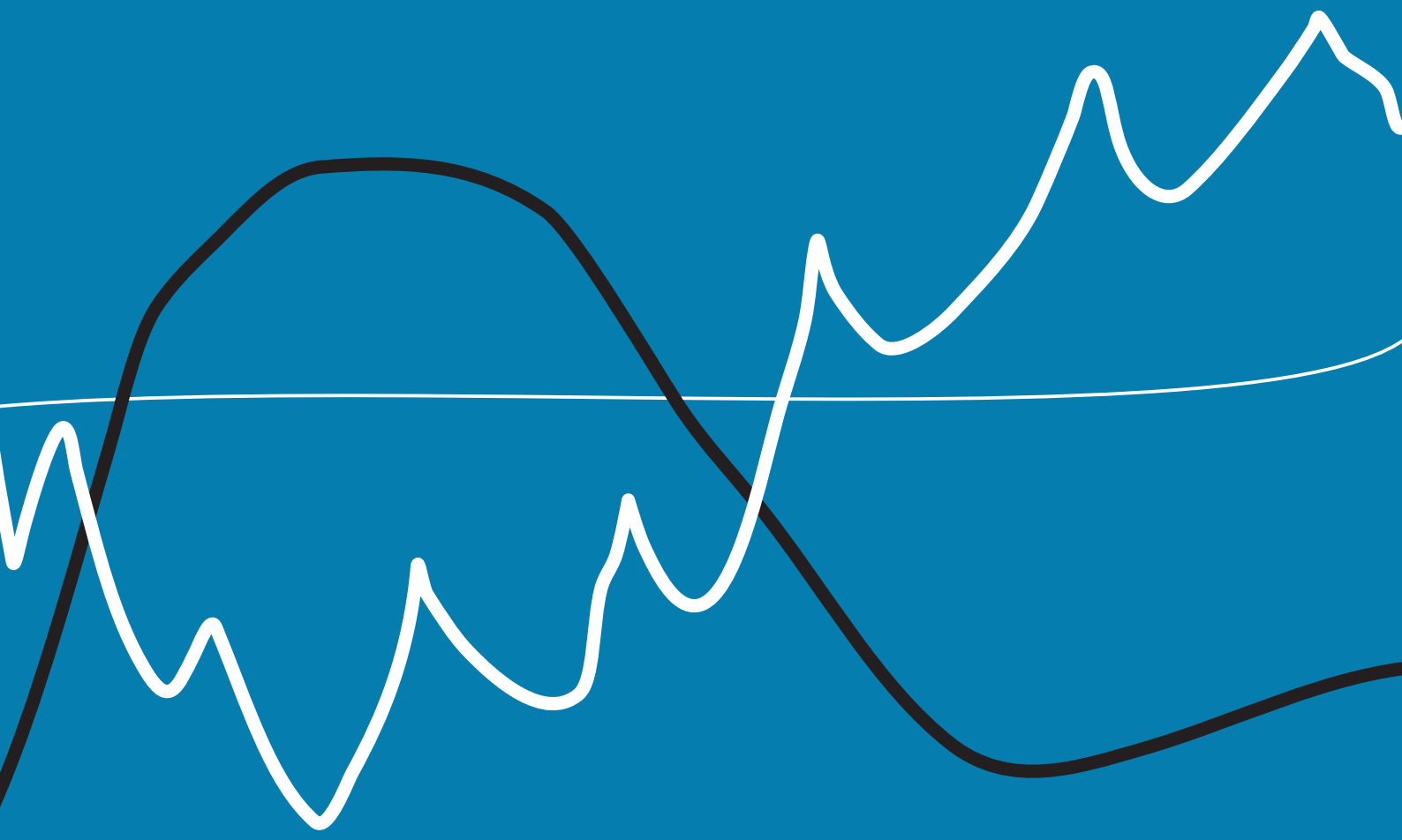
DTU Compute
Section of Cognitive Systems

Investigation of Deep Probabilistic Surrogate-models in Bayesian Optimization

Master Thesis



Abstract

Bayesian optimization (BO) is the leading methodology in sample-efficient optimization and widely used in hyperparameter tuning in machine learning and deep learning.

From a decision theoretical standpoint, assuming the probabilistic surrogate model to be correct is crucial for the correct decision. We will investigate if other surrogate models could be better than the BO default: Gaussian process (GP).

This thesis investigates surrogate models such as Mixture regression and Bayesian neural networks(BNN) and compares them against GP. To have a good chance of analyzing the models, the focus is first to understand and present the theory and second to apply the models in a BO setting.

After understanding and presenting the different surrogate models and corresponding inference methods. Tests were conducted on four self-made 1D Problems and a commonly used benchmarking problem set. The mixture and BNN regression results were promising for classes of problems with discontinuities and multimodality, and the GP was found best for most tests. All surrogate models were Gaussian approximated in the BO tests, especially this compromisation of the mixture regression models sparks even more promise for their positive results.tests

Contents

1 Introduction

Optimization plays an important role in our everyday life, science development, product design, and much more. Examples of optimization could be choosing the optimal way to commute from A to B, deciding what songs should land on your playlist, or constructing the strongest possible bridge using limited material. In general, optimization is the methodology of choosing the best decision among a set of possible decisions. Often we try to quantify how good a decision is: A specific bus takes 20 min to go from A to B, you rate a specific song 4 out of 5, and a particular bridge design costs 10 million DKK. Suppose it is possible to come up with a quantification of how good a decision is in terms of a real number. In that case, we can formulate the optimization problem as a *mathematical optimization problem* [1, p. 1]:

$$\min_{x \in \mathcal{X}} f(x),$$

where the functional $f : \mathcal{X} \to \mathbb{R}$ is called the *objective function* and $\mathcal{X}$ is the set of possible decisions. Throughout this thesis, we refer to optimization as minimizing the objective function. Solving the mathematical optimization problem is an active research field, and many algorithms have been developed to find the minimum of the function $f(\cdot)$ [1].

Evaluation of the objective function can be cheap (e.g., if it just requires summing/multiplying numbers) or highly expensive (e.g., if it involves human rating, large simulation, or physical experiments). In the latter case, we want to avoid evaluating the objective function as much as possible - we want to use *sample-efficient* optimization. The overall topic of this thesis, *Bayesian optimization* (BO), is one of the preferred frameworks for sample-efficient optimization [2].

Bayesian optimization is a probabilistic surrogate-based optimization methodology: Assuming some initial samples $\{(x_1, f(x_1)), \ldots, (x_n, f(x_n))\}$ from a highly expensive objective, a cheap (surrogate) function is fitted to the samples. The next sample is found by minimizing the surrogate, and the process is repeated. Bayesian optimization seeks to enhance this procedure with probability theory, where the surrogate function becomes a probabilistic (Bayesian) regression model. The most common surrogate model is a Gaussian Process (GP), as it encapsulates the uncertainty very well, and its inference procedure (computing answers to probability queries like $p(y|x)$) is exact.

Even though GP has proven suitable for many cases, there will be problems where its assumptions do not hold. For example, the commonly seen GP with an isotropic kernel (covariance between two points is invariant to translation in input) yields a strong assumption about the continuity of the objective function and that the objective function behaves similarly throughout the domain $\mathcal{X}$. In Figure 1.1 we see an example of how the GP's uncertainty quantification becomes unreasonable due to a discontinuity in the underlying objective function. In some areas, it is prevalent to have these discontinuities, for instance, in material discovery [3]. To accommodate the strong assumptions of the GP the literature introduces more flexible kernel functions; however, this introduces additional hyperparameters. Since we often only deal with a small amount of data, tuning and computation of these more complex GPs can be significantly challenging [3].

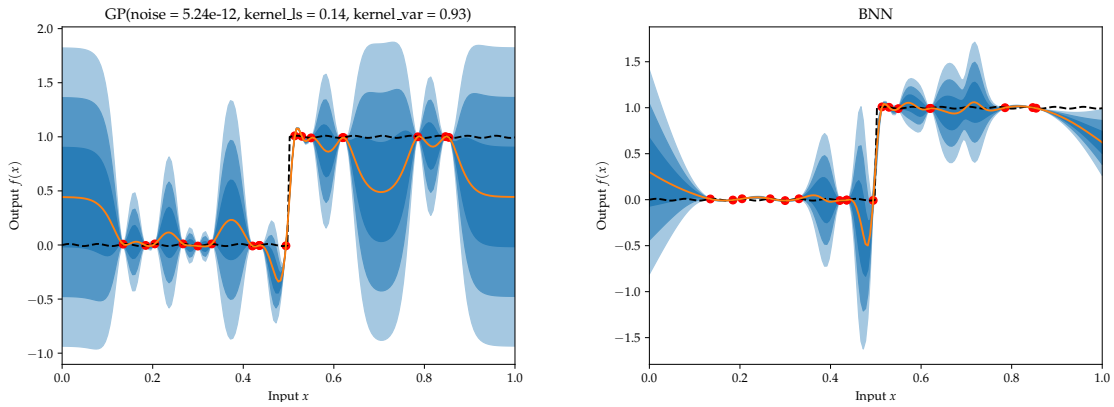


Figure 1.1: Gaussian process (GP) and Bayesian neural network (BNN) fitted to 18 data points. The objective function is the dashed black line. This examplifies how a discontinuity makes the standard implemented GP (optimized with emperical Bayes) expresses large uncertainties to all other areas in the domain $[0, 1]$, while the Bayesian NN only express uncertainty where the discontinuity happens at $x = 0.5$

GPs are attractive models since they allow for exact inference, a closed-form expected improvement (see Section 2.3.1) and, in general, provide a satisfying uncertainty quantification. However, as we saw in the Figure 1.1 we can maybe do better — especially if we have some insights into the behavior of the objective function (i.e. yielding a more transparent black-box optimization problem). A better surrogate model ultimately implies fewer samples to reach an optimal solution, and can, thereby, potentially save work hours, money, or energy. Therefore, the investigation of the performance of other surrogate models is highly relevant. In this thesis, we aim to create surrogate models alternative to GPs, which can perform better on certain classes of (complex) problems, and have comparable performance on most classes of problems.

1.1 Project limitation and contibution

Even though it would be interesting to examine different types of GPs such as deep kernel learning [**deepkernel**], we limit the scope of this project to deal with the following surrogate models:

- GP with Matérn kernel
- Bayesian neural network (BNN)
- Mixture regression (Gaussian mixture model, kernel density estimator, sum-product networks)

and we choose only to test the Bayesian optimization with the widely used acquisition function: *expected improvement*. Furthermore, we only deal with problems in continuous domains $\mathcal{X} \in \mathbb{R}^m$. The proposed hypothesis is,

1. Mixture regression models and BNNs can be effective surrogate models performing better than GPs in some complex BO problems.

Note that we here use ”performance” to describe *sample efficiency*, i.e., how few evaluations/samples of the objective function are necessary to find the minima. So here, it is assumed that the objective function is so expensive (whatever that means to the stakeholder) that its cost outweighs the power and time spent on the surrogate modeling and optimization.

We test the performance of the surrogates models on 4 self-made problems and

1.2 Related work

Might delete this Here we give a short overview of research in different surrogates for Bayesian optimization and the very related field of active learning. The different surrogate models, which are in the research we found,

Mostly the work done on surrogate models are

- Gaussian process
- Bayesian neural network
- Random forest regression
- Kernel density estimator
- Bayesian multivariate adaptive regression splines (BMARS)
- Bayesian additive regression trees (BART)

Since inference time of the GP scales cubic with the number of samples, often research in alternative surrogate models has its primary focus on lowering the computational complexity of inference while showing the sample efficiency is (or almost is) as good as the GPs. A popular model in the current time is a (deep) neural network and using a neural network as a probabilistic surrogate model (i.e. Bayesian Neural Network [4] or as basis functions in linear regression [5]) yields only linear complexity. However, as mentioned above, this thesis assumes that the objective function is always more costly than the inference cost. And cubic complexity for a GP does not matter for the small number of samples, which is often the case for highly expensive Bayesian optimization problems.

Inference complexity is also the main focus of the Ph.D. thesis "Sample-efficient Optimization Using Neural Networks" from 2020 [6], but his chapter 3 showcases empirically that using Bayesian neural networks as surrogate models performed better, or at least comparable to GPs on a wide number of problems. The performance difference was more evident for high-dimensional problems.

The 2021 paper "Bayesian optimization with adaptive surrogate models for automated experimental design" [3], focus on the sample efficiency of the BO applied to autonomous materials discovery, which yields a relatively high-dimensional design space and non-smooth patterns of objective functions. The paper shows that using Bayesian multivariate adaptive regression splines and Bayesian additive regression trees as alternative surrogate models outperform GP significantly, for complex BO problems.

Active learning is closely related to Bayesian Optimization, but here the focus is on learning the underlying function using as few samples as possible instead of just finding its minima. In active learning, a Gaussian process is also very common, but the paper "Active Learning with Statistical Models" [7] investigates using Gaussian mixtures and kernel estimator in active learning, i.e. as a surrogate model for selecting the next samples. These regression models are not seen much in the literature; They are modeling the joint distribution of x and y, and using the conditional distribution $p(y|x)$ as the regression model. The results were ... ??

1.3 Structure of this thesis

The structure of this thesis is as follows: Firstly, build up the foundation around Bayesian optimization. Secondly, present different surrogate/Bayesian regression models. Thirdly, show the results on different types of problems on Bayesian regression performance and on Bayesian optimization performance. Finally, a discussion and conclusion are presented. In more detail the different chapters are,

1. **Introduction**
2. **Bayesian optimization**: First we establish the relevance for BO as a sample-efficient solver. Secondly, we explain the two components of BO: A surrogate/Bayesian regression model and an acquisition function.
3. **Discriminative models surrogates**: Gaussian processes and Bayesian neural networks are presented and discussed in terms of inference procedures and regression performance.
4. **Generative models as surrogates**: Since the mixture models are not inherently Bayesian, we present the conditional distribution in a Bayesian setting. Next, the three different mixture regression models are presented: Kernel density regression, Gaussian mixture regression, and the novel (in a regression setting) Sum-product networks.
5. **Results**: Testing regression and Bayesian Optimization performance (sample-efficiency) on 4 homemade test functions and the popular benchmark COCO.
6. **Discussion** and **conclusion**

1.4 Notation

Following is the general notation used in this thesis, but some chapters (e.g. about GPs) will alter the notation a bit.

- $y \in \mathbb{R}$: Observation or sample of $f(x)$ (potentially with noise).
- $f(\cdot) : \mathcal{X} \to \mathbb{R}$: Objective function.
- $x \in \mathcal{X}$: Location or point in the sample space.
- $\mathbf{x}$: Set of location or points in the sample space.
- $\mathbf{y}$: Set of observations or samples for corresponding $\mathbf{x}$.
- $\mathcal{D}$: Collection of samples in the optimization landscape.
- $\mathcal{N}(x|\mu, \Sigma)$: The density of a normal distribution evaluated in x.
- *optimization landscape*: joint set of points in the domain and the objective function evaluated in the points, i.e. $\{(x, f(x)) \in \mathcal{X} \times \mathbb{R} | x \in \mathcal{X}\}$.
- $y \sim p(y)$: A realization of y with the density $p(y)$

1.4.1 Bayesian notation

Throughout this thesis we will be using Bayesian notation, i.e. $p(x) := P(X = x)$ is the probability density function of the random variable X evaluated in x and $p(y|x) := P(Y = y|X = x)$. Furthermore, writing $p(y^2|x)$ means $P(Y^2 = y^2|X = x)$ and **not** $P(Y = y^2|X = x)$. Another noteable notation is the expecation of $g(y)$ with respect to the preditive distribution

$$E_{p(y|x,\mathcal{D})}[g(y)] = \int g(y)p(y|x, \mathcal{D})dy.$$

2 Bayesian Optimization

This chapter will introduce Bayesian optimization. We start with a general introduction to the concept of optimization (mainly based on [8]), which culminates with the introduction of the idea of Bayesian optimization (BO). Next, we dive into the first BO component: The Bayesian regression methodology. Finally, the concept of an acquisition function is introduced, with a focus on expected improvement and a brief description of the other types.

2.1 Optimization methodology

Given an objective function $f : \mathcal{X} \to \mathbb{R}$, where the domain $\mathcal{X}$ could be a subset of $\mathbb{R}^n$[1], optimization is a methodology which seeks to find an optimal point, x^*, and value $f^* = f(x^*)$, given as

$$x^* \in \arg\min_{x \in \mathcal{X}} f(x), \qquad f^* = \min_{x \in \mathcal{X}} f(x) = f(x^*). \tag{2.1}$$

Solving this problem (close to) exact is often intractable except for rare cases e.g. if f is convex and analytically directly solvable or the domain of f is very limited. The following example with linear least squares is an example of such a problem.

Example: Direct solution method
The unconstrained linear least squares [1],

$$\min_{x \in \mathbb{R}^n} f(x) := ||Ax - b||_2^2$$

where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$, is a convex problem, i.e. finding x^* such that $\nabla f(x^*) = 0$ is equivalent to finding the solution to the problem. Assuming $A^T A$ is invertable, linear least squares can be solved directly by the normal equations,

$$\nabla f(x) = 2A^T Ax + 2b^T A = 0 \quad \Leftrightarrow \quad x^* = (A^T A)^{-1} A^T b$$

Even if the gradient is given analytically, the problem can be non-convex, implying that the solution is found among a potentially infinitely large set of stationary points ($\nabla f(x) = 0$) - this might be tedious or impossible. Therefore, when the problem is not directly solvable, mathematical optimization takes an indirect approach: Design a sequence of experiments that reveal information about the objective function. This information can hopefully lead us to the solution of (2.1). This general way of sequentially solving is presented in Algorithm 1 [8].

Algorithm 1 Sequencial Optimization [8]

Input: Initial dataset $\mathcal{D}$ ▷ can be empty
while Temination is not reached **do**
  $x \leftarrow \text{policy}(\mathcal{D})$ ▷ select next observation location
  $y \leftarrow \text{observe}(x)$ ▷ observe objective function at chosen location
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(x, y)\}$ ▷ update dataset
return: $\mathcal{D}$

[1] The thesis only work with this subset $\mathcal{X} \subset \mathbb{R}^n$

Given data points in the *optimization landscape* a policy selects a location $x \in \mathcal{X}$ where we make our next observation. Policies can be deterministic or probabilistic; examples of each type could be grid search and random search. The next observation provides us a $y = f(x)$ value[2], which combined with x is included in the available data $\mathcal{D}$. Finally, a stopping criterion decides whether to repeat or terminate the procedure. We will now present how different examples of well-known optimization routines fits into Algorithm 1.

Grid search
In grid search values along each dimention in $\mathcal{X}$ is seleced and combined with each other, which thereby defines a parrellel grid in the optimization domain $\mathcal{X}$. All grid points are ordered and systematically selected. In the context of algorhtim 1 we define the grid search policy as

$$\text{policy}_{GS}(\mathcal{D}) = x_{|\mathcal{D}|+1},$$

assuming $x_1, x_2, \ldots, x_m \in \mathcal{X}$ are the ordered grid points and the size of the obtained data is $|\mathcal{D}|$. Termination will happen when $|\mathcal{D}| = m$.

Random search
In random search a point is randomly drawn e.g. from a uniform distribution supported over the domain space $\mathcal{X}$,

$$\text{policy}_{RS}(\mathcal{D}) = x, \quad x \sim \text{Unif}(\mathcal{X}).$$

Gradient descent
Gradient descent (GD) is the most simple gradient-based optimization approach. The gradient of a continuous function points in the most ascending direction at the location it is evaluated. GD iteratively minimize the objective function by taking steps using opposite gradient direction, i.e. the most descending direction, weighted with a stepsize η. This yields the policy:

$$\text{policy}_{GD}(\mathcal{D}) = x_n - \eta \nabla f(x_n)$$

where we, for a brief moment, in Algorithm (1) modify y to be a vector since the observation is given as:

$$\text{observe}_{GD}(x) = [f(x), \nabla f(x)]$$

2.1.1 Sample-efficient optimization

Note that grid search, random search, and gradient descent are policies that entirely ignore the available data. Ignoring potentially valuable information is a shame if the objective function is expensive to evaluate. And indeed, improvements of the gradient descent algorithm, such as momentum and quasi-newton methods, indirectly remember and exploit obtained data, $\mathcal{D}$. They are examples of so-called *sample-efficient* solvers since they need fewer y-samples to minimize $f(\cdot)$. Choosing a more sample-efficient solver ultimately costs extra time/energy, due to the extra work of storing and exploiting the collected information for every iteration. In the end, solving the optimization problem can be divided into the following components,

- N_{iter}: The number of iterations to reach an acceptable solution. This number depends on the solver. If N_{iter} is relatively small, the solver is called sample-efficient.

[2]Since the evaluation of $f(\cdot)$ might be noisy or inprecise, we more generally write $y = observe(x)$. The typical connection between $f(x)$ and y are $y = f(x)$ or $y = f(x) + \epsilon$, where ϵ is additive white noise, but different connections might be present as well.

- C_{policy}: The solvers cost per iteration, i.e. what is the cost of calculating policy($\mathcal{D}$), where cost is typically in terms of time and power usage.
- C_{observe}: Cost per evaluation of the objective function, i.e. the cost of observe(x), which can be in terms of power consumption, human resources, simulation time etc.

Assuming same cost pr iterations and for all evaluations, the total optimization cost, C_{total}, is given as

$$C_{\text{total}} = N_{\text{iter}} \cdot \left[C_{\text{policy}} + C_{\text{observe}}\right]$$

A sample efficient solver has a large C_{policy} and a small N_{iter}, which is completely opposite for a simple optimization scheme like random search. Choosing the right optimization solver depends highly on C_{observe}. For small C_{observe} it is favorable to find a good trade-off between N_{iter} and C_{policy}: In deep learning the solver Adam is very popular, due to its cheap but advanced policy [9]. This project deals with a dominating observation cost, i.e. the cost of the policy is assumed neglectable $C_{\text{observe}} >> C_{\text{policy}}$. So focus is not on finding a cheap policy, but rather on improving the number of iterations to reach the minima N_{iter}.

Surrogate-based optimization

In surrogate-based optimization all available data is fitted by a cheap-to-evaluate approximation to the objective function - this approximation is called a *surrogate model*, $f_{\text{sur}}(x)$. Examples of surrogate models could be a radial basis function or a support vector regressor [10]. The next point is chosen as the point where the surrogate model is minimized.

$$\text{policy}_{\text{sur}}(\mathcal{D}) = \min_x f_{\text{sur}}(x)$$

where $f_{\text{sur}}(x) \approx f(x)$ for x close to the data $\mathcal{D}$. And we hope the approximation holds for x far away from the the data.

Figure 2.1 illustrates the idea of a sample-efficient solver; the solver adapts the obtained data into its decision and, thereby, faster discovers the minimum.

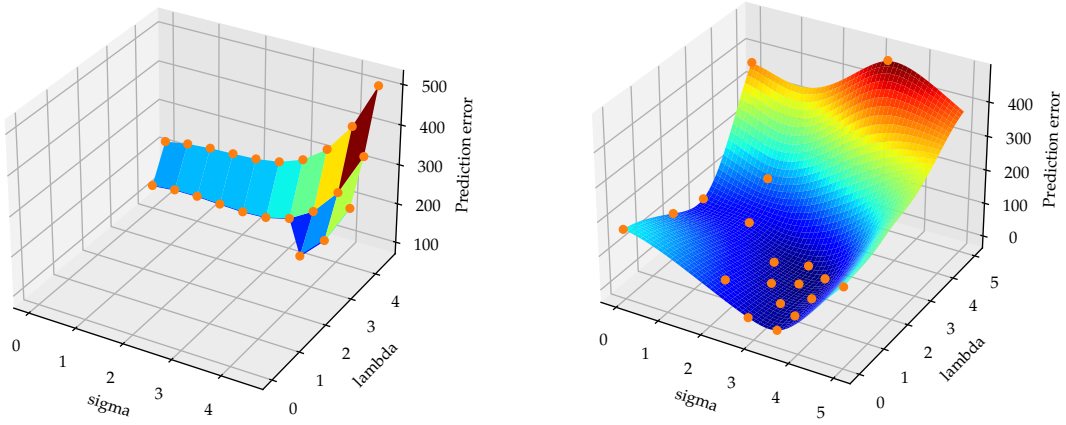


Figure 2.1: Example of an optimization task tuning a parameterised regression model with parameters λ and σ, on a test set, i.e. minimization of prediction error. We see the first 23 evaluations out of 100 in grid search vs 23 evaluations using a sample-efficient solver (Bayesian optimization).

2.1.2 Exploitation and exploration

A carefully balanced exploration and exploitation trade-off is key to effective global optimization [8, p. 11]. They are explained here:

1. Exploitation: Exploiting the obtained information $\mathcal{D}$ to find the best point, i.e. sampling in areas where the objective function is expected to be low.
2. Exploration: Obtaining new information about the characteristics of the objective function i.e. improve $\mathcal{D}$ for future decisions.

Essentially, the only explorative phase of the deterministic surrogate optimization is the first initial samples from the objective, i.e. when creating the initial $\mathcal{D}$. If done poorly, the optimization might get stuck in a local minimum. This is a consequence of their nonexisting ability to express uncertainty about the objective function. Deterministic surrogate models do not pay attention to less-explored regions, which is the essential characteristic of a global optimization method [11].

Bayesian formalism is a powerful and sound tool for incorporating uncertainty into the surrogate model. Prior to any observed data, we can easily integrate our beliefs about the distribution of the objective function. When observing data, the likelihood and prior collaborate to create a predictive (posterior) distribution, $p(y|x, \mathcal{D})$. The next location x to explore is now determined with a theoretically motivated policy (acquisition function) which might balance the exploitation and exploration tradeoff.

Moreover, Bayesian optimization (or *probabilistic* surrogate-based optimization) is motivated by its ability to deal with noisy objective functions.

2.1.3 Noisy objective functions

Many optimization algorithms (e.g. gradient descent and grid search) assume *exact* evaluations of the objective function. However, this assumption is often wrong, especially for objective functions with real-life experiments, imperfect simulations, and human interaction where measurement noise is well known. A potentially noisy objective function is the main reason why we in Algorithm (1) use the terminology observe(x) and not just evaluate$(f(x))$. The observation model is typically noisy and described as

$$y = f(x) + \epsilon,$$

where ϵ is the measurement error, this is typically assumed to be Gaussian with zero mean and a variance σ^2 (which could depend on x in a heteroskedastic setting) and implies a Gaussian observation model,

$$p(y|f(x), \sigma_x^2) = \mathcal{N}(y|f(x), \sigma_x^2),$$

where $f(x)$ is the mean value and σ_x^2 is the variance (with the subscript indicating the potentially dependency on x).

> **Note: "Sampling" from objective function**
> The terminology *to sample* is refering to a probabilistic observation model. Formally, we can extend this model to deal with noiseless observations as well, simply by setting $\sigma = 0$ and letting the model colaps into a Direct delta distribution, $p(y|f(x)) = \delta(y - f(x))$, i.e. all probability mass for y is on the value $f(x)$ giving the observation sample $y = f(x)$.

Bayesian optimization or probabilistic surrogate-based optimization deals with both noiseless and noisy objective functions, as it defines a Bayesian regression model over the observations.

Bayesian Optimization

Bayesian optimization is a *probabilistic* surrogate-based optimization methodology. Here a cheap probabilistic regression model $p(y|x)$ is fitted to the the observations $\mathcal{D}$ and in contrast to (deterministic) surrogate-based optimization, it is not possible right away to find the minima in the cheap surrogate model; first, we need to interpret the meaning of minima in a probabilistic regression model. This interpretation is done through a so-called acquisition function (AQ) - more about this later. The policy in the context of Algorithm 1 is as follows,

$$\text{policy}_{BO}(\mathcal{D}) = \max_x AQ(p(y|x, \mathcal{D})) \tag{2.2}$$

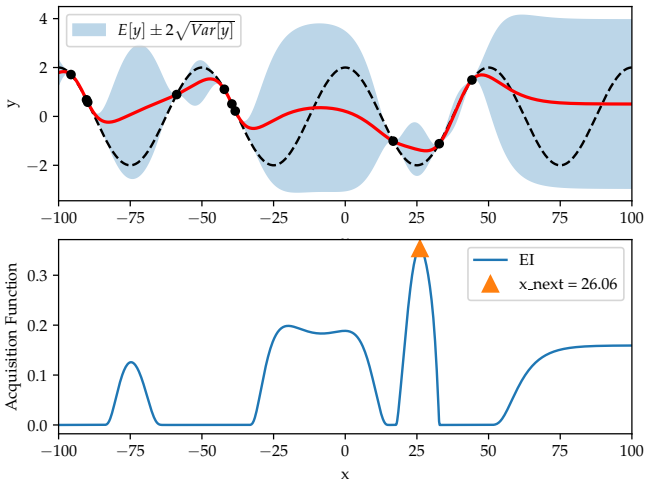


Figure 2.2: Top: Bayesian regression model (Gaussian Process) is fitted to the observed data, which are sampled from the underlying objective (black sin-function). Bottom: The expected improvement *acquisition function* is maximied at the orange arrow, i.e. the location of the next sample. $\text{policy}_{BO}(\mathcal{D}) = 26.06$

2.2 Bayesian regression

Whereas traditional regression workflow is the following: Given data, choose the best fitting model parameters, make predictions using those parameters. The Bayesian framework allows us to skip the dependency of a single set of parameters and instead use *all possible* parameters by treating the set of parameters as a random quantity, $\theta \sim p(\theta|\mathcal{D})$, where some values/realizations of θ are more probable than others given data. In Bayesian regression we are interest is the predictive posterior distribution,

$$p(y|x, \mathcal{D}) = \int p(y, \theta|x, \mathcal{D}) d\theta \tag{2.3}$$

$$= \int p(y|x, \theta) p(\theta|\mathcal{D}) d\theta, \tag{2.4}$$

where the posterior $p(\theta|\mathcal{D})$ gives weighting to the proposed regression model $p(y|x, \theta)$. Note that the second equation is true because of the probability chain rule and that y is fully described by the parametric model $p(y|x, \theta)$ and the parameters θ are fully described by the posterior distribution $p(\theta|\mathcal{D})$.

Background: Bayesian methodology

In Bayesian modelling, $p(\theta|\mathcal{D})$ is the posterior distribution, and it is linked to the likelihood

$p(\mathcal{D}|\theta)$ and prior $p(\theta)$ via Bayes rule,

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \propto p(\mathcal{D}|\theta)p(\theta),$$

where the evidens $p(\mathcal{D})$ act as a propotionality constant since $p(\theta|\mathcal{D})$ is only a function of θ. In the case of regression, we always condition on $\mathbf{x}$[a].

$$p(\theta|\mathcal{D}) = \frac{p(\mathbf{y}|\mathbf{x},\theta)p(\theta|\mathbf{x})}{p(\mathbf{y}|\mathbf{x})} \propto p(\mathbf{y}|\mathbf{x},\theta)p(\theta|\mathbf{x})$$

The modeling task to is to specify a likelihood $p(\mathbf{y}|\mathbf{x},\theta)$, which encodes how likely the model θ explian the data, and a prior $p(\theta|\mathbf{x})$, which encodes our prior belief about the model.

[a] By abusing notation, we could define the data as "conditional data", $\mathcal{D} := \mathbf{y}|\mathbf{x}$

2.2.1 Surrogate model

A surrogate model in a Bayesian optimization setting (i.e. $p(y|x,\mathcal{D})$ in (2.2)) is a Bayesian regression model. The most used surrogate model is a Gaussian Process. But there have been investigations on other surrogates, such as Bayesian neural networks and Bayesian regression trees. These are all discriminative models, and another approach we focus on in this project is to model y and x jointly in a so-called generative model, $p(x,y)$. A generative model can be used implicitly as a surrogate from the conditional distribution of y given x, $p(y|x)$.

In this thesis, the Bayesian regression models investigated as Bayesian optimization surrogates are the following:

- Gaussian process (GP)
- Bayesian neural network (BNN)
- Kernel density regression (KDE)
- Gaussian mixture regression (GMR)
- Sum-product networks (SPN)

We now introduce the concept of inference, which is necessary for using the probabilistic surrogate models in Bayesian Optimization.

2.2.2 Inference of surrogate models

Inference is the process of computing answers to queries about a probabilistic model after observing data. In Bayesian regression, the query is the predictive distribution, $p(y|x,\mathcal{D})$, as we are interested in the distribution of y given x and already observed data, $\mathcal{D}$. This often indirectly create the posterior query, $p(\theta|\mathcal{D})$, the probability of model parameters θ given data $\mathcal{D}$. Lastly, it is also inference when we train a Gaussian mixture model or SPN using the expectation-maximization algorithm (EM) since we are iteratively answering the query $E_{p(z|\theta^{(k)})}[z|\theta]$.

We distinguish between two different ways of inference: Exact and approximate inference. It is *exact inference* when a probabilistic query is calculated exact. It is possible to calculate exact inference on the predictive distribution for the Gaussian mixture model, Sum product network, and Gaussian processes. Models which allow for exact inference have a powerful advantage over the models with approximate inference since we can guarantee the answers to the queries are correct; however, they are usually also less expressive (unable to explain complicated models).

When it is not possible to answer a probabilistic query exact, we can approximate the answer using *approximate inference*. When dealing with complicated and expressive statistical models, exact

inference is often intractable, and we need to use approximate inference. Approximate inference is a broad category of methods, which includes variational inference and Markov chain Monte Carlo (MCMC). The two Bayesian Neural networks, we deal with in this project, Bohamiann and Numpyro BNN are similar regression models but are inferred using two different MCMC methods, Hamiltonian Monte Carlo - giving different results. As revealed later (see in section 3.2) approximate inference might be flawed and inexact.

Model	**Predictive inference**	**Learning**
GP	Exact $O(n^3)$	Emperical Bayes
Numpyro BNN	No U-Turn Sampler	
Bohamiann BNN	Adaptive stochatic HMC	
Kernel density regression	Exact $O(n)$	
Gaussian mixture regression	Exact $O(K)$	EM
SPN	Exact $O(E)$	EM $O(E)$

Table 2.1: Overview of inference methods applied on the statistical models used in this project. E is the number of edges in the SPN. n is the number of data points. $K \leq n$ is the number of mixture components. We will soon learn that for an SPN the number of mixture components is exponentially larger than the number of edges i.e. $E << K$. In theory MCMC methods samples from true the posterior distribution, and do not need any fitting/learning.

2.3 Acquisition function

Given a correct predictive distribution $p(y|x\mathcal{D})$ the next step in Bayesian optimization is to select the next location $x \in \mathcal{X}$ to sample from. The next location is chosen according to a so-called acquisition function (AQ function), which balances out the well-known concept of exploitation and exploration. It is exploitation if the next chosen location is found according to its average improvement. It is exploration if the next point is chosen in a region of high uncertainty and thereby helps lower the overall uncertainty. First, we will look at the acquisition function used in the thesis: Expected improvement. Secondly, we shortly present other different types of acquisition functions. In Figure 2.3 we see three different acquisition functions. The expected improvement (EI) is known for being biased towards exploitation. In contrast, the negative lower confidence bound (LCB) has a parameter β, which can be tuned to make it focus more on exploration.

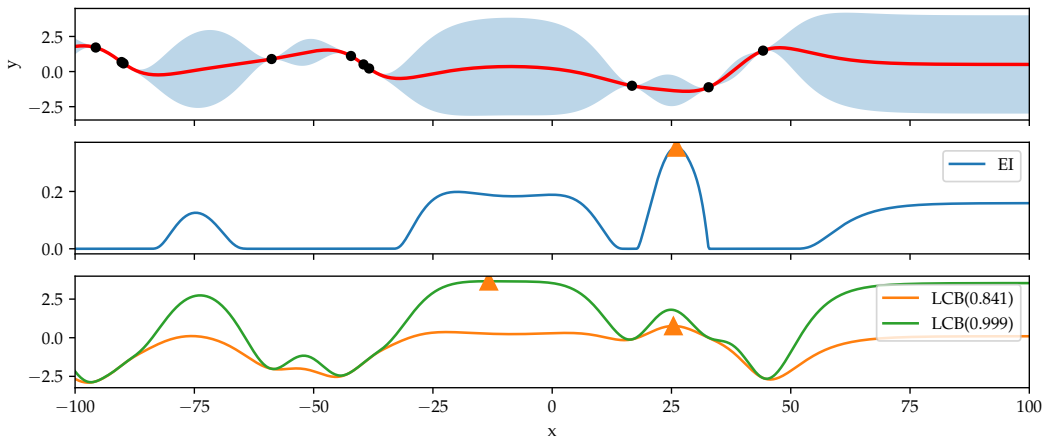


Figure 2.3: The same regression model and points as Figure 2.2, but with three different acquisition functions: Expected improvement and negative lower confidence bound for two different lower quantiles 0.841 and 0.999. The latter yields more exploration.

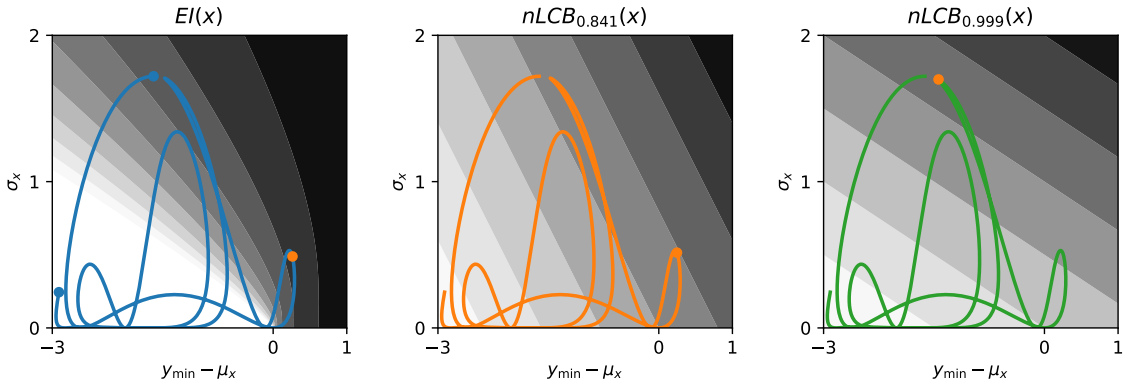


Figure 2.4: Contourplot of expected improvement (EI) and lower confidence bound (LCB) for two different quantiles for different (Gaussian) predictive uncertainties $\sigma_x = \sqrt{\mathbb{V}ar_{p(y|x,\mathcal{D})y]}}$ versus the average improvement $y_{\text{min}} - \mu_x$, where $\mu_x = E_{p(y|x,\mathcal{D})}[y]$. Darker colors indicates higher values. The colored lines are the mapping $x \mapsto (\sigma_x, y_{\text{min}} - \mu_x)$ for $x = [-100, 100]$ for the Bayesian regression function in Figure 2.3 - and thereby explains how the acquisition functions balances exploitation and exploration. The orange dot represent the point maximizing the acquisition function

2.3.1 Expected improvement

A popular choice of acquisition function is expected improvement,

$$EI(x) = \mathbb{E}_{p(y|x,\mathcal{D})}[\max(0, y_{\text{min}} - y)]$$

where we only consider the values y, which improves the current best value in the expectation of the predictive distribution, $p(y|x, \mathcal{D})$. Therefore, a x which yield a bad predictive mean value $\mathbb{E}_{p(y|x,\mathcal{D})}[y] > y_{\text{min}}$ might still be maximizing the expectated improvement, if the predictive uncertainty is very large at x. Figure 2.4 illustrates that a large uncertainty in the predictive distribution (represented as the predictive variance) can lead to relative large values even for non-improving mean predictions.

> **Note: Why defining expected improvment with max**
> Note that $\max(0, \cdot)$ is important since the Bayesian optimization othervise reduces to a simple non-probabilistic surrogate-based optimization method,
>
> $$\mathbb{E}_{p(y|x,\mathcal{D})}[y_{\text{min}} - y] = y_{\text{min}} - \mathbb{E}_{p(y|x,\mathcal{D})}[y]$$
>
> i.e. maximizing the above is equivalent to maximizing the predictive mean, and thereby we loose all the valuable information about the predictive uncertainties from the Bayesian regression model.

Exact expected improvement

In the following derivation we assume the predictive distribution can be approxiamted by a normal distribution dependent on the point of interest x and the data $\mathcal{D}$ (note for the GP it is in fact not an approximation),

$$p(y|x, \mathcal{D}) \approx \mathcal{N}(y|\mu(x, \mathcal{D}), \sigma^2(x, \mathcal{D}))$$

where we will change to a less clumpsy notation $\mathcal{N}(y|\mu_x, \sigma_x^2) := \mathcal{N}(y|\mu(x, \mathcal{D}), \sigma^2(x, \mathcal{D}))$. This is completely fine since x is fixed (and $\mathcal{D}$ is fixed) when evaluating the expected improvement in a point x. Furthermore, the density of a standard normal distribution is denoted $\phi(\cdot) := \mathcal{N}(\cdot|0, 1)$, and the cumlative density function (CDF) of a standard normal distribution is denoted, $\Phi(\cdot) :=$

$\int_{-\infty}^{\cdot} \phi(\epsilon)d\epsilon$. We will now see that the normal approximation of the predictive distribution yiels closed form solution to the expected improvement function,

$$\begin{aligned}
\mathbb{E}_{\mathcal{N}(y|\mu_x,\sigma_x^2)}[\max(0, y_{\min} - y)] &= \int \max(0, y_{\min} - y)\mathcal{N}(y|\mu_x, \sigma_x^2)dy \\
&= \int_{-\infty}^{y_{\min}} (y_{\min} - y)\frac{1}{\sigma_x}\phi\left(\frac{y-\mu_x}{\sigma_x}\right)dy \\
&= \int_{-\infty}^{\frac{y_{\min}-\mu_x}{\sigma_x}} (y_{\min} - \mu_x - \sigma_x\epsilon)\frac{1}{\sigma_x}\phi\left(\epsilon\right)\sigma_x d\epsilon \\
&= \int_{-\infty}^{u} \sigma_x \cdot (u - \epsilon)\phi(\epsilon)d\epsilon \\
&= \sigma_x \cdot \left(u \cdot \int_{-\infty}^{u} \phi(\epsilon)d\epsilon + \int_{-\infty}^{u} (-\epsilon)\phi(\epsilon)d\epsilon\right) \\
&= \sigma_x[u\Phi(u) + \phi(u)]
\end{aligned}$$

where $u := \frac{y_{\min}-\mu_x}{\sigma_x}$.

Note: Derivation details
To understand the identity $\phi(u) = \int_{-\infty}^{u}(-\epsilon)\phi(\epsilon)d\epsilon$ used in the last equality, we first see that the antiderivative is $\phi(\epsilon) = \frac{1}{\sqrt{2\pi}}\exp\left(\frac{-\epsilon^2}{2}\right)$,

$$\frac{d}{d\epsilon}\phi(\epsilon) = \frac{1}{\sqrt{2\pi}}\frac{d}{d\epsilon}\exp\left(\frac{-\epsilon^2}{2}\right) = \frac{1}{\sqrt{2\pi}}\exp\left(\frac{-\epsilon^2}{2}\right)(-\epsilon) = -\epsilon\phi(\epsilon)$$

and evaluating the rieman integral is equivalent to evaluate the antiderivative in its boundaries, giving the solution,

$$\int_{-\infty}^{u}(-\epsilon)\phi(\epsilon)d\epsilon = [\phi(\epsilon)]_{-\infty}^{u} = \phi(u) - 0 = \phi(u)$$

We can also explicily write the expected improvement as,

$$EI(x) = (y_{\min} - \mu_x)\Phi\left(\frac{y_{\min}-\mu_x}{\sigma_x}\right) + \sigma_x\phi\left(\frac{y_{\min}-\mu_x}{\sigma_x}\right)$$

where the first part can be interpretted as exploitation (favouring points with a large average improvement $I(x) := (y_{\min} - \mu_x)$) and the second part can be seen as exploitation (favouring points with high uncertainty.). This can also be seen in Figure (2.4), where it is clear that the expected improvement is growing for increasing average improvement $I(x)$ and also for increasing prediction uncertainty σ_x.

Approximate expected improvement
If the predictive distribution is non-Gaussian, it is either possible to approximate it as a Gaussian (By using the mean and variance of the predictive distribution to define the Gaussian approxima-

tion) or calculate the expected improvement approximately as follows,

$$E_{p(y|x,\mathcal{D})}[\max(0, y_{\min} - y)] = \int \max(0, y_{\min} - y)p(y|x, \mathcal{D})dy$$
$$\approx \frac{1}{K}\sum_{k=1}^{K} \max(0, y_{\min} - y^{(k)})$$

where $y^{(k)}$ are samples from the predictive distribution.

2.3.2 Other acquisition functions

Expected improvement is just one choice of acquisition function, we now shortly present three different acquisition functions, Lower confidence bound, entropy search (mutual information acquisition) and probability of improvement. As mentioned, we only use expected improvement in the experiments of this thesis.

Lower confidence bound

Lower confidence bound acquisition function [8, p. 145][3] is parameterised by a confidence parameter $\pi \in [0, 1]$ which defines the preditive $(1 - \pi)$-quantile at x,

$$q_{1-\pi}(x) = \inf\{y^* | \mathbb{P}(y \leq y^* | x, \mathcal{D}) \geq (1 - \pi)\},$$

i.e. the prediction $y \sim p(y|x, \mathcal{D})$ will only be be less than the lower bound $q_{1-\pi}(x)$ with a tunable probability of $1 - \pi$. The acquisition function is simply defined as the negative lower quantile

$$LCB_\pi(x) = -q_{1-\pi}(x).$$

It is negative since we want to find the next location which maximizes the acquisition function. Choosing a confidence level close to 1, i.e. $\pi \approx 1$ yields exploration, as seen in figure 2.3. For a Gaussian predictive distribution this the lower confidence bound is simply given as,

$$LCB(x) = -(\mu_x - \beta\sigma_x)$$

where $\beta = \Phi^{-1}(\pi)$.

Entropy search

Optimization policies known as policy search utilize information theory to select the next point which will provide the most information (i.e useful knowledge) about the objective function. More specifically their acquisition function is *mutual information* [8, pp. 135–140],

$$I(x, y) = \int\int p(x, y) \log \frac{p(x, y)}{p(y)p(x)} dydx, \tag{2.5}$$

which is a measurement of dependency between the random variables. If x and y are independent, then $p(x, y) = p(x)p(y)$, i.e. the fraction in (2.5) becomes 1 and thereby $I(x, y) = 0$.

Probability of improvement

Probability of improvment acquisitionfunction is defined as follows,

$$PI(x) = \mathbb{P}p(y|x, \mathcal{D})(\max(0, y_{\min} - y) > 0) = \mathbb{P}p(y|x, \mathcal{D})(y < y_{\min})$$

i.e. the probability of the prediction is an actually improvement. It does not take the magnitude of the improvement into consideration (as Expected improvement). It rather just if there is an improvement or not. In the case of a Gaussian predictive probability it is given on closed form

$$PI(x) = \Phi \cdot \left(\frac{y_{\min} - \mu_x}{\sigma_x}\right)$$

where $p(y|x\mathcal{D}) = \mathcal{N}(y|\mu_x, \sigma_x^2) = \mu_x + \sigma_x \cdot \mathcal{N}(y|0, 1)$

[3] [8, p. 145] deals with an maximization problem, and an upper confidence bound acquisition function is presented, however, this formulation is equivalent.

2.4 Summary

In this chapter, we introduced Bayesian optimization and found its relevance among the large number of optimization algorithms: It is a leading methodology for sample-efficient optimization. Then we dived into the details of a surrogate model in a BO setting. This was nothing else than a cheaply evaluated Bayesian regression model. Finally, we introduced the second component of BO, i.e. an acquisition function. We derived the need closed-form of expected improvement, and shortly presented alternative acquisition fucntion. In the following chapter, we will introduce discriminative surrogate models.

3 Discriminative surrogate models

When talking about a probabilistic surrogate model we are always implicitly talking about a discriminative model: A statistical model of the conditional distribution of the observation, y, conditional on x often parameterized by parameters θ, i.e. $p(y|x,\theta)$ which, in a Bayesian context, is utilized in the *predictive posterior distribution*:

$$p(y|x,\mathcal{D}) = \int p(y|x,\theta)p(\theta|\mathcal{D})d\theta, \tag{3.1}$$

where we take all possible models $\theta \in \text{Dom}(\theta)$ into account weighted accordingly to how probable the model is $p(\theta|\mathcal{D})$ (the posterior distribution). Gaussian processes and Bayesian neural networks are both discriminative models and they both assume that the observation is noisy in the following way,

$$y = f_{\mathbf{w}}(x) + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, \sigma^2).$$

They are, however, using two very different approaches to define a discriminative model. A BNN define $f_{\mathbf{w}}(x)$ as the the neural network output for a specific realization of the network weights and biases $\mathbf{w}$. Given a realization of $\theta = (\mathbf{w}, \sigma^2)$, the BNN likelihood is defined as

$$p_{\text{BNN}}(y|x,\theta) = \mathcal{N}(y|f_{\mathbf{w}}(x), \sigma^2).$$

A GP takes a different approach and directly model the noisefree prediction $f_* := f(x)$ as a random variable. Given a realization of $\theta = (f_*, \sigma^2)$, the likelihood for a GP is given as

$$p_{\text{GP}}(y|x,\theta) = \mathcal{N}(y|f_*, \sigma^2). \tag{3.2}$$

In the following chapter, we will provide more details on both models, how they are defined, can be trained and used for predictions and discuss their properties. First, we dive into GPs.

3.1 Gaussian process surrogate

The most popular surrogate model is the Gaussian process, which we soon will understand in more detail. Typically, a probabilistic regression model is on the from

$$y = f_{\mathbf{w}}(x) + \epsilon$$

where weights are trained. f could describe a linear model, $f(x) = \mathbf{w}^T x$ or a polynomial $f(x) = \sum_i \mathbf{w}_i \cdot x_i^2$ etc. The Gaussian process takes a completely different approach; its noisefree prediction[1], $f(x)$, does not depend any parameters $\mathbf{w}$, instead it depends on the vector $\mathbf{f} := [f(x_1), \ldots, f(x_n)]$ defined on the input/location data $\mathbf{x} = [x_1, \ldots, x_n]$. We assign $\mathbf{f}$ a multivariate normal distribution,

$$\mathbf{f}|\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}),$$

where $\boldsymbol{\mu}$ typically is 0 and the covariance matrix, is dependent on the input, $\mathbf{x}$,

$$\boldsymbol{\Sigma} = c(\mathbf{x},\mathbf{x}) = \begin{bmatrix} c(x_1,x_1) & \ldots & c(x_1,x_n) \\ \vdots & \ddots & \\ c(x_n,x_1) & \ldots & c(x_n,x_n) \end{bmatrix}, \qquad c(x,y) := Matern(x,y)...$$

[1] More correctly we here mean *predictive distribution*

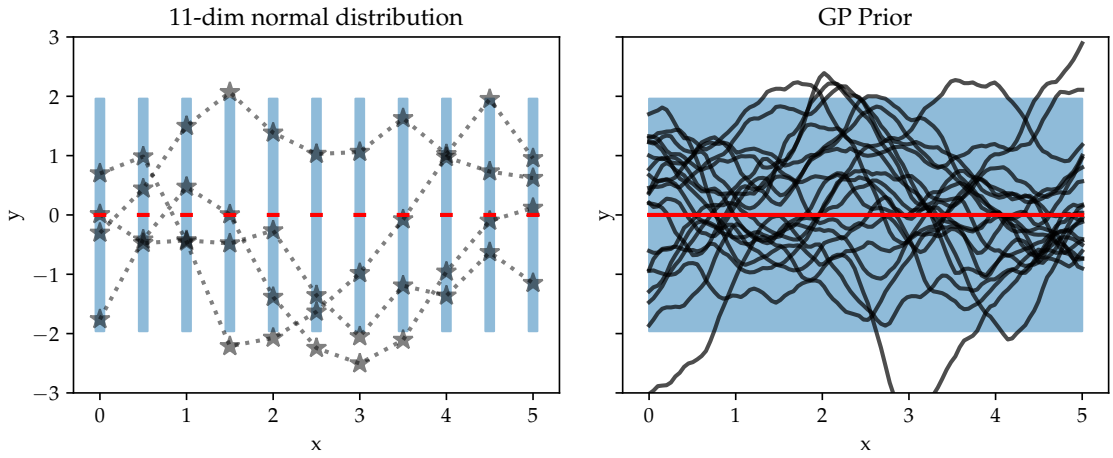


Figure 3.1: Left: Samples from $\mathcal{N}(\mathbf{f}|0, \kappa(x_1, \ldots, x_{11}))$ where $x_i = 0.5(i-1)$. Illustration that a samples from a Gaussian process is just samples from the multivariate normal distribution. We could potentially choose $\mathbf{x}$ to be all of the real line, which will give us the GP - an infinitely large multivariate normal distribution (right).

this means that a realization of the vector $\mathbf{f}$ is often close to 0 with a variance of the diagonal $\boldsymbol{\Sigma}$, but also with a correlation between the elements in $\mathbf{f}$ given by the off-diagonals in $\boldsymbol{\Sigma}$. This is a very important ingredience of a GP, if $c(x_1, x_2) \approx 1$ (assuming that the variances of $\mathbf{f}$ are 1, then $\boldsymbol{\Sigma}$ is a *correlation matrix*) then realizations of $\mathbf{f}$ always lead to similar values of $f(x_1)$ and $f(x_2)$ (this can be seen in Figure 3.1). This encapsulates the idea of a GP: similarities (could be distance or other measures) in x should lead to similarities in $f(x)$. Now, in the case of extending it to a regression model, we need to introduce the observation y_* for an arbitrary location x_*[2]. Prior to observing any data, we assume that the corresponding $f_* = f(x_*)$ is just a new element in the multivariate normal distribution,

$$p(f_*, \mathbf{f}|x_*, \mathbf{x}) = \mathcal{N}\left(\begin{bmatrix} f_* \\ \mathbf{f} \end{bmatrix} \middle| \begin{bmatrix} 0 \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} c(x_*, x_*) & c(x_*, \mathbf{x}) \\ c(\mathbf{x}, x_*) & c(\mathbf{x}, \mathbf{x}) \end{bmatrix}\right), \tag{3.3}$$

which yields many possible outcomes for f_* with an average $E[f_*] = 0$ and variance of $V[f_*] = 1$. Note that given a realization of $\mathbf{f}$, then the distribution of f_* is changed due to the correlation $c(x_*, \mathbf{x})$. Fortunately, any conditional distribution from a multivariate normal distribution is easy to deal with. Using appendix Todo include in appendix the conditional distribution of f_* given $\mathbf{f}$ is,

$$p(f_*|\mathbf{x}, \mathbf{f}) = \mathcal{N}(f_*|c(x_*, x_*)^{-1}c(x_*, \mathbf{x})\mathbf{f}, c(x_*, x_*)^{-1}). \tag{3.4}$$

3.1.1 Exact predictive distribution

What we want is the predictive posterior distribution, $p(y_*|x_*, \mathcal{D})$, i.e. by marginalizing out the random variable $f_* := f(x_*)$ (as seen in (3.1)),

$$p(y_*|x_*, \mathcal{D}) = \int \mathcal{N}(y_*|f_*, \sigma^2)p(f_*|\mathcal{D})df_*. \tag{3.5}$$

We will soon see that the posterior $p(f_*|\mathcal{D})$ is also a normal distribution with mean μ_* and variance σ_*^2 so using (3.6) we end up with the closed form Normal distribution,

$$p(y_*|x_*, \mathcal{D}) = \mathcal{N}(y_*|\mu_*, \sigma_*^2 + \sigma^2)$$

[2]The star subscript is included to help the reader easily relate f_*, y_* and x_*.

So now, we want to calculate $p(f_*|\mathcal{D})$, this can be done using the neat properties of Gaussian distributions.

Background: Trick with normal distributions [12, p. 93]
Given a marginal Gaussian distribution of x and a conditional Gaussian distribution of y given x of the form,

$$p(x) = \mathcal{N}(x|\mu, \Lambda^{-1})$$
$$p(y|x) = \mathcal{N}(y|Ax + b, L^{-1})$$

then the marginal distribution of y and the conditional distribution of x given y have the form,

$$p(y) = \mathcal{N}(y|A\mu + b, L^{-1} + A\Lambda^{-1}A^T) \tag{3.6}$$
$$p(x|y) = \mathcal{N}(x|\Gamma[\Lambda\mu + A^T L(y - b)], \Gamma) \tag{3.7}$$
$$\Gamma := (\Lambda + A^T LA)^{-1} \tag{3.8}$$

Posterior function distribution
From observing the data $\mathcal{D} = \{x_1, y_1, \ldots, x_n, y_n\} = (\mathbf{x}, \mathbf{y})$, we can marginalize over the noisefree predictions $\mathbf{f} = [f(x_1), \ldots, f(x_n)]$,

$$p(f_*|\mathcal{D}) = \int p(f_*|\mathbf{x}, \mathbf{f})p(\mathbf{f}|\mathcal{D})d\mathbf{f}. \tag{3.9}$$

From (3.4) we already have $p(f_*|\mathbf{x}, \mathbf{f})$ as a Gaussian distribution, so we just need to calculate the posterior $p(\mathbf{f}|\mathcal{D})$, this is done through the prior and likelihood,

$$p(\mathbf{f}|\mathcal{D}) \propto p(\mathbf{f}|\mathbf{x})p(\mathbf{y}|\mathbf{f}). \tag{3.10}$$

As mentioned $f(\cdot)$ is the noisefree prediction, i.e. $y = f(x) + \epsilon$. So assuming iid data, and that ϵ is additive Gaussian noise with variance σ^2, we get the likehood,

$$p(\mathbf{y}|\mathbf{x}, \mathbf{f}) = \prod_{i=1}^{n} p(y_i|x_i, \mathbf{f}_i) = \prod_{i=1}^{n} \mathcal{N}(y_i|\mathbf{f}_i, \sigma^2) = \mathcal{N}(\mathbf{y}|\mathbf{f}, \sigma^2 I).$$

From (3.3) the prior of $\mathbf{f}$ is defined using similarities between its corresponding $\mathbf{x}$, giving a multivariate normal distribution, $p(\mathbf{f}|\mathbf{x}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, c(\mathbf{x}, \mathbf{x}))$ so we can specify the unomalized posterior (3.10),

$$p(\mathbf{f}|\mathcal{D}) \propto \mathcal{N}(\mathbf{f}|\mathbf{0}, c(\mathbf{x}, \mathbf{x}))\mathcal{N}(\mathbf{y}|\mathbf{f}, \sigma^2 I).$$

Noticing that the random quantaties $\mathbf{y}$ and $\mathbf{f}$ are related such that we can use (3.7), we have that the posterior is the following Gaussian:

$$p(\mathbf{f}|\mathcal{D}) = \mathcal{N}(\mathbf{f}|\Gamma\sigma^{-2}\mathbf{y}, \Gamma) \quad \Gamma := \left(c(\mathbf{x}, \mathbf{x})^{-1} + \sigma^{-2} I_n\right)^{-1},$$

where Γ is defined according to (3.8). Finally, we see that both terms in the integral (3.9), are related such that it is possible to use (3.6) [3], for arriving at,

$$p(f_*|\mathcal{D}) = \mathcal{N}(f_*|\mu_*, \sigma_*^2)$$
$$\mu_* = A\Gamma\sigma^{-2}\mathbf{y}$$
$$\sigma_*^2 = c(x_*, x_*)^{-1} + A\Gamma A^T),$$

where we define $A := c(x_*, x_*)^{-1}c(x_*, \mathbf{x})$ agreeing with A in (3.6). We now have a fully specified Gaussian process posterior function, which we sample from in Figure 3.2.

[3] This can be seen by the relation between $\mathbf{f}$ and f_* in (3.9)

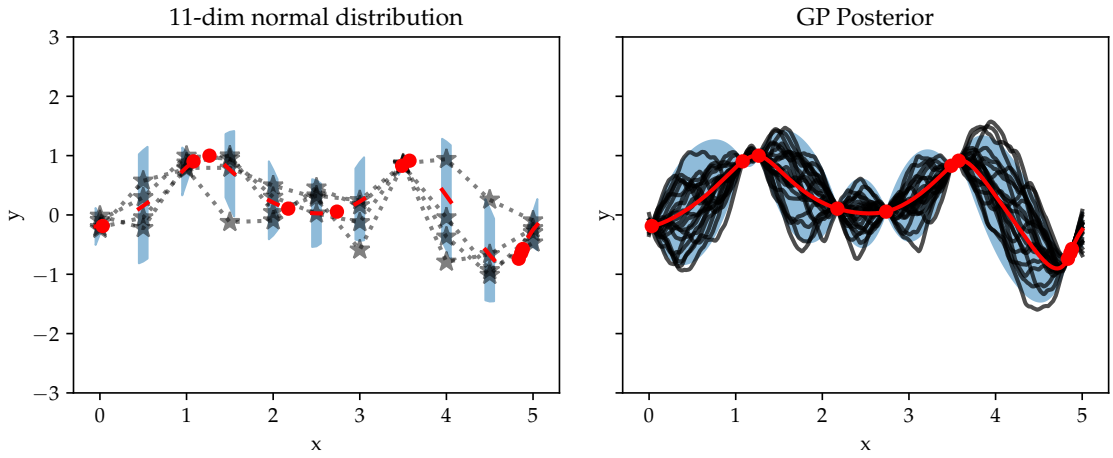


Figure 3.2: Left: Samples from posterior function distribution for $\mathcal{N}(\mathbf{f}_*|m(\mathbf{x}), V(\mathbf{x}))$ where $\mathbf{x} = \{x_1, \ldots, x_{11}\}$ and $x_i = 0.5(i-1)$. Illustration that a samples from a Gaussian process is just samples from the multivariate normal distribution. We could potentially choose $\mathbf{x}$ to be all of the real line, which will give us the GP - an infinitely large multivariate normal distribution.

3.1.2 Learning - Empirical Bayes inference

The above derivation of the prediction with the Gaussian process only acknowledged f_* as the unknown variable, however, in (3.2) we included the observation variance σ^2 as the unknown parameter set θ. Additionally, the similarity measurement, which is done in the kernel, $c(\cdot,\cdot)$, is also parameterized. We assumed those additional parameters were known because of the resulting closed-form predictive posterior. Instead we now call the variance σ^2 and kernel parameters hyperparameters. Very often GP hyperparameters are chosen using Emperical bayes, which is simply to choose the hyperparameters ν, which maximize the marginalized likehood,

$$\nu = \arg\min_{\nu} p(\mathbf{y}|\mathbf{x}, \nu) \tag{3.11}$$

where for a Gaussian process the marginal is given as

$$p(\mathbf{y}|\mathbf{x}, \nu) = \int p(\mathbf{y}, \mathbf{f}|\mathbf{x}, \nu) d\mathbf{f} = \int p(\mathbf{y}|\mathbf{f}, \nu) p(\mathbf{f}|\mathbf{x}, \nu) d\mathbf{f},$$

where $p(\mathbf{y}|\mathbf{f}, \nu) = \mathcal{N}(\mathbf{y}|\mathbf{f}, \sigma^2)$ and the prior is just the Gaussian $p(\mathbf{f}|\mathbf{x}, \nu) = \mathcal{N}(\mathbf{f}|0, c_\nu(\mathbf{x}, \mathbf{x}))$ And now we can easily perform the integration using (3.6),

$$p(\mathbf{y}|\mathbf{x}, \nu) = \mathcal{N}(\mathbf{y}|0, I\sigma^2, c_\nu(\mathbf{x}, \mathbf{x})).$$

Understanding why it makes sense to maximize the marginalized likelihood in order to get the best model is explained in [12, p. 165], we will skip this theory, and look at the implications.

In Figure 3.3 we see that using the same data, the (gradient-based) optimization of (3.11) converge to two different results, $\log p(y; \nu_1) = -14.29$, and $\log p(y; \nu_2) = -16.70$, and more local optima might exist. Therefore we optimize the GP with several restarts, when using it as a surrogate model.

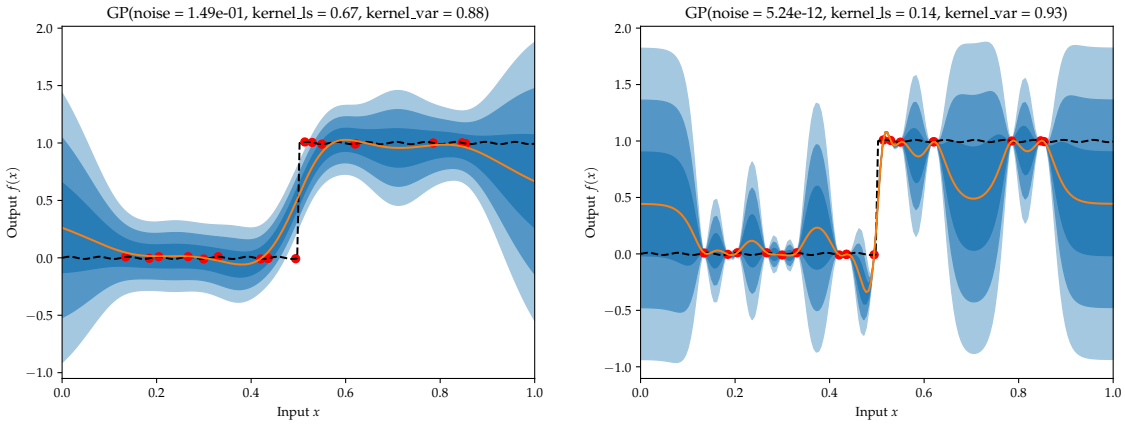


Figure 3.3: The same data fitted with two quite different GPs - note the different lenghtscales in the kernel and observation variance σ^2. Left is chosen 19 out of 20 times and has and marginalized log likelihood of -16.70, whereas 1 out of 20 runs the solution at right is given with marginalized log likelihood of -14.29.

3.2 Bayesian neural network regression

Neural networks have become increasingly popular in recent years, due to their ability to approximate any function arbitrary well (showed via the universal approximation theorem) and with the amount of data in the world today, neural networks are very powerful regression models for finding complex patterns. Bayesian neural networks are essentially neural networks, but instead of point estimates, each weight is assigned a distribution, which provides a probabilistic regression model (see Figure 3.5). Prior to any observed data, the weights are assigned a distribution (typically a standard Gaussian). After observing data a joint (posterior) distribution of the weights are inferred, such that the regression model is fitting the data and provides good uncertainty estimation (note that the weights might be complicated and correlated). In Figure 3.4 we see how the regression model can do predictions even when no data is observed, i.e. prior samples (left), when observing data the posterior combines the likelihood and the prior and we get the predictive posterior distribution (right).

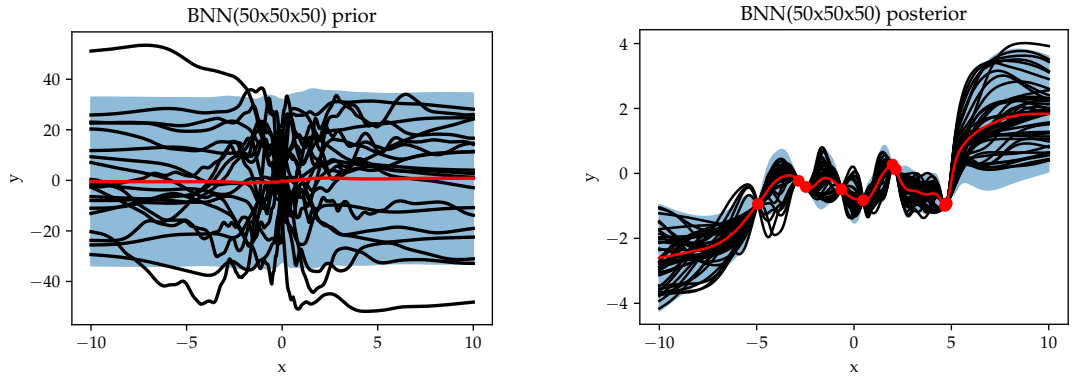


Figure 3.4: Left: 20 black predictive *prior* samples from a 3 layers BNN with 50 tanh nodes, with standard Gaussian prior distributions. Right: 20 predictive *posterior* samples. The blue areas are $\pm$ 2 predictive standard deviations from the red predictive mean.

The likelihood of a Bayesian Neural network is typically defined as a normal distribution with mean equal the neural network output and a observation variance σ^2 (both are random variables) which in the thesis is assumed constant throughout the domain.

$$p_{\text{BNN}}(y|x,\theta) = \mathcal{N}(y|f_{\mathbf{w}}(x), \sigma^2)$$

The prior distribution of the weights (including the biases) are typically assumed uncorrelated standard normal distributed and the observation variance σ^2 can be assigned a lognormal or half-Cauchy, with support on the positive real domain. We can write the priors of the BNN model as

$$p(\theta = (w, \sigma)) = \mathcal{N}(w; \mathbf{0}, I) \log \mathcal{N}(\sigma; \dots)$$

Now we can define the posterior distribution, i.e. the probability of the unknown quantaties given the observations. We define it using Bayes rule,

$$\begin{aligned} p(\theta|x,y) &= \frac{p(\theta, y|x)}{p(y|x)} \\ &= \frac{p_{\text{BNN}}(y|x,\theta)p(\theta|x)}{p(y|x)}. \end{aligned}$$

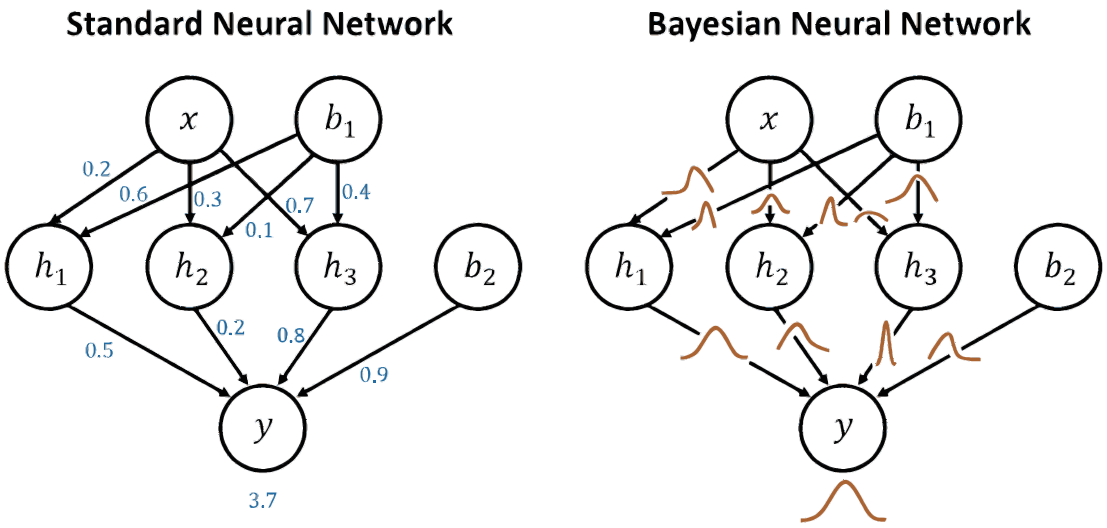


Figure 3.5: Graphical representation of a Bayesian neural network compared to a neural network. The weights are assigned a probability density. Note that we often prior assume no correlation and a standard normal distribution, but the posterior (after observing data) might contain correlations between the weights.

Note that the prior distribution $p(\theta|x)$ just like the likelihood can depend on x - we could for example extend σ to depend on the location of the sample x and define it as $\sigma(x)$. Note that the posterior distribution of the weights and σ^2 is complex and correlated due to the multiplication of the compllicated likelihood and the prior. Therefore, exact inference of BNNs is intractable and we approximate the predicitve posterior distribution with Monte Carlo samples from posterior,

$$\begin{aligned} p(y_*|x_*, \mathcal{D}) &= \int p_{\text{BNN}}(y_*|x_*, \theta) p(\theta|\mathcal{D}) d\theta \\ &\approx \frac{1}{K} \sum_{k=1}^{K} p_{\text{BNN}}(y_*|x_*, \theta^{(k)}) \end{aligned}$$

where the integral is intractable. As indecated in the second line, we can approximated the integral with Monte Carlo sampling: $\theta^{(k)}$ are iid samples from the posterior distribution, $\theta^{(k)} \sim p(\theta|\mathcal{D})$.

Background: Monte Carlo approximation

Assuming we have a number of iid samples, $\theta^{(1)}, \ldots, \theta^{(K)}$ drawn from the distribution $p(x)$, then the following appriximation

$$E[f(x)] \approx \frac{1}{K} \sum_{k=1}^{K} f(x^{(k)}) =: \Theta_K(f)$$

holds according to the law of large numbers. In fact,

$$E[f(x)] = \lim_{K \to \infty} \Theta_K(f)$$

and the central limit theorem ensures that the variance of the unbiased estimator of the expectation decreases with number of samples, K, i.e.

$$p(\hat{\Theta}) \approx \mathcal{N}(\hat{\Theta}|\mu_f, \frac{\sigma_f^2}{K}),$$

where $\mu_f := E[f(x)]$ and $\sigma_f^2 = Var(f(x))$. [13, p. 708]

We try to get iid samples from the posterior distribution using MCMC.

Posterior samples

For BNN the joint distribution $p(\mathcal{D}, \theta)$ is available, but calulating the posterior distribution requires the marginalized likehood, $p(\mathcal{D}) = \int_\theta p(\mathcal{D}, \theta)$. This integral is often intractable since the space of θ typically is abnomous - so not even nummerical approximations of the intergral is tractable. From Bayes rule, we have the equality,

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}, \theta)}{p(\mathcal{D})} \propto p(\mathcal{D}, \theta),$$

where the propotional sign is true, since $p(\theta|\mathcal{D})$ is a function of θ. Knowing the joint distribution, $p(\mathcal{D}, \theta)$, allow for using Markov chain Monte Carlo (MCMC) to obtain samples from the posterior distribution.

Background: Markov chain Monte Carlo (MCMC)

We can use MCMC for sampling from a probability density $p(x)$, with only the knowledge of a proportional/unnormalised density $\hat{p}(x) \geq 0$ i.e

$$\hat{p}(x) = c \cdot p(x) \propto p(x), \qquad c = \int \hat{p}(x)dx,$$

where $\int \hat{p}(x)dx$ may be intractable. The MCMC procedure constructs an **ergodic** Markov chain/process, such that its **stationary** distribution is exactly $p(x)$, but only with the knowledge of $\hat{p}(x)$. Here is a short explaination of the ergodicity and stationary:

- **Ergodicity**: A Markov process which spans/visits all the space.
- **Stationary distribution**: A Markov process has reached stationary distribution if we stay there at the next sample.
- **Detailed balance relation**: A way to check if a Markov chain is stationary is by the detailed ballance equation:

  $$p(x)p(x \to y) = p(y)p(y \to x)$$

  note we introduce the notation $p(x \to y)$ for the transition probablity from x to y, this can also be interpreted as a conditional distribution of state y given state x

So since the known joint distribution is proportional to the posterior distribution, we can use MCMC to get samples from the posterior distribution, even when it is intractable. To better introduce what MCMC is, we here present a simple MCMC procedure, the Metropolis-Hasting algorithm (HM), and give proof of why its produced Markov chain stays in its stationary distribution.

MCMC example: Metropolis-Hasting (MH)

At iteration n we already have a sample x_n,

1. Propose $\hat{x}$ from a proposal density $q(\cdot|x_n)$
2. Compute acceptance probability

$$\alpha(x_n, \hat{x}) = \min\left(1, \frac{p(\hat{x})}{p(x_n)} \frac{q(x_n|\hat{x})}{q(\hat{x}|x_n)}\right)$$

3. Set the next sample

$$x_{n+1} = \begin{cases} \hat{x} & \text{with probability } \alpha(x_n, \hat{x}) \\ x_n & \text{with probability } 1 - \alpha(x_n, \hat{x}) \end{cases}$$

note that $\alpha(x_n, \hat{x})$ uses $p(x)$ in the fraction $\frac{p(\hat{x})}{p(x_n)} = \frac{p(\hat{x})\cdot c}{p(x_n)\cdot c} = \frac{\hat{p}(\hat{x})}{\hat{p}(x_n)}$, so we only need $\hat{p}(x)$.

Proof: Assuming discrete states, the transition probability between the states are given as,

$$p(x \to y) = \begin{cases} q(y|x)\alpha(x,y) & \text{if } x \neq y \\ q(x|x) + \sum_{z \neq x} q(z|x)(1 - \alpha(x,z)) & \text{if } x = y \end{cases}$$

Now, let us look at the detailed balance relation. Assume $x \neq y$,

$$\begin{aligned} p(x)p(x \to y) &= p(x)q(y|x)\alpha(x,y) \\ &= p(x)q(x,y)\min\left(1, \frac{p(\hat{x})}{p(x_n)}\frac{q(\hat{x}, x_n)}{q(x_n, \hat{x})}\right) \\ &= \min(p(x)q(x,y), p(y)q(y,x)) \end{aligned}$$

Observing that the right hand side yields symmetric result in x and y, therefore we obtain,

$$p(x)p(x \to y) = p(y)p(y \to x)$$

and summing over x on both sides yields,

$$\sum_x p(x)p(x \to y) = p(y)\sum_x p(y \to x) \tag{3.12}$$

$$\implies \quad p(y) = \sum_x p(x)p(x \to y). \tag{3.13}$$

The detailed balance is trivially obtained for $x = y$, all in all, this reveals that $p(x)$ is in fact invariant for the chain $\{x_1, \ldots, x_n\}$ and thereby that MH is a MCMC algorithm.

HM with its random walk transition is very simple and it comes with some serious disadvantages: Slow convergence speed, it might stay in the same region for a long time and produce highly correlated samples(i.e. not the iid samples we need for the Monte Carlo approximation to be correct) [13]. The gold standard in MCMC, Hamiltonian Monte Carlo (HMC), replaces HMs random walk with gradient-guided movements and interpret the probability landscape as a physical system.

Hamiltonian Monte Carlo (HMC)

HMC exploits arguments from classical mechanics around the Hamiltonian equations. This method leads to more efficient sampling as the Hamiltonian interpretation allows the system to consider regions with high probability more - this is obtained using a gradient of the probability landscape, $\frac{-\partial E(x)}{\partial x}$.

We introduce the potential energy $E(x)$ by defining the joint probability as,

$$p(x) = \frac{1}{Z_E}\exp(-E(x))$$

Now, a latent vector q is introduced in order to represent the momentum of the system, which

gives us the kinetic energy of the system.

$$K(q) = \frac{1}{2}\sum_{i=1}^{l} q_i^2$$

Combining the kinetic and potential energy yields, the the Hamilton function and its corresponing distribution

$$H(x, q) = E(x) + K(q)$$

and

$$p(x, q) = \frac{1}{Z_H}\exp(-H(x, q)) \tag{3.14}$$

$$= \frac{1}{Z_E}\exp(-E(x))\frac{1}{Z_K}\exp(-K(x)) \tag{3.15}$$

$$= p(x)p(q) \tag{3.16}$$

The desired distribution $p(x)$ is found as the marginal of $p(x, q)$. The sampling procedure is as follows,

1. At the (x_n, q_n) sample momentum q_n by gibbs sampling
2. Simulate the Hamiltonian dynamics with Leap-frog integration (i.e. walking the contour lines of $p(x, q)$). New position is (x_{n+1}, q_{n+1})
3. Accept new position with MH acceptance probability $\min(1, \frac{p(x_{n+1},q_{n+1})}{p(x_n,q_n)})$

3.2.1 MCMC used in thesis

While being a popular sampling method for BNN inference, Hamiltonian Monte Carlo is very sensitive to the two parameters used in the leapfrog integration (physical simulation): the step size and the number of steps. These parameters are typically tuned by conducting a number of initial experiments, i.e. data set samples, which is not appropriate in a BO setting with an expensive objective function. The NUTS sampling method (No-U-Turn-Sampler) [14] is an adaptive version of HMC, which terminates the leapfrog integration when the simulation reaches the starting point. Thereby we avoid manually tuning of the number of steps.

The paper [4] introduces a Bayesian optimization framework called Bayesian Optimization with Hamiltonian Monte Carlo Artificial Neural Networks (BOHAMIANN), which demonstrates state of the art performance on a wide range of optimization task. It is stochastic gradient HMC where the hyperparameters are tunes adaptively doing the burn-in phase. It is implemented using

we look a bit at the two versions used in the thesis. Our implementation in Numpyro and the one from the paper, BOHamiANN, [4].

Often the physical simulation in HMC goes back and forth on the same path (a u-turn), and we risk getting bad samples. No U-turn (NUTS) sampling avoids this, by forcing the HMC path to not take u-turns.

BOHAMIANN is using an adaptive stochastic Hamiltonian monte carlo method to train the BNN.

Historically, Hamiltonian Monte Carlo (HMC) [157] has been the most popular sampling method used in BNNs. However, its performance is very sensitive to a proper initialization of its two user-specified parameters, namely step size and number of steps, which are normally set by carrying out

a number of initial experiments. This makes HMC inappropriate for the Expensive Optimization application, where new observations are incrementally added and the surrogate model undergoes change in each iteration. Fortunately, the No-U-Turn-Sampler (NUTS) [86], an adaptive version of HMC, resolves this limitation by eliminating the need for setting the number of steps in advance.

3.2.2 Design and properties of Bayesian neural network

The architecture of a BNN is a large topic to discuss. There are certain tradeoffs to be taken into consideration: How expressive should the network be (i.e. how deep and how many nodes per layer) versus how much time do we have for the sampler to converge to the true posterior distribution? A challenging part of MCMC is that it is difficult to know when the samples are true samples from the posterior. A consideration when training deterministic neural networks are overfitting however this is not a big consideration when fitting Bayesian neural networks; Choosing a prior around 0 will regulate the parameters, and thereby postpone overfitting.

This thesis is inspired by the PhD thesis [6], which uses an architecture of a 2 layers with 50 sigmoid nodes in each layer, and the BOHAMIANN paper [4], default uses an architecture of 3 layers with 50 tanh-nodes in each layer. As we want to make sure always to do the inference correctly, we want to be able to take a proper amount of samples, while also having an explicit model. Figure (3.6) shows prior samples of BNN with a different number of tanh-nodes on each of the 3 layers, this provides an intuition that choosing a larger BNN leads to a more expressive regression model. When doing Bayesian optimization the model as a small amount of training data, i.e. complicated patterns in data is not possible to discover, and hence highly expressive models are not important.

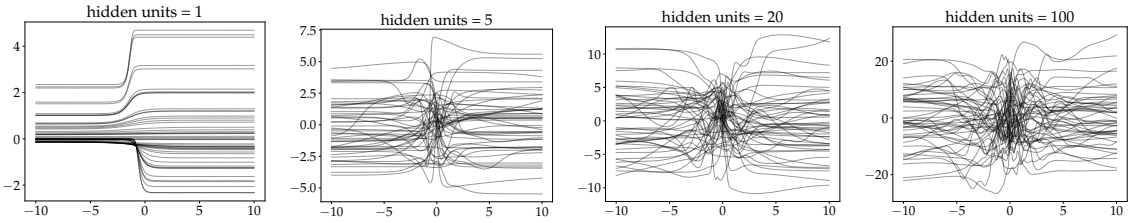


Figure 3.6: Prior samples from 3 layer BNN. The number of units in each of the three layers have a large influence on the BNNs ability to be expressive. Note that the inference becomes more computationally demanding with BNN size.

$\mathcal{N}(\mathbf{w}|\mathbf{0}, I)$, and the prior distribution of the observation variance is following a inverse gamma distribution InvGa$(\sigma^2|1000, 1)$ yielding a quite informative prior of $\sigma^2 \approx 0$, i.e. believing that there is no oberservation noise.

3.3 Summary

This chapter introduced the discriminative models, which will be tested as surrogate models in a Bayesian optimization setting. First, we introduced the difference between a Gaussian process and the more common way of defining a Bayesian regression model, i.e. a deterministic regression model which is made Bayesian by assigning the weights and regression output a joint uncertainty. We define the Bayesian neural network in this common way. Next, we introduced the Gaussian processes and found that they are nothing else than multivariate Gaussian distributions, yielding nice exact inference. We shortly discussed the hyperparameter tuning of GPs using empirical Bayes, and that we need to take care of local optima. Finally, we introduced Bayesian Neural networks, which are trained with MCMC sampling on joint probability factorized to the prior distribution and a likelihood function. a bit more?

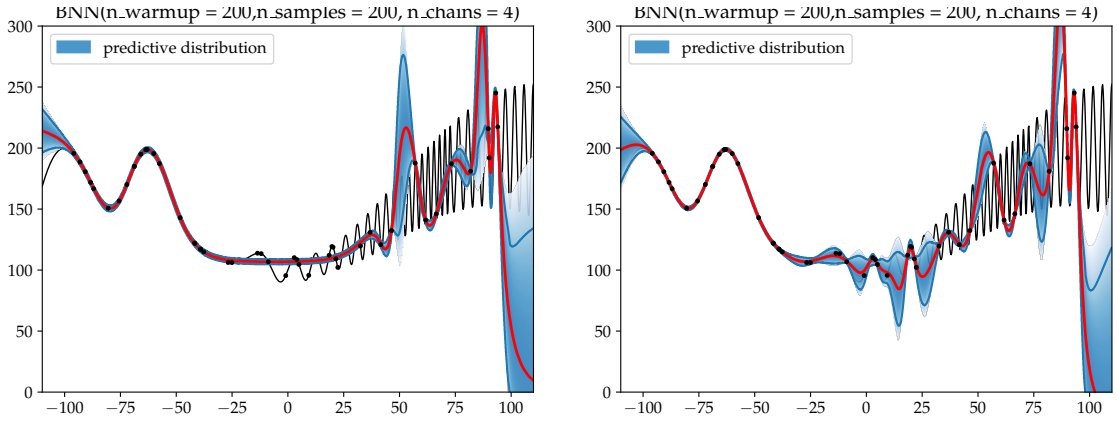


Figure 3.7: Example where 100 nodes on each of three layers, lead to a much more expressive model. σ^2 follows a informative prior $InvGamma(1000, 1)$, i.e. prior mean $E[\sigma^2] \approx \frac{1}{1000}$, and variance $\approx \frac{1}{1000^3}$, however since the data is distributed in such a complex way the limited expressiveness of the model, forces the model to infer make σ^2 large, i.e. including the data in the noise

4 Generative models as surrogate

Generative models are statistical models of the joint distribution $p(x, y)$ We need, however, a discriminative model for regression, i.e. a model of the conditional distribution of y given x, i.e. $p(y|x)$. All generative models we deal with in this thesis allow for exact inference of the conditional distribution. So given a well-fitted generative model, one could immediately think they would be feasible to use as surrogate models. However, in this project, we only look at Gaussian mixture models as generative models - and they have a problem for x-values where the probability of the observed input data, the marginal $p(x)$, is low. Recall the conditional distribution is

$$p(y|x) = \frac{p(x, y)}{p(x)}$$

and can be interpreted as a slice of the joint distribution $p(x, y)$ for a fixed value of x, but normalized with $p(x) = \int p(x, y)dy$. So even if there is a very small probability of the data, the conditional probability $p(y|x)$ gets artificially certain in the case of Gaussian mixtures. We, therefore, need to introduce a prior distribution for y, which will take over in areas for no data, i.e. small $p(x)$. This is discussed in section 4.1.

Using generative models as regression models is not used much in the literature. Using the conditional of a Gaussian mixture model (or kernel estimator) for regression has been discussed briefly in [15] and using kernel estimator [7] and [16] for active learning. According to these sources, the good reasons for using the mixtures for regression are that they can be used to represent any relations between the variables, e.g., $p(y|x)$ or $p(x|y)$. They are both applicable in supervised and unsupervised machine learning. And they are good at dealing with incomplete data, i.e. missing values in the data set <change this>. We hypothesize that it will allow for an expressive surrogate model, which competently can deal with complex BO tasks, as they do not assume continuity. In this thesis we will first look at the most simple approach to a generative model, i.e. putting an equally weighted Gaussian mixture component on each data point. This is also referred to as a kernel density estimator (some might know this from kde-plots/estimating a distribution from data), but with a twist of including a prior distribution to it. Next, we look at the more intelligent models, Gaussian mixture models, which hopefully can capture some correlations between the variables. And finally, we look at the more complicated sum-product networks, which introduce a generalization element and have a flavor of a neural network. To summarize, the mixture regression models are:

- Kernel density estimator regression (KDE),
- Gaussian mixture regression (GMR),
- Sum-product networks (SPN).

4.1 Conditional distribution in a Bayesian setting

As mentioned above the conditional itself is not enough as a probabilistic regression model used as a surrogate model. This is showcased in the middle right illustration in Figure 4.1, where the conditional distribution would have the same distance between its confidence bounds even for those x far away. To our knowledge, this problem has not been dealt with in the literature before. We want to manipulate the conditional distribution to transform into a very uncertain prior probability for y in areas where there is small evidence of the data $p(x)$. Two of the ideas for manipulating the conditional distributions (denoted $\hat{p}(y|x)$),

1. Include a new Gaussian mixture component with zero mean and large variance, $p_{prior}(y)$, and choose an x-depended weighting, $\alpha_x \in [0, 1]$, such that

$$\hat{p}(y|x) = \alpha_x p(y|x) + (1 - \alpha_x) p_{prior}(y)$$

2. Assuming both the conditional and prior distribution to be a Gaussian, we could choose an- x depended weighting, $\alpha_x \in [0, 1]$ such that the manipulate conditional is a Gaussian with mean $\hat{\mu}$ and variance $\hat{\sigma}^2$ such that,

$$\hat{\mu} := \alpha \cdot \mu_{y|x} + (1 - \alpha) \cdot \mu_{prior}$$
$$\hat{\sigma}^2 := \alpha \cdot \sigma^2_{y|x} + (1 - \alpha) \cdot \sigma^2_{prior}$$

We define x-depending weighting to be a function of the evidence $p(x)$ and how much data is observed N, and we additionally introduce a parameter $\Delta > 0$,

$$\alpha_x = \frac{N \cdot p(x) \cdot \Delta}{N \cdot p(x) \cdot \Delta + 1}$$

note that $\alpha_x = 0$ for $p(x) = 0$ and $\alpha_x \to 1$ for $N \cdot p(x) \cdot \Delta \to \infty$. Illustration of idea 1 can be seen in left illustration in Figure 4.1 where we call $S(x) := N \cdot p(x) \cdot \Delta$ the scaling.

Idea 1 would work on any kind of conditional distribution, while idea 2 would be an intuitive transformation but only if the conditional is Gaussian, alternative, as we are working with mixtures of Gaussians, these could be manipulated in the same way.

Note: Idea 1 defines a valid distribution
The manipulated conditional in idea 1, is a convex combination of valid distributions, and it is always positive and integrates to 1, as easily seen here,

$$\begin{aligned}\int_y \hat{p}(y|x)dy &= \int_y \left[\alpha_x p(y|x) + (1 - \alpha_x) p_{prior}(y)\right] dy \\ &= \alpha_x \int_y p(y|x)dy + (1 - \alpha_x) \int_y p_{prior}(y)dy \\ &= \alpha_x + (1 - \alpha_x) = 1.\end{aligned}$$

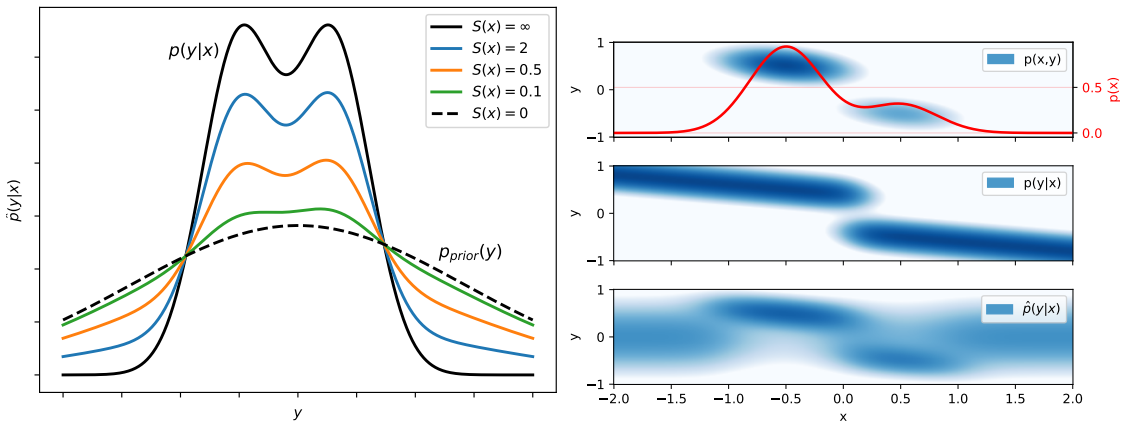


Figure 4.1: Left: Illustration of how the preditive distribution is manipulated according the the scaling function $S(x) := p(x) \cdot N \cdot \Delta$. Right: Illustration of why it makes sense to manipulate the predictive distribution $p(y|x)$, if there is a small amount of input data at a region, then the predictive distribution should transform into the uncertain prior

4.1.1 Mean and variance of predictive distribution v1

If we just are interested in a Gaussian approximation of the predictive distribution, this can be easily done assuming we know the mean, variance and the second moment of the conditional distribution, first the predictive mean is calculate,

$$\begin{aligned} E_{\hat{p}(y|x)}[y] &= \int y \cdot (\alpha_x \cdot p(y|x) + (1-\alpha_x) p_{prior}(y))\, dy \\ &= \alpha_x \cdot E_{p(y|x)}[y] + (1-\alpha_x) \cdot E_{p_{prior}(y)}[y] \end{aligned}$$

And the predictive variance is calcualted, using the definition of variance, $V_{\hat{p}(y|x)}[y] = E_{\hat{p}(y|x)}[y^2] - E_{\hat{p}(y|x)}[y]^2$. So we only need to calculate the second moment,

$$\begin{aligned} E_{\hat{p}(y|x)}[y^2] &= \int y^2 \cdot \alpha_x \cdot p(y|x) + (1-\alpha_x) p_{prior}(y) dy \\ &= \alpha_x \cdot E_{p(y|x)}[y^2] + (1-\alpha_x) \cdot E_{p_{prior}(y)}[y^2] \\ &= \alpha_x \cdot (Var_{p(y|x)}[y] + E_{p(y|x)}[y]^2) + (1-\alpha_x) Var_{p_{prior}(y)}[y] \end{aligned}$$

Assuming $E_{p_{prior}(y)}[y] = 0$.

> **Note: Implementation**
> If we use $\alpha_x \propto p(x)$, then it is not necessary to calculate the conditional distribution at all. Assuming c is a constant in y.
>
> $$\hat{p}(y|x) = \frac{c \cdot p(x) \cdot p(y|x) + p_{prior}(y)}{c \cdot p(x) + 1} = \frac{c \cdot p(x,y) + p_{prior}(y)}{c \cdot p(x)}$$

4.2 Conditional of mixture model

To exploit a generative model as a surrogate model in Bayesian optimization, we need to calculate the condtional distribution. Fortunately, all generative models used in this thesis are mixture models, which simplifies the upcomming deveriations. We define a general mixture model with Z mixture components as,

$$p(x,y) = \sum_{z=1}^{Z} \lambda_z p_z(x,y)$$

where $p_z(x,y)$ are mixture components, i.e. simpler generative models with same support, $(x,y) \in \mathcal{X} \times \mathbb{R}$. The goal is to define the conditional distribution exact for all the mixture models. As we will soon see, this is again a mixture model,

$$p(y|x) = \sum_z \gamma_z(x) p_z(y|x).$$

with $\sum_z \gamma_z(x) = 1$ and $\gamma_z(x) \in [0,1]$. First, we calcalculate the marginal distribution $p(x)$ of the mixture,

$$p(x) = \int p(x,y) dy = \sum_z \lambda_z \int p_z(x,y) dy = \sum_z \lambda_z p_z(x).$$

Next, we can calculate the conditional in terms of the conditional of the individual mixture components,

$$\begin{aligned} p(y|x) &= \frac{p(y,x)}{p(x)} \\ &= \sum_z \frac{\lambda_z}{p(x)} p_z(x,y) \\ &= \sum_z \frac{\lambda_z p_z(x)}{p(x)} p_z(y|x) \\ &= \sum_z \underbrace{\frac{\lambda_z p_z(x)}{\sum_{z^*} \lambda_{z^*} p_{z^*}(x)}}_{\gamma_z(x)} p_z(y|x). \end{aligned}$$

So we see that the conditional of a mixture model is again a mixture model. We also see that $\sum_z \gamma_z(x) = 1$ and hence we can interpret the above as the following,

$$p(y|x) = p_z(y|x), \qquad z \sim Cat(\gamma_1(x), \dots, \gamma_Z(x)).$$

And we name $p(z|x) = \gamma_z(x) \in [0,1]$ the *responsibility* of mixture component z at a given location $x \in \mathcal{X}$, (The probability that y to belong to component z at a given location x). For implementation we notice that the denominator in $\gamma_z(x)$ can be reused for all components.

Gaussian approximation of mixture conditional

As discussed in Section 4.1.1, in order to obtain the closed-form solution in the expected improvement, we can approximate the mixture with a gaussian distribution, i.e. calculation of the conditional mean and variance.

The mean of the conditional is just

$$\begin{aligned} E_{p(y|x)}[y] &= \sum_z \gamma_z(x) \int y \cdot p_z(y|x) dy \\ &= \sum_z \gamma_z(x) E_{p_z(y|x)}[y]. \end{aligned}$$

The variance is found using the variance definition $V[y] = E[y^2] - E[y]^2$,

$$\begin{aligned} E_{p(y|x)}[y^2] &= \sum_z \gamma_z(x) \int y^2 p_z(y|x) dy \\ &= \sum_z \gamma_z(x) (Var_{p_z(y|x)}[y] + E_{p_z(y|x)}[y]^2). \end{aligned}$$

We will now present all the models and show how their conditional distributions are calculated concretely.

4.3 Kernel density estimator regression

Maybe the most simple mixture model one could think about is to put a small variance Gaussian mixture component around all data points and weight all the components equally. So for n datapoints, $\{(x_i, y_i)\}_{i=1}^n$, the generative model is given as,

$$p(x,y) = \frac{1}{N}\sum_{i=1}^{n} \mathcal{N}\left(\begin{bmatrix}x\\y\end{bmatrix}\middle|\begin{bmatrix}x_i\\y_i\end{bmatrix}, \sigma^2 I\right) = \frac{1}{N}\sum_{i=1}^{n} \mathcal{N}(x|x_i,\sigma^2 I)\mathcal{N}(y|y_i,\sigma^2),$$

where σ^2 is refered as the bandwidth, when the literature refers to the above as a kernel estimator. Small σ^2 yields a complex model and large σ^2 yields a simple model. Therefore chosing σ^2 just rigth is crucial for a good model.

4.3.1 Conditional of Kernel density estimator

Since the kernel density estimator is just a Gaussian mixture model, with no correlation between any of the variables, yeilding i.e. $p_z(y|x) = p_z(y)$, therefor the conditional distribution is given as,

$$p(y|x) = \sum_{z=1}^{n} \gamma_z(x)\mathcal{N}(y|\mu_y^{(z)}, \Sigma_{yy}^{(z)}), \tag{4.1}$$

$$\gamma_z(x) = \frac{\lambda_z \mathcal{N}(x|\mu_x^{(z)}, \Sigma_{xx}^{(z)})}{\sum_{z^*} \lambda_{z^*}\mathcal{N}(x|\mu_x^{(z^*)}, \Sigma_{xx}^{(z^*)})}. \tag{4.2}$$

The computational complexity of calculating the conditional or the predictive distribution is $O(n)$, since we reused the denominator of $\gamma_z(x)$ for all components z.

4.4 Gaussian mixture regression

Extending the kernel estimator regression with covariance between the variables, only $K \leq N$ components and different weighting on each component, we arrive at a Gaussian mixture model. The conditional of GMM gives the Gaussian mixture regression model [17].

We can model our data, as a generative model $p(x,y)$,

$$p(x,y) = \sum_{z=1}^{K} \lambda_z \mathcal{N}(x,y|\mu^{(z)}, \Sigma^{(z)}), \qquad \mu^{(z)} = \begin{bmatrix}\mu_x^{(z)}\\ \mu_y^{(z)}\end{bmatrix},\ \Sigma^{(z)} = \begin{bmatrix}\Sigma_{xx}^{(z)} & \Sigma_{xy}^{(z)}\\ \Sigma_{yx}^{(z)} & \Sigma_{yy}^{(z)}\end{bmatrix},$$

where $\sum_{z=1}^{K} \lambda_z = 1$. The parameters $\left(\lambda_z, \mu^{(z)}, \Sigma^{(z)}\right)_{z=1}^{K}$ need to be trained, which is done using the EM algorithm. We will now show how the conditional is calculated exactly.

4.4.1 Conditional of Gaussian mixture model

Since the components are multivariate Gaussian distributions, we use <REF> and can define the conditional of a multivariate Gaussian as

$$p_z(y|x) = \mathcal{N}(y|\mu_{y|x}^{(z)}, \Sigma_{y|x}^{(z)}) \tag{4.3}$$

$$\mu_{y|x}^{(z)} := \mu_y^{(z)} + \Sigma_{yx}^{(z)}(\Sigma_{xx}^{(z)})^{-1}(x - \mu_x^{(z)}) \tag{4.4}$$

$$\Sigma_{y|x}^{(z)} := \Sigma_{yy}^{(z)} - \Sigma_{yx}^{(z)}(\Sigma_{xx}^{(z)})^{-1}\Sigma_{xy}^{(z)}. \tag{4.5}$$

Now, the conditional is defined straight forward from Section (4.2),

$$p(y|x) = \sum_{z=1}^{K} \gamma_z(x)\mathcal{N}(y|\mu_{y|x}^{(z)}, \Sigma_{y|x}^{(z)}) \tag{4.6}$$

$$\gamma_z(x) := \frac{\lambda_z \mathcal{N}(x|\mu_x^{(z)}, \Sigma_{xx}^{(z)})}{\sum_{z^*=1}^{K} \lambda_{z^*}\mathcal{N}(x|\mu_x^{(z^*)}, \Sigma_{xx}^{(z^*)})} \tag{4.7}$$

The computational complexity is $O(K \cdot d^3)$ (d is the dimension of x), since the matrix inversion of the covariance matrix $(\Sigma_{xx}^{(z)})^{-1}$ is the dominating cost and it happens for all the K components.

4.5 Sum product networks

A sum-product network (SPN) is a mixture model, which allows for exponentially many mixture components, but with only <linearly many parameters> and, thereby, tractable inference (i.e. conditionalization and marginalization queries). In short, the SPN consists of a computational graph, with tractable leaf distributions, which are combined using products and sums nodes, recursively. To keep the inference of the SPN tractable, we want to maintain certain properties when designing the SPN graph. First we need to define a scope.

Scope of nodes in SPN

A scope (sc) of a leaf node is the set of random variables among each dimension of $x = \{x_1, \ldots, x_{\text{dim}(x)}\}$ and y of which the leaf distribution, $p_i(\cdot)$, defines a distribution function (in our implementation the leaf scopes are all singletons). The scope of a sum og product node, i, are defined recursively, $sc(i) = \cup_{j \in ch(i)} sc(j)$.

- A sum nodes children must have the same scope (completeness).
- A product nodes children must have distinct scopes (decomposability).
- Leaf nodes must have tractable inference.

The density of the mixture models is calculated in the following way,

Calculation of $p(x, y)$

Input: Fully trained SPN, with leaf distributions $p_i(\cdot)$ for $i \in \mathcal{L}eaf(S)$ and weights $w_{i,j}$ for $(i, j) \in \{(i, j) | i \in \mathcal{S}um(S), j \in ch(i)\}$
function Eval(node i)
  if $i \in \mathcal{L}eaf(S)$ **then**
    return: $p_i(x, y)$ ▷ evaluate leaf distributions at point (x, y)
  if $i \in \mathcal{S}um(S)$ **then**
    return: $\sum_{j \in ch(i)} w_{i,j}$Eval$(j)$
  if $i \in \mathcal{P}rod(S)$ **then**
    return: $\prod_{j \in ch(i)}$ Eval(j)
$p(x, y) =$ Eval(root node)

In this thesis, we implement the SPN similar to the RAT-SPN presented in [18], which ensures that we have a Complete and decomposable SPN. The following is the structure of the RAT-SPN,

1. Define C leaf distributions per random variable[1] (C is called ”channels”).
2. Pair up the elements in the set of random variables in a random way. If uneven set, then one element pairs with the empty set.
3. For each pair: Define C^2 product nodes by combining each combination of leaf nodes.
4. For each pair: Give the C^2 product nodes the same C sum node parents.
5. Now, pair up the pairs and repeat step 3 (with sum-nodes instead of leaf nodes) and 4.
6. At the final iterations: Give the C^2 product nodes 1 sum node parent.

This is a scalable and easy way to construct the SPN. For high-dimensional problems, we can limit the number of pairs defined in step 2 and instead, combine more of the randomly defined SPNs in several tracks T. This allows the model to be lucky in the case e.g. dimension x_3 and x_5 was a

[1] Note that we define the random variables as the dimensions in the joints distribution, i.e. $\{x_1, ..., x_{\text{Dim}(x)}, y\}$.

powerful combination. Figure 4.2 illustrates the concept of RAT-SPN for only one track $T = 1$ (more tracks would shuffle the pairs).

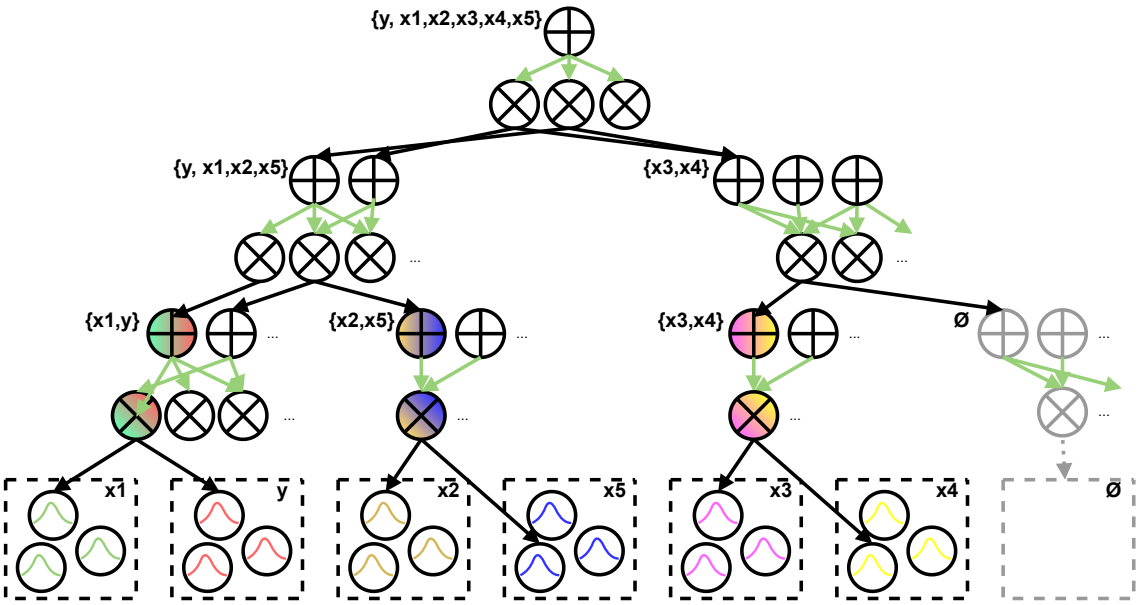


Figure 4.2: Illustration of random constructed sum-product network (RAT-SPN) for the joint distribution $p(x_1, \ldots, x_5, y)$. The product nodes, $\otimes$, always combine 2 nodes from different scopes, while sum nodes, $\oplus$, sum all of the product nodes for similar scopes. Note the drawing is not complete; This illustarates a RAT-SPN with 3 channels, so every cluster of product nodes has size 9 and every cluster of sum nodes (except the root) has size 3.

Figure 4.4 illustrates how 3 simple Gaussian distributions from two different scopes x and y can be multiplied together and defined as many mixtures as the product of the numbers of distributions in each scope (i.e. 9). So by only training parameters for 6 distributions we obtain 9 distributions. In the middle figure, we see a data distribution with no need for all 9 mixture components and the weighting ensures that the unnecessary components are turned off. Figure 4.3 illustrates the graphical representation of the SPN for 2 dimensions - and shows that the SPN for small dimensions is not deep or complicated. In fact, if we look at the right figure in Figure 4.4, the SPN is just adding less flexibility.

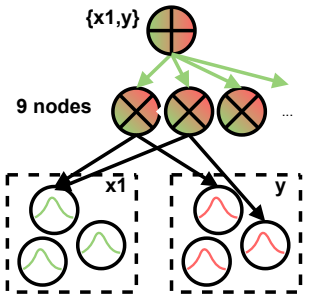


Figure 4.3: Graphical representation of the SPN used in Figure 4.4. The 9 green arrows are weighted and sums to 1

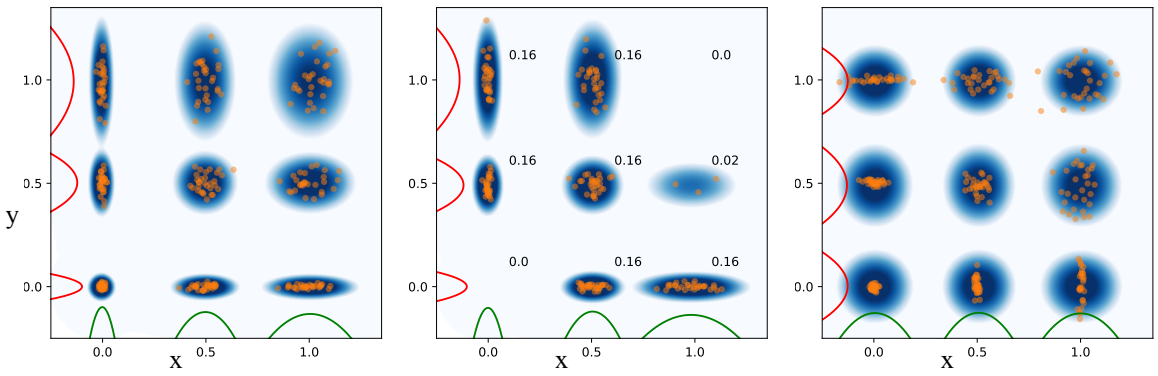


Figure 4.4: SPN on the joint probalility $p(x, y)$ with 3 leaf distributions in each scope (shown on the axes), trained on 3 different data sets. Left: The data lies perfect for the SPN. Middle: Numbers in the graph represent how the weight of each mixture component is weighted. Right: The data is distributed badly for the SPN.

> **Note: SPN as neural network**
> <Lav om> The interpretation of an SPN as a neural network citevergari (what motivated us to look at them in the first place) Here, imagine the weights of the sum nodes are parameters, leaf distributions are input neurons, root node is output and all other nodes correspond to hidden neurons. Note: the depth is at most $\log_2(\#variables)$, yielding not much depth in Bayesian optimization tasks, where the number of dimensions typically is small.

4.5.1 SPN as a mixture model

Each sum-node can be interpreted as a categorical variable [**??**], i.e. a weighted dice. So each mixture component can be found by starting at the root sum-node and rolling the dice of which green arrow to continue the path through the SPN. If the path meets a product-node all children are included in the path. If the path meets a sum-node we roll a dice. Finally, if the path meets a leaf-node it terminates. The defined path is referred to as a sub-network, S_z, in the SPN and is equivalent to a mixture component. The total number mixture components equal the product of all sum-nodes children, i.e. $Z = \prod_{i \in \mathcal{S}um(S)} |ch(i)|$. i.e. an exponentially large amount.

Denote the set of edges in the sub-network $\mathcal{E}(S_z)$. Now the we define a mixture coeficient, λ_z and component for each S_z as

$$\lambda_z := \prod_{(i,j) \in \mathcal{E}(S_z)} w_{i,j}, \qquad p_z(x, y|\theta) := \prod_{i \in \mathcal{L}(S_z)} \phi_i(x, y),$$

where $\phi_i(x, y)$ is the leaf distribution at leaf node i paramitised with θ. It can now be proven that the SPN can be interpreted as the following mixture model,

$$p(x, y|w, \theta) = \sum_{z=1}^{Z} \lambda_z(w) p_z(x, y|\theta)$$

i.e. by the weighted sum of all Z sub-networks.

4.5.2 Conditional of SPN

Conviniently we can define the mixture model as

$$\begin{aligned} p_z(x,y) &= \prod_{i \in x\mathcal{L}eaf(z)} \phi_i(x) \prod_{i \in y\mathcal{L}eaf(z)} \phi_i(y) \\ &=: p_{z_x}(x) p_{z_y}(y) \end{aligned}$$

<obs connect til forgående sections> giving the conditional of the mixture and the responsibility

$$p(y|x) = \sum_{z=1}^{n} \gamma_z(x) p_{z_y}(y) \tag{4.8}$$

$$\gamma_z(x) = \frac{\lambda_z p_{z_x}(x)}{\sum_{z \in \Sigma(S)} \lambda_z p_{z_x}(x)} \tag{4.9}$$

calculation of responsibility

The responsibility of a datapoint to belong to one mixture component, is given by

$$\gamma_z(x) = \frac{\lambda_z p_z(x)}{\sum_{z^*} \lambda_{z^*} p_{z^*}(x)}$$

We can prove that the responsibility is equal to the gradient of the log likehood,

$$L := \sum_n \log \sum_z \lambda_z \exp \psi_z(x_n)$$

where we define $\psi_z(x_n) = \log p_z(x_n)$. Take the gradient

$$\frac{\partial L}{\partial \psi_z(x_n)} = \frac{\lambda_z p_z(x_n)}{\sum_{z^*} \lambda_{z^*} p_{z^*}(x)}$$

Note that the gradient easily can be found using automatic differentiation.

4.6 Mixture model training

The following section presents the expectation-maximization algorithm, which is used to train the Gaussian mixture model and the SPN.

4.6.1 Expectation-maximization for mixture models

Mixture models can be seen as probibalistic graphical models, <fig> there one mixture component is picked according to the realization of a catagorical variable $\mathbf{Z}$ with parameters according the the mixture weights, i.e we can reformelate,

$$p(x) = \sum_{k=1}^{K} w_k p_k(x) \tag{4.10}$$

$$\iff \qquad p(x) = p_z(x), \qquad z \sim Cat(w_i, \dots, w_K). \tag{4.11}$$

In fact $p_z(x)$ is a conditial distribution, $p(x|z)$, and combined with the distribution of Z we can define the joint

$$p(x, z) := p_z(x)p(z)$$

The the case of a statistical model, data $\mathcal{D}$ is fitted by the mixture model by tuning the model parameters $\theta = \{w, \text{paramers for} p_i\}$. Then the joint distribtuion $p(\mathcal{D}, z|\theta)$ is refered as the *complete-data* likehood in the EM algorithm.

$$p(\mathcal{D}, z|\theta) := p(\mathcal{D}|z, \theta)p(z|\theta)$$

When fitting model parameters we essentially want to find the parameters, that maximize the probability of the parameters given the data, $p(\theta|\mathcal{D})$. Assuming an unimformative/flat prior $p(\theta)$,

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}$$

$$\Rightarrow \arg\max_\theta p(\theta|\mathcal{D}) = \arg\max_\theta p(\mathcal{D}|\theta)$$

we arrive at the maximum likelihood estimate (MLE). The task of finding the MLE is conviniently done using EM algorithm, since we can look at the likehood as the marginalized *complete-data* likehood,

$$p(\mathcal{D}|\theta) = \sum_z p(\mathcal{D}, z|\theta)$$

Background: Expectation-maximization EM <based on [12]

Expectation maximization is a convinient method for finding ML (or MAP) estimate of a latent variable model. We consider a probibalistic model paramitised with θ,

$$p(\mathbf{X}, \mathbf{Z}|\theta)$$

where we denote all latent variables $\mathbf{Z}$, and observed variables $\mathbf{X}$. Our goal is to find the maximum of the likehood,

$$p(\mathbf{X}|\theta) = \int p(\mathbf{X}, \mathbf{Z}|\theta)\mu(d\mathbf{Z})$$

maximizating the likehood itself $p(\mathbf{X}|\theta)$ is assumed dificult but maximizating of the *complete-data* likehood $p(\mathbf{X}, \mathbf{Z}|\theta)$ is much easier. The algorithm iterates over two steps: The expecation (E) step and the maximization (M) step, defined in the following way for iteration t,

E-step

Define the functional $Q(\theta, \theta^{(t)})$, to be the expected value of the complete-data log likehood (log likehood function of θ), with respect to the only random quantaty $\mathbf{Z}$, which is assumed to follow a distribtuion with the density $p(\mathbf{Z}|\mathbf{X}, \theta^{(t)})$, i.e. the conditional distribution of $\mathbf{Z}$ given $\mathbf{X}$ and the current parameter point estimate $\theta^{(t)}$:

$$Q(\theta, \theta^{(t)}) := E_{p(\mathbf{Z}|\mathbf{X},\theta^{(t)})}[\log p(\mathbf{X}, \mathbf{Z}|\theta)]$$

M-step

After the E-step we find the point estimate $\theta^{(t+1)}$ which maximizes $Q(\cdot|\theta^{(t)})$, i.e.

$$\theta^{(t+1)} = \arg\max_{\theta} Q(\theta|\theta^{(t)})$$

(local) maximization of $p(\mathcal{D}|\theta)$

Input: dataset $\mathcal{D}$, joint model $p(\mathcal{D}, \mathbf{Z}|\theta)$
while Not converged **do**
  $Q(\cdot, \theta^{(t)}) \leftarrow E_{p(\mathbf{Z}|\mathcal{D},\theta^{(t)})}[\log p(\mathcal{D}, \mathbf{Z}|\cdot)]$ ▷ E-step
  $\theta^{(t+1)} \leftarrow \arg\max_\theta Q(\theta|\theta^{(t)})$ ▷ M-step
return: $\theta^{(end)}$

Proof of correctness

We will now give a short proof that maximizing $Q(\cdot|\theta^{(t)})$ maximizes the likelihood $p(\mathbf{X}|\theta)$, where we assume that $\mathbf{Z}$ is a random vector with a discrete distribution. This allow us to use Gibbs inequality:

$$\sum_z p_1(z) \log p_1(z) \geq \sum_z p_1(z) \log p_2(z)$$

where $p_1(\cdot)$ and $p_2(\cdot)$ are densities belonging to two discrete distributions of Z, equality if $p_1(\cdot) = p_2(\cdot)$. From now on we will alter the subscript on the expecations, just have in mind that

$$E_{\theta^{(t)}}[g(Z)] := E_{p(\mathbf{Z}|\mathbf{X},\theta^{(t)})}[g(Z)] = \sum_z g(z) p(\mathbf{z}|\mathbf{X}, \theta^{(t)})$$

Now to the proof: From bayes rule $p(\mathbf{X}|\theta) = \frac{p(\mathbf{X},\mathbf{Z})}{p(\mathbf{X})}$ we can write

$$\log p(\mathbf{X}|\theta) = \log p(\mathbf{X}, \mathbf{Z}) - \log p(\mathbf{Z}|\mathbf{X}, \theta)$$

Now, taking the expecation of the above w.r.t. $p(\mathbf{Z}|\mathbf{X}, \theta^{(t)})$, yields,

$$\begin{aligned}\log p(\mathbf{X}|\theta) &= E_{\theta^{(t)}}[\log p(\mathbf{X}, \mathbf{Z}|\theta)] - E_{\theta^{(t)}}[\log p(\mathbf{Z}|\mathbf{X}, \theta)] \\ &= Q(\theta, \theta^{(t)}) + E_{\theta^{(t)}}[\log p(\mathbf{Z}|\mathbf{X}, \theta)]\end{aligned}$$

Since the above equation holds for any θ, it also holds for $\theta^{(t)}$ now we have,

$$\log p(\mathbf{X}|\theta^{(t)}) = Q(\theta^{(t)}, \theta^{(t)}) + E_{\theta^{(t)}}[\log p(\mathbf{Z}|\mathbf{X}, \theta^{(t)})]$$

Subtracting the two equations, we get,

$$\log p(\mathbf{X}|\theta) - \log p(\mathbf{X}|\theta^{(t)}) = Q(\theta, \theta^{(t)}) - Q(\theta^{(t)}, \theta^{(t)}) + E_{\theta^{(t)}}[\log p(\mathbf{Z}|\mathbf{X}, \theta)] - E_{\theta^{(t)}}[\log p(\mathbf{Z}|\mathbf{X}, \theta^{(t)})]$$

From Gibb's inequality we have that $E_{\theta^{(t)}}[\log p(\mathbf{Z}|\mathbf{X}, \theta^{(t)})] \leq E_{\theta^{(t)}}[\log p(\mathbf{Z}|\mathbf{X}, \theta)]$ where equality only holds for $\theta^{(t)} = \theta$, giving

$$\log p(\mathbf{X}|\theta) - \log p(\mathbf{X}|\theta^{(t)}) \geq Q(\theta, \theta^{(t)}) - Q(\theta^{(t)}, \theta^{(t)})$$

so optimizing $Q(\theta, \theta^{(t)})$ will optimize $\log p(\mathbf{X}|\theta)$ as least as much.

4.6.2 EM for Gaussian mixture

For a Gaussian mixture the p_k distributions in (4.10) is substituted by Gaussain pdfs, i.e. $p_k(x) = \mathcal{N}(x|\mu_k, \Sigma_k)$ and the density of a catagorical distribution is $p(z) = \sum_{k=1}^{K} 1_{z=k} w_k = w_z$, combining the two we get the joint distribution,

$$p(x, z|w, \mu, \Sigma) = w_z \mathcal{N}(x|\mu_z, \Sigma_z)$$

Taking the log and defning $\theta = \{\mu_1, \Sigma_1, w_1, \ldots, \mu_K, \Sigma_K, w_K\}$, and assuming iid data

$$\log p(X, Z|\theta) = \sum_{i}^{n} (\log(w_{z_i}) + \log(\mathcal{N}(x_i|\mu_{z_i}, \sigma_{z_i})))$$

Now we are ready to calculate $Q(\theta, \theta^{(t)})$, by taking the expectation of the complete-data log likehood with respect to the distribution, $p(Z|X, \theta)$,

$$\begin{aligned} E_{p(Z|X,\theta^{(t)})}[\log p(X, Z|\theta)] &= \sum_{i}^{n} E_{p(Z|X,\theta^{(t)})}[p(X_i, Z_i|\theta)] \\ &= \sum_{i}^{n} E_{p(z_i|x_i,\theta^{(t)})}[p(x_i, z_i|\theta)] \end{aligned}$$

Expectation with repect to unnecessary variables

the last equation holds since taking expecation over a function of a random variable x with respect to a that random variable and more random variabels, x, y, is equivalent to the expecation with respect to just x, i.e.

$$\begin{aligned} E_{x,y}[g(x)] &= \int\int g(x)p(x, y)dydx \\ &= \int g(x) \int p(x, y)dydx \\ &= \int g(x)p(x)dy = E_x[g(x)] \end{aligned}$$

the posterior distribution is calculated the following way,

$$\begin{aligned} p(z|x, \theta^{(t)}) &= \frac{p(x, z|\theta^{(t)})}{p(x|\theta^{(t)})} \\ &= \frac{p(x, z|\theta^{(t)})}{\sum_z p(x, z|\theta^{(t)})} \\ &= \frac{w_z^{(t)} \mathcal{N}(x|\mu_z^{(t)}, \Sigma_z^{(t)})}{\sum_{k=1}^{K} w_k^{(t)} \mathcal{N}(x|\mu_k^{(t)}, \Sigma_k^{(t)})} \end{aligned}$$

For simplificaiton we will denote, $\gamma^{(t)}(z_i) := p(z_i|x_i, \theta^{(t)})$, interpreted as the probability of datapoint x_i to belong to class z_i. Bishop [12] calls this probability function the *responsibility*. We can now conclude the **E-step**.

$$
\begin{aligned}
Q(\cdot, \theta^{(t)}) &= \sum_{i=1}^{n} p(x_i, z_i|\cdot)\gamma^{(t)}(z_i) \\
&= \sum_{i=1}^{n} \left[\gamma^{(t)}(z_i)\log(\cdot_{z_i}) + \gamma^{(t)}(z_i)\log(\mathcal{N}(x_i|\cdot_{z_i}, \cdot_{z_i}))\right]
\end{aligned}
$$

or more concretely $\theta = \{\mu_1, \Sigma_1, w_1, \ldots, \mu_K, \Sigma_K, w_K\}$,

$$
Q(\theta, \theta^{(t)}) = \sum_{i=1}^{n} \gamma^{(t)}(z_i)\log(w_{z_i}) + \gamma^{(t)}(z_i)\log(\mathcal{N}(x_i|\mu_{z_i}, \Sigma_{z_i})).
$$

$Q(\cdot, \theta^{(t)})$ is a concave function - the Gaussian is log-concave and a sum of concave functions is also concave - so it is sufficient and nessesary to find its maxima by the root of its derivative,

$$
\frac{d}{d\theta}Q(\theta^*, \theta^{(t)}) = 0 \iff \theta^* = \arg\max_{\theta} Q(\theta, \theta^{(t)})
$$

Giving the updates...

EM for SPN

4.7 Summary

In the chapter, we introduced the seemingly, novel idea of using a generative model as a surrogate model in Bayesian optimization. We limited our scope to (Gaussian) mixture models, and quickly realized the problem of naively using the conditional as a predictive distribution for low-density areas. This was solved by introducing a prior (background) zero-mean Gaussian with high variance. How much the prior will influence the predictive distribution will be treated as a hyper parameter in the upcoming experiements. We continued by deriving the mean and variance of the predictive distribution (which will be used in the Expected improvement). We shortly introduced the different mixtures. illustrations of the models (in 1D) are found in the result chapter. The newly rising class of mixture models, SPNs, were introduced, illustrated and plotted. Finally we also shortly covered the training of the mixture models, i.e. using the EM algorthm to maximize the likelihood. With this chapter, we are soon ready to test all the surrogate models, but first, we need to specify the models more.

5 Implementation

As the models defined in the previous chapters have not been specified enough, we here present the specified surrogate models and methodologies for the upcoming results. Furthermore, we shortly explain the implementation details of the Bayesian optimization routine.

5.0.1 Bayesian Optimization

The Bayesian optimization framework is implemented as a python class. It is possible to pass the class two different types of problem objects (COCO and sklearn test problems), and additionally, the class needs to be given one of the defined surrogate models (also an object). By defining the BO methodology in this modular fashion, we make sure that we are treating all experiments equally.

Initially, 5 data points are sampled randomly from the domain $\mathcal{X}$ and evaluated in $f(\cdot)$ (all defined in the problem class). Next, an optimization step is performed repeatedly, until a predefined budget (i.e. how many samples from f the user allows) is used up. The optimization step is simply a fitting procedure, followed by finding the point x_{next} which maximizes the acquisition function, and including the point and the evaluation $f(x_{\text{next}})$ into the dataset. The maximization of the acquisition function is done by testing $d \cdot 5000$ random points on the domain, where d is the dimension of the problem.

5.0.2 Surrogates

All surrogate models are defined as Python classes, with similar function names, i.e. *predict*, *fit* and *name*, such that they fit into the BO framework (described in the above section). In the following, we present how all the surrogate models are implemented. First, we describe the implementation of the discriminative surrogate models (GP, BNN, BOHamiANN). Next, we describe the implementation of the generative/mixture surrogate models (GMR, KDE, SPN).

Discriminative surrogates

The Gaussian process regression is implemented using the GPy framework, with the Matern kernel with $\nu = 1.5$. The hyperparamteres, i.e. the lengthscale, variance and observation variance are optimized by maximizing the marginalized likelihood using the limited memory quasi-newton solver with bounds $(10^{-3}, 2)$. l-bfgs-b max 1000 iterations with 20 restarts. The bounds are reasonable since the data is always standardized.

The Bayesian neural network(BNN) is a fully connected neural network with 3 layers for 50 tanh nodes. It is implemented using Numpyro - which is a python library for probabilistic machine learning, developed by some of the people behind Pyro. Different from Pyro, Numpyro uses Jax (and not PyTorch) as the backend. This allows for a significantly large speedup when doing MCMC, i.e. NUTS sampling. The MCMC routine is still quite slow and we limit the sampling size to 500 burn-in samples and 500 (hopefully) posterior samples with 4 chains - this showed good empirical results on 1D problems. The prior distribution for weights and bias is a standard normal distribution. This is reasonable since the data is always standardized. As we are (mostly) dealing with noise-free problems, we put an informative prior of $InvGa(1000, 1)$ on the observation variance i.e. giving a very small observation variance.

The BOHamiANN model is presented in the paper [4] and is implemented as it is implemented default in ”github.com/automl/pybnn”, which is a fully connected Bayesian neural network with 50 tanh nodes on 3 layers. I.e. Same architecture as we use for our implementation, BNN.

Model	Specification	Training
BNN	3 layers of 50 tanh nodes $p(\mathbf{w}, b) \sim \mathcal{N}(0, 1)$ $p(\sigma) \sim$ InvGa$(1000, 1)$	NUTS(500 burn-in, 500 samples)
BOHamiANN	3 layers of 50 tanh nodes $p(\mathbf{w}, b) \sim \mathcal{N}(0, \sigma_w)$ $p(\sigma) \sim$ LogNormal$(??)$ $p(\sigma_w) \sim$ Gamma$(??)$	Adaptive stochatic HMC (1000 burn-in, use every 10 of 2000 samples)
GP	Mantern kernel 52	Maximize marginalized likelihood 20 restarts of BFGS optimization

Table 5.1: Overview of chosen discriminative surrogate models

Generative surrogates
We now present how the generative/mixture models (GMR, KDE, SPN) are implemented more concretely. Firstly, all the models have some hyperparameters, which we will not fix,

- GMR: Number of components, prior weight.
- KDE: Variance of x and y components for all component, prior weight.
- SPN: Prior on the variance of x and y components, prior weight.

We select the hyperparameters using 30 fold (or leave one out for data size smaller) cross-validation, where the models are scored on their mean predictive log probability, i.e. uncertainty quantification. We test 40 different points using Bayesian optimization outside the crossvalidation loop. This is done using the Python package ”scikit-optimize”.

The Gaussian mixture regression (GMR) is implemented using the software[17], which takes in mixture components and weights trained using sklearn's Gaussian mixture model. Which we train using 20 restarts.

The kernel decision estimator regression is implemented using numpy and fast vector operations. As mentioned the standard deviation of all is mixture components in the x and y direction is found using cross-validation in the range between $(10^{-3}, 0.3)$.

SPN is implemented using software similar to RAT-SPN but implemented by Mikkel N. Schmidt. It is trained using the EM algorithm to maximize the posterior i.e. finding the MAP estimate. The posterior is found in closed form since the component distributions are Gaussian and the prior is given as an inverse gamma (i.e. a conjugate prior). Inverse gamma distributions have two parameter the shape $\alpha > 0$ and the scale $\beta > 0$, and a mean, $\frac{\beta}{\alpha-1}$ for $\alpha > 1$, and a variance $\frac{\beta^2}{(\alpha-1)^2(\alpha-2)}$ for $\alpha > 2$. Looking at plots of the distribution we choose to fix $\beta = 1$ and select $\alpha \in [1, 200]$ using cross-validation.

Model	Specification	Training
GMR	Training sklearn GMM, EM of MLE with 3 restarts. pluggin into GMR software [17].	BO Cross-validation 30 fold, tuning number of components and prior weight
KDE	Gaussian around all datapoints with a x-variance and y-variance	BO Cross-validation 30 fold, testing x-variance and y-variance and prior weight
SPN	Trained using 3 restarts of EM of MAP,	BO Cross-validation 30 fold, tuning: hyperpriors and prior weight

Table 5.2: Overview of chosen generative surrogate models

Whereas a true Bayesian would not choose a specific type of model, but use all models , however, this would indeed be infeasible. We need to limit the scope, For the GP, we do 20 restarts in the emperical maximization. And the manteen kernel prior with $\nu = 2.5$. BNN, is a 50x50x50 tanh layers with standard Gaussain priors on. And an inverse gamma (1000, 1) prior on the noise. Trained with 100 burn-in and 100 samples. BOHAMIANN is similar but uses a different noise prior, and trains using the adaptive HMC method. The mixture models are trained using crossvalidation scored on the mean predictive log .. posterior (formally speaking this is not a likelihood, but we will keep it as it is almost the same) The tuning parameters We do 40 iterations SPN is trained using EM with maximum 1000 epochs (is terminated if the progress is stalling). it is trained 2 times. Lav den historie GMR is trained using EM but this is done by the sklearn software.

5.0.3 Standardized data

Before the data reach any of the models, it is standardized. this is a scaling and a translation of the data, such that the datas empirical mean and standard deviation are 0 and 1, respectively. Using standardized data makes it much easier to control the parameters in the models since the data used in fitting and prediction is always on the same scale. The first time the models see the data, the empirical mean and standard deviation are recorded both for x and y, (note x can be a vector), giving μ_x,μ_y, $\sigma_x$and σ_y. When fitting the data, all data is transformed using the transformation,

$$T(x, y) := \left(\frac{x - \mu_x}{\sigma_x}, \frac{y - \mu_y}{\sigma_y} \right).$$

When doing prediction i.e. calculating $p(y|x)$, we use the above transform on the input x, and the model output y is transformed back to the original domain using the inverse transform,

$$T_y^{-1}(\hat{y}) := \hat{y} \cdot \sigma_x + \mu_x.$$

6 Results

In the following chapter, we want to test the surrogate models both as regression models and as surrogate models in a Bayesian optimization setting. Firstly we stay in the 1D domain on self-defined test problems, which will give an intuition about the capability of the generative models as well as the discriminative models. We have designed 4 test functions, which are defined to give an advantage to all the regression models. Secondly, we go to the higher dimensional problems from the BBOB noiseless problem set. This is a popular set of problems with 24 different problems for continuous black-box optimization algorithms.

6.1 Regression analysis methodology

As described in the previous sections, an essential part of Bayesian optimization and the decision theory built around Bayesian optimization is that the regression model is correct. We will therefore look at how good the regression models are in terms of accurate prediction and correct uncertainty estimation. The following provides details on how exactly this is done.

6.1.1 Uncertainty quantification

Well-calibrated uncertainty estimation is important for a probabilistic regression model. A good fit is scored on the model's ability to assign a non-zero (hopefully large) probability to every test point, i.e. we hope for a given test point (x_t, y_t), that $p(y_t|x_t, \mathcal{D})$ is large.

One way to score the fit on a large test set $(x_t, y_t)_{t=1}^T$, is to simply calculate the average probability assigned to all $y_1, ..., y_T$, i.e.

$$Score_1 = \frac{1}{T}\sum_{t=1}^{T} p(y_t|x_t, \mathcal{D}),$$

however, if the regression model is really certain about one test point and guess this point correct, we could have potentially $Score_1 = \infty$ as $p(y_t|x_t, \mathcal{D})$ could be arbitraty certain about this 1 point. Meanwhile, test points with assigned probability, i.e $p(y_t|x_t, \mathcal{D}) = 0$, will not matter.

Another way to score the fit on the large test, is to score the joint predictive probability (with assumed iid test data),

$$p(\mathbf{y}|\mathbf{x}, \mathcal{D}) = \prod_{y=1}^{T} p(y_t|x_t, \mathcal{D})$$

Now, if just one test point is assigned zero probability the regression model has failed overall. Additional, on a computer with finite-precision small probabilities might underflow to 0 and course the score to be 0. We overcome underflow with a log transformation and take the mean value to compare with different amounts of test data.

$$Score_2 = \frac{1}{T}\log p(\mathbf{y}|\mathbf{x}, \mathcal{D}) = \frac{1}{T}\sum_{t=1}^{T} \log p(y_t|x_t, \mathcal{D})$$

this score, the *mean predictive log probability*, we will use in the experiments conducted in this thesis.

6.1.2 Predictive power

The performance of a deterministic regression model $f_w(\cdot)$ is typically measured in mean squared error or mean absolute error, i.e.

$$MAE = \frac{1}{T}\sum_{t=1}^{T}|f_w(x_t) - y_t|.$$

For the probabilistic regression model, we want a similar measurement - especially if the uncertainty quantification fails. It makes sense to use the most probable outcome of the predictive distribution, i.e. the mode. However, if it is Gaussian the mode is equal the mean. The mean absolute used in this project is given as, $MAE := \frac{1}{T}\sum_{t=1}^{T}|\mu_{\mathcal{D}}(x_t) - y_t|$. However to make the performance relateable across problems, we use the mean relative error,

$$\text{Mean relative error} := \frac{1}{T}\sum_{t=1}^{T}\frac{|\mu_{\mathcal{D}}(x_t) - y_t|}{|y_t|}.$$

6.1.3 Benchmark surrogate model

Before moving on with the results, we want to introduce an important benchmark surrogate model, the *emperical mean and std* model (always plotted in black). This surrogate's predictive distribution is a simple normal distribution with mean and variance of the training y-data i.e. $\mathbf{y}$. More formally it is defined as,

$$p(y|x\mathcal{D}) = \mathcal{N}(y|\bar{\mathbf{y}}, \bar{\sigma}^2(\mathbf{y})),$$

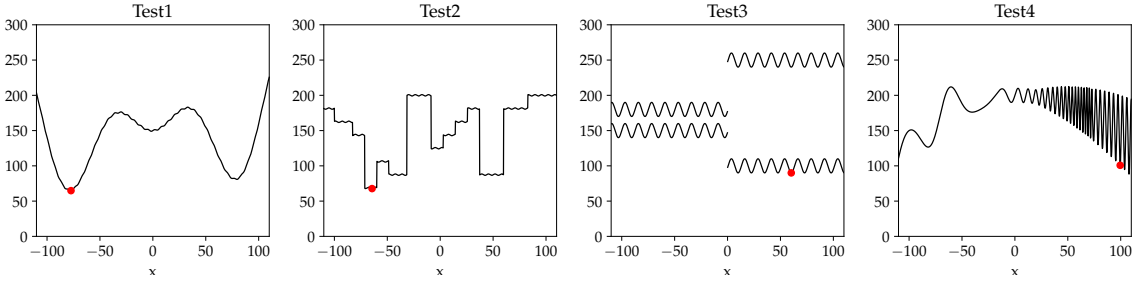
where $\bar{\mathbf{y}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{y}_i$ and $\bar{\sigma}^2(\mathbf{y})) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{y}_i - \bar{\mathbf{y}})^2$. In the context of Bayesian optimization, it is imidiately a bad model since it sees as all x points/locations as equally good. We will use this model in the benchmark for the Bayesian optimization as it will corespond to random search.

6.2 Model comparison on test problems (regression)

Bayesian optimization is typically performed on black-box functions, and thereby the Bayesian regression model is trained on data from all kinds of underlying problems. Now we want to test the performance of different regression models on different types of 1D problems, all in the interval $x \in [-100, 100]$:

- Test 1: A well-behaved sine-function,
- Test 2: A highly discontinuous function,
- Test 3: A multi-modal objective function,
- Test 4: An anisotropic objective function.

The functions are illustrated here,



It is common to have many parameters to tune in Bayesian optimization (in 1D, a human could potentially assist the surrogate model since the regression can be plotted and judged by eye). Nevertheless, we will keep the domain in 1D to establish a more informative evaluation of the model performance. Along with tests of predictive power and uncertainty quantification, we have figures of the corresponding predictive distribution for every problem, model, number of training data and across all 10 random seeds.

In the following regression tests, we increase the number of training points (all seeded randomly selected), i.e. $n_{test} \in \{10, 15, 23, 36, 56, 86, 133, 205, 316\}$. Then we fit all regression/surrogate models to the training data, and then 10.000 (also random seeded) test points in the x-interval $x \in [-100, 100]$ are used to evaluate predictive power (mean relative error) and uncertainty quantification (mean predictive log probability)[1] for all the models. This is repeated for 10 different random seeds, to avoid one model being especially lucky with one instance of the training data.

Test1: Well-behaved problem

The first problem will establish a benchmark since this is a simple function to fit. Its exact definition is,

$$f(x) = x \cdot \sin(\frac{x}{50}\pi) + 150 + \frac{x}{10} + \sin(x) \qquad x \in [-100, 100],$$

giving a smooth wave function, with the optimium around -77. Note that $\sin(x) \in [-1, 1]$ contribute with some small high frequency surface waves.

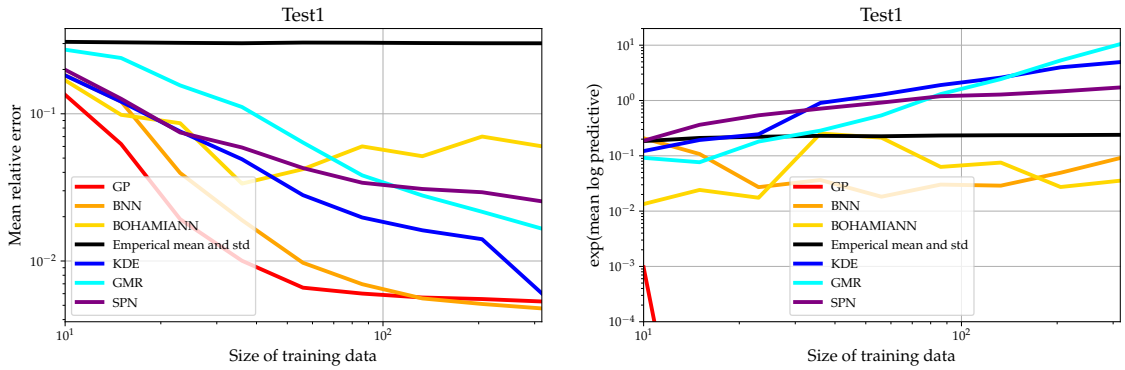


Figure 6.1: Mean relative error (point estimate using the predictive mean) (left) and exponential mean predictive log probalility (right) as functions of number of training data points. All curves reprecent the average across 10 seeded runs for each model.

First of all, we see that all the methods are indeed performing better than the mean prediction (black). The average relative error for the mean prediction is constant at 30%. All models, besides the Gaussian mixture regression (GMR), have an average relative error of less than 10% after just 20 training points. Including more data, the Bayesian neural network and the Gaussian process have the best mean prediction, the third discriminative model, BOHamiANN, seems here to be a weak learner. at 30 data points, it stays at the same error - even when being presented for 316 training data points in the end.

Looking at the right plot in Figure 6.1, we see that the generative models after 20 training points, have an increasingly better uncertainty quantification compared with using the empirical standard deviation. The discriminative models on the other hand perform worse than the empirical standard deviation, especially the GP is doing exceptionally bad - its overconfident, the reason can be seen in

[1] More concretely, we exp-transform this number to make it to plot

Figure 6.2, showing one of the 10 runs for 23 data points. The GP is so certain on some predictions that it assigns almost 0 probability to some of the 10000 test points (i.e the we test all of the line from -100 to 100), yielding very low mean log predictive likelihood.

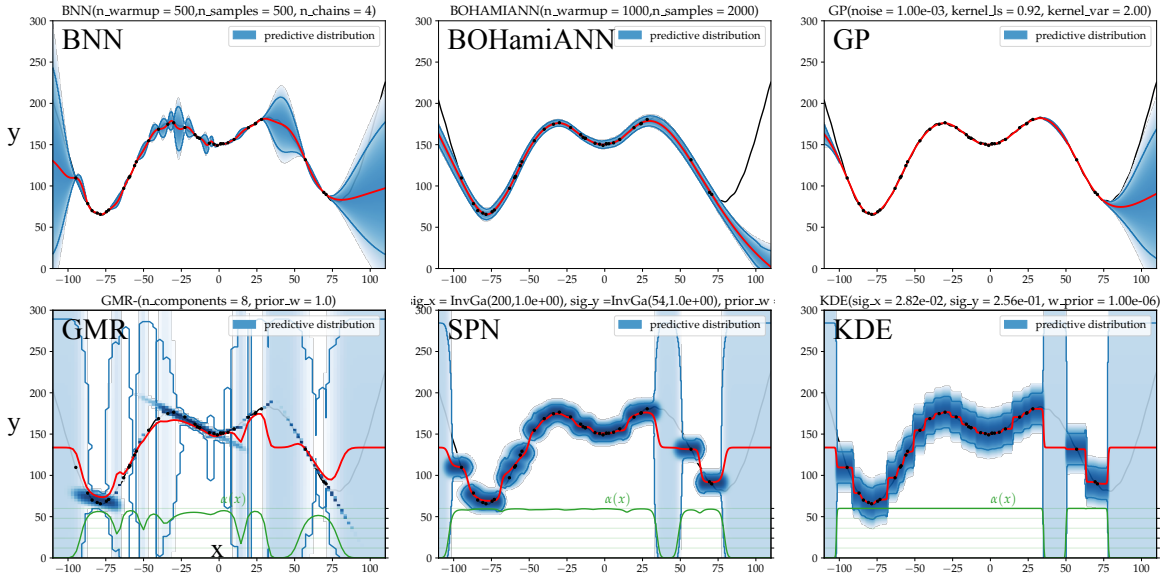


Figure 6.2: Plots visulising the results in figure at $n_{data} = 23$

Test2: Discontinuous problem

As GP and BNN have a natural assumption of continuity we want to investigate their performance on a discontinuous problem. We want to investigate their performance on the folloing step function, defined as

$$f(x) = \sum_{i=1}^{13} u_i \mathbf{1}(I_{i+1} \leq x < I_{i+1}) + sin(x) \qquad x \in]-100, 100]$$

where $\mathbf{1}$ denoted the indicator function and

$$I = [-100, -83, -71, -60, -49, -31, -9, 3, 14, 26, 37, 60, 83, 100],$$
$$u = [181, 162, 144, 69, 106, 88, 200, 125, 144, 162, 181, 88, 181, 200].$$

Outside and on the boundaries $\{-100, 100\}$ the function value is constant 200. This leads to a more deficult problem than case 1.

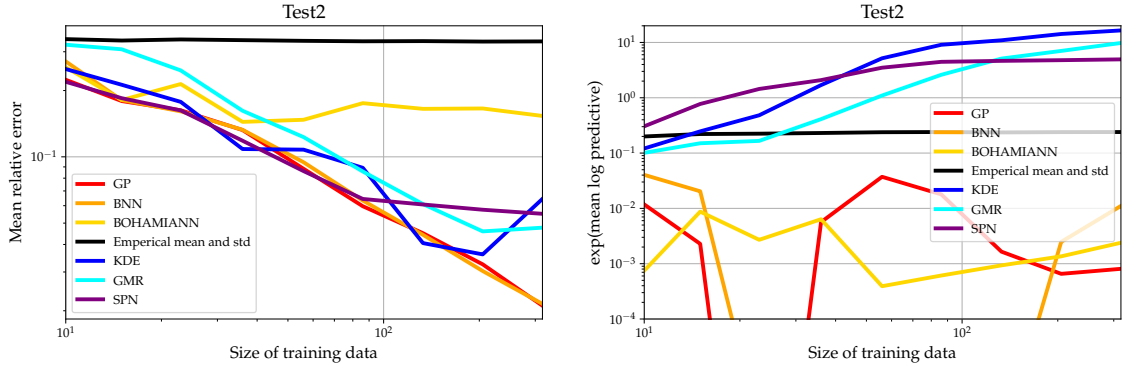


Figure 6.3: Mean relative error (point estimate using the predictive mean) (left) and exponential mean predictive log probalility (right) as functions of number of training data points. All curves reprecent the average across 10 seeded runs for each model.

Looking at Figure 6.3 left we see that this is indeed harder to fit than Test 1, as the reative error is not going down as fast. Comparing to Test 1 we here see that the mixtures are doing as well and some times better than the discriminative models. If instead we used the mode for the mixture regression models, we might see superior performance, as the prediction, will not be continuous. Looing at the Figure 6.3 right we see the same story as Test1 that the mixtures regression has the best uncertainty quantification. Here SPN and KDE perfroms better than Gaussian mixture regression, since the mixture components has no correlation beween x and y, which fits into the. Same story for BOHamiANN as Test1.

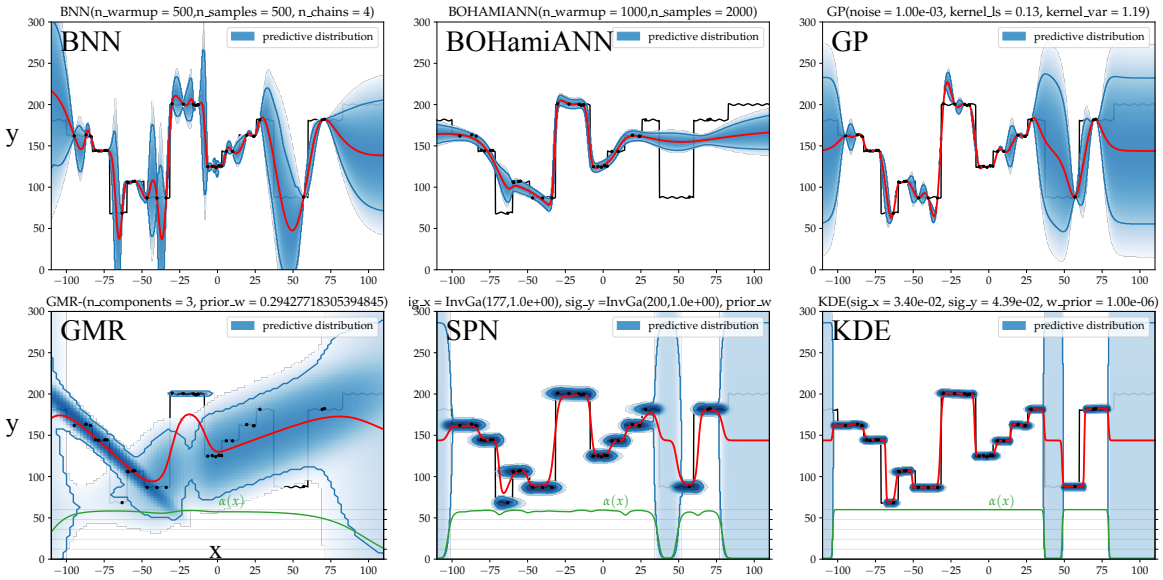


Figure 6.4: Plots visulising the results in Figure 6.3 at n_{data} =?

Test3: multi-modal problem

The following problem is corresponding to a multimodal problem, which could occure in simulations with two equally likely outcomes. We construct this problem to give the mixture models, an advantage over the discriminative models. It is defined as follows,

$$f(x) = 50 * g(x) + 100 + 10 * \sin(0.5 \cdot x) + \epsilon \cdot (30 - g(x) \cdot 90), \qquad x \in [-100, 100],$$

where $\epsilon \sim \text{Cat}(0.5, 0.5)$ and $g(x) = \text{sign}(x) + 1$ ($g(x)$ is 0 for negative x and 2 for positive x).

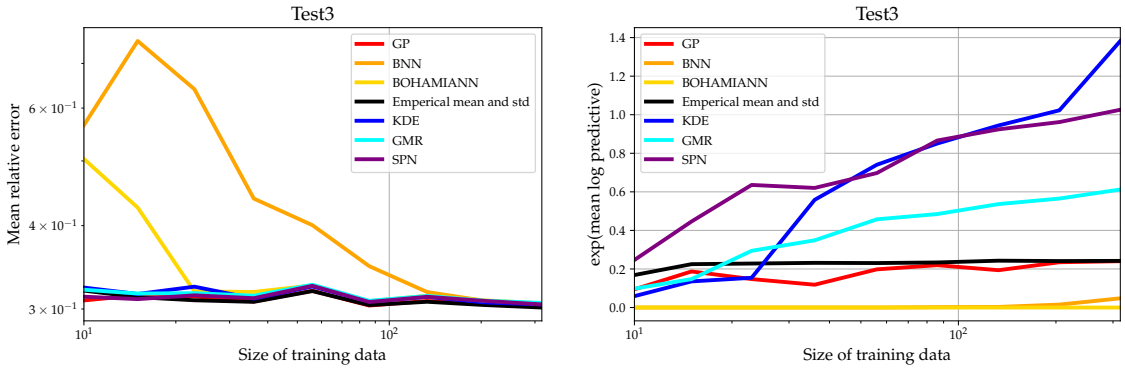


Figure 6.5: Plots visulising the results in above figure.

In Figure 6.5 left we see that none of the models are doing better than the mean prediction. This is

obvious since the data is symmetric and jumping random between two states. The Bayesian neural networks are trying the fit the models, the GP is just turning itself into a mean prediction. In the right plot we see just like previous that the data is parallel with the axis making it perfect the SPN and KDE to fit.

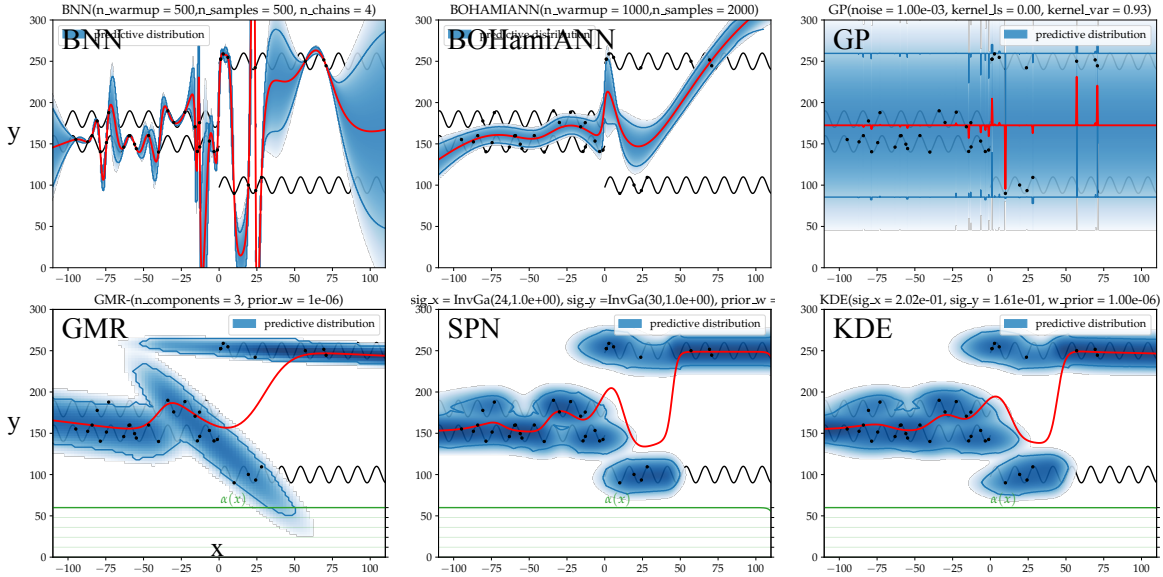


Figure 6.6: Plots visulising the results in figure at $n_{data} =?$

6.2.1 Test4: anisotropic problem

Finally, this problem has a low frequency in the beginning but ends out with a very high frequency. This is designed to be a difficult problem for the GP as this would need a different lengthscale throughout the space.

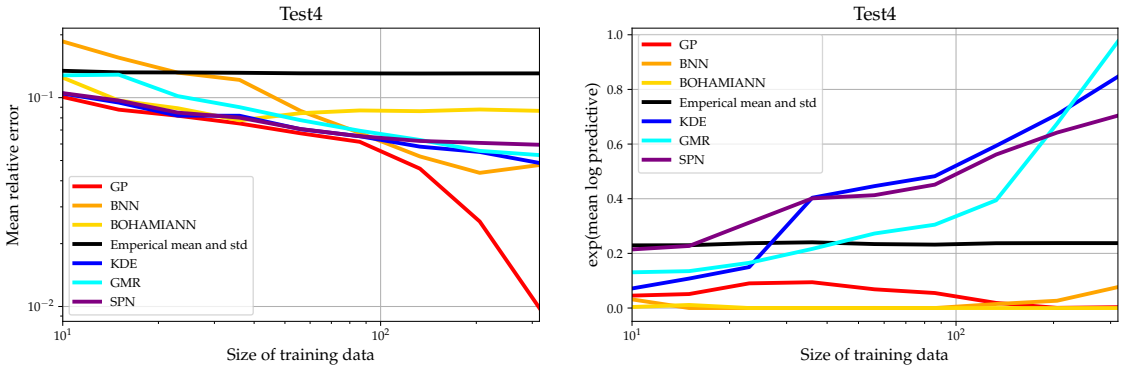


Figure 6.7: Plots visulising the results in above figure.

In Figure B.4 we see that the GP does have a hard time fitting the 36 points, however, when more and more data becomes dense, the GP will lower its lengthscale and fit the data very well. This is seen in appendix Appendix.

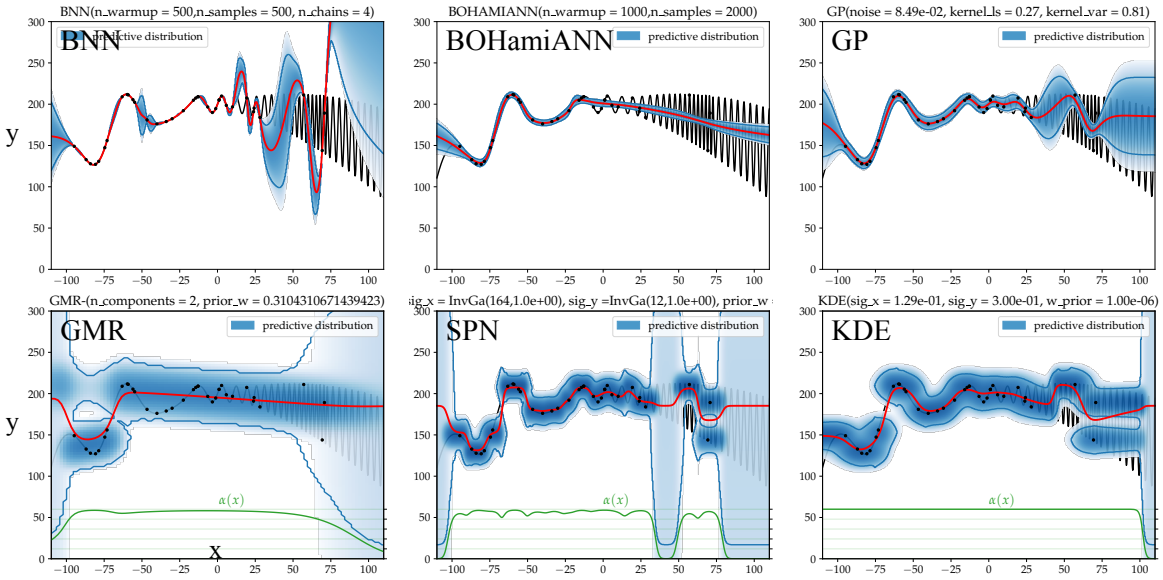


Figure 6.8: Plots visulising the results in figure at $n_{data} =?$

6.3 Model comparison on test problems (Bayesian Optimization)

Now we are ready to test the models in a Bayesian optimization setting. Test3 contains multiple equal minima, and Test4 has a close to minima point on the boundary $x = 100$, so testing them would not be informative. We redefine Test3 and Test4 as seen in Figure 6.9. They still have the same overall structure, so the connecting to the regression analysis should be okay.

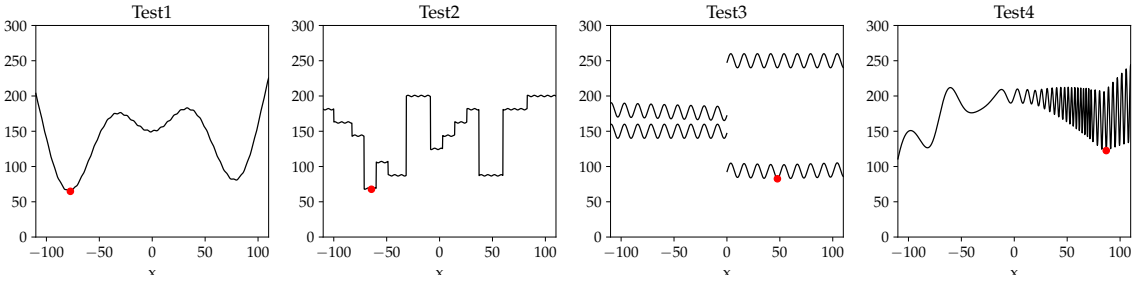


Figure 6.9: All test function used in Bayesian Optimization. We redefine Test3 and Test4

Bayesian optimization using 20 restarts, we see that the GP is in fact the best model in the simple problem Test1; however, for the more complicated problems, the GP is not the preferred model.

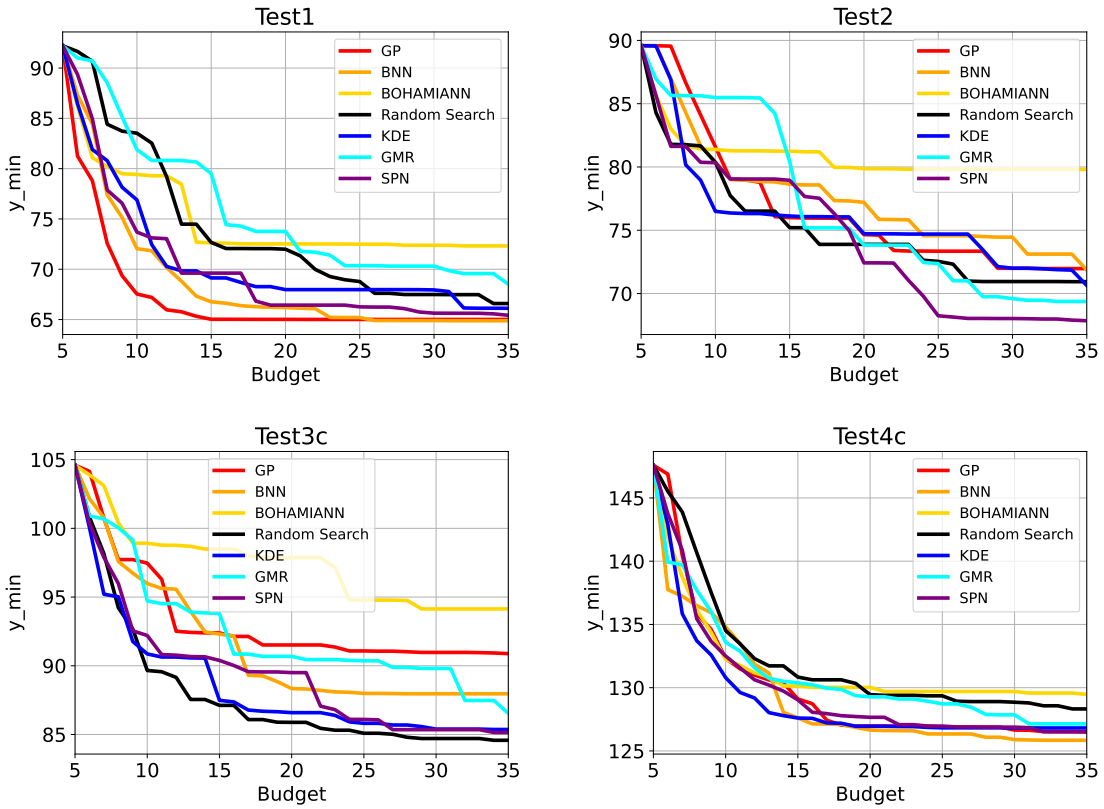


Figure 6.10: Bayesian optimization of the Test functions average accross 20 different samples of initial dataset of size 5.

Looking at Figure 6.10 it is important first of all compare the surrogates to the black random search. E.g. All of the models did worse than random search in Test3. As the regression task failed for test3 this makes sense. However, noticeably, the GP in Test3 is the worst model (together with BOHamiANN) after 16 function evaluations. Test1 is the benchmark and here it becomes evident that BOHamiANN does not work, and GMR is as good as Random search. GP is a clear winner of test1. Test 2 shows after 35 iterations that random search is better than the GP and that all the mixture models and (barely) the BNN performs better than the GP. Finally, test4 shows approximatly the same results for KDE, SPN, BNN and GP, all being better than Random search and BOHamiANN.

6.4 Test on BBOB in higher dimensions

So far, we have looked at our self-defined 1D problems and found problems, where the GP is not the preferred model. We now want to go up in higher dimensions to test the models on more realistic problems. The tests are conducted on 2,3,5,10 dimensions on the *Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions* (BBOB) [19], which consists for 24 functions divided into 5 different categories of difficulty and are scaleable in dimensions. These functions are available through Comparing Continuous Optimizers framework COCO [20], which is a popular benchmarking testbed for continuous black-box optimization algorithms. Due to time and resource constraints on this thesis, we select the functions, f3, f9, f15 and f21 which are plotted in 2D in Figure 6.11.

- f3: Rastrigin Function.
- f9: Rotated Rosenbrock Function.

- f15: Rastrigin Function transformed to be non-separable
- f21: Gallagher's Gaussian 101-me Peaks Function.

For more details, all the 24 functions are described here [19]. First, we test the regression performance of surrogate models on the four benchmark problems in dimensions 2,3,5,10. Secondly, we test Bayesian optimization on them.

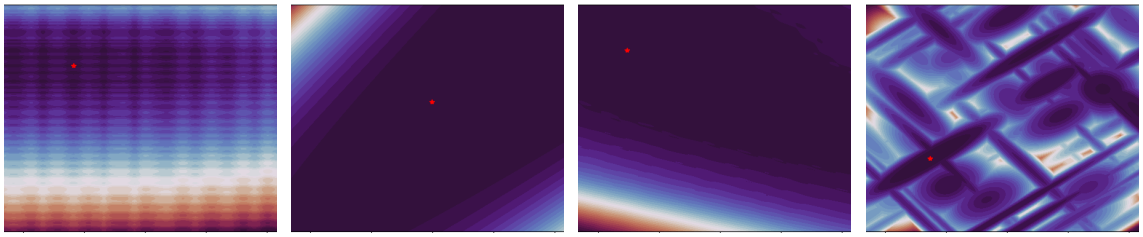


Figure 6.11: COCO Test functions f3, f9,f15, and f21 in 2 dimensions

6.4.1 Regression performance on BBOB

The testing regression performance is the same procedure as the 1D test problems. With 10 seeded random samples, we select $\{10, 15, 23, 36, 56, 86, 133, 205, 316\}$ points to fit the 4 BBOB problems in 4 dimensions (2,3,5,10). All results can be seen in the Appendix. The results are in general just like the 1D test problems, that the GP has the best predictive power, the uncertainty quantification is, however, more equal between the generative and discriminative models. We have selected some noticeable plots here.

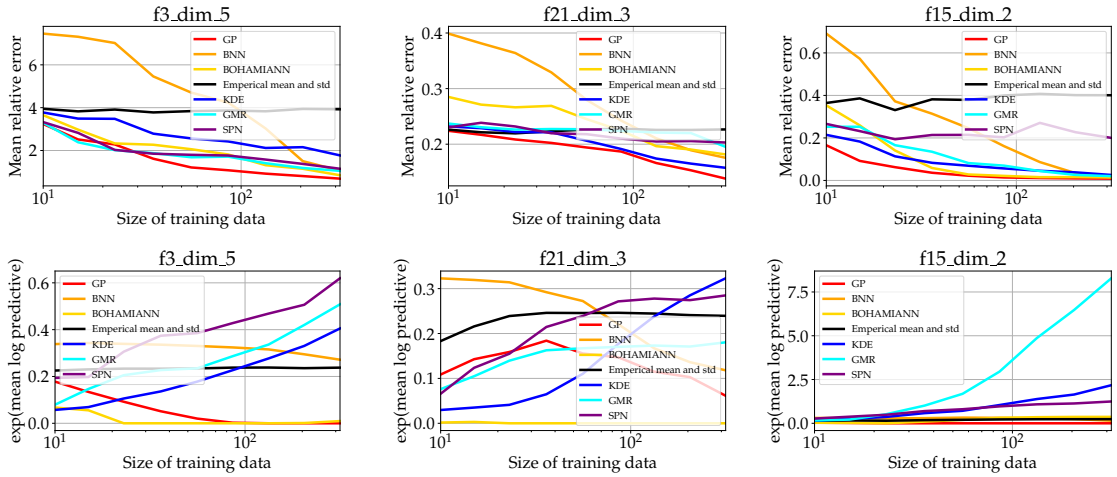


Figure 6.12: Regression performance of selected problems of BBOB. Top: Predictive power. Bottom: Uncertainty quantification. Average across 10 different samples of seeded samples from the problems.

6.4.2 Bayesian optimization performance on BBOB

The Bayesian optimization procedure is as follows: 1) sample 5 random points, 2) do 30 Bayesian optimization iterations. We want to limit optimization paths of being lucky by doing 20 restarts (preferable more, however, due to time and resource limitations). The optimization of the acquisition function is done by $d \cdot 5000$ random guesses. To be fairer across the dimensions, this should have been 5000^d; however, this would have been computational infeasible.

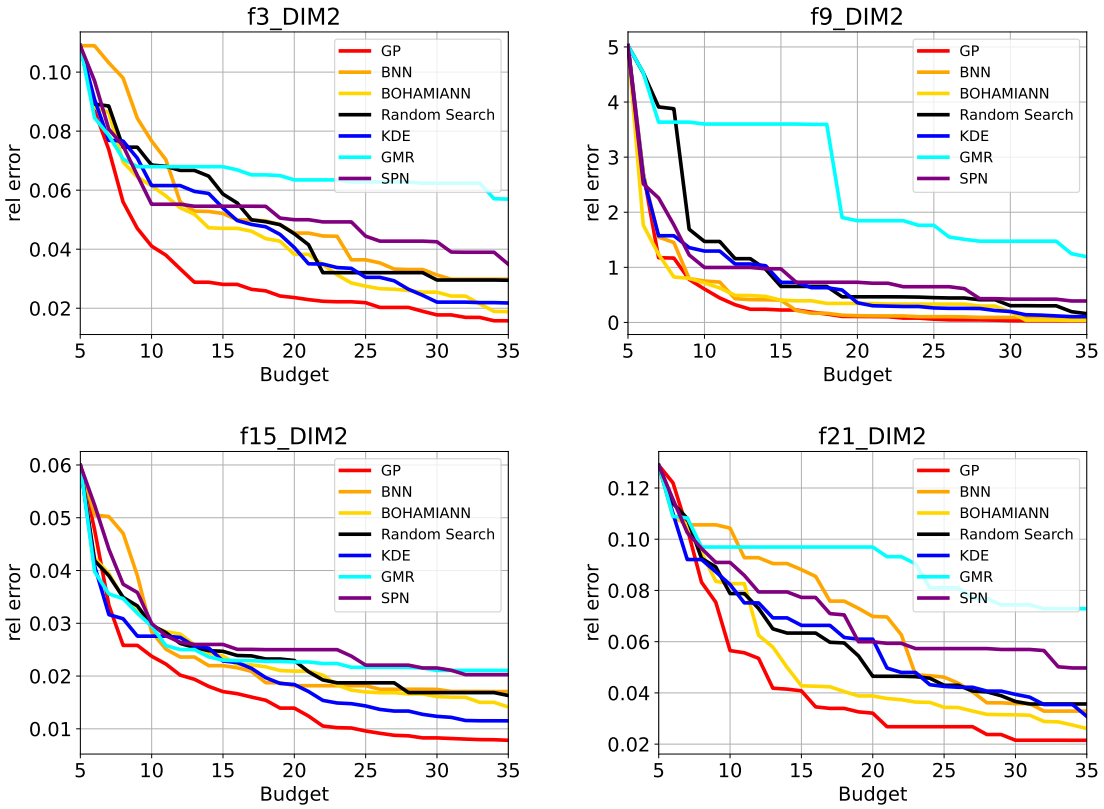


Figure 6.13: Bayesian optimization of BBOB in 2 dimensions, average accross 20 different samples of initial dataset of size 5)

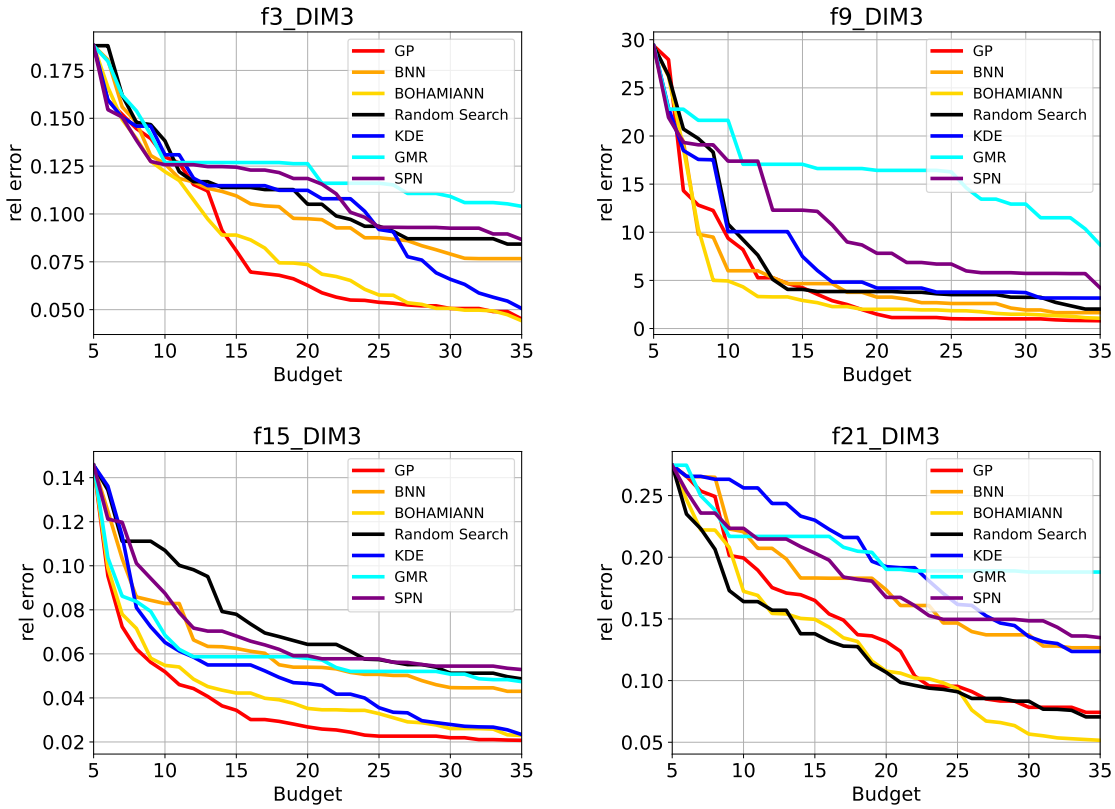


Figure 6.14: Bayesian optimization of BBOB in 3 dimensions average accross 20 different samples of initial dataset of size 5)

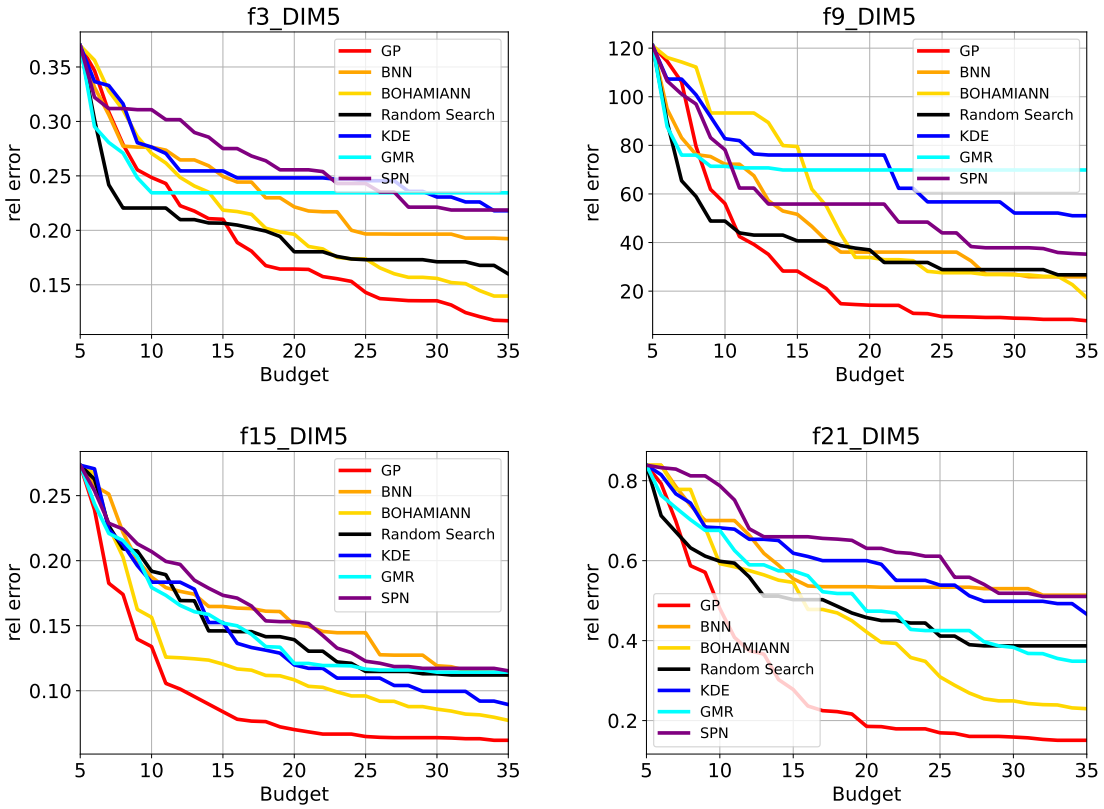


Figure 6.15: Bayesian optimization of BBOB in 5 dimensions average accross 20 different samples of initial dataset of size 5.

In large dimensions there are need for more data for the regression model to be correct? Course of dimensionality..?

In general, across the BBOB tests, we see that GP is the best surrogate model. The other models are doing worse than random search on several tests. This is unfortunate.

7 Discussion

In the following, we will discuss the results has been conducted in the previous chapter and the surrogate models in general.

After the results have been conducted, it is evident that the Gaussian process is a preferred model across the tested problems. We did, however, find problems, where the GP was not the best, those are Test2 and Test3. Therefore, we indicate that the hypothesis is right. There might be more, and in fact, we only looked at 4 self-defined 1D problems and 4 BBOB problems tested. An obvious next step would be to test the surrogate models on all 24 BBOB problems (we test on 4) and for all dimensions (2,3,5,10,20,40). But first of all, we need to discuss the validation of the BO test. A better test would be to,

- Conduct a larger number of BO iterations (we use 30)
- Use more restarts (we use 20)

Especially more restarts is important, in Figure **??** we have plotted the BO optimization paths, which reveals a lot of variation depending on the restarts, it indicates that the BO result might be noisy. .. The PhD thesis [6] uses only 15 restarts, however, here the tests are conducted on 100 iterations and with only 2 initial points. We select to use 5 random samples .. no argument.

We established a connection between the performance of Bayesian optimization and the surrogate model's performance as a regression model. Note that for very small σ^2 which was the case for the Gaussian process (lower bounded with a observation variance of 10^{-3}) the expected improvement is

$$EI(x) = (y_{\text{min}} - \mu_x)\Phi\left(\frac{y_{\text{min}} - \mu_x}{\sigma_x}\right) + \sigma_x\phi\left(\frac{y_{\text{min}} - \mu_x}{\sigma_x}\right) \approx y_{\text{min}} - \mu_x,$$

since the fraction $\frac{y_{\text{min}}-\mu_x}{\sigma_x}$ goes to infinity, the standard Gaussian pdf and cdf become 0 and 1, respectively. The connection between the mixture regression performance and as Bayesian optimization should not be as clear. Indeed Test3 shows an example where the mixtures ... this is due to the Gaussian approximation. Furthermore, we chose the easily available mean value to do predictive power of the regression models. This is a shame, as the model's predictive distribution is richer than a simple mean and variance. This would, however, involve tests using the approximate expected improvement. The predictive distribution of the Bayesian neural networks is also approximated with a Gaussian. This is however a bit more reasonable as the Bayesian neural networks in the limit are equal to a Gaussian process. Futhermore, experiments using the approximated expected improvement with the Bayesian Neural networks were conducted in [6], with no promising results.

In this thesis, we tried using mixture regression models as surrogate models. Especially, when breaking the GP and BNN's assumptions of a Gaussian observation noise they proved effective. This class of problems might occur in real-life simulations where the objective function can have multiple values, which are not just nearby values coursed by measurement noise. If one knew the "generative story" of the objective function values (e.g. that Test3 jumps between 2 functions), it might be possible to think about combining more Gaussain proceses in a probalistic graphical model, i.e. flipping a coin on what GP to use. But in the case where this generative story was not given mixture regression might indeed be a smart choice.

Discriminative models are built with this one purpose of regression, whereas generative models conduct different kinds of probabilistic queries, which are useful. Discriminative models are good for finding the paterns in data.

Variance of the prior was sat equal 10. I.e. when doing cross validation it was more keen on actually fitting a good model. crossvalidation on low amount of data is... Not a general smart thing? We always learn that we should use Bayesian methods on small amounts of data.

7.1 Models

BOHamiANN is in general performing bad.. But got a comback for higher dims..?!.. It is designed to have a fast inference, while second to have on pair performance with the GP but with only linearly scaling inference.

Sum product networks are the motivation why we call this deep.. However, the seem to put unnecessary constraints on the model.

GMR was overconfident.. Should have been Bayesian.

GPs we never discussed the kernel. But the Matern kernel is chosen as the default in ... BayesOpt book.

Looking a regression plots and there is a clear connection between being a good regression model and surrogate model.! However, we actually ran BO till more than 35 data points, i.e. all the figures shown above is from 36 points.

The Bayesian neural network training is difficult to know when it has converged, in general the more samples the better, and the more weights the more complicated model there for more samples. We chose 500 warm-up and 500 samples, using 4 chains, since that we have 4 cpus. Also to make sure the fitting we sat a very informative prior on a very small observation variance.

BOHamiANN is should have been more in focus. More samples? Or Better parameters.

Mixture regression for small amount of data does not seem smart. GMR is trained using MLE, has the most discriminative power, however, is also failing a lot. Essentially it is just a bit smarter version of random search, which indeed migth be better than an overconfident GP in certain problems. This is what we saw in ... Even for the MAP model SPN, the prior matters a lot.

Compartment between resluts should have used the mode instead for the mean.

emperical expecpected improvement should be used..! Where the mode is not equal the mean and where we us the predictive distribution.

Even though SPN is an interesting model, its relevance is only for large dimensions, where the number for data points or in certain cases where we want a more constraint model, i.e. that y can only come from 2 distributions thorughout the data. The terminology deep is not really relevant for the SPN as they are constrained by the number of dimensions..!

crossvalidation for selecting the hyper paramters is not really smart for only few data. It would make more sence to just place a hyperprior on the priors instead!.

8 Conclusion and further work

The following chapter concludes the thesis and my work over the last five months.

First of the all, the hypothesis conducted in the beginning was,

- Mixture regression models and BNNs can be effective surrogate models performing better than GPs in some complex BO problems.

In the results we did find two problems, where the GP is not a preferred model. This confirms that we successfully implemented both BNNs and Mixture regression models in the Bayesian optimization setting. However, the tests were not exhaustive enough to conclude whether or not the new methods, in general, are a prefered model in these cases.

This thesis has motivated the relevance for Bayesian optimization, developed different alternative surrogate models, i.e. a Bayesian NN and mixture regression models and tested those on some test functions. Among the 4 selected BBOB problems in higher dimensions, it was not possible to show that GP was not the preferred model. However, for the 4 small 1D test problems, we found problems, where the GP. is not preferable. To the story is that the developed generative surrogates all have been approximated by a Gaussian.

The SPN was understood in the context of regression, which to our knowledge is a novel approach. However, when diving into the definition of the SPN it became clear that it was nothing but a discretized regression model.

The kernel density estimation is an uncorrelated GMM with a component on each datapoint, this made it robust, but also very naive, without the prior this model would provide a local mean prediction with mean equal the nearest data point and variance is selected.

The Gaussian mixture regression was included to make the more intelligent mixture components, which would allow for better predictive power, however, without much data the model collapsed into very spiky distributions, and the prior was often sat high to help out the uncertainty quantification.

The BOHamiANN did a surprisingly good job in the Bayesian optimization (for the larger high-dim problems), even though it had a

We showed that there is a strong connection between being a good regression model and a good Bayesian optimization algorithm. Unfortunately, even though the mixture regression models showed good uncertainty quantification, they were approximated by gaussians to perform the fast closed form Expected improvement, therefore the connection is lost.

8.1 further work

As mentioned, the mixture models were forcefully approximated by a Gaussian in the BO setting, but also when testing preditive power with mean instead of mode. Therefore, an obvious next step in further work is to test the mixture models using approximate expected improvement, which uses samples from their true predictive distributions.

The investigation of this thesis was conducted to understand what model to choose. However, the standard answer of a Bayesian should be: ”Why choose?” If there is uncertainty about them, then we can just average them. Picing the maximum would be frquencialistic. Could we combine GP and SPN? Ensemble of models.

Avoiding cross-validation and instead include the hyperpriors and do it the Bayesian way.

Many problems have dimensions of no relevance. Here This might be relevant to investigate

More dimensions of prior weighting?

<Using mixtures of more complicated components, i.e. hundebens distribution>

<Parrallel Bayesian OPtimization>

Bibliography

[1] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[2] Bobak Shahriari et al. "Taking the human out of the loop: A review of Bayesian optimization". In: *Proceedings of the IEEE* 104.1 (2015), pp. 148–175.

[3] Bowen Lei et al. "Bayesian optimization with adaptive surrogate models for automated experimental design". In: *npj Computational Materials* 7.1 (2021), pp. 1–12.

[4] Jost Tobias Springenberg et al. "Bayesian Optimization with Robust Bayesian Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016. url: https://proceedings.neurips.cc/paper/2016/file/a96d3afec184766bfeca7a9f989fc7e7-Paper.pdf.

[5] Jasper Snoek et al. *Scalable Bayesian Optimization Using Deep Neural Networks*. 2015. arXiv: 1502.05700 [stat.ML].

[6] Mashall Aryan. "Sample-efficient Optimization Using Neural Networks". https://researcharchive.vuw.ac.nz/handle/10063/9035. PhD thesis. Victoria University of Wellington, New Zealand, 2020.

[7] David A. Cohn, Zoubin Ghahramani, and Michael I. Jordan. "Active Learning with Statistical Models". In: *CoRR* cs.AI/9603104 (1996). url: https://arxiv.org/abs/cs/9603104.

[8] Roman Garnett. *Bayesian Optimization*. in preparation. Cambridge University Press, 2022.

[9] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. doi: 10.48550/ARXIV.1412.6980. url: https://arxiv.org/abs/1412.6980.

[10] Alexander IJ Forrester and Andy J Keane. "Recent advances in surrogate-based optimization". In: *Progress in aerospace sciences* 45.1-3 (2009), pp. 50–79.

[11] Donald R Jones. "A taxonomy of global optimization methods based on response surfaces". In: *Journal of global optimization* 21.4 (2001), pp. 345–383.

[12] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[13] Sergios Theodoridis. *Machine learning: a Bayesian and optimization perspective*. Academic press, 2015.

[14] Matthew D. Hoffman and Andrew Gelman. "The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo". In: *Journal of Machine Learning Research* 15.47 (2014), pp. 1593–1623. url: http://jmlr.org/papers/v15/hoffman14a.html.

[15] Christopher M Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[16] Zoubin Ghahramani and Michael Jordan. "Supervised learning from incomplete data via an EM approach". In: *Advances in Neural Information Processing Systems*. Ed. by J. Cowan, G. Tesauro, and J. Alspector. Vol. 6. Morgan-Kaufmann, 1993. url: https://proceedings.neurips.cc/paper/1993/file/f2201f5191c4e92cc5af043eebfd0946-Paper.pdf.

[17] Alexander Fabisch. "gmr: Gaussian Mixture Regression". In: *Journal of Open Source Software* 6.62 (2021), p. 3054. doi: 10.21105/joss.03054. url: https://doi.org/10.21105/joss.03054.

[18] Robert Peharz et al. *Probabilistic Deep Learning using Random Sum-Product Networks*. 2018. doi: 10.48550/ARXIV.1806.01910. url: https://arxiv.org/abs/1806.01910.

[19] Nikolaus Hansen et al. "Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions". PhD thesis. INRIA, 2009.

[20] Nikolaus Hansen et al. “COCO: a platform for comparing continuous optimizers in a black-box setting”. In: *Optimization Methods and Software* 36.1 (Aug. 2020), pp. 114–144. doi: 10.1080/10556788.2020.1808977. url: https://doi.org/10.1080%2F10556788.2020.1808977.

[21] Jasper Snoek et al. *Scalable Bayesian Optimization Using Deep Neural Networks*. 2015. arXiv: 1502.05700 [stat.ML].

[22] Jost Tobias Springenberg et al. “Bayesian Optimization with Robust Bayesian Neural Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016. url: https://proceedings.neurips.cc/paper/2016/file/a96d3afec184766bfeca7a9f989fc7e7-Paper.pdf.

A Gaussian mixture rules

B Additional results

B.1 All COCO results

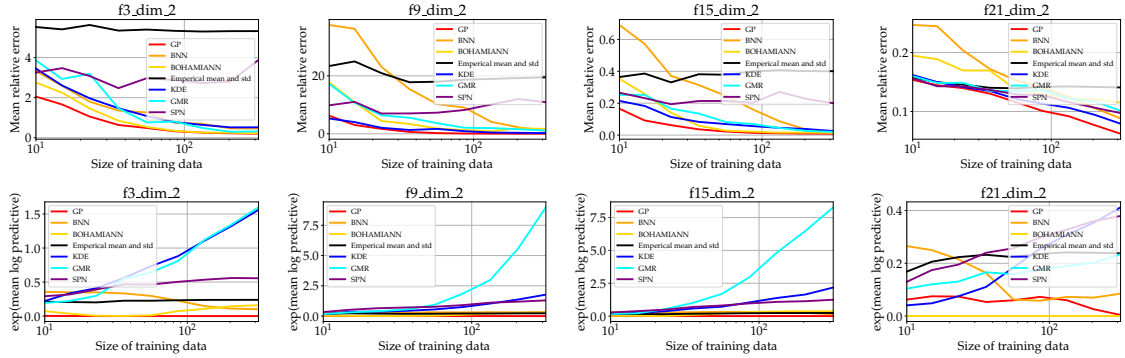


Figure B.1: Plots visulising the results in figure at $n_{data} =?$

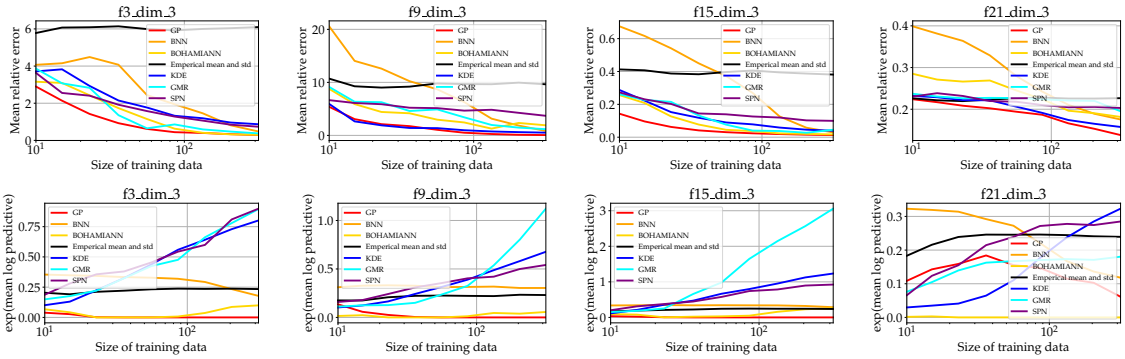


Figure B.2: Plots visulising the results in figure at $n_{data} =?$

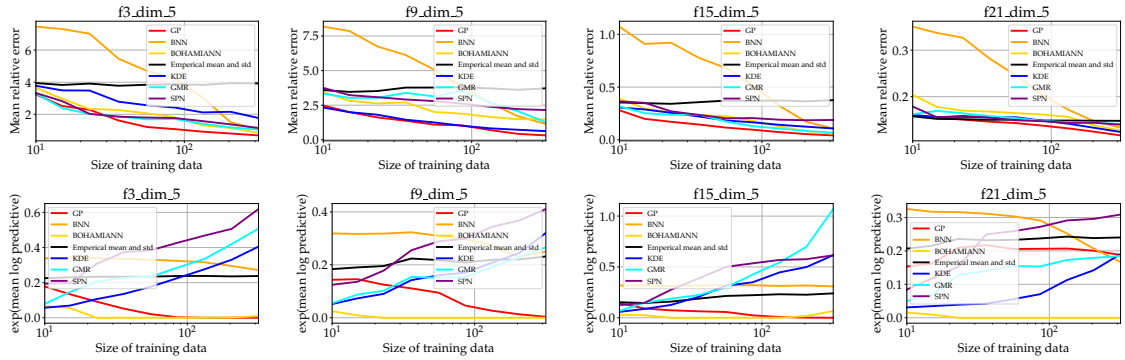


Figure B.3: Plots visulising the results in figure at $n_{data} =?$

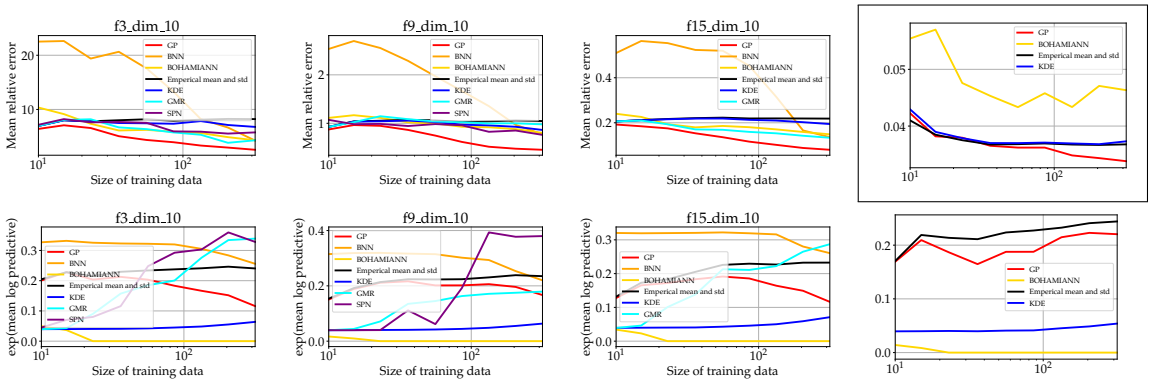


Figure B.4: Plots visulising the results at dim = 10

Bayesian optimization is an effective framework for finding optimizers of a highly expensive (in terms of money, time, human attention or computational processing) or noisy objective function. First a prior is defined over possible functions and updated to a posterior according to already obtained observations/samples of the objective function. Next an acquisition function uses this posterior, also called an surrogate model, and is then utilized to find the next location in the optimization landscape to sample from. The far most common surrogate model is Gaussian Process (GP), partially due to its ability to represent its posterior in closed form. However, it also comes with short comings: Its inference, although it is exact, scales cubic with amount of samples and it impose strong assumptions of a well behaved objective function.

This thesis aims to investigate surrogate models different from GPs in order to improve on either the accuracy of the surrogate model or the inference cost of it. Meanwhile Bayesian Neural Networks (BNN) already have proven useful as surrogate models [6][21][22] (with cost of inference, which scales linearly and less strong assumptions) this thesis additionally wants to investigate Sum Product networks (SPN). An SPN is - similarly to a BNN - a deep probabilistic model and still expressive but with tractable inference, which potentially could lead to advantages over BNNs.

Technical
University of
Denmark

Building 321
2800 Kgs. Lyngby
Tlf. 4525 1700

www.github.com/SimonKruse0/master-thesis