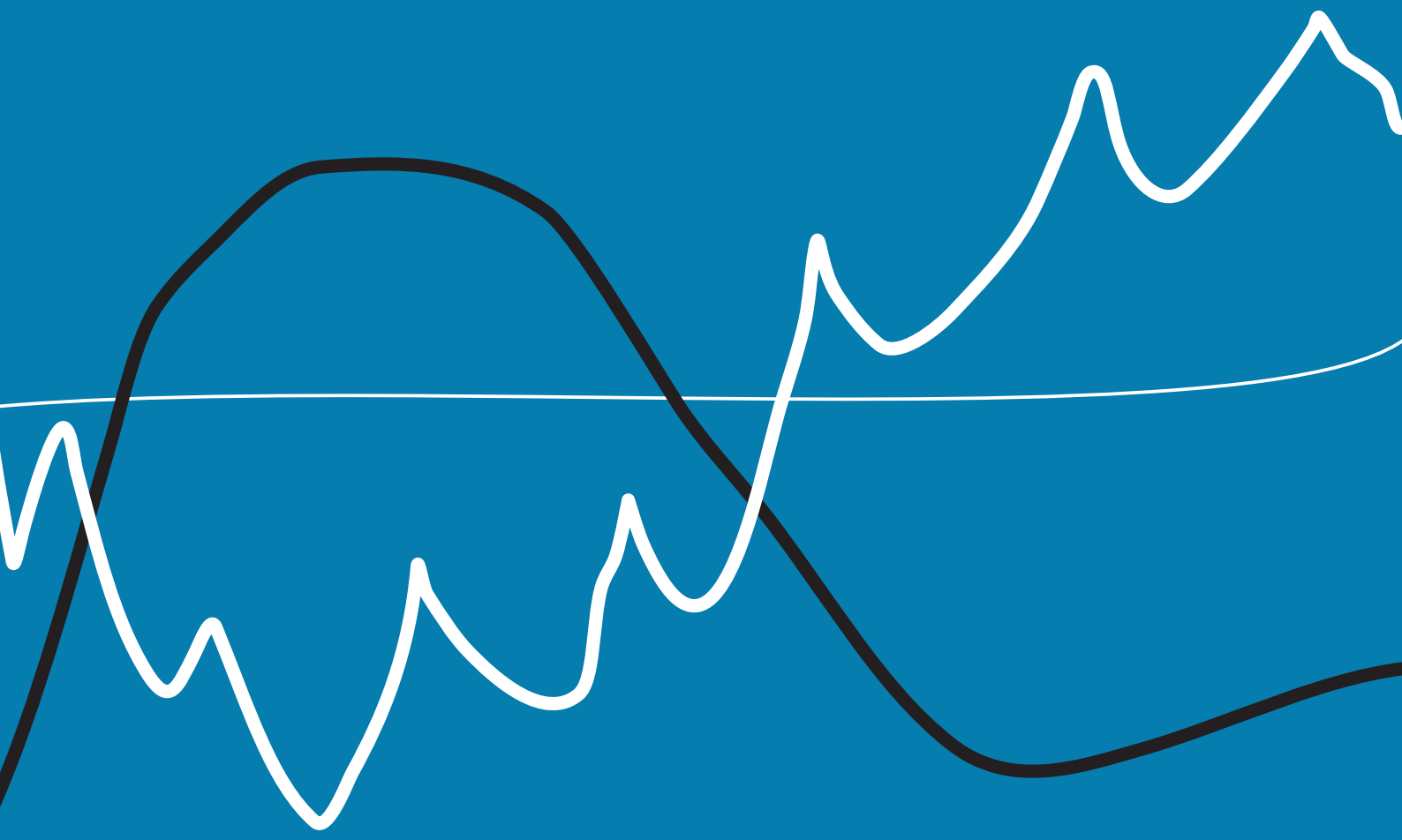
DTU Compute
Section of Cognitive Systems

Investigation of Deep Probibalistic Surrogate-models in Bayesian Optimization

Master Thesis



Abstract

Bayesian optimization is the leading way in sample-efficient optimization and widely used in hyperparameter tuning in machine learning and deep learning.

However, from a decision theory 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 a GP.

This thesis investigates surrogate models such as Mixture regression, SPNs and BNN and compare against GP.

Contents

1 Introduction

Optimization plays an important part 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 formluate the optimization problem as a *mathematical optimization problem*:

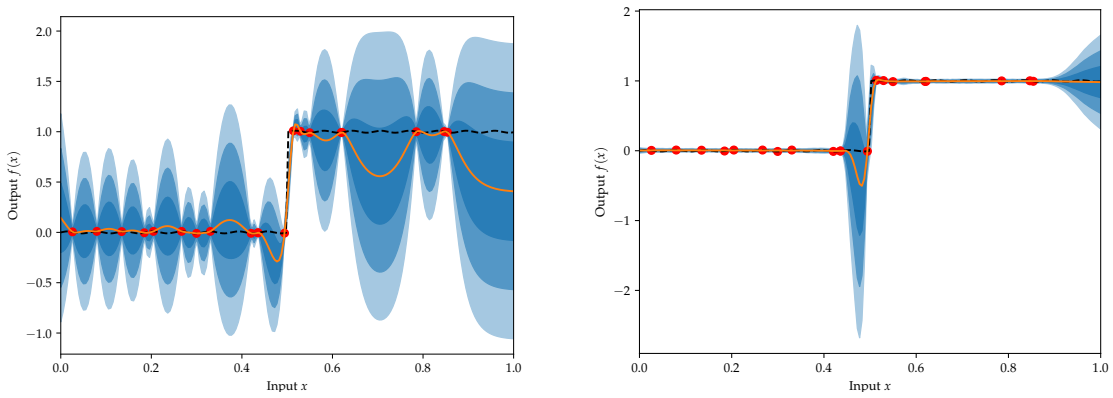
$$\min_{x \in \mathcal{X}} f(x)$$

where the functional $f : \mathcal{X} \to \mathbb{R}$ is called the objective function and $\mathcal{X}$ is the set of possible decisions (or decisions you consider). Note that the optimization problem is formulated as a minimization problem. If one identifies the optimal decision as a maximum of f (instead of the minima), finding the minima in the negated objective function $-f(x)$ is equivalent. 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 $f(\cdot)$.

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*, is one of the preferred frameworks for sample-efficient optimization.

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 (in a Bayesian manner). 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, 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 $\mathcal{X}$. However, in Figure 1.1 we see an example of how the GP's uncertainty quantification is going wild due to a discontinuity in the underlying objective function. in some areas, it is prevalent to have these discontinuities, for instance, in material discovery, where it is well known that materials often change very suddenly [1]. 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 [1].



(a) GP (b) BNN

Figure 1.1: Left: GP fitted to 20 data points. Right: A Bayesian neural network fitted to the same points. The objective function is the dashed black line. This examplifies how a discontinuerity makes the standard implemented GP (optimized with emperical bayes) overreact to all other areas in the domain $[0, 1]$, while the Bayesian NN only express uncertainty where the discontinuerity happen $x = 0.5$

GPs are attractive models since they allow for exact inference, a closed-form expected improvement (more on this later) and, in general, give a good uncertainty quantification. However, as we saw in the figure **??** we can maybe do better - especially if we have some insights into the behavior of the objective function (i.e. a more transparent the black-box optimization problem). Suppose it is possible to evaluate the expensive objective function a few times. Then it can potentially save work hours, lots of money, or energy (depending on what makes the objective function expensive). It is, therefore, relevant to investigate if other models will perform better. As mentioned, the assumptions of GP can be too strong. In this thesis, we aim to create models with less strong assumptions, which can perform better on certain classes of (complex) problems, and just as well as the GP on most classes of problems.

We limit the scope of the project to deal with the following surrogate models,

- GP with Matérn kernel (Scikit-learn implementation)
- Bayesian Neural network (Numpyro implementation)
- Bayesian Neural network (BOHAMIANN)
- Mixture regression (Gaussian mixture, 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 $\mathcal{X} \in \mathbb{R}^m$.

1.1 Contribution

This thesis investigates surrogate models alternative to the isotropic GP - more concretely Bayesian NN and mixture regression. The proposed hypotheses are,

1. Neural Networks perform better applied to complex BO problems than GPs.
2. Mixture regression models like SPN can be an effective surrogate model performing better than GPs and Neural Networks in some complex cases.

Note what is meant by performance is *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 outweigh the power and time spend on the surrogate modeling and optimization.

1.2 Related work

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,

- 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 have 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 [2] or as basisfunctions in linear regression [3]) 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 [4], 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 nature paper ”Bayesian optimization with adaptive surrogate models for automated experimental design” [1], 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” [5] investigates using Gaussian mixtures and kernel estimator in active learning, i.e. as a surrogate model for selecting the next samples. These are regression models not seen much in the literature, which is 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 the thesis

The structure of the thesis is first to build up the foundation around Bayesian optimization, then present different surrogate/Bayesian regression models, and then results on different types of problems are shown first on Bayesian regression and next on Bayesian optimization. Finally, a discussion and conclusion.

1. **Introduction to Bayesian optimization**, first by establishing its relevance as a sample-efficient solver, next by understanding its components: An acquisition function and a surrogate/Bayesian regression model.
2. **Discriminative models as surrogates**: Gaussian processes and Bayesian neural networks are presented and discussed in terms of inference procedures and regression performance.
3. **Generative(/mixture) models as surrogates**: Since the mixture models are not inherently Bayesian, we present the conditional distribution of 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.
4. **Results** on regression performance and on Bayesian Optimization performance (sample-efficiency)
5. **Discussion** and **conclusion**

1.4 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 $p(y|x) := P(Y = y|X = x)$ or $p(y|x) := P(Y|X = x)$. And writing $p(y^2|x)$ means $P(Y^2 = y^2|X = x)$ and **not** $P(Y = y^2|X = x)$

The density of a normal distribution evaluated in x is denoted as, $\mathcal{N}(x|\mu, \Sigma)$.

We will not distinguish between vectors and scalars notation-wise unless it is not clear from the context.

2 Bayesian Optimization

This chapter will introduce Bayesian optimization. We start with a general introduction to the concept of optimization.

2.1 Optimization methodology

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

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

Note that the above formulation is a minimization problem, which is equivalent to a maximization problem maximizing $-f(\cdot)$. Throughout this thesis, we choose to only work with a minimization problem. Solving this problem exactly 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.

Example: Direct solution method
The unconstrained linear least squares,

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

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,

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

Most optimization problems are non-convex with multiple local minima. And even if the gradient is given analytically, the solution is still found among a potentially infinitely large set of stationary points ($\nabla f(x) = 0$) and boundary points ($x \in \partial\mathcal{X}$) - 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 the book Bayesian Optimization by Roman Garnett [6] and presented here as Algorithm 1.

Algorithm 1 Sequencial Optimization [6]

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: $\mathcal{D}$

Given data points in the optimization landscape[1] a policy selects a point $x \in \mathcal{X}$ where we make our next observation. Policies can be deterministic or probabilistic, e.g. grid search and random search are examples of policies. The next observation provides us a y value, which combined with x is included in the available data $\mathcal{D}$. Finally, a stopping criterion decides whether to continue or terminate. We will now present how different examples of well-known optimization rutines fits into Algorithm (1).

Example: 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 points are ordered and systematically selected. In the content of algorhtim 1 we define the grid search policy as

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

assuming $x_1, x_2, \ldots, x_n \in \mathcal{X}$ are the ordered grid points and the size of the obtained data is $|\mathcal{D}|$.

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

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

Example: Gradient descent
Gradient descent is the most simple gradient-based optimization approach. The gradient of a continuous function points in the most ascending direction from the location where it is evaluated. In the minimization task, (2.1), we can iteratively use the opposite gradient direction, i.e. the most descending direction. 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 model is given as:

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

2.1.1 When to use sample-efficient optimization

Note that grid search, random search, and gradient descent are policies that entirely ignore the available data. Not using all 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 previous observations, $\mathcal{D}$. Choosing a more sample-efficient solver is a question of how much extra time/energy per iteration it will cost to store and exploit the collected information. In the end solving the optimization problem can be divided into the following components,

- N_{iter}: 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.
- C_{policy}: The solvers cost per iteration, i.e. what is the cost of calculating policy($\mathcal{D}$) - cost typically measured in time and power.
- C_{observe}: Cost per evaluation of the objective function, i.e. the cost of observe(x)

[1] "Optimization landscape" defined as the joint set of points in the domain and the objective function evaluated in the points, i.e. $\{(x, f(x)) \in \mathcal{X} \times \mathbb{R} | x \in \mathcal{X}\}$

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

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

where the a sample efficient solver has a large C_{policy} but a small N_{iter}, and a simple optimization scheme like grid search has very small C_{policy} but also large N_{iter}. So choosing the right optimization solver depends highly on C_{observe}. In machine learning the framework ADAM is very popular, due to its cheap but advanced policy, yielding a good trade-off between N_{iter} and C_{policy}.

This project deals with a dominating observation cost, i.e. the cost of the policy is assumed neglectable,

$$C_{\text{observe}} >> C_{\text{policy}}.$$

Example: 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 neural network or a random forest. 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.

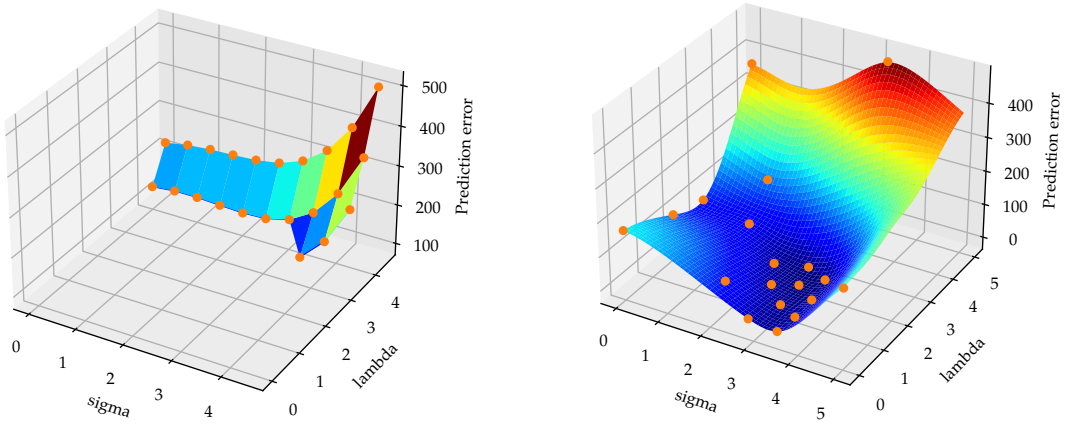


Figure 2.1: Example of an optimization task tuning a parameterised regression model with parameters λ and σ, on a test set, i.e. minimization of prediction error. We see the first 23 evaluations out of 100 in grid search vs 23 evaluations using a sample-efficient solver (Bayesian optimization). This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

Noisy objective functions

Many optimization algorithms assume *exact* evaluations of the objective function. However, this assumption is often wrong, especially for objective functions with real-life experiments, imperfect simulations, human interaction where measurement noise is well known. A potentially noisy objective function is the main reason why we in Algorithm (1) use the termology 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,

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

> **Note: "Sample" or "evaluate" objective function**
>
> The terminology *to sample* is also refering to a probabilistic observation model - we can, however, extend this model to deal with noiseless observations as well, simply by setting $\sigma = 0$ and let 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.

Example: 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 oppose 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 (more about this later). The policy is as following,

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

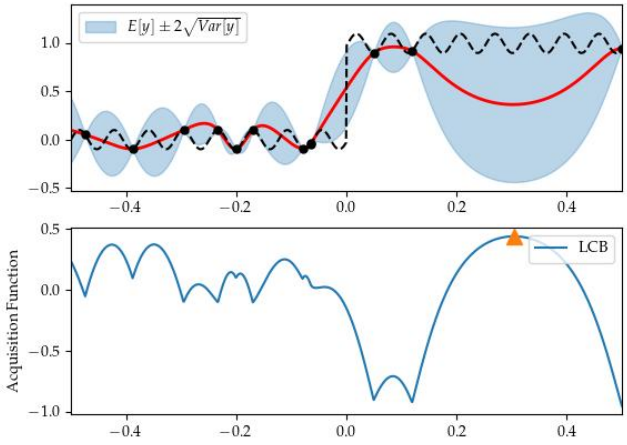


Figure 2.2: Top: Bayesian regression model (Gaussian Process) is fitted to the observed data. Bottom: The negated lower confidense bound acquisition function, $LCB(p(y|x, \mathcal{D}))$, the maxima, i.e. the orange point, is the location of the next sample

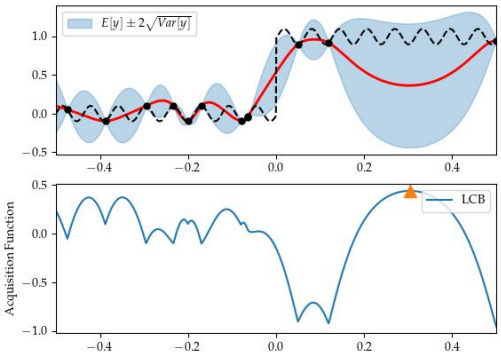


Figure 2.3: Top: Bayesian regression model (Gaussian Process) is fitted to the observed data. Bottom: The negated lower confidense bound acquisition function, $LCB(p(y|x, \mathcal{D}))$, the maxima, i.e. the orange point, is the location of the next sample

Algorithm 2 Bayesian Optimization

Input: Initial dataset $\mathcal{D}$, Bayesian regression model.
while Temination is not reached **do**
Fit Bayesian regression model:

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

$x \leftarrow \max_x AQ(p(y|x, \mathcal{D}))$ ▷ Find point with highest acquisition
$y \leftarrow \text{observe}(x)$ ▷ observe objective function at chosen location
$\mathcal{D} \leftarrow \mathcal{D} \cup \{(x, y)\}$ ▷ update dataset
return: $\mathcal{D}$

2.2 Bayesian regression (surrogate)

Whereas traditional regression workflow is the following: From data, fit model parameters, make predictions using the 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, Θ, where some values/realizations of Θ are more probable than others. What is of interest is the predictive posterior distribution,

$$p(y|x, \mathcal{D}) = \int p(y, \theta|x, \mathcal{D})d\theta \tag{2.2}$$

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

where 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 is fully described by the posterior distribution $p(\theta|\mathcal{D})$.

2.2.1 surrogate model

A surrogate model in a Bayesian optimization setting is just a Bayesian regression model. The most used surrogate model is a Gaussian Process. But there have been investigations on other types of surrogates, such as Bayesian neural networks and Bayesian regression trees. These are all discriminative models, and another approach, which 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 (discriminative)
- Bayesian Neural network (discriminative)
- Kernel density regression (generative)
- Gaussian mixture regression (generative)
- SPN (generative)

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. It is possible to make exact inference of SPN, Gaussian Process, and Gaussian Mixture Regression.

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, Laplace approximation, 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 versions of the MCMC method, Hamiltonian Monte Carlo. As revealed later (see result section), approximate inference might indeed 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 datapoints. $K \leq n$ is the number of mixture comonents. We will soon learn that for an SPN the number of mixture compenets is exponential larger than number of edges i.e. $E << K$. In theory MCMC methods samples from true the posterior distribution, and do not need any fitting/learning.

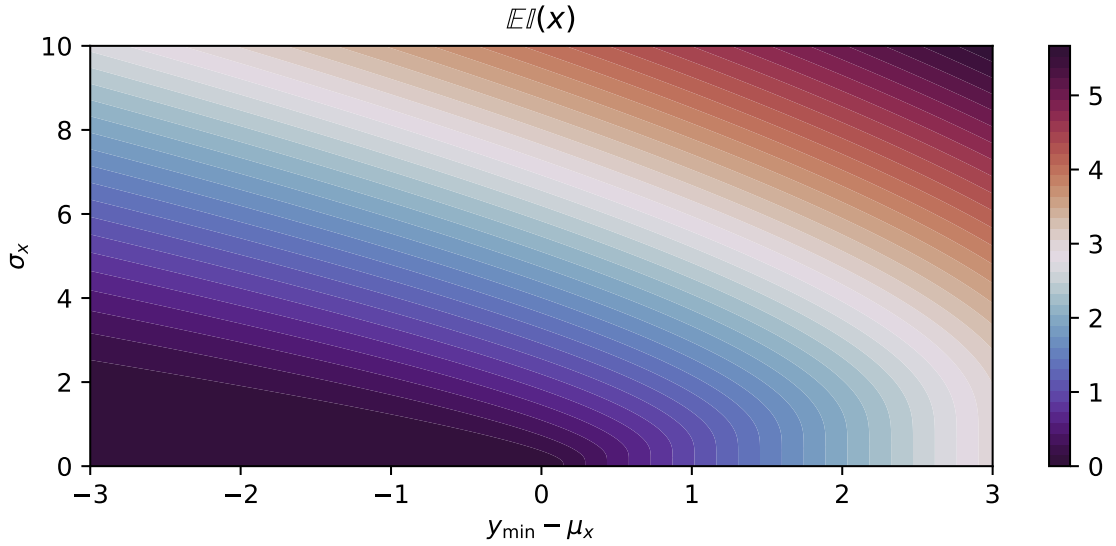


Figure 2.4: When then predictive distribution is Gaussian, expected improvement becomes closed form. The figure illustrates the values of expected improvement for different predictive uncertainties $\sigma_x = \sqrt{\mathbb{V}ar_{p(y|x,\mathcal{D})}[y]}$ versus the average improvement $y_{\min} - \mu_x$, where $\mu_x = E_{p(y|x,\mathcal{D})}[y]$

2.3 Acquisition function

Given a well fitted predictive distribution $p(y|x\mathcal{D})$ (2.2). The next sample point is chosen according to a so-called acquisition function, which balances out the well-known concept of exploitation and exploration. Where exploitation will be choosing the next location, $x \in \mathcal{X}$ according to its average improvement, and exploration will be choosing the next point in a region of high uncertainty and thereby help lower the overall uncertainty. We will first look at the acquisition function used in the thesis: Expected improvement. But also shortly go over different types of acquisition functions.

2.3.1 Expected improvement

A popular choice of acquisition function is expected improvement,

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

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 x which yield a bad predictive mean value $\mathbb{E}_{p(y|x,\mathcal{D})}[y] > y_{\min}$ might still be maximizing the expectated improvement, if its uncertainty is very large. Figure 2.4 illustrates that a large uncertainty in the predictive distribution could 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,
>
> $$\mathbb{E}_{p(y|x,\mathcal{D})}[y_{\min} - y] = y_{\min} - \mathbb{E}_{p(y|x,\mathcal{D})}[y]$$
>
> 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 deveration 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 we 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,

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

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 $I(x) := (y_{\text{min}} - \mu_x)$) and the second part can be seen a 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 growing average improvement $I(x)$ and also for growing 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 approximation) or calculate the expected improvement approximately as follows,

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

where $y^{(k)}$ are samples from the predictive distribution. In the case of a parametric model like a Bayesian NN, then we already got posterior samples from posterior $\theta^{(k)} \sim p(\theta|\mathcal{D})$, giving,

$$p(y|x, \mathcal{D}) \approx \frac{1}{K_2} \sum_{k=1}^{K_2} p(y|x, \theta^{(k)})$$

where $p(y|x, \mathcal{D} = \mathcal{N}(y|NN_w, \sigma))$. So essentially we should sample from a sampled distribution, but instead the PhD thesis [4], just sample the mean and set $K_2 = 1$...

2.3.2 Lower confidense bound

Lower confidence bound is defined by a confidence paramter $\pi \in [0, 1]$ and then the lower quantile $\int_{\infty}^{x} p(y|x, \mathcal{D}) dy = (1 - \pi)$ is chosen as the acquisition. For a Gaussian predictive distribution this is simply

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

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

2.3.3 Entropy search

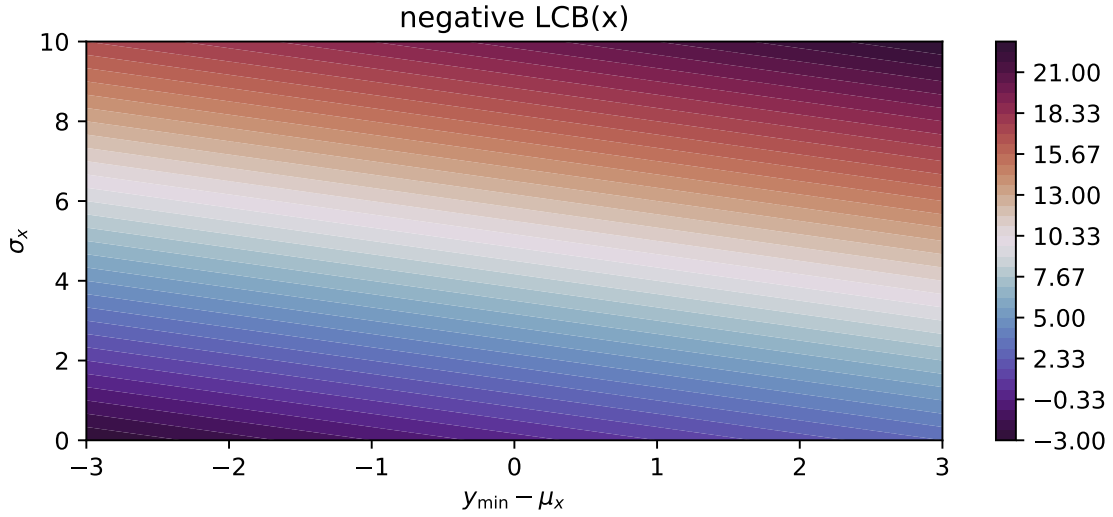


Figure 2.5: Illustration of the values of the negative lower confidence bound for different predictive uncertainties $\sigma_x = \sqrt{\mathbb{V}ar_{p(y|x,\mathcal{D})}[y]}$ versus the average improvement $y_{\min} - \mu_x$, where $\mu_x = E_{p(y|x,\mathcal{D})}[y]$

3 Discriminative surrogate models

When talking about a surrogate model we usually refer to a discriminative model, meaning that we model the conditional distribution of y given x directly, typically paramatised by some parameters, θ.

$$p(y|x, \theta)$$

When talking about a probabilistic surrogate model we are always implicit talking about a discriminative model: A model of the conditional distribution of the observation, y, conditional on x, i.e.,

$$p(y|x)$$

we also refer to this as the predictive distribution. Gaussian processes and Bayesian neural networks are both discriminative models. There is no distribution over the input x, it is just given. This is indeed sufficient for a surrogate model, where we are interested in the predictive distribution.

<Better illustration of GP> <Illustration of BNN>

3.1 Gaussian process surrogate

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

$$y = f_w(x) + \epsilon$$

where weights are trained. f could describe a linear model, $f(x) = w^T x$ or a polynomial $f(x) = w \cdot x^2$ etc. the performance of the regression model is very dependend on the regression model. The Gaussian process takes a completely different approach, its model f does not include any parameters. If we collect the data in a vector $\mathbf{f} := [f(x_1), \ldots, f(x_n)]$ then the GP assign it a multivariate normal distribution,

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

where $\boldsymbol{\mu}$ typically is 0 and the covariance matrix, is depeneded on the input, $x_1, \ldots, x_n$,

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

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 $\mathbf{f}$ is 1, then $\boldsymbol{\Sigma}$ is a correlation error) then realizations of $\mathbf{f}$ always lead to similar values of $f(x_1)$ and $f(x_2)$. 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 bring a unobserved y_* for a arbitrary location x_* into the game. Prior to observing any data, we assume that the corresponding $\mathbf{f}_*$ is just a new element in the multivariate normal distribution,

$$p(f(\cdot), \mathbf{f}|\mathbf{x}) = \mathcal{N}\left(\begin{bmatrix} f(\cdot) \\ \mathbf{f} \end{bmatrix} \middle| \begin{bmatrix} 0 \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} c(\cdot, \cdot) & c(\cdot, \mathbf{x}) \\ c(\mathbf{x}, \cdot) & c(\mathbf{x}, \mathbf{x}) \end{bmatrix} \right) \tag{3.1}$$

yielding many possible outcomes for $\mathbf{f}_*$, but with an average $E[\mathbf{f}_*] = 0$ and variance of $V[\mathbf{f}_*] = 1$. but note, we treate $\mathbf{f}$ as random quantaty, but if get a realization of $\mathbf{f}$, then the distribution of $\mathbf{f}_*$ is changed,

$$p(f(\cdot)|\mathbf{x}, \mathbf{f}) = \mathcal{N}(f(\cdot)|c(\cdot,\cdot)^{-1}c(\cdot, \mathbf{x})\mathbf{f}, c(\cdot,\cdot)^{-1})$$

What we want is the predictive posterior distribution, of a new point $p(y_*|x_*, \mathcal{D})$, i.e. by marginalizing out the random variabel $f(x_*)$,

$$p(y_*|x_*, \mathcal{D}) = \int \mathcal{N}(y_*|f(x_*), \sigma^2)p(f(x_*)|\mathcal{D})df(x_*) \tag{3.2}$$

where we assume that ϵ from **??** is a Gaussian with zero mean and variance σ^2. We will soon see that the posterior $p(f(x_*)|\mathcal{D})$ is also a normal distribution so using <trick> we end up with the distribution.

$$p(y_*|x_*, \mathcal{D}) = \mathcal{N}(y_*|E[f(x_*)], Var(f(x_*)) + \sigma^2)$$

So now want to calculate $p(f(x_*)|\mathcal{D})$,

$$p(f(x_*)|\mathcal{D}) = \int p(f(x_*)|\mathbf{x}, \mathbf{f})p(\mathbf{f}|\mathcal{D})d\mathbf{f}. \tag{3.3}$$

<image>

Assuming corr

Using a GP as a surrogate model is very popular in Bayesian Optimization, one of the reasons are its exact inference, and often its assumption is a very good characteristic of the true function.

3.1.1 Exact predictive distribution

We now show how the preditive distribution is calculated exact for Gaussian Processes, i.e.

$$p(y|x, \mathcal{D}) = \int \mathcal{N}(y|f(x), \sigma^2)p(f(x)|\mathcal{D})df(x) \tag{3.4}$$

we will soon see that $p(f(x)|\mathcal{D}) = \mathcal{N}(f(x)|.., ...)$ and and thereby that we have a marginal Gaussian distribution for $f(x)$ and a conditional Gaussian distribution of y given $f(x)$, giving us the marginalized distribtuion, $p(y|x, \mathcal{D})$, using formulars (3.5).

Background: Trick with normal distributions [from Bishops book?]

Given a marginal Gaussian distribution of x and a conditional Gaussian distribution of y given x of the form,

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

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

$$p(y) = \mathcal{N}(y|A\mu + b, L^{-1} + A\Lambda^{-1}A^T) \tag{3.5}$$
$$p(x|y) = \mathcal{N}(x|\Gamma\mu + \Gamma[A^T L(y - b)], \Gamma) \tag{3.6}$$
$$\Gamma := (\Lambda + A^T LA)^{-1} \tag{3.7}$$

Posterior function
Recall we assume $\mathbf{f} = (f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n))$ is the parameters in our model, therefore we call $p(\mathbf{f}|\mathcal{D})$ the posterior distribution. However, what is of real interest is the function values on unobserved locations, thereby we extend $\mathbf{f}$ to be a function, i.e. an infinitely dimentional vector. We call this quantaty *the posterior function*

$$p(f(\cdot)|\mathcal{D}) = \int p(f(\cdot)|\mathbf{x}, \mathbf{f}) p(\mathbf{f}|\mathcal{D}) d\mathbf{f}. \tag{3.8}$$

Prior we assume that the function takes values accoring to

$$p(\mathbf{f}|\mathbf{x}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, c(\mathbf{x}, \mathbf{x}))$$

where the covariance is defined at kernel evaluation for each pair of $\mathbf{x}$, where $c(\cdot, \cdot)$ is a covariance function, chosen to be the Matérn covariance function,

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

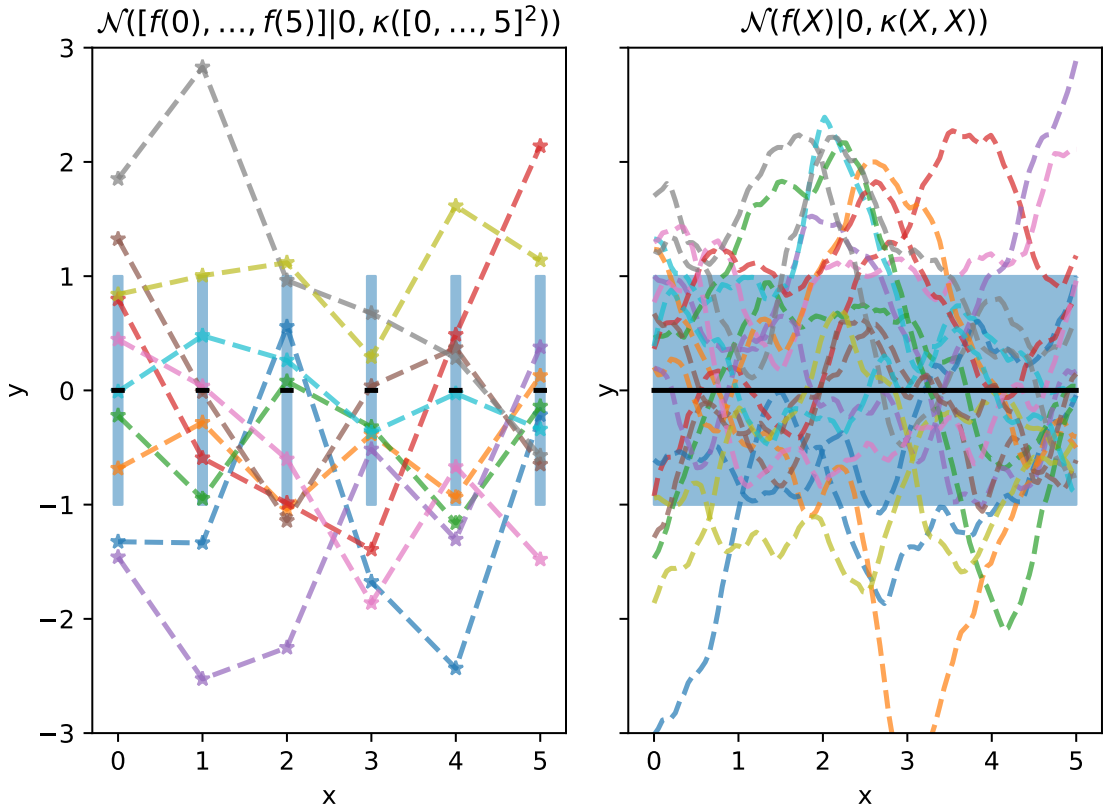


Figure 3.1: Realisations> Illustartaion of GP is just samples from a multivariate normal distribution. In the limit the we have a infinite-variate normal distribution, which we call a GP.

Appendix <...> tries to give the intuition why this makes sense.

We now calculate the first term in the integral (3.8), $p(f(\cdot)|\mathbf{x}, \mathbf{f})$ using that we have the joint prior

distribution,

$$p(f(\cdot), \mathbf{f}|\mathbf{x}) = \mathcal{N}\left(\begin{bmatrix} f(\cdot) \\ \mathbf{f} \end{bmatrix} \middle| \begin{bmatrix} 0 \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} c(\cdot,\cdot) & c(\cdot,\mathbf{x}) \\ c(\mathbf{x},\cdot) & c(\mathbf{x},\mathbf{x}) \end{bmatrix}\right) \tag{3.9}$$

And the conditonal of a joint Gaussian is given using <ref>

$$p(f(\cdot)|\mathbf{x}, \mathbf{f}) = \mathcal{N}(f(\cdot)|c(\cdot,\cdot)^{-1}c(\cdot,\mathbf{x})\mathbf{f}, c(\cdot,\cdot)^{-1})$$

Next we calculate the last term in the integral (3.8), $p(\mathbf{f}|\mathcal{D})$, i.e. the posterior distribution. Assuming iid data, i.e. $p(y_1,\ldots,y_n|x_1,\ldots,x_n,\mathbf{f}) = \prod_{i=1}^{n} p(y_i|x_i,\mathbf{f}_i)$ and that the likelihood is Gaussian with mean $\mathbf{f}$ and variance $\sigma^2 I_n$.

$$\begin{aligned} p(\mathbf{f}|\mathcal{D}) &\propto p(\mathbf{f}|x)\prod_{i=1}^{n} p(y_i|x_i,\mathbf{f}_i) \\ &= \mathcal{N}(\mathbf{f}|\mathbf{0}, c(\mathbf{x},\mathbf{x}))\prod_{i=1}^{n}\mathcal{N}(y|\mathbf{f}_i,\sigma^2) \\ &= \mathcal{N}(\mathbf{f}|\mathbf{0}, c(\mathbf{x},\mathbf{x}))\mathcal{N}(\mathbf{y}|\mathbf{f},\sigma^2 I_n) \end{aligned}$$

now from <ref> we have that the posterior is the following Gaussian:

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

Now we found that both term in the integral (3.8), and they are related such that it is possible to use (3.5) for arriving at (we define $A := c(\cdot,\cdot)^{-1}c(\cdot,\mathbf{x})$),

$$p(f(\cdot)|\mathcal{D}) = \mathcal{N}(f(\cdot)|AM^{-1}\sigma^{-2}\mathbf{y}, c(\cdot,\cdot)^{-1} + AM^{-1}A^T)$$

Finally we found that both terms in the integral (3.4) also is related in a simlar way, and we use (3.5), again to arrive at the predictive distribtuion,

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

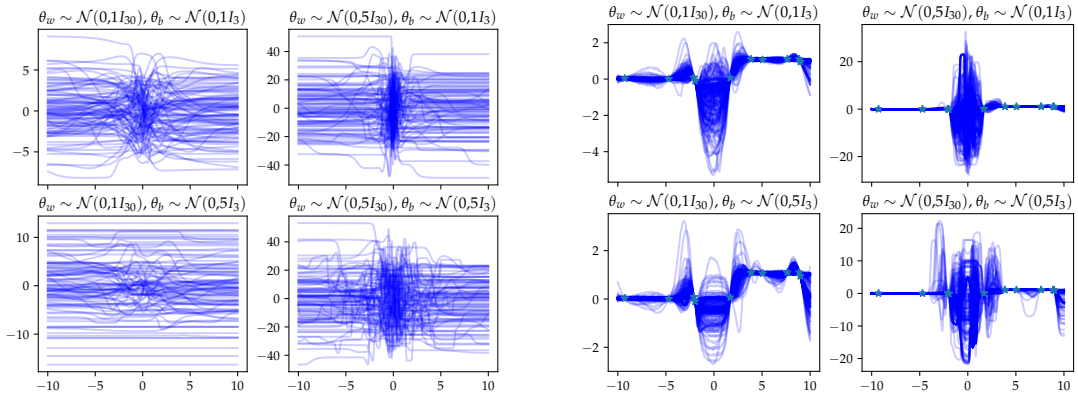
Some questions about a naive approach..!

Learning - Emperical bayes inference
Anther inference which is done is then optimizing the hyper parameters using emperical bayes i.e. the variance and length scale for the kernel. Here we optimize the marginalized likelihood function

$$p(y_1,\ldots,y_n|x_1,\ldots,x_n,\theta) = -\frac{1}{2}[(y-\mu)^T(\Sigma+N)^{-1}(y-\mu) + \log|\Sigma+N| + n\log 2\pi]$$

<and how to get to there?>

Model assessment becomes trivial in light of the model posterior if we simply establish preferences over models according to their posterior probability. When using the uniform model prior (4.6) the model posterior is proportional to the marginal likelihood alone, which can be then used directly for model assessment. ??! Youtube - good video! Go through that!



(a) prior (b) posterior

Figure 3.2: BNN with $\tan^{-1}(\cdot)$ activation functions and different prior distributions.

3.2 Bayesian Neural Networks

Neural networks have become increasingly popular in the recent years, due to their ability to approximate any function arbitrary well. And with the amount of data in the world today, neural networks are very powerful regression models for finding the complex patterns. Bayesian neural network, essentially neural networks, but instead of point estimates, each weight is assigned a distribution (usually a standard normal distribution), which provides a regression model, prior to any observed data. Choosing the prior of a standard normal distribution of a 3 layers with 10 nodes on each layer the we get the following regression model,

when observing data the distribution of the weights is adjusted accordingly - note that there can be correlations between the weights (corresponding to dimension in the distribution). The likelihood of a Bayesian Neural network is typically defined as a normal distribution with mean equal the neural network output (which itself also is a random variabel) and a variance random variable σ which prior is assumed independent from the input and assigned a half-caucy or log-normal distribution,

$$p(y|x,\theta) = \mathcal{N}(y|NN_w(x), \sigma^2)$$

Note that the posterior distribution of the weights and σ might not be a normal distribution or anything of the known analytically distributions, it might be highly complex and correlated. Therefor BNNs are examples of probabilistic models with intractable inference. The predictive density is given as,

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

where the integral is intractable as θ can live in a highly dimensional space. The approximation sign is true, since we can aproximate the integral with monte carlo sampling: $\theta^{(k)}$ are iid samples from the posterior distribution, $\theta^{(k)} \sim p(\theta|\mathcal{D})$. We can get samples from the posterior distriution using MCMC

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 accoring to the law of large numbers in fact

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

and the central limit theorem, <OBS refere!>

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

which ensures that the variance of the unbiased estimator of the expecation decreases with number of samples, K. Left is to sample the iid samples from the distribution $p(x)$

Posterior samples

For both models 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 appriximations 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 $p(\mathcal{D}, \theta)$ joint distribution thereby allow for using Markov chain Monte Carlo for sampling from the posterior distribution.

Background: Markov chain Monte Carlo

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

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

where $\int \hat{p}(x) dx$ is a possible intractable integral. An ergodic Markov chain/process is constructed, such that its stationary distribution is exactly $p(x)$, but only with the knowledge of $\hat{p}(x)$.

Example: Metropolis-Hasting (MH)

The most simple MCMC method is the Metropolis-Hasting algorithm. At iteration n we have a sample x_n,

1. Propose $\hat{x}$ from a proposal density $q(x_n, \cdot)$

2. Compute accptance probability

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

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 $\alpha(x_n, \hat{x})$ requires $p(x)$, but since the algorithm only requires the fraction $\frac{p(\hat{x})}{p(x_n)} = \frac{p(\hat{x})\cdot c}{p(x_n)\cdot c} = \frac{\hat{p}(\hat{x})}{\hat{p}(x_n)}$ we only need $\hat{p}$.

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

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

Now, let us look at the so-called *detailed balance* relation, i.e. that if we are sampling from the stationary density we stay there at the next state. Assume $x \neq y$,

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

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,

$$\sum_x p(x)p(x \to y) = p(y) \sum_x p(y \to x) \tag{3.10}$$

$$\implies \quad p(y) = \sum_x p(x)p(x \to y) \tag{3.11}$$

similar conclusion will be 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 random walk transition is very simple and it comes with some serious disadvantages: slow convergence speed, might stay in the same region for a long time and produces highly correlated sampels. We can do better by replacing the random walk with gradient-guided movements and intepretate the probability landscape as a physical system.

Example: HMC
The golden standard in MCMC is the Hamilton monte carlo, which exploits arguments from classical mechanics around the Hamiltonian equations. This method leads to more efficient sampling as the Hamiltonian intepretation allows the system to take regions with high probability

mass into acound - this is optained using gradient of the probability landscape. $\frac{-\partial E(x)}{\partial x}$

We define PDF

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

where $E(x)$ is interpretted as the systems potential energy. Now, an latent vector q is introduced in order to represent the momentum of the system, which gives us the kinetic energy of the system.

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

Giving the Hamilton function and its coresponing distribution

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

and

$$p(x, q) = \frac{1}{Z_H} \exp(-H(x, q)) \tag{3.12}$$

$$= \frac{1}{Z_E} \exp(-E(x)) \frac{1}{Z_K} \exp(-K(x)) \tag{3.13}$$

$$= p(x)p(q) \tag{3.14}$$

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

since some intuition is now established around Hamiltonian Monte Carlo, we look a bit on the two versions used in Numpyro-BNN and Bohamiann,

3.2.1 No U-Turn sampling

othen the physical simulation in HMC goes forth and back the same path, and we risk getting bad samples. No U-turn (NUTS) sampling avoid this.

3.2.2 Adaptive stochatic HMC

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

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 [7] and using kernel estimator [5] and [8] 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 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 estimator regression,
- Gaussian mixture regression,
- Sum-product networks.

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,

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

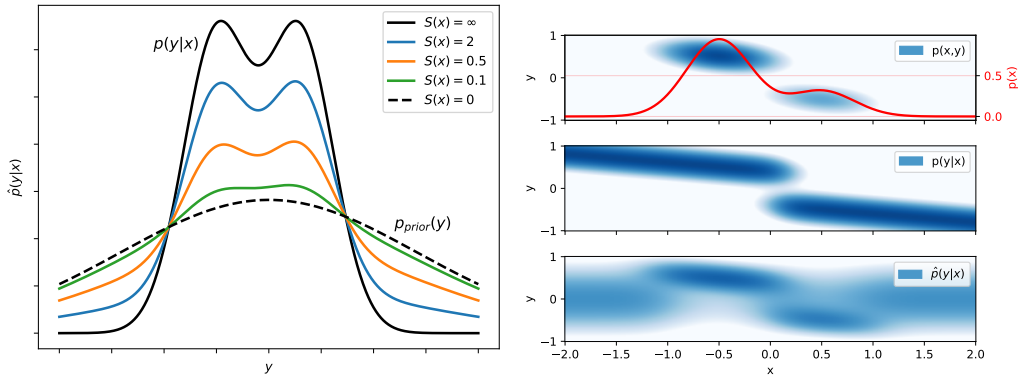


Figure 4.1: Left: Illustration of how the preditive distribution is manipulated according the the scaling function $S(x) := p(x) \cdot N \cdot \Delta$. Right: Illustration of why it makes sense to manipulate the predictive distribution $p(y|x)$, if there is a small amount of input data at a region, then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

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 as,

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

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,

$$p(y|x) = \frac{p(y,x)}{p(x)} \tag{4.1}$$

$$= \sum_z \frac{\lambda_z}{p_z(x)} p_z(x,y) \tag{4.2}$$

$$= \sum_z \frac{\lambda_z p_z(x)}{p(x)} p_z(y|x) \tag{4.3}$$

$$= \sum_z \underbrace{\frac{\lambda_z p_z(x)}{\sum_{z^*} \lambda_{z^*} p_{z^*}(x)}}_{\gamma_z(x)} p_z(y|x) \tag{4.4}$$

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.

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

4.3 Kernel 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,

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

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,

$$p(y|x) = \sum_{z=1}^n \gamma_z(x) \mathcal{N}(y|\mu_y^{(z)}, \Sigma_{yy}^{(z)}) \tag{4.5}$$

$$\gamma_z(x) = \frac{\lambda_z \mathcal{N}(x|\mu_x^{(z)}, \Sigma_{xx}^{(z)})}{\sum_{z^*} \lambda_{z^*} \mathcal{N}(x|\mu_x^{(z^*)}, \Sigma_{xx}^{(z^*)})} \tag{4.6}$$

Giving a the computational complexity of $O(n)$, 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. This conditional of this gives the Gaussian mixture regression model.

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

$$p_z(y|x) = \mathcal{N}(y|\mu_{y|x}^{(z)}, \Sigma_{y|x}^{(z)}) \tag{4.7}$$

$$\mu_{y|x}^{(z)} := \mu_y^{(z)} + \Sigma_{yx}^{(z)}(\Sigma_{xx}^{(z)})^{-1}(x - \mu_x^{(z)}) \tag{4.8}$$

$$\Sigma_{y|x}^{(z)} := \Sigma_{yy}^{(z)} - \Sigma_{yx}^{(z)}(\Sigma_{xx}^{(z)})^{-1}\Sigma_{xy}^{(z)} \tag{4.9}$$

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

$$p(y|x) = \sum_{z=1}^{K} \gamma_z(x)\mathcal{N}(y|\mu_{y|x}^{(z)}, \Sigma_{y|x}^{(z)}) \tag{4.10}$$

$$\gamma_z(x) := \frac{\lambda_z \mathcal{N}(x|\mu_x^{(z)}, \Sigma_{xx}^{(z)})}{\sum_{z^*=1}^{K} \lambda_{z^*} \mathcal{N}(x|\mu_x^{(z^*)}, \Sigma_{xx}^{(z^*)})} \tag{4.11}$$

note here the computational complexity is $O(K \cdot dim(x)^3)$ 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 is a mixture model, which allows for exponentially many mixture components, but only with linearly many parameters. Figure 4.2 illustrates how simple 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. In the SPN implementation used in this thesis, the number of distribution in each scope is called the number of channels.

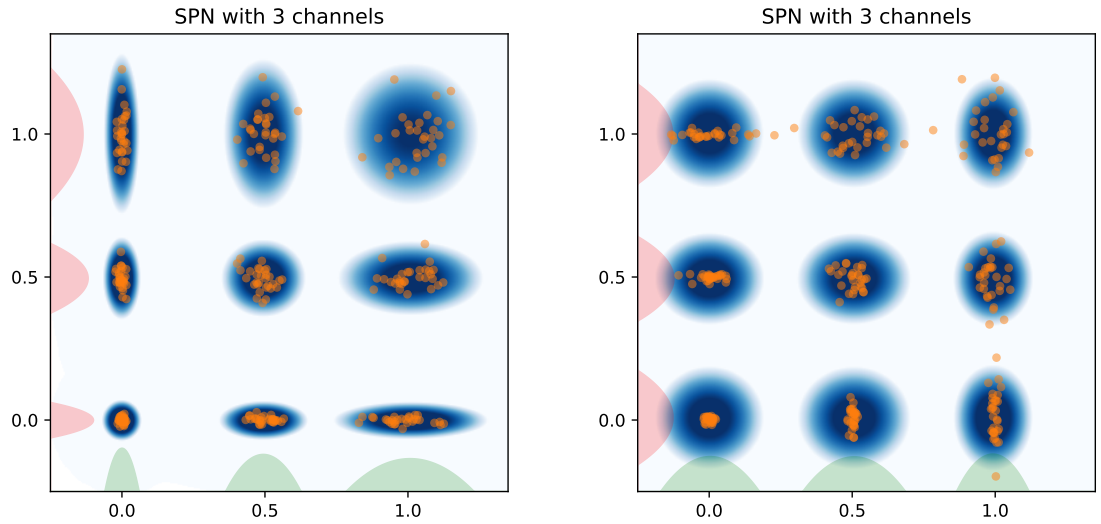


Figure 4.2: left: the data lies perfect for the SPN. Right: The data distribution is not suited for SPN

Figure 4.3 shows that even though we are getting some mixture components for free, then they can be turned off or given a small weight if no data is present there.

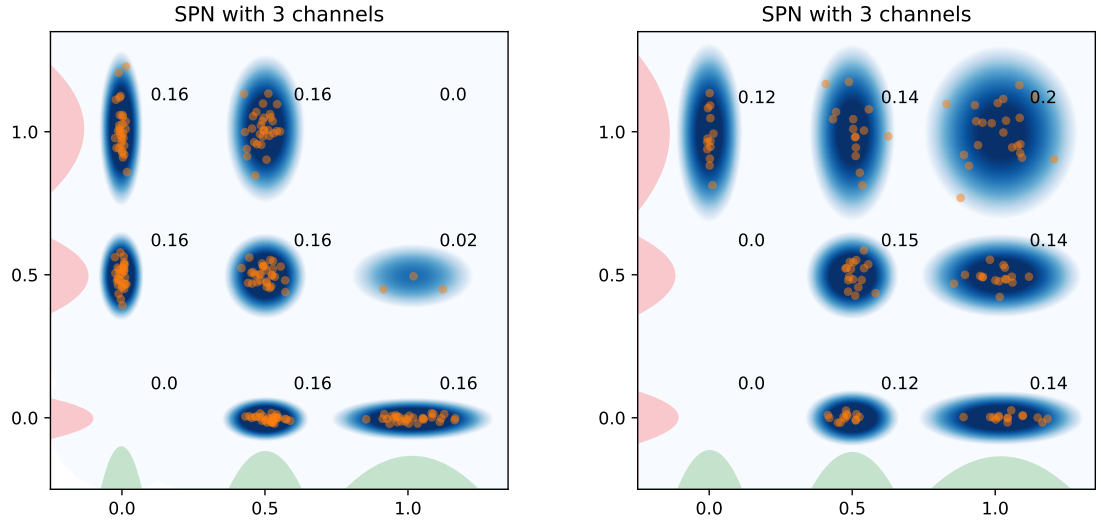


Figure 4.3: Example of how the weight of each mixture component is turned up and down according the amount of data observed.

4.5.1 SPN as mixture regression

We will to a large extend just see SPN as a large mixture model. This is a valid observation.

Definition 1 *A sub-network $\bar{S}_z$ of S is an SPN, which includes the root S and then includes nodes according to the following recursive scheme:*

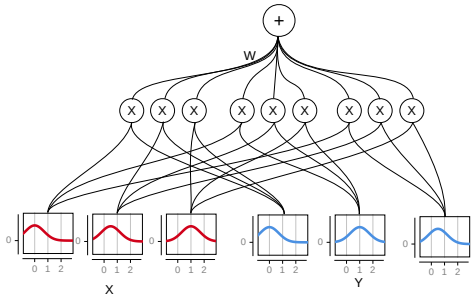


Figure 4.4: The SPN trained in Figure 4.2. Number of channels are 3, giving 9 product nodes, which are weighted in the final sum node.

Collection of sub-network S_z of S

---

function Process(node i, S_z)
  if $i \in \mathcal{L}eaf(S)$ **then**
    return:
  if $i \in \mathcal{S}um(S)$ **then**
    $S_z = S_z.add(j \in ch(i))$ ▷ include one child of node i
    return: Process(j, S_z)
  if $i \in \mathcal{P}rod(S)$ **then**
    $S_z = S_z \cup \{j | j \in ch(i)\}$ ▷ include all childen of node i
    for $j \in ch(i)$ **do**
      return: Process(j, S_z)
  return: S_z
S_z = Process(root,Ø)

So we see that at each sum node the number of different sub-networks multiplies with the number of children for that sum node. And thereby, the total number of sub-networks is

$$Z = \prod_{i \in \mathcal{S}um(S)} |ch(i)|$$

i.e. an exponentially large amount of sub-networks. This is the amount of mixture components implicitly defined in an SPN. 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

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

where $p_i(x, y)$ 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. For convinience we define each sum component as $p(z, x, y|w, \theta) := \lambda_z(w) p_z(x, y|\theta)$. Evaluation of $p(x, y|w, \theta)$ will never be done as the sum over Z components, instead there is a proposition.

Proposition 1 *Consider a SPN, S, a sum node $q \in \mathcal{S}um(S)$ and a child $i \in ch(q)$, then the following relation holds,*

$$\sum_{z:(q,i)\in\mathcal{E}(S_z)} \lambda_z(w) p_z(x, y|\theta) = w_{i,q} \frac{\partial S}{\partial v(q)} v(i)$$

SPNs can also be interpreted as a deep neural network [@vergari]. 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

4.5.2 SPN as a mixture model

<source> we can interpret an SPN as the mixture model,

$$p(x, y) = \sum_{z\in\Sigma(S)} \lambda_z p_z(x, y)$$

where $\lambda_z = \prod_{(q,j)\in\mathcal{E}(z)} w_{q,j}$. And where the mixture components are given as

$$\begin{aligned} p_z(x, y) &= \prod_{l\in\mathcal{L}eaf(z_x)} \phi_l(x) \prod_{l\in\mathcal{L}eaf(z_y)} \phi_l(y) \\ &=: p_z(x) p_z(y) \end{aligned}$$

where ϕ_l is the density of the l'th leafs tractable distribution. The last equation is splitting the products up in the two marginals $p_z(x)$ and $p_z(y)$ since they are uncorrelated.

The responsibility

$$p(z|x) = \frac{\lambda_z p_{z_x}(x)}{\sum_{z\in\Sigma(S)}}$$

is calculated easily using autograd.

4.5.3 Conditional of SPN

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

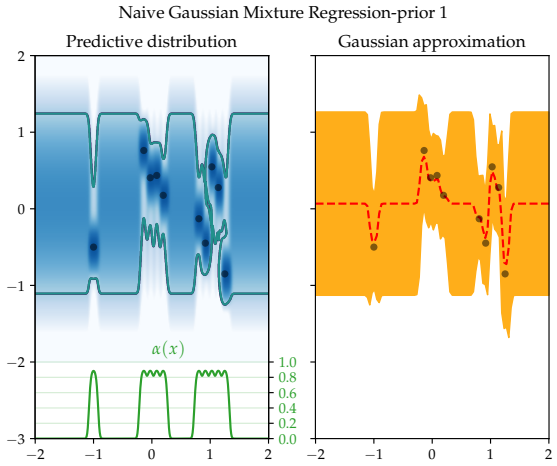
where ϕ_l is the density of the l'th leafs tractable distribution. Recall that we can interpret an SPN as the mixture model,

$$p(x, y) = \sum_{z\in\Sigma(S)} \lambda_z p_z(x, y)$$

where $\lambda_z = \prod_{(q,j)\in\mathcal{E}(z)} w_{q,j}$.

$$p(y|x) = \sum_{z\in\Sigma(S)} \gamma(x) p_{z_y}(y)$$

With $\gamma(x) = \frac{\lambda_z p_{z_x}(x)}{\sum_{z\in\Sigma(S)} \lambda_z p_{z_x}(x)}$



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 Gaussian approximation of mixture regression

4.6.1 Mean and variance of conditional SPN

The mean of the conditional is just

$$\begin{aligned} E_{p(y|x)}[y] &= \sum_{z \in \Sigma(S)} \gamma(x) \int y p_{z_y}(y) dy \\ &= \sum_{z \in \Sigma(S)} \gamma(x) \prod_{l \in \mathcal{L}eaf(z_y)} E_{\phi_l}[y] \end{aligned}$$

and the variance is found using the second moment,

$$\begin{aligned} E_{p(y|x)}[y^2] &= \sum_{z \in \Sigma(S)} \gamma(x) \int y^2 p_{z_y}(y) dy \\ &= \sum_{z \in \Sigma(S)} \gamma(x) \prod_{l \in \mathcal{L}eaf(z_y)} (Var_{\phi_l}[y] + E_{\phi_l}[y]^2) \end{aligned}$$

4.7 Mixture model training

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

4.7.1 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,

$$p(x) = \sum_{k=1}^{K} w_k p_k(x) \tag{4.12}$$

$$\iff \qquad p(x) = p_z(x), \qquad z \sim Cat(w_i, \dots, w_K). \tag{4.13}$$

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 [9]

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.

EM for Gaussian mixture

For a Gaussian mixture the p_k distributions in (4.12) 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 $p(z) = \sum_{k=1}^{K} 1_{z=k} w_k = w_z$, com-

bining 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 [9] 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

5 Results

<reproducer resultater fra Arayns thesis> Egne forsøg med SPN og mixture regression

5.1 implementation

The Gaussian process regression is implemented using the Scikit-learn Gaussian process implementation with the Matern kernel with $\nu = 1.5$ and the y-values are normalized in order for the prior distribution to be reasonable (mean 0 and variance 1). The lengthscale is optimized by maximizing the marginalized likelihood using the limited memory quasi newton solver with bounds l-bfgs-b with 200 restarts.

The Bayesian neural network is implemented using Numpyro - which is a python library for probabilistic machine learning, developed by some of the people behind Pyro, but instead of using pyTorch as backend, Numpyro uses Jax. This allows for significantly large speedup when doing MCMC, i.e. NUTS sampling. This was however still very slow and we therefore limited the network size to a small 3 layer with 10 nodes on each layer network. The prior distribution for weights is a standard normal distribution, and the bias normal priors are sat a bit less restrictive with a standard deviation of 2 instead. This is reasonable since the data is always standardized.

5.1.1 standardized data

Before the data reach any of the models, it is standardized. Which is a scaling and translation such that the datas empirical mean and standard deviation are 0 and 1, respectively. In that way is much easier to control the parameters in the models, since the data it fits and predicts is always the same scale. The transformation is as following, first time the models sees the data, the empirical mean and standard deviation is recorded both for x and y, (note x can be a vector), giving μ_x,μ_y, $\sigma_x$and σ_y. next all data is transformed using the transformation,

$$T_x(\cdot) := \frac{\cdot - \mu_x}{\sigma_x}$$

When we predict the x is transformed, the model output a prediction and then the inverse transform is mapping the prediction from the standardised domain to the original domain, $T_y^{-1}(\hat{y}) := \hat{y} \cdot \sigma_x + \mu_x$

5.2 Regression analysis

As described in the previous sections an essential part of Bayesian optimization and the decision theory build around Bayesian optimization, that the regression model is correct. We will therefore look at how good the regression models are, both in terms of correct prediction and in terms of correct uncertainty estimation.

5.2.1 uncertainty quantification

The probibalistic model is given as

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

where the mean, and variance functions are different for all the models. Given a trained model, we quantify its ability to capture uncertainty with the average predictive likelihood/ probability of

the observation, given a test input x_i the the density of predictive distribution is evaluated in the coresponding test output y_i,

$$\overline{p(y_i|x_i,\mathcal{D})} := \frac{1}{n}\sum_{i=1}^{n} p(y_i|x_i,\mathcal{D})$$

there the bigger the mean predictive likelihood is, the better. it is not called preditive likelihood!!

5.2.2 prediction quantification

Here we use the mean absolute error to quantify the prediction error. The i'th test point is (x_i, y_i) and the mean absolute error is given as, $MAE := \frac{1}{n}\sum_{i=1}^{n} |\mu_{\mathcal{D}}(x_i) - y_i|$

5.2.3 regression benchmark

We will benchmark our different regression models againt a simple emperical mean and emperical std. normal distribution benchmark,

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

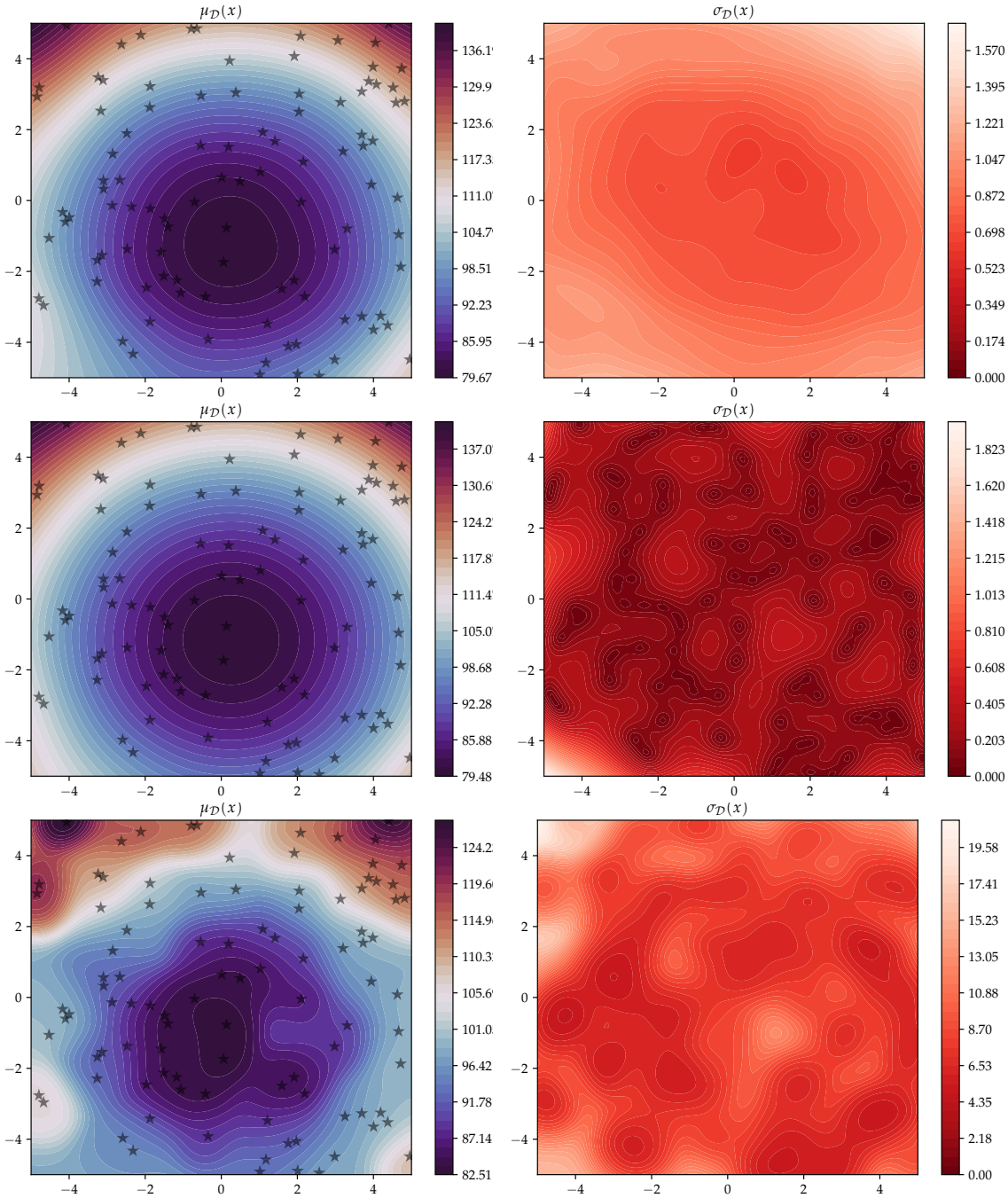
where $\bar{\mathbf{y}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{y}_i$ and $\bar{\sigma}^2(\mathbf{y})) = \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{y}_i - \bar{\mathbf{y}})^2$ this is however not a good model for Bayesian optimization, as it will not provide a new candidate point, as all points are equally good.

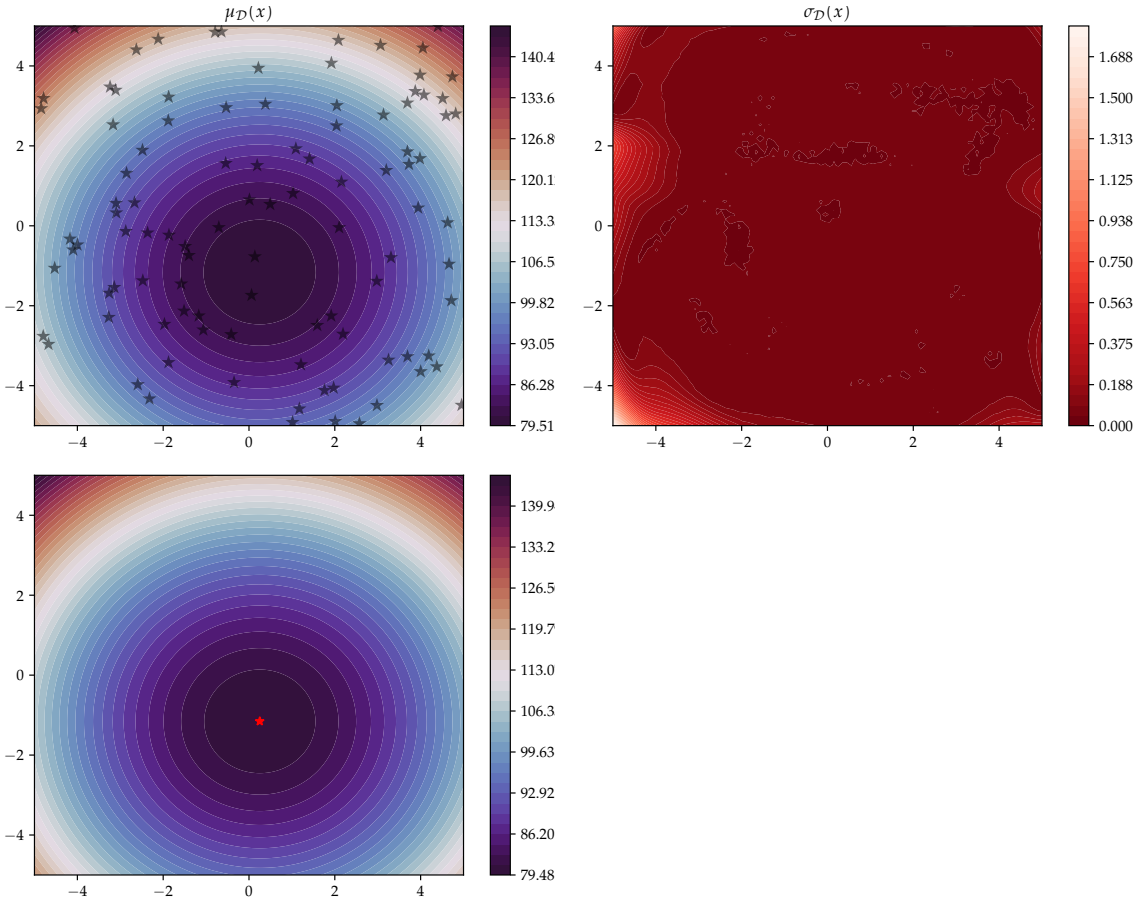
5.3 Regression analysis of GP, BOHAMIANN and NumpyNN 1D

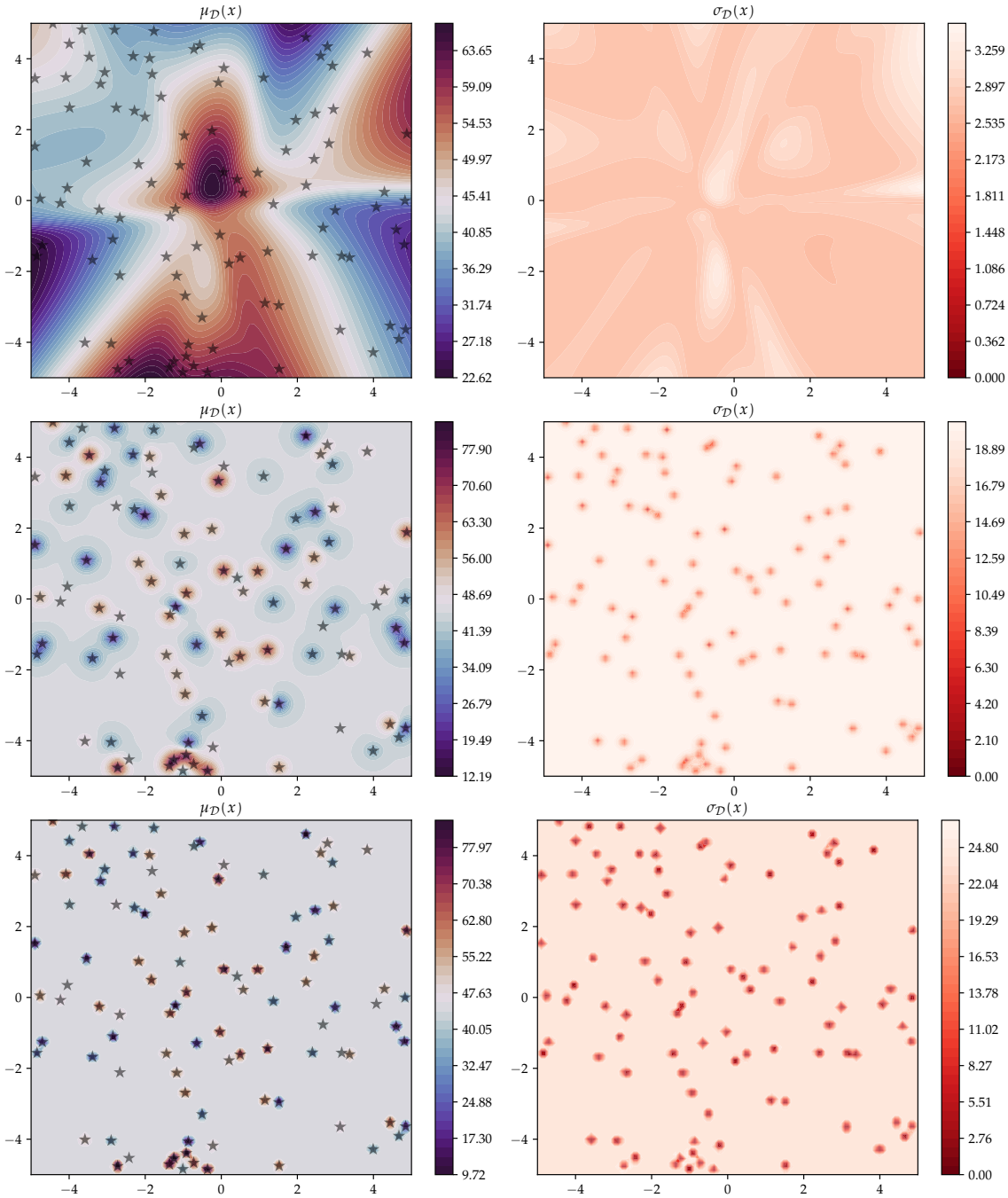
5.4 Regression analysis of GP, BOHAMIANN and NumpyNN 2D

5.5 Mixture regression on simple functions

Choosing a good set of design parameters is crucial, the manipulation...


$\mu_{\mathcal{D}}(x)$
$\sigma_{\mathcal{D}}(x)$
$\mu_{\mathcal{D}}(x)$
$\sigma_{\mathcal{D}}(x)$
$\mu_{\mathcal{D}}(x)$
$\sigma_{\mathcal{D}}(x)$


$\mu_{\mathcal{D}}(x)$
$\sigma_{\mathcal{D}}(x)$
140.4
133.6
126.8
120.1
113.3
106.5
99.82
93.05
86.28
79.51
1.688
1.500
1.313
1.125
0.938
0.750
0.563
0.375
0.188
0.000
139.9
133.2
126.5
119.7
113.0
106.3
99.63
92.92
86.20
79.48


$\mu_{\mathcal{D}}(x)$
$\sigma_{\mathcal{D}}(x)$
$\mu_{\mathcal{D}}(x)$
$\sigma_{\mathcal{D}}(x)$
$\mu_{\mathcal{D}}(x)$
$\sigma_{\mathcal{D}}(x)$

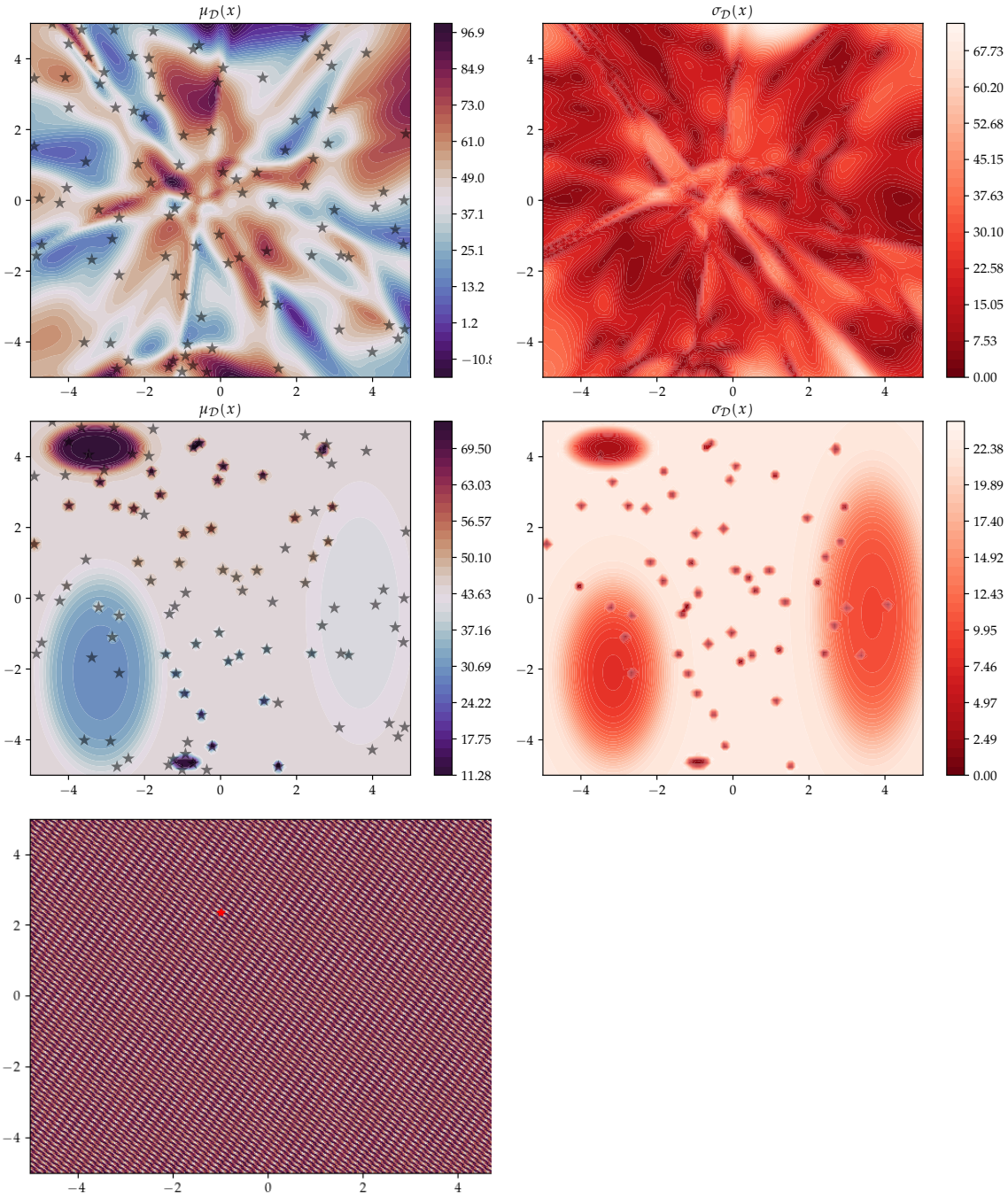


Figure 5.1: 2D Regression plot for a few of the problems, with the correct

Figure 5.2: Regression plot for one problem with all dims: Number of data points / dims, mean squared error, with bayesline mean prediction

Figure 5.3: Regression plot for anther problem

	f1	f2	f3	f4	f5
BO_numpy	4.62(13.0)	897.13(39.0)	2.07(36.0)	10.14(39.0)	0.03(28.0)
BO_BOH	0.33(32.5)	552.83(27.0)	5.51(33.0)	10.33(37.0)	0.03(28.0)
BO_Gaus	0.89(8.0)	21.48(31.5)	21.86(21.0)	16.32(25.5)	0.03(22.5)
BO_Naiv	3.25(9.0)	72571.84(28.0)	19.41(15.0)	19.61(8.5)	4.35(21.0)
BO_em	1.23(24.5)	455.28(10.0)	6.74(8.5)	5.98(30.5)	6.00(22.5)

Figure 5.4: Distance to optima $f(x^{best}) - f^*$ using Bayesian optimization with different surrogates and using expected improvement with a budget of 40 samples for group 1: "5 separable functions"

	f6	f7	f8	f9
BO_numpy	1.31(28.0)	2.65(10.5)	nan(nan)	nan(nan)
BO_BOH	0.05(33.0)	1.68(17.0)	10.70(16.0)	2.31(39.0)
BO_Gaus	6.53(32.0)	0.60(12.0)	1.14(18.8)	0.27(25.0)
BO_Naiv	4.74(20.0)	4.71(25.5)	11.87(2.0)	20.09(15.5)
BO_em	5.43(20.5)	1.97(24.5)	5.11(24.0)	68.97(8.0)

Figure 5.5: Distance to optima $f(x^{best}) - f^*$ using Bayesian optimization with different surrogates and using expected improvement with a budget of 40 samples for group 2: "4 functions with low or moderate conditioning"

	f10	f11	f12	f13	f14
BO_numpy	nan(nan)	nan(nan)	nan(nan)	nan(nan)	nan(nan)
BO_BOH	286.57(30.0)	7.48(35.0)	2770.49(31.0)	4.98(16.0)	0.03(30.0)
BO_Gaus	16.52(33.2)	638.31(32.0)	3033.71(27.8)	8.84(23.8)	0.01(36.5)
BO_Naiv	49166.58(27.0)	2848.42(22.0)	23714.17(24.0)	52.21(2.0)	0.61(31.0)
BO_em	61391.71(14.0)	10317.74(20.5)	673.63(12.5)	10.04(19.0)	0.80(28.0)

Figure 5.6: Distance to optima $f(x^{best}) - f^*$ using Bayesian optimization with different surrogates and using expected improvement with a budget of 40 samples for group 3: "5 functions with high conditioning, unimodal"

	f15	f16	f17	f18	f19
BO_numpy	nan(nan)	nan(nan)	nan(nan)	nan(nan)	nan(nan)
BO_BOH	10.05(20.0)	9.02(33.0)	4.00(4.0)	3.55(38.0)	2.05(27.0)
BO_Gaus	17.12(28.0)	2.86(10.0)	3.15(18.5)	3.26(10.0)	13.31(27.0)
BO_Naiv	34.09(20.5)	3.20(20.5)	1.69(9.5)	11.72(24.0)	3.36(5.0)
BO_em	14.75(24.5)	10.70(19.0)	2.27(21.5)	9.63(13.5)	1.99(9.5)

Figure 5.7: Distance to optima $f(x^{best}) - f^*$ using Bayesian optimization with different surrogates and using expected improvement with a budget of 40 samples for group 4: "5 multi-modal functions with adequate global structure"

	f20	f21	f22	f23	f24
BO_numpy	nan(nan)	nan(nan)	nan(nan)	nan(nan)	nan(nan)
BO_BOH	2.44(34.0)	1.99(1.0)	1.98(25.0)	2.64(15.0)	8.98(14.0)
BO_Gaus	1.67(29.0)	1.23(22.5)	2.61(12.8)	5.24(17.2)	9.99(19.2)
BO_Naiv	2.43(15.5)	0.27(21.5)	2.04(4.0)	5.15(5.0)	7.15(26.5)
BO_em	3.16(17.0)	1.07(15.0)	0.90(22.0)	5.66(13.5)	3.82(6.5)

Figure 5.8: Distance to optima $f(x^{best}) - f^*$ using Bayesian optimization with different surrogates and using expected improvement with a budget of 40 samples for group 5: ”5 multi-modal functions with weak global structure”

Figure 5.9: BayesOpt plot: Number of iterations, distance to optima using EI, with bayesline random search

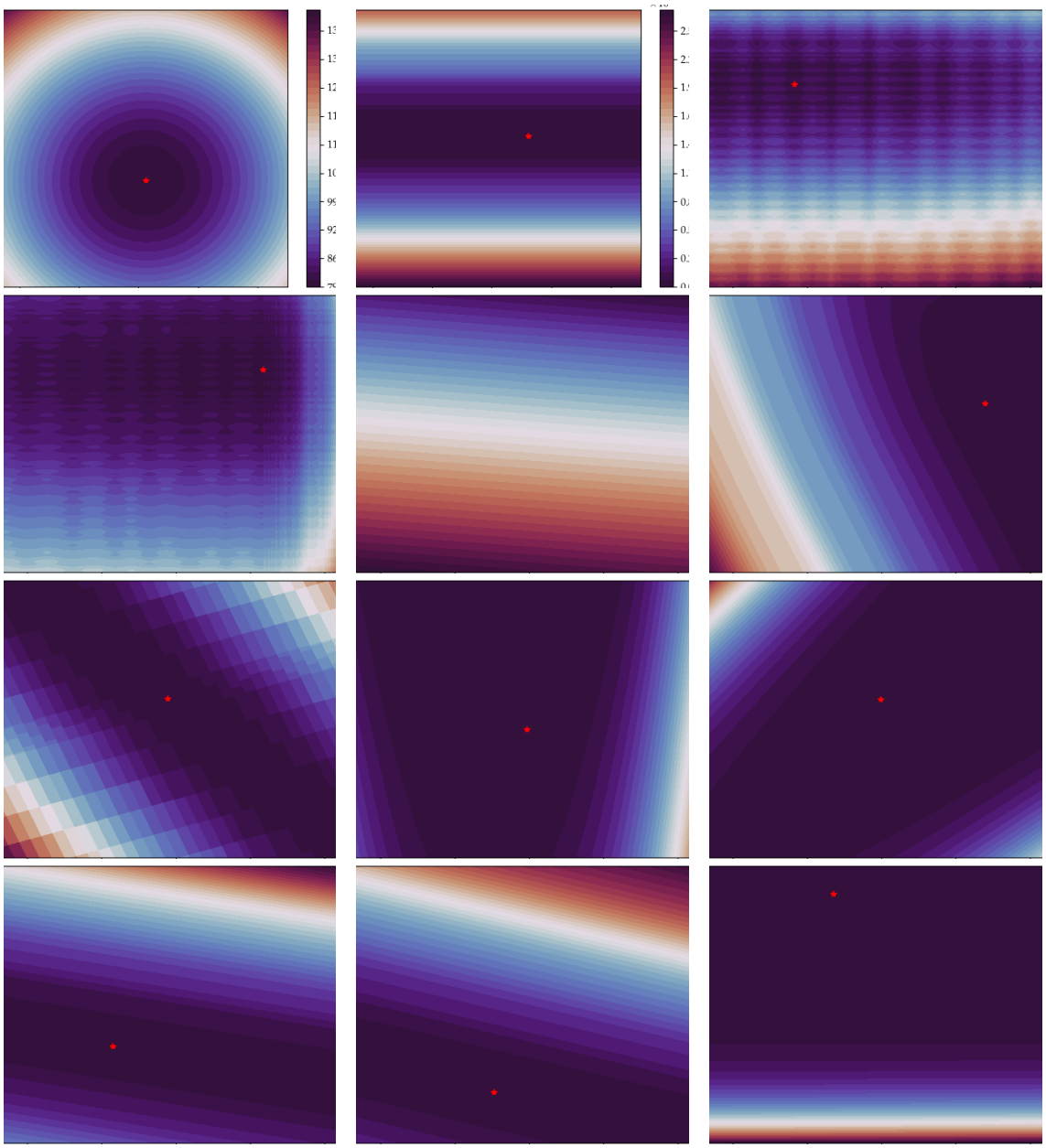


Figure 5.10: Test functions f1 to f12

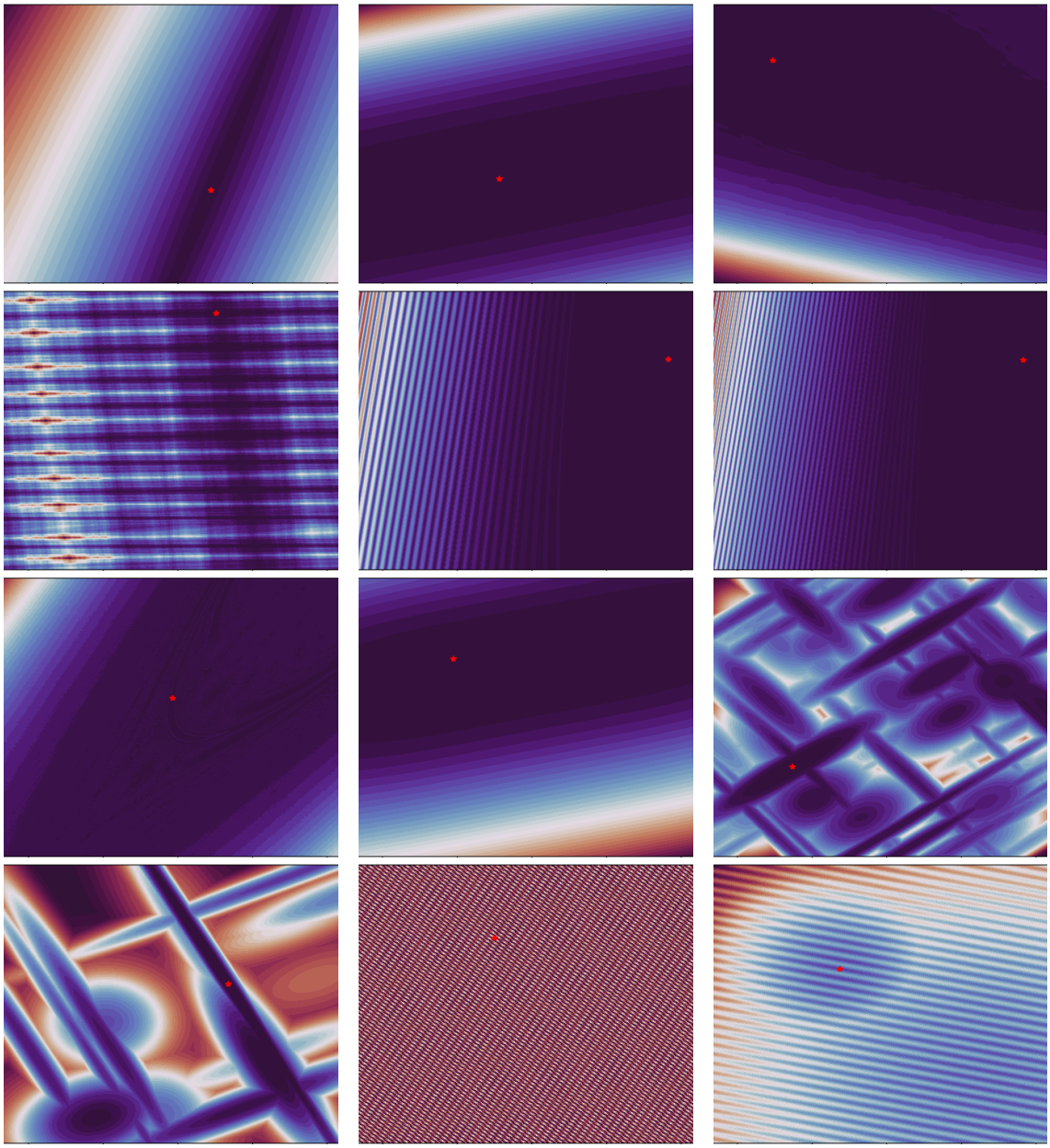


Figure 5.11: Test functions f13-24

6 Conclusion and further work

discussion: Using a generative model as a surrogates model is a novel idea - maybe it is not a good idea.. Discretized regression

Maybe manipulation of variance could be done, or same mean prediction?!? Or mean prediction using the means of the close by x-values.

Bibliography

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

[2] 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.

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

[4] 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.

[5] 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.

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

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

[8] 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.

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

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

[11] 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.

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 [4][10][11] (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