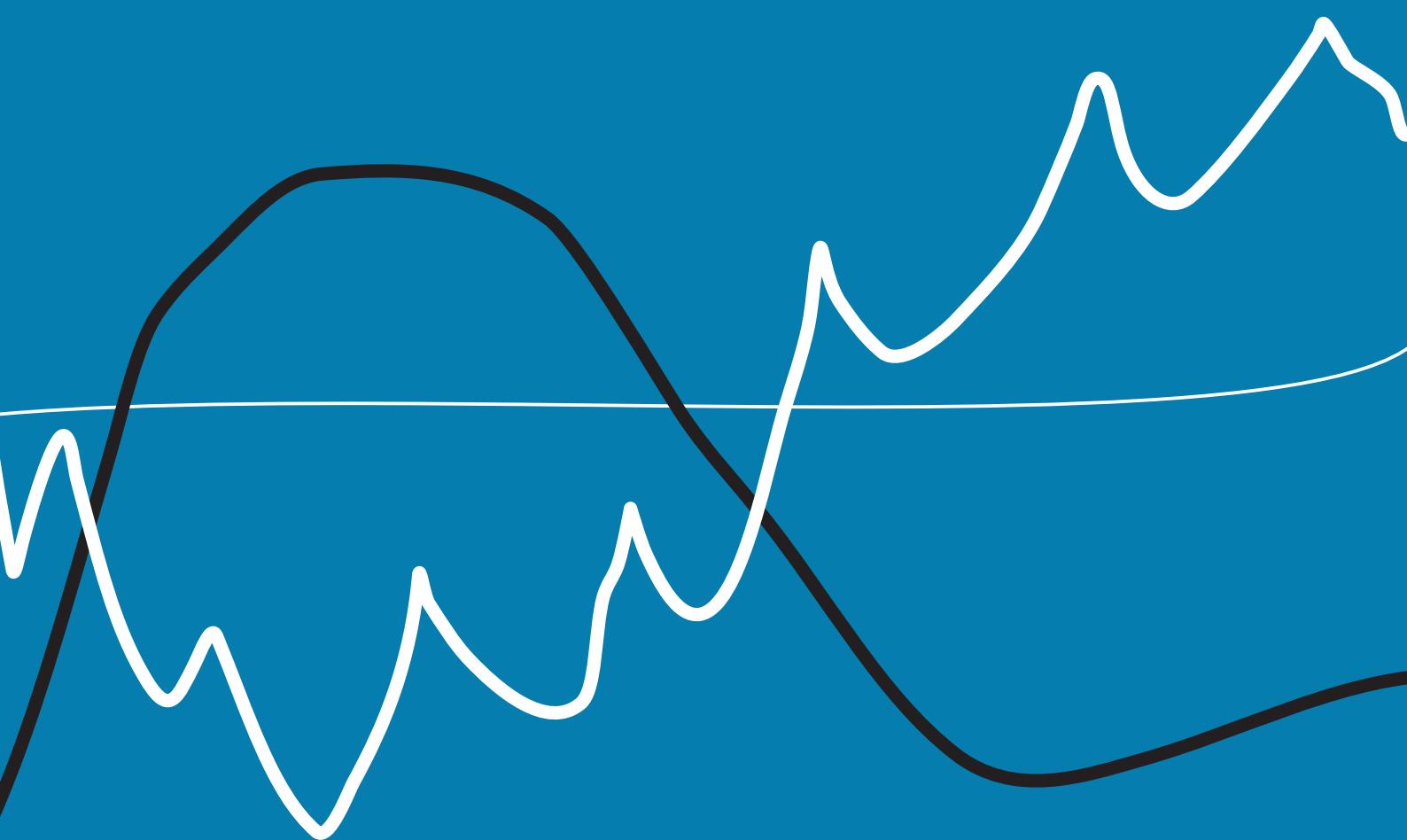


Investigation of Deep Probabilistic Surrogate-models in Bayesian Optimization

Master Thesis



Investigation of Deep Probabilistic Surrogate-models in Bayesian Optimization

Master Thesis

June, 2022

By

Simon Saugmandsgaard Kruse

Copyright:	Reproduction of this publication in whole or in part must include the customary bibliographic citation, including author attribution, report title, etc.
Cover photo:	Vibeke Hempler, 2012
Published by:	DTU, Section of Cognitive Systems, Building 321, 2800 Kgs. Lyngby Denmark www.github.com/SimonKruse0/master-thesis
ISSN:	[0000-0000] (electronic version)
ISBN:	[000-00-0000-000-0] (electronic version)
ISSN:	[0000-0000] (printed version)
ISBN:	[000-00-0000-000-0] (printed version)

Contents

1	Introduction	1
1.1	Contribution	2
1.2	Related work	2
1.3	Structure of the thesis	3
1.4	notation	3
2	Bayesian Optimization	5
2.1	Optimization methodology	5
2.2	Bayesian Optimization	8
2.3	Acquisition function	8
2.4	surrogate model	10
2.5	Inference of surrogate models	10
3	Discriminative surrogate models	13
3.1	Gaussian process surrogate	13
3.2	Bayesian Neural Networks	15
3.3	Regression analysis	20
4	Generative model surrogates	21
4.1	”Bayesianizing” conditional distribution	21
4.2	Kernel estimator regression	23
4.3	Gaussian mixture regression	23
4.4	Sum product networks	24
4.5	Mixture model training	30
5	Results	35
5.1	Regression analysis	35
5.2	Regression analysis of GP, BOHAMIANN and NumpyNN 1D	35
5.3	Regression analysis of GP, BOHAMIANN and NumpyNN 2D	35
5.4	Mixture regression on simple functions	35
6	Conclusion and further work	43
Bibliography		44
A Implementation		45

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 art of choosing the best decision among a set of possible decisions. Often we try to quantify how good a decision is: A bus takes 20 min, a car takes 15 min to go from A to B, How would you rate the song? This bridge construction costs 10 million kroner. If it is possible to come up with a quantification of how good a decision is in terms of a real number, then we can define the optimization problem as *mathematical* optimization problem:

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

where the functional $f : \mathcal{X} \rightarrow \mathbb{R}$ 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), then finding the minima in the negating the objective function $-f(x)$ is equivalent. Throughout this thesis, we refer to optimization as minimizing the objective function. The optimization problem is now in the domain of numbers and here 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 samples from a highly expensive objective, then a cheap (surrogate) function is used to fit the samples. The next sample is found by minimizing the surrogate and the process is repeated. Bayesian optimization seeks to enhance this procedure with probability theory, where the surrogate function becomes a probabilistic (Bayesian) regression model. The most common choice is a Gaussian Process, as it encapsulates the uncertainty very well, but also because its inference procedure (computing answers to probability queries like $p(y|x)$) is exact.

Even though GP has proven good for many cases, there will be problems where its assumptions do not hold. E.g the commonly seen GP with an isotropic kernel (covariance between two points is invariant to translation in input), yields a strong assumption about the continuity of the objective function and that the objective function behaves similarly throughout the domain \mathcal{X} . In Figure 1.1 we see an example of how the GP's uncertainty quantification is going wild due to a discontinuity in the underlying objective function (a small varying step-function). In some areas is it very common to have these discontinuities, for instance, in material discovery where it is well known that materials often change very suddenly [1]. To accommodate these challenges the literature introduces more flexible kernel functions for the GP, however, this introduces additional hyperparameters and since we often only deal with a small amount of data, tuning and computation can be significantly challenging [1].

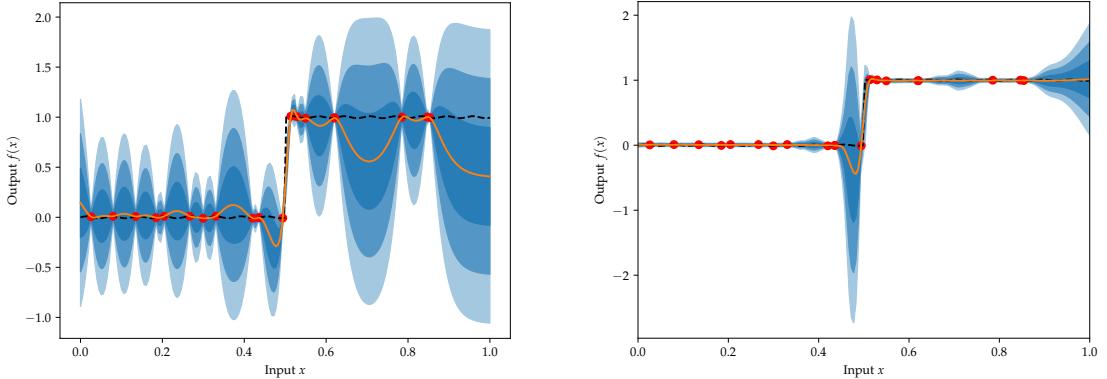


Figure 1.1: Left: GP fitted to 20 data points. Right: A Bayesian neural network fitted to the same points. The objective function is the dashed black line. This exemplifies how a discontinuity makes the standard implemented GP (optimized with empirical bayes) overreact to all other areas in the domain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

GPs are very attractive models since they allow for exact inference, a closed-form expected improvement (more on this later) and in general good uncertainty quantification. However, as we saw in the figure 1.1 maybe we can do better! If it is possible to sample a few times less from the objective function, it can potentially save hours of work, lots of money, or energy (Depending on what is demanded to evaluate the objective function). 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 types of (complex) problems, and just as well as the GP on most types of problems.

1.1 Contribution

This thesis investigates surrogate models alternative to 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 employed as 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 of the objective function are necessary to find the minima. So even though the surrogate modeling and optimization can be really slow, it is assumed that the objective function is very expensive, so a slow surrogate model would not matter.

1.2 Related work

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

- Gaussian Process
- Bayesian neural network
- Random forest
- Kernel estimator
- Bayesian multivariate adaptive regression splines (BMARS)

- Bayesian additive regression trees (BART)

Often the main focus has been on lowering the computational complexity of inference while showing performance is (or almost is) as well as GPs. Inference time of the GP scales cubic with the number of data points and only linear for Bayesian Neural Network. Lowering the inference time is also the main focus of the PhD thesis "Sample-efficient Optimization Using Neural Networks" from 2020 [2], but 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 clear for high-dimensional problems. <mention BOHAMIANN and DNGO>

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" [3] 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

- The first chapter introduces the concept of optimization in general, Bayesian optimization and how surrogates/Bayesian regression models are relevant.
- The second chapter introduces GP and Bayesian Neural Networks
- The third chapter introduces the mixture regression and the novel new SPN regression.
- The fourth chapter results
- Finally a 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. However, we start with an introduction to the optimization framework in general.

2.1 Optimization methodology

Given a cost/objective function $f : \mathcal{X} \rightarrow \mathbb{R}$, 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

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

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 is intractable except for rare cases e.g. if f is convex and analytically directly solvable or the domain of f is very limited. Hm

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 found among a potentially infinitely large set of stationary points ($\nabla f(x) = 0$) and boundary points - this might be tedious or impossible. When the problem is not directly solvable, mathematical optimization takes an indirect methodology: Design a sequence of experiments that reveal information of 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 [4] and presented here as Algorithm 1. W

Algorithm 1 Sequencial Optimization [4]

```

Input: Initial dataset  $\mathcal{D}$  ▷ can be empty
while Termination 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 landscape¹ a policy selects a point $x \in \mathcal{X}$ where we make our next observation. Policies can be deterministic or probibalitic, e.g. grid search and random

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

search are examples of policies. The next observation provides us a y value, which combined with x is included to the available data \mathcal{D} . Finally, a stopping criterion decides whether to continue or terminate.

<example of random search is half as efficient as BO>

Example: Grid search

In grid search values along each dimension in \mathcal{X} are selected and combined with each other, which thereby defines a grid in the parameter space. All points are ordered and systematically selected. In the content of algorithm 1 we define the grid search policy as

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

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

Example: Random search

In random search a uniform distribution is laid over the domain space \mathcal{X} and a random point is selected from the distribution.

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

Note that grid search and random search are policies which completely ignores the available data. This is a shame and we can do better.

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 modify y to be a vector, since the observation model is given as:

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

Example: Surrogate-based optimization

<SVR> <Radial Basis function> <Polynomial model/response surface> 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. 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.

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

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

2.1.1 When to use Bayesian optimization

What if $f(x)$ took several days to evaluate. What if $f(x)$ is noisy? what if discrete points? <more here>

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 observations \mathcal{D} and opposed 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,

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

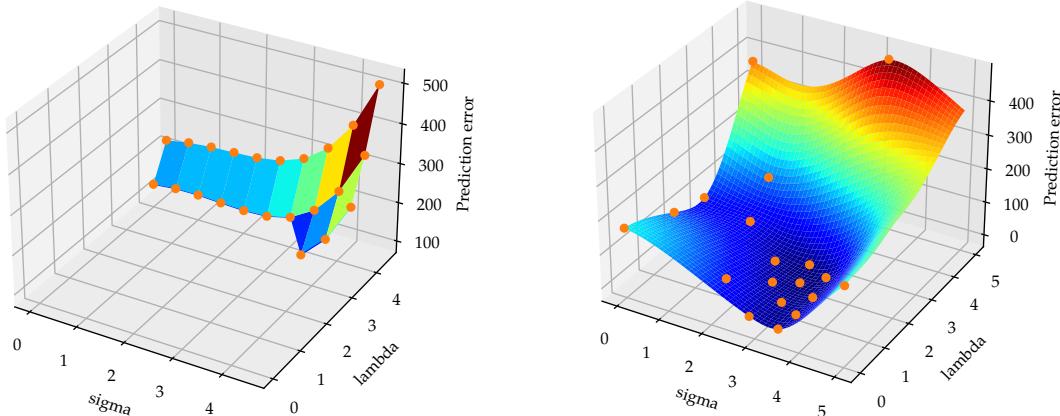


Figure 2.1: Hyper parameter tuning of a model $M(\lambda, \sigma)$, 23 evaluation in grid search vs 23 evaluations using Bayesian optimization

additionally, Bayesian Optimization allow for observation noise,

2.1.2 Observation model

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 a well known. 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 heterostodatic setting) and implies a Gaussian observation model,

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

we can 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,

$$p(y|x, f(x)) = \delta(y - f(x))$$

i.e. all probability mass for y is on the value $f(x)$ giving the observation sample $y = f(x)$

2.2 Bayesian Optimization

<Cope with inaccuracies> i.e. allows for stochastic objective function. <Uncertainty measure with prediction based on simple and clear prior assumptions about the characteristic about the objective function. > <Provides an adequate termination condition for the opt. process>.

<kilde 151. Bayes Opt is assumes superior to other global optimization techniques with limited budget>

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 sets of parameters by treating the set of parameters as a random quantity, θ . What is of interest is the predictive posterior distribution,

$$p(y|x, \mathcal{D}) \quad (2.2)$$

Bayesian optimization is essentially two steps: First, a probabilistic surrogate-model is fitted to the available data \mathcal{D} giving the predictive distribution (2.2). Second, the next sample point is chosen according to a so-called acquisition function, which in a sense balances the well-known concept, exploitation and exploration. Where exploitation will be choosing the next point according to its expected 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 this thesis, which is also the most commonly used: Expected improvement.

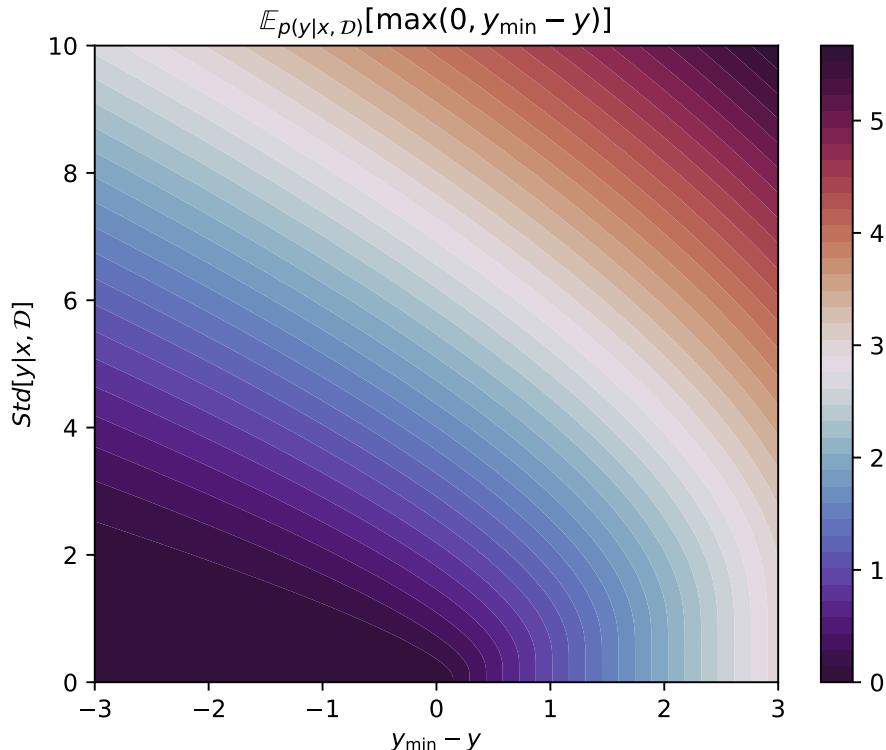
2.3 Acquisition function

2.3.1 Expected improvement

A popular choice of acquisition function is expected improvement,

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

this will only look at the expectation of the values y , which improves the current best value.



In the following derivation we assume the predictive distribution can be approximated 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 clumsy 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 cumulative 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 yields a closed form solution to the expected improvement function,

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

where $u := \frac{y_{\min} - \mu_x}{\sigma_x}$. 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 $\phi(\epsilon) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\epsilon^2}{2}\right)$,

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

and evaluating the Riemann 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 explicitly 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 interpreted as exploitation (favouring points with a large improvement $I(x) := (y_{\min} - \mu_x)$) and the second part can be seen as exploitation (favouring points with high

uncertainty.) taking the derivative with respect to $I(x) := (y_{\min} - \mu_x)$ and σ_x , we see that expected improvement is increasing if the improvement grows or if the variance σ_x grows, i.e.

$$\frac{\partial EI(x)}{\partial I(x)} = \Phi\left(\frac{y_{\min} - \mu_x}{\sigma_x}\right) > 0, \quad \frac{\partial EI(x)}{\partial \sigma_x} = \phi\left(\frac{y_{\min} - \mu_x}{\sigma_x}\right) > 0$$

<obs mistake in the book!!!?>

$$\begin{aligned} \mathbb{E}_{y_*|\mathbf{x}_*, D_n} [\max(0, y_{\min} - y_*)] &= ?? \\ \mathbb{E}[\min(0, y_{\min} - y_*)|\mathbf{x}_*, D_n] &= \int_{-\infty}^{\infty} \min(0, y_{\min} - y_*) p(y_*|\mathbf{x}_*, D_n) dy_* \\ &= \int_{-\infty}^{y_{\min}} (y_{\min} - y_*) p(y_*|\mathbf{x}_*, D_n) dy_* \\ &\approx \frac{1}{N} \sum_{\theta \in \Omega} [y_{\min} - f_{\theta}(x)] \end{aligned}$$

where $\Omega = \{\theta | f_{\theta}(x) < y_{\min}\}$

2.3.2 Entropy search

2.4 surrogate model

As mentioned in the previous sections, the first of two repeated steps in Bayesian optimization is to create a good Bayesian regression model. i.e. finding the probability of prediction for an arbitrary point x given datapoints $\mathcal{D} = \{x_1, y_1, \dots, x_n, y_n\}$,

$$p(y|x, \mathcal{D})$$

The surrogate model of choice in Bayesian optimization is a Gaussian Process, and Bayesian Neural Network. These are discriminative models, however, another approach, which we focus on in this project, is to model y and x jointly in a so-called generative model.

2.4.1 Discriminative model as surrogate model

When talking about a probabilistic surrogate model we are always implicitly 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.

2.4.2 Using a generative model as surrogate model

..

2.5 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 creates 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 Bohamian and Numpyro BNN are similar regression models, but are inferred using two different versions of the MCMC method, Hamiltonian Monte Carlo. As it will be revealed later (see result section) approximate inference might indeed be flawed and inexact.

Model	Predictive inference	Learning
GP	Exact $O(n^3)$	Emperical Bayes
SPN	Exact $O(E)$	EM $O(E)$
Gaussian Mixture Regression	Exact $O(K)$	EM
Bohamiann	Adaptive stochastic HMC	
Numpyro BNN	No U-Turn Sampler	

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 components. We will soon learn that for an SPN the number of mixture components is exponential larger than number of edges i.e. $E \ll K$. In theory MCMC methods samples from true the posterior distribution, and do not need any fitting/learning.

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 parametrised by some parameters, θ .

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

3.1 Gaussian process surrogate

The most popular surrogate model is the Gaussian process, we will now understand model in more detail.

3.1.1 Exact predictive distribution

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

$$p(y|x, \mathcal{D}) = \int \mathcal{N}(y|f(x), \sigma^2) p(f(x)|\mathcal{D}) df(x) \quad (3.1)$$

we will soon see that $p(f(x)|\mathcal{D}) = \mathcal{N}(f(x)|\dots, \dots)$ 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 distribution, $p(y|x, \mathcal{D})$, using formula (3.2).

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,

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

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

$$p(y) = \mathcal{N}(y|A\mu + b, L^{-1} + A\Lambda^{-1}A^T) \quad (3.2)$$

$$p(x|y) = \mathcal{N}(x|\Gamma\mu + \Gamma[A^T L(y - b)], \Gamma) \quad (3.3)$$

$$\Gamma := (\Lambda + A^T L A)^{-1} \quad (3.4)$$

Posterior function

Recall we assume $\mathbf{f} = (f(\mathbf{x}_1), \dots, f(\mathbf{x}_n))$ 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 dimensional vector. We call this quantity *the posterior function*

$$p(f(\cdot)|\mathcal{D}) = \int p(f(\cdot)|\mathbf{x}, \mathbf{f}) p(\mathbf{f}|\mathcal{D}) d\mathbf{f}. \quad (3.5)$$

Prior we assume that the function takes values according 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,

$$c(\mathbf{x}, \mathbf{x}) = \begin{bmatrix} c(x_1, x_1) & \dots & c(x_1, x_n) \\ \vdots & \ddots & \vdots \\ c(x_n, x_1) & \dots & c(x_n, x_n) \end{bmatrix} \quad c(x, y) := \text{Matern}(x, y) \dots$$

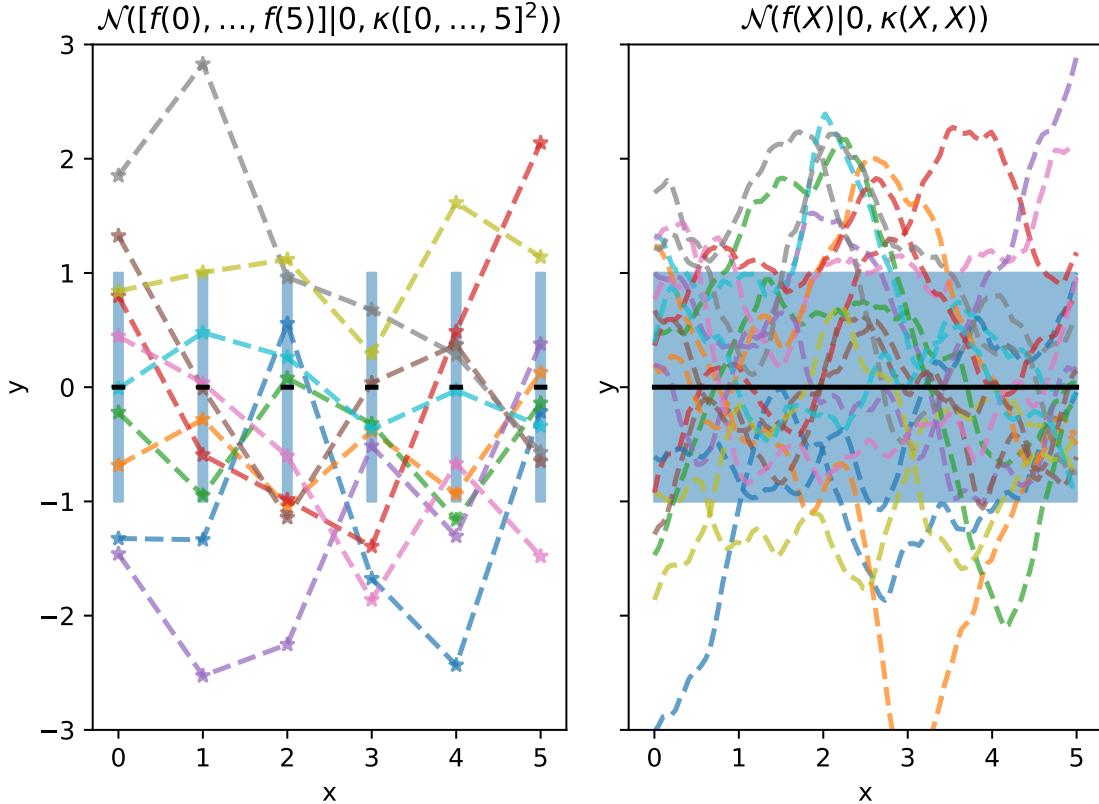


Figure 3.1: Realisations > Illustration of GP is just samples from a multivariate normal distribution. In the limit we have an infinite-variate normal distribution, which we call a GP.

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

We now calculate the first term in the integral (3.5), $p(f(\cdot)|\mathbf{x}, \mathbf{f})$ using that we have the joint prior distribution,

$$p(f(\cdot), \mathbf{f}|\mathbf{x}) = \mathcal{N}\left(\begin{bmatrix} f(\cdot) \\ \mathbf{f} \end{bmatrix} \middle| \begin{bmatrix} 0 \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} c(\cdot, \cdot) & c(\cdot, \mathbf{x}) \\ c(\mathbf{x}, \cdot) & c(\mathbf{x}, \mathbf{x}) \end{bmatrix}\right) \quad (3.6)$$

And the conditional 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 (3.5), $p(\mathbf{f}|\mathcal{D})$, i.e. the posterior distribution. Assuming iid data, i.e. $p(y_1, \dots, y_n|x_1, \dots, x_n, \mathbf{f}) = \prod_{i=1}^n p(y_i|x_i, \mathbf{f}_i)$ and that the likelihood is Gaussian with mean \mathbf{f} and variance $\sigma^2 I_n$.

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

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

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

Now we found that both term in the integral (3.5), and they are related such that it is possible to use (3.2) 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 (3.1) also is related in a simlar way, and we use (3.2), 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 - Empirical 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

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

<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!

3.2 Bayesian Neural Networks

Bohamiann and numpyro-BNN 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 first integral is intractable as θ can live in a highly dimensional space, second 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

Background: Monte Carlo approximation

Assuming we have a number of iid samples, $\theta^{(1)}, \dots, \theta^{(K)}$ 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

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

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 likelihood, $p(\mathcal{D}) = \int_{\theta} p(\mathcal{D}, \theta)$. This integral is often intractable since the space of θ typically is abnorous - so not even numerical 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

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

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 acceptance probability

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

3. Set the next sample

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

note that $\alpha(x_n, \hat{x})$ 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,

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

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 $x \neq y$,

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

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

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

and summing over x on both sides yields,

$$\sum_x p(x)p(x \rightarrow y) = p(y) \sum_x p(y \rightarrow x) \quad (3.7)$$

$$\implies p(y) = \sum_x p(x)p(x \rightarrow y) \quad (3.8)$$

similar conclusion will be obtained for $x = y$, all in all this reveals that $p(x)$ is in fact invariant for the chain $\{x_1, \dots, x_n\}$ 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 samples. We can do better by replacing the random walk with gradient-guided movements and interpretate 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 interpretation allows the system to take regions with high probability mass into account - this is obtained 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 interpreted 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

$$p(x, q) = \frac{1}{Z_H} \exp(-H(x, q)) \quad (3.9)$$

$$= \frac{1}{Z_E} \exp(-E(x)) \frac{1}{Z_K} \exp(-K(x)) \quad (3.10)$$

$$= p(x)p(q) \quad (3.11)$$

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 stochastic HMC

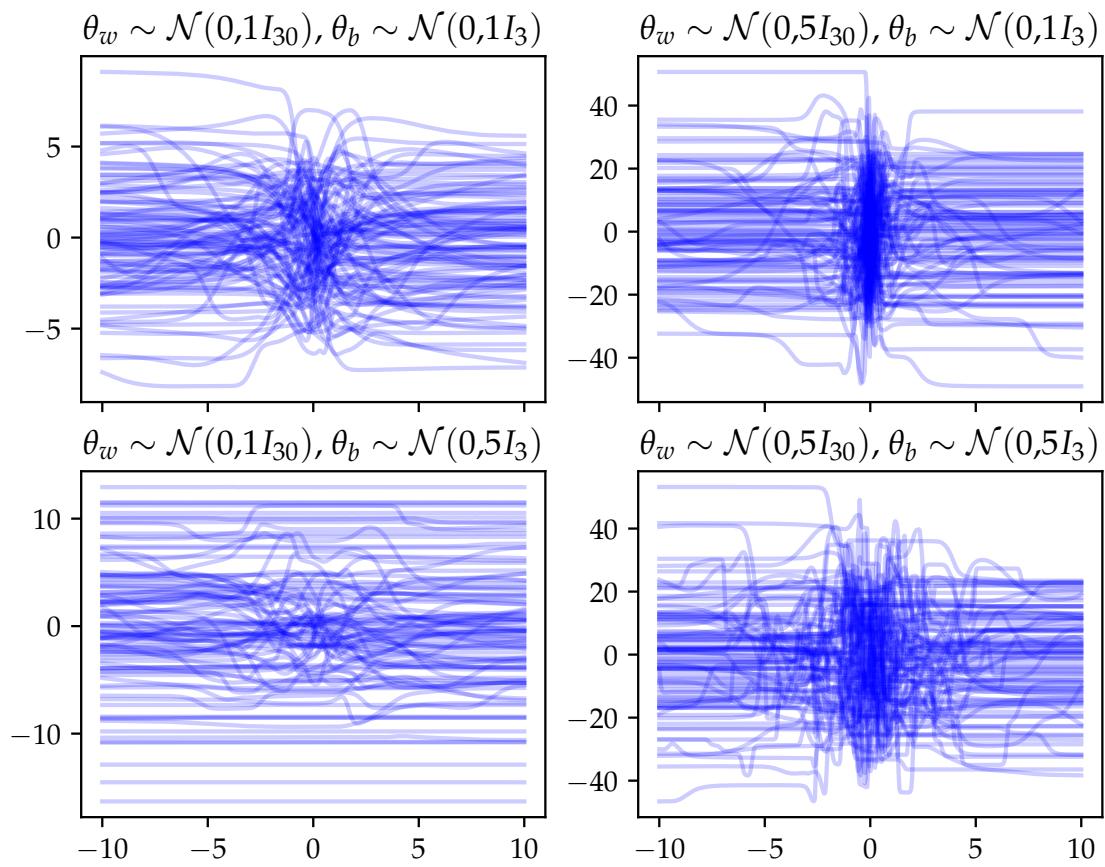


Figure 3.2: Prior samples

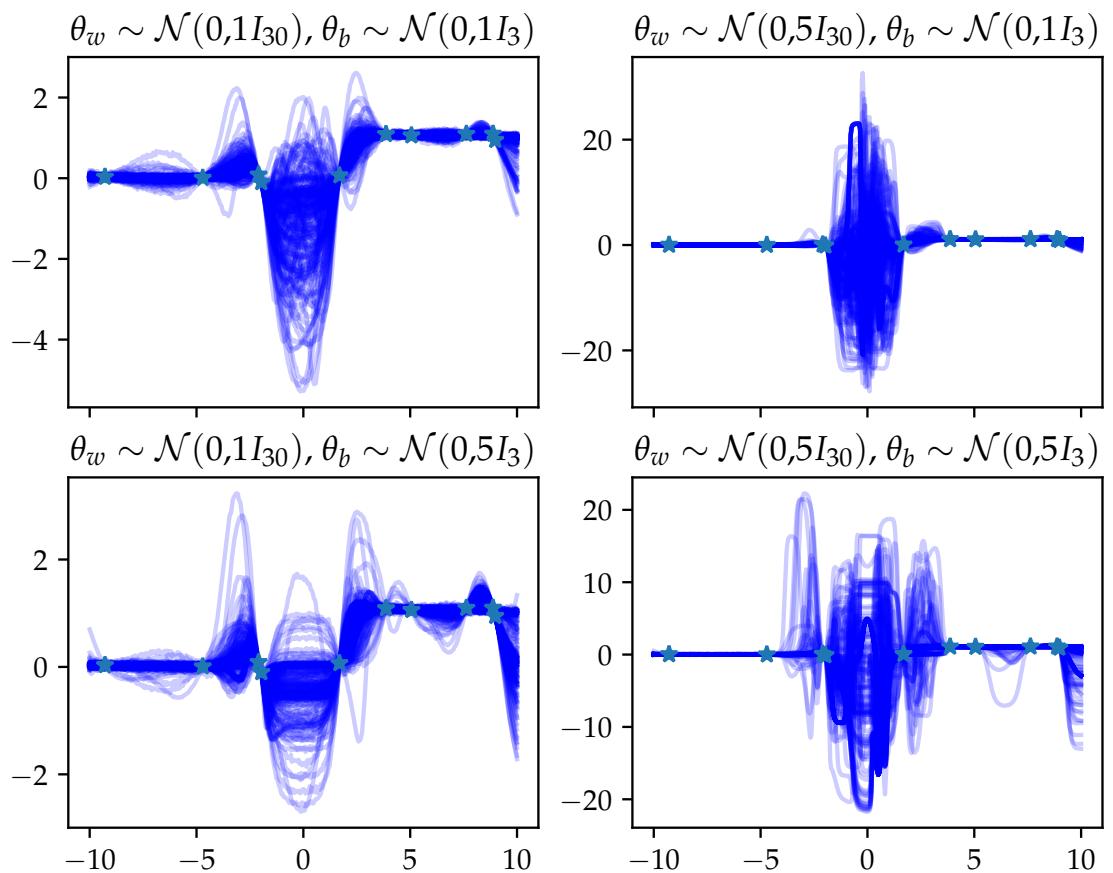


Figure 3.3: Posterior samples after observing

3.3 Regression analysis

4 Generative model surrogates

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

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 [5] and using kernel estimator [3] 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 ml. And they are good at dealing with incomplete data, i.e. missing values in the data set. [6]. However is it a hypothesis that it will allow for an expressive surrogate model, allowing for dealing 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 then from doing 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:

- Naive Gaussian mixture regression (Kernel estimator regression),
- Gaussian mixture regression,
- Sum-product networks.

4.1 "Bayesianizing" conditional distribution

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,

$$\begin{aligned}\hat{\mu} &:= \alpha \cdot \mu_{y|x} + (1 - \alpha) \cdot \mu_{prior} \\ \hat{\sigma}^2 &:= \alpha \cdot \sigma_{y|x}^2 + (1 - \alpha) \cdot \sigma_{prior}^2\end{aligned}$$

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 $\alpha_x \rightarrow 1$ for $N \cdot p(x) \cdot \Delta \rightarrow \infty$. 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,

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

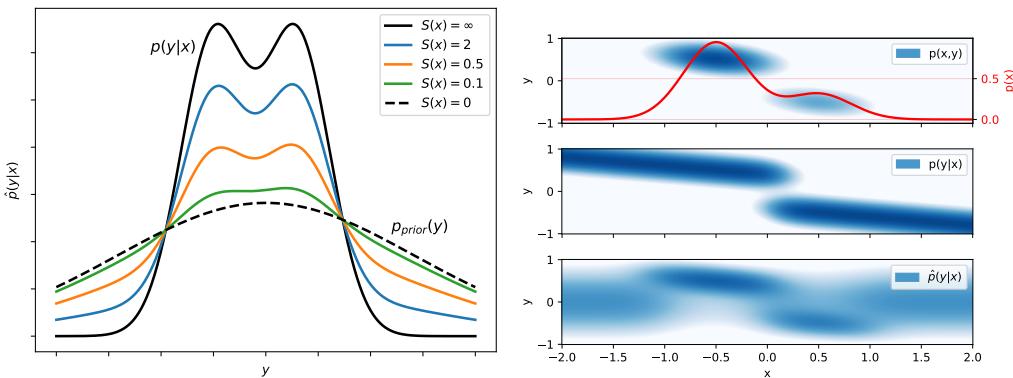


Figure 4.1: Left: Illustration of how the predictive distribution is manipulated according the the scaling function $S(x) := p(x) \cdot N \cdot \Delta$. Right: Illustration of why it makes sense to manipulate the predictive distribution $p(y|x)$, if there is a small amount of input data at a region,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

Placing a Gaussian around all data points.

4.3 Gaussian mixture regression

Taking a convex combination of a set of multivariate Gaussian distributions is a Gaussian mixture model

$$p(z) = \sum_{k=1}^K \pi_k \mathcal{N}(z|\mu_k, \Sigma_k)$$

Defining $z := (x, y)$ we can model our data, as a generative model $p(x, y)$,

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

This is trained using the EM algorithm. We will now show how the conditional is calculated exact.

4.3.1 GMR - prediction

We will now formulate the conditil distribution in terms of conditional and marginals of the individual mixture components. First of all the marginal distribution $p(x)$ of the mixture is given

as,

$$\begin{aligned}
p(x) &= \int p(x, y) dy \\
&= \sum_{k=1}^K \pi^{(k)} \int \mathcal{N}(x, y | \mu^{(k)}, \Sigma^{(k)}) dy \\
&= \sum_{k=1}^K \pi^{(k)} \mathcal{N}(x | \mu_x^{(k)}, \Sigma_{xx}^{(k)})
\end{aligned}$$

next the joint distribution can be decomposed with the probability chain rule,

$$\begin{aligned}
p(x, y) &= p(x)p(y|x) \\
\Rightarrow \quad \mathcal{N}(x, y | \mu, \Sigma) &= \mathcal{N}(x | \mu_x, \Sigma_{xx}) \mathcal{N}(y | \mu_{y|x}, \Sigma_{y|x})
\end{aligned}$$

And we can formulate the conditional in terms of individual multivariate Gaussians,

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

$$= \sum_{k=1}^K \frac{\pi^{(k)}}{p(x)} \mathcal{N}(x, y | \mu^{(k)}, \Sigma^{(k)}) \quad (4.2)$$

$$= \sum_{k=1}^K \frac{\pi^{(k)} \mathcal{N}(x | \mu_x^{(k)}, \Sigma_{xx}^{(k)})}{p(x)} \mathcal{N}(y | \mu_{y|x}^{(k)}, \Sigma_{y|x}^{(k)}) \quad (4.3)$$

$$= \sum_{k=1}^K \pi_{y|x}^{(k)} p(y|x, \mu_{y|x}^{(k)}, \Sigma_{y|x}^{(k)}) \quad (4.4)$$

where $\pi_{y|x}^{(k)} := \frac{\pi^{(k)} \mathcal{N}(x | \mu_x^{(k)}, \Sigma_{xx}^{(k)})}{\sum_{i=1}^K \mathcal{N}(x | \mu_x^{(i)}, \Sigma_{xx}^{(i)})}$. So we see that the conditional of a Gaussian mixture model is again a Gaussian mixture model.

Background: Conditional og multivariate Gaussian

The conditional distribution is defined as [7] ..?

$$p(y|x, \mu, \Sigma) = \mathcal{N}(y | \mu_{y|x}, \Sigma_{y|x}) \quad \mu = \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \quad \Sigma = \begin{bmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_{yy} \end{bmatrix}$$

where

$$\mu_{y|x} = \mu_y + \Sigma_{yx} \Sigma_{xx}^{-1} (x - \mu_x) \quad (4.5)$$

$$\Sigma_{y|x} = \Sigma_{yy} - \Sigma_{yx} \Sigma_{xx}^{-1} \Sigma_{xy} \quad (4.6)$$

4.4 Sum product networks

4.4.1 SPN is a mixture model

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:

Collection of sub-network S_z of S

```

function Process(node i,  $S_z$ )
  if  $i \in \text{Leaf}(S)$  then
    return:
  if  $i \in \text{Sum}(S)$  then
     $S_z = S_z.add(j \in ch(i))$                                  $\triangleright$  include one child of node  $i$ 
    return: Process( $j, S_z$ )
  if  $i \in \text{Prod}(S)$  then
     $S_z = S_z \cup \{j | j \in ch(i)\}$                            $\triangleright$  include all children of node  $i$ 
    for  $j \in ch(i)$  do
      return: Process( $j, S_z$ )
  return:  $S_z$ 
 $S_z = \text{Process}(\text{root}, \emptyset)$ 

```

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

$$Z = \prod_{i \in \text{Sum}(S)} |ch(i)|$$

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 we define a mixture coefficient, λ_z and component for each S_z as

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

where $p_i(x, y)$ is the leaf distribution at leaf node i parametrised 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 convenience 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 $q \in \text{Sum}(S)$ and a child $i \in ch(q)$, then the following relation holds,

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

4.4.2 Conditional of SPN

We will soon see how it is possible to write the conditional distribution as the mixture,

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

where $\Sigma(S)$ is the set of all sub-networks in the SPN, S - IT IS EXPONENTIALLY LARGE. And where $p_{z_y}(y)$ is defined through $p_z(x, y)$,

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

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 $\lambda_z = \prod_{(q,j) \in \mathcal{E}(z)} w_{q,j}$. First we calculate the marginal density, $p(x)$,

$$\begin{aligned} p(x) &= \int p(x, y) dy \\ &= \int \sum_{z \in \Sigma(S)} \lambda_z p_z(x, y) dy \\ &= \sum_{z \in \Sigma(S)} \lambda_z p_{z_x}(x) \int p_{z_y}(y) dy \\ &= \sum_{z \in \Sigma(S)} \lambda_z p_{z_x}(x) \end{aligned}$$

Now we are ready to calculate the conditional density,

$$\begin{aligned} p(y|x) &= \frac{p(x, y)}{p(x)} \\ &= \frac{\sum_{z \in \Sigma(S)} \lambda_z p_z(x, y)}{p(x)} \\ &= \sum_{z \in \Sigma(S)} \frac{\lambda_z p_{z_x}(x)}{p(x)} p_{z_y}(y) \\ &= \sum_{z \in \Sigma(S)} \frac{\lambda_z p_{z_x}(x)}{\sum_{z \in \Sigma(S)} \lambda_z p_{z_x}(x)} p_{z_y}(y) \\ &= \sum_{z \in \Sigma(S)} \gamma(x) p_{z_y}(y) \end{aligned}$$

So we defined $\gamma(x) = \frac{\lambda_z p_{z_x}(x)}{\sum_{z \in \Sigma(S)} \lambda_z p_{z_x}(x)}$ and this is very convinient, as we will see soon is equivalent to the derivative of the log-likelihood of the SPN, which is easily obtained by automatic differentiation.

4.4.3 calculation of responsibility, $\gamma(x)$

expectation maximization of a mixture model, is given by Bishop.. the responsibility of a datapoint to belong to one mixture component, is given by

$$\gamma(z_{nk}) = \frac{w_k p_j(x_n)}{\sum_i w_i p_i(x_n)}$$

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

$$L := \sum_n \log \sum_j w_j \exp \psi_j(x_n)$$

where we define $\psi_j(x_n) = \log p_j(x_n)$. Take the gradient

$$\frac{\partial L}{\partial \psi_j(x_n)} = \frac{w_k p_j(x_n)}{\sum_i w_i p_i(x_n)}$$

Note that the gradient easily can be found using autograd.

4.4.4 Mean and variance of conditional SPN

The mean of the conditional is just

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

and the variance is found using the second moment,

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

SPN is an exponential large mixture model, with linear inference - unlike GMM. !? [Write naive bayesian mixture model as a Sum Product Network](#)

This thesis will only work with RAT spn. Which takes combinations of each dimension. and...
sum nodes play a role of mixtures over their children distribution, similar to a classic mixture model
Product nodes on the other hand, are equivalent to factorizations over independent distributions as they are combining disjoint RVs.

SPNs can also be interpreted as deep feed forward 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

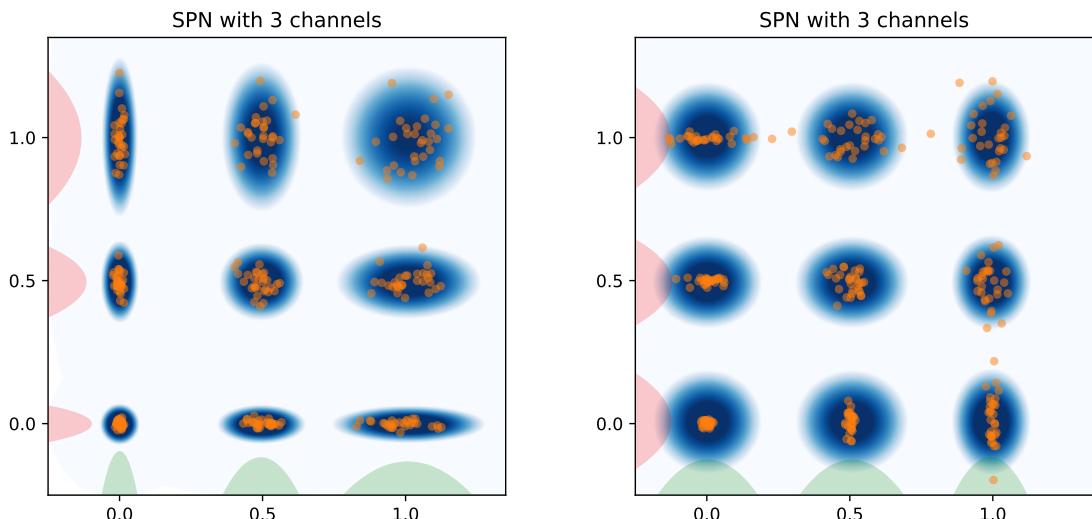


Figure 4.2: left: the data lies perfect for the SPN. Right: The data distribution is not suited for SPN

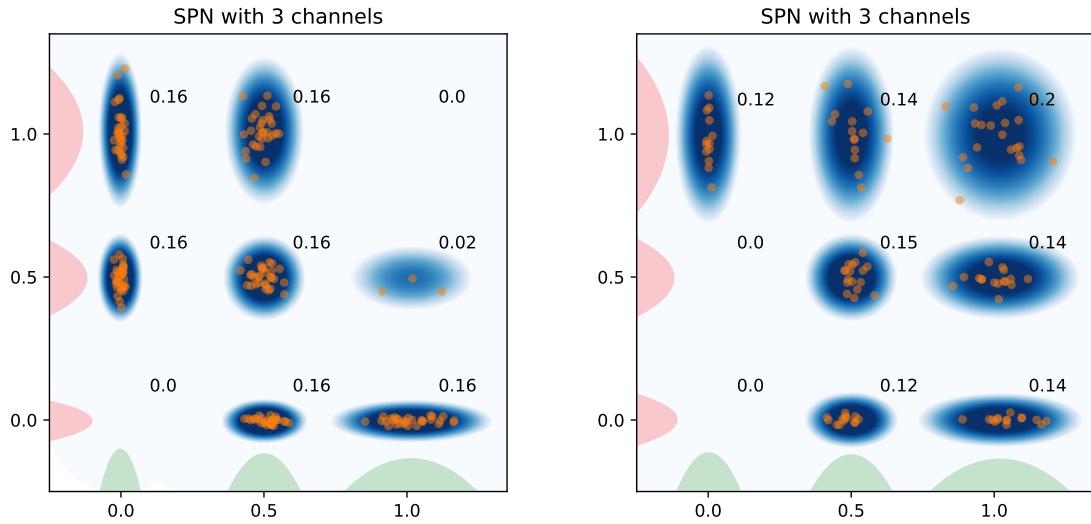
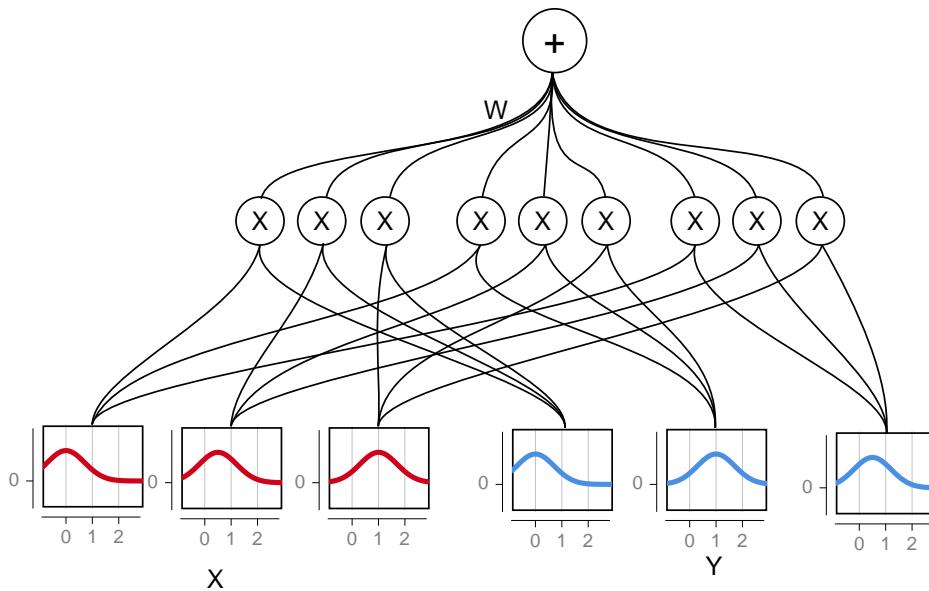


Figure 4.3: Example of how the weight of each mixture component is turned up and down according the amount of data observed.



4.4.5 SPN - prediction

Prior to the inference of the predictive distribution, we assume that the SPN, S , is trained, i.e. trained leaf distributions $p_j(\cdot)$ for all leaf nodes, $j \in \text{Leaf}(S) := \{j \in \mathcal{V}(S) | \text{pa}(j) = \emptyset\}$ and weights $w_{i,j}$ for the connections between every sum nodes $i \in \mathcal{S}$ and its children, $j \in \text{ch}(i)$.

The joint and the marginal distribution are evaluated in the following recursive way

Calculation of $p(x, y)$

Input: Fully trained SPN, with leaf distributions $p_i(\cdot)$ for $i \in \text{Leaf}(S)$ and weights $w_{i,j}$ for $(i, j) \in \{(i, j) | i \in \text{Sum}(S), j \in \text{ch}(i)\}$

function Eval(node i)

- if** $i \in \text{Leaf}(S)$ **then**
 - return:** $p_i(x, y)$ ▷ evaluate leaf distributions at point (x, y)
- if** $i \in \text{Sum}(S)$ **then**
 - return:** $\sum_{j \in \text{ch}(i)} w_{i,j} \text{Eval}(j)$
- if** $i \in \text{Prod}(S)$ **then**
 - return:** $\prod_{j \in \text{ch}(i)} \text{Eval}(j)$

$p(x) = \text{Eval}(\text{root})$

Calculation of $p(x)$

Input: Fully trained SPN, with leaf distributions $p_i(\cdot)$ for all leaves i and weights $w_{\cdot,\cdot}$

function Eval(node i)

- if** $i \in \text{Leaf}(S)$ **then**
 - if** node handle x **then**
 - return:** $p_i(x, y)$ ▷ evaluate leaf distributions at point (x, y)
 - else**
 - return:** 1 ▷ set node equal 1 at point (x, y)
- if** $i \in \text{Sum}(S)$ **then**
 - return:** $\sum_{j \in \text{ch}(i)} w_{i,j} \text{Eval}(j)$
- if** $i \in \text{Prod}(S)$ **then**
 - return:** $\prod_{j \in \text{ch}(i)} \text{Eval}(j)$

$p(x) = \text{Eval}(\text{root})$

So after doing two slightly different forward passes through the SPN, $p(x)$ and $p(x, y)$, using Bayes rule, we can combined the two queries into the conditional distribution:

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

The predictive distribution is found with a cost of just $O(E + E + 1) = O(E)$, where E is number of edges/connections in the SPN.

4.4.6 SPN - learning

It is not enouth to do predictive inference on a SPN, we also need to fit it on the data. It is possible interpret sum-product network as a large mixture model and therefore use expectation-maximization to train the model. We will introduce that idea now. The Paper [SPN_EM]... defines SPN as a mixture of all sub-networks of an SPN.

Definition 2 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:

Collection of sub-network S_z of S

```

function Process(node i,  $S_z$ )
  if  $i \in \text{Leaf}(S)$  then
    return:
  if  $i \in \text{Sum}(S)$  then
     $S_z = S_z.add(j \in ch(i))$                                  $\triangleright$  include one child of node  $i$ 
    return: Process( $j, S_z$ )
  if  $i \in \text{Prod}(S)$  then
     $S_z = S_z \cup \{j | j \in ch(i)\}$                              $\triangleright$  include all children of node  $i$ 
    for  $j \in ch(i)$  do
      return: Process( $j, S_z$ )
  return:  $S_z$ 
 $S_z = \text{Process}(\text{root}, \emptyset)$ 

```

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

$$Z = \prod_{i \in \text{Sum}(S)} |ch(i)|$$

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 we define a mixture coefficient, λ_z and component for each S_z as

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

where $p_i(x, y)$ is the leaf distribution at leaf node i parametrised 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 convenience 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 2 Consider a SPN, S , a sum node $q \in \text{Sum}(S)$ and a child $i \in ch(q)$, then the following relation holds,

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

4.5 Mixture model training

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

4.5.1 Expectation-maximization for mixture models

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

$$p(x) = \sum_{k=1}^K w_k p_k(x) \quad (4.7)$$

$$\iff p(x) = p_z(x), \quad z \sim \text{Cat}(w_1, \dots, w_K). \quad (4.8)$$

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

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

In 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 distribution $p(\mathcal{D}, z|\theta)$ is referred as the *complete-data* likelihood 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 uninformative/flat prior $p(\theta)$,

$$\begin{aligned} p(\theta|\mathcal{D}) &= \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \\ \Rightarrow \arg \max_{\theta} p(\theta|\mathcal{D}) &= \arg \max_{\theta} p(\mathcal{D}|\theta) \end{aligned}$$

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

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

Background: Expectation-maximization EM <based on [7]

Expectation maximization is a convenient method for finding ML (or MAP) estimate of a latent variable model. We consider a probabilistic model parametrised 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 likelihood,

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

maximizing the likelihood itself $p(\mathbf{X}|\theta)$ is assumed difficult but maximizing of the *complete-data* likelihood $p(\mathbf{X}, \mathbf{Z}|\theta)$ is much easier. The algorithm iterates over two steps: The expectation (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 likelihood (log likelihood function of θ), with respect to the only random quantity \mathbf{Z} , which is assumed to follow a distribution 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 expectations, 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 $p(\mathbf{X}|\theta) = \frac{p(\mathbf{X}, \mathbf{Z})}{p(\mathbf{Z})}$ 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 (4.7) is substituted by Gaussian pdfs, i.e. $p_k(x) = \mathcal{N}(x|\mu_k, \Sigma_k)$ and the density of a categorical distribution is $p(z) = \sum_{k=1}^K 1_{z=k} w_k = w_z$, combining the two we get the joint distribution,

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

Taking the log and defining $\theta = \{\mu_1, \Sigma_1, w_1, \dots, \mu_K, \Sigma_K, w_K\}$, 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 likelihood 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 respect to unnecessary variables

the last equation holds since taking expectation over a function of a random variable x with respect to a that random variable and more random variables, x, y , is equivalent to the expectation 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 simplification we will denote, $\gamma^{(t)}(z_i) := p(z_i|x_i, \theta^{(t)})$, interpreted as the probability of data-point x_i to belong to class z_i . Bishop [7] 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 $\theta = \{\mu_1, \Sigma_1, w_1, \dots, \mu_K, \Sigma_K, w_K\}$,

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

$Q(\cdot, \theta^{(t)})$ is a concave function - the Gaussian is log-concave and a sum of concave functions is also concave - so it is sufficient and necessary 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 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.

5.1.1 uncertainty quantification

The probabilistic model is given as

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

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 density of predictive distribution is evaluated in the corresponding 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 predictive likelihood!!**

5.1.2 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, $MAE := \frac{1}{n} \sum_{i=1}^n |\mu_{\mathcal{D}}(x_i) - y_i|$

5.1.3 regression benchmark

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

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

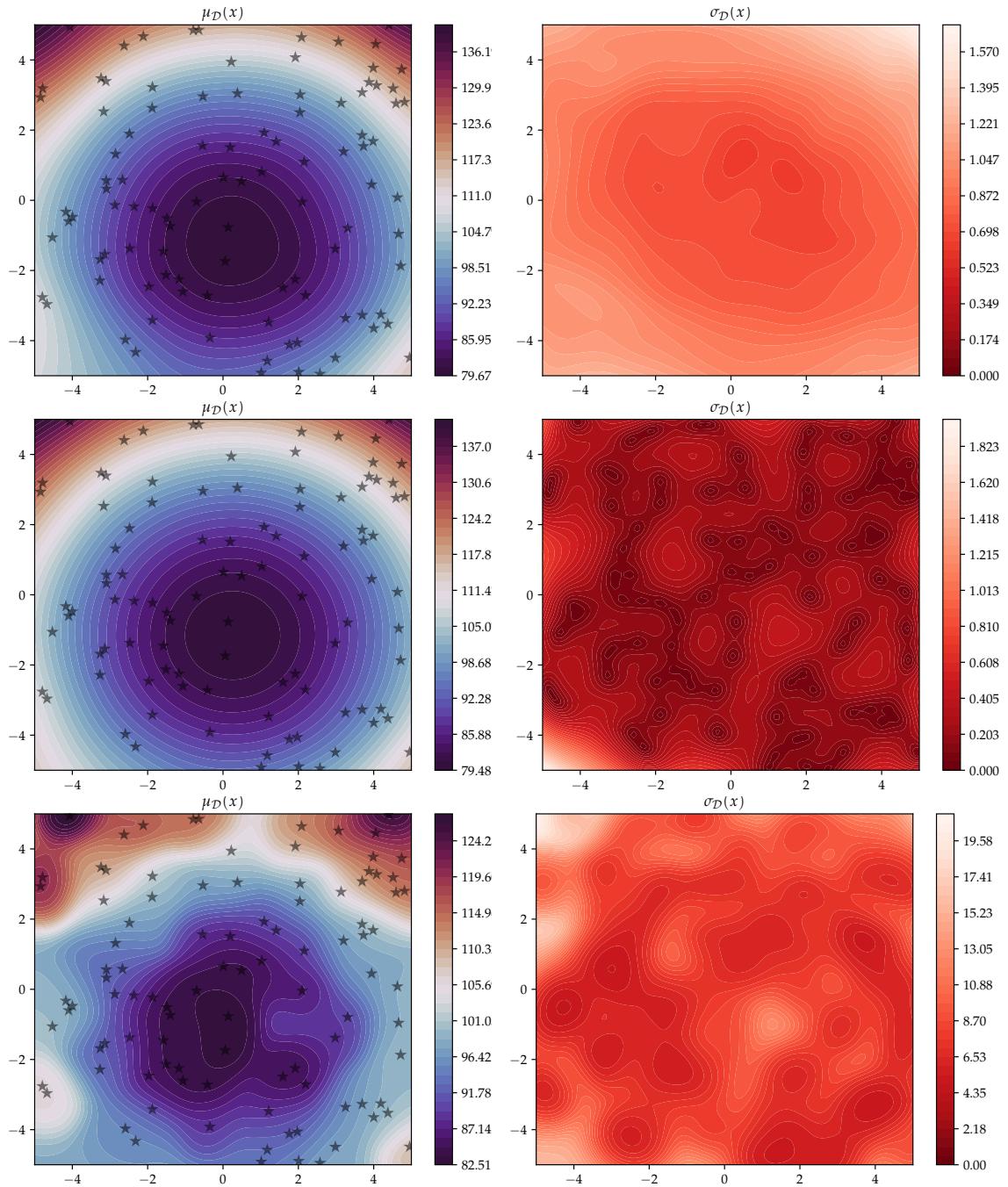
where $\bar{\mathbf{y}} = \frac{1}{n} \sum_{i=1}^n \mathbf{y}_i$ and $\bar{\sigma}^2(\mathbf{y}) = \frac{1}{n-1} \sum_{i=1}^n (\mathbf{y}_i - \bar{\mathbf{y}})^2$ this is however not a good model for Bayesian optimization, as it will not provide a new candidate point, as all points are equally good.

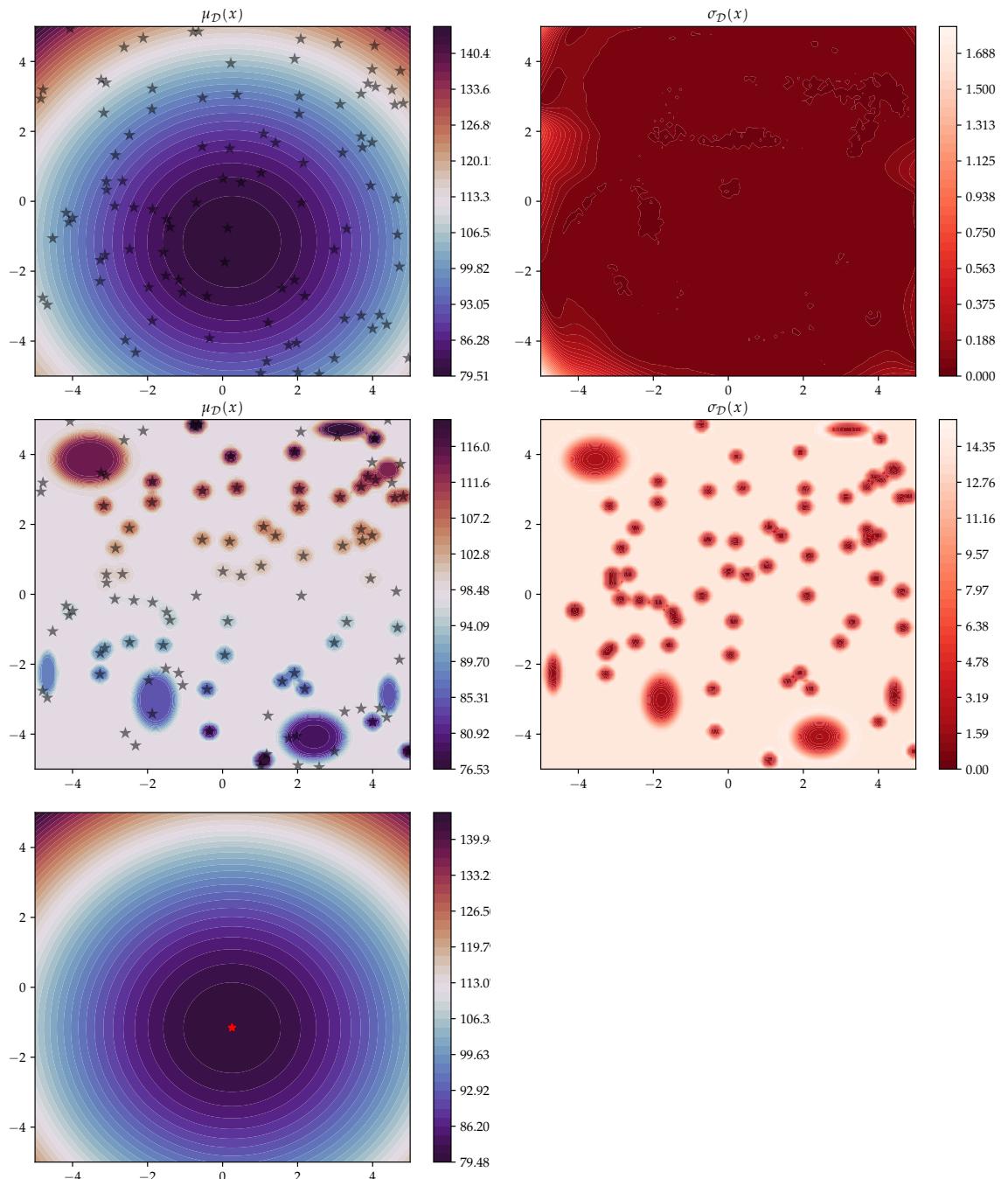
5.2 Regression analysis of GP, BOHAMIANN and NumpyNN 1D

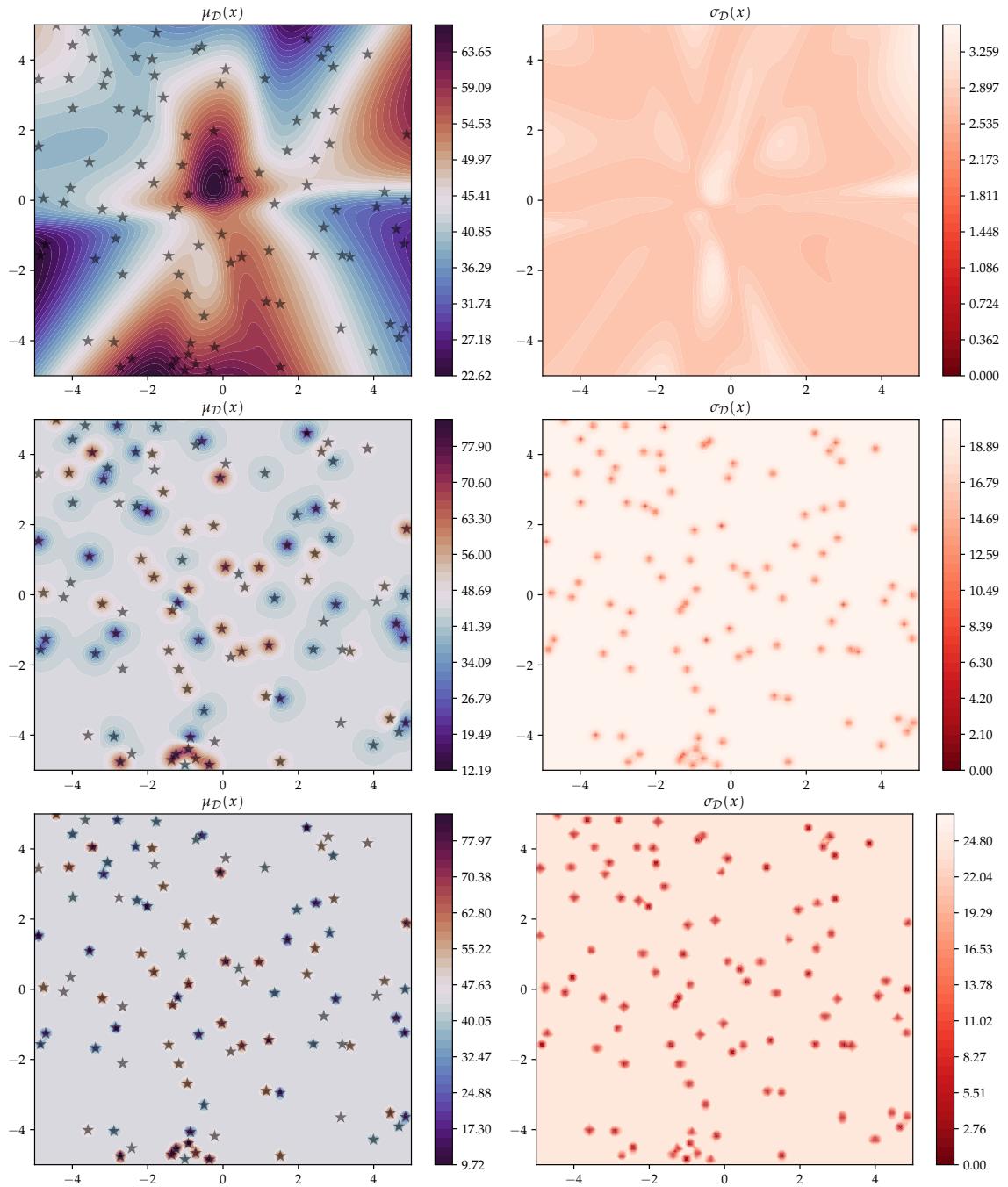
5.3 Regression analysis of GP, BOHAMIANN and NumpyNN 2D

5.4 Mixture regression on simple functions

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







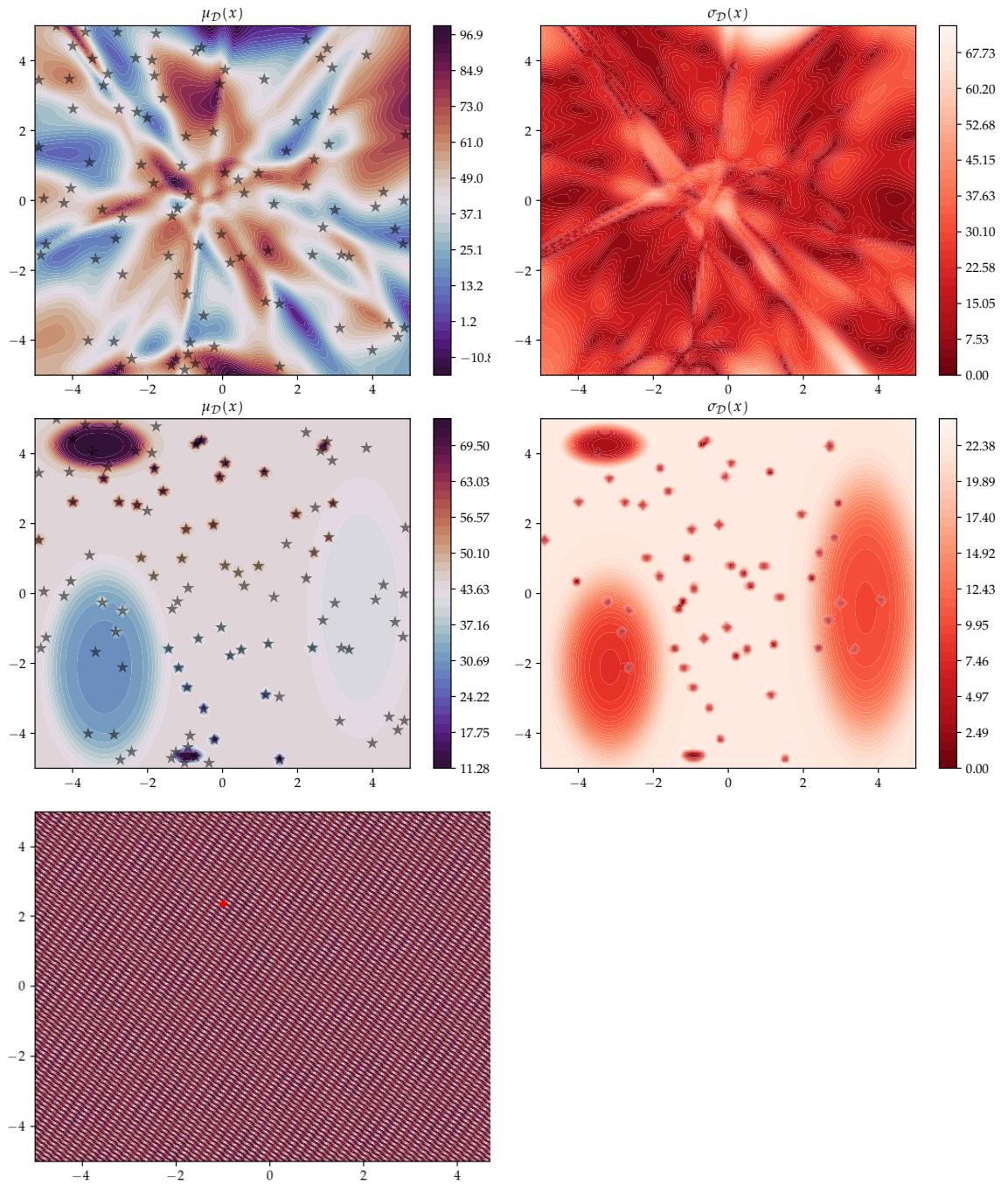


Figure 5.1: 2D Regression plot for a few of the problems, with the correct

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 another 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

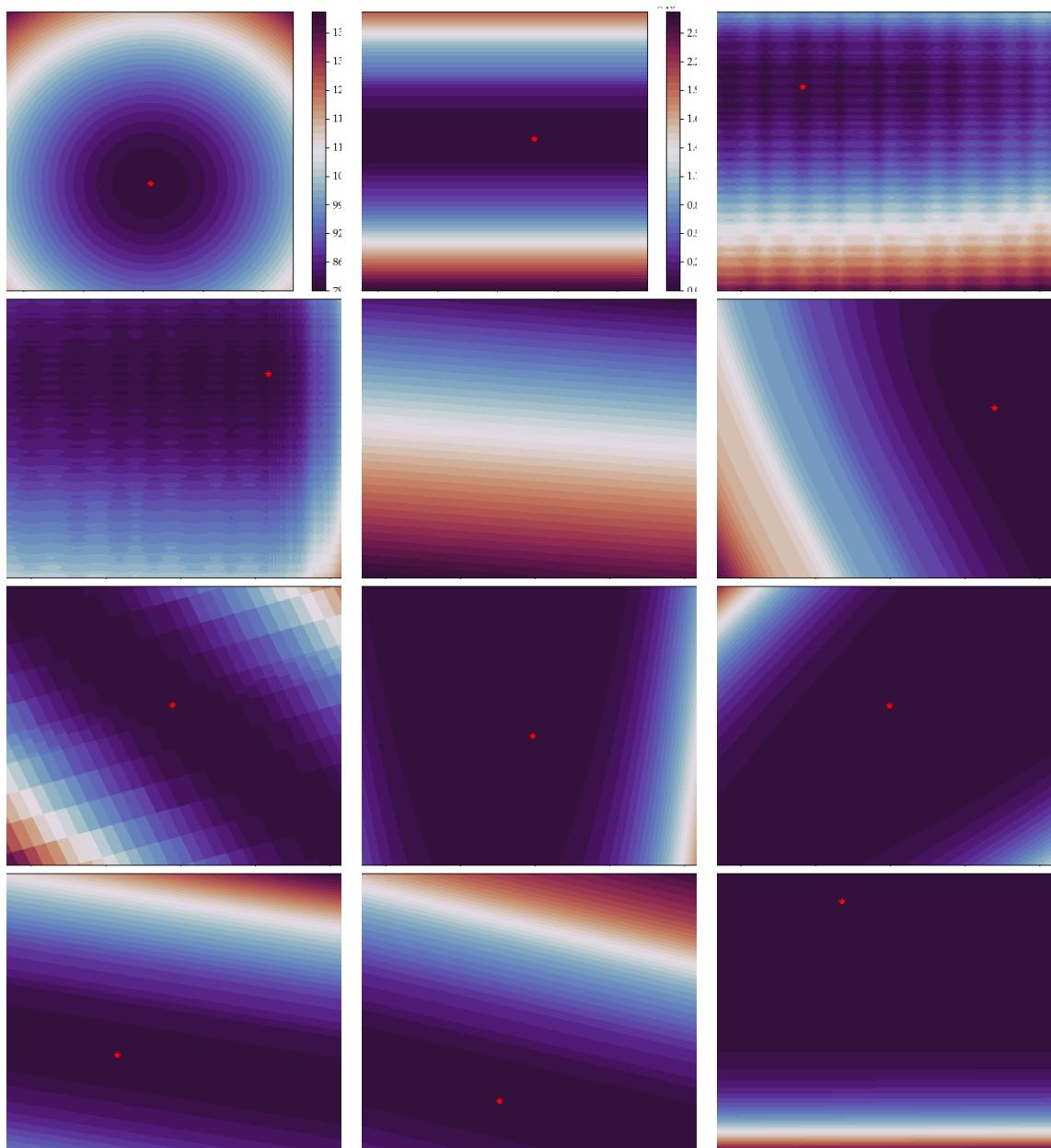


Figure 5.10: Test functions f1 to f12

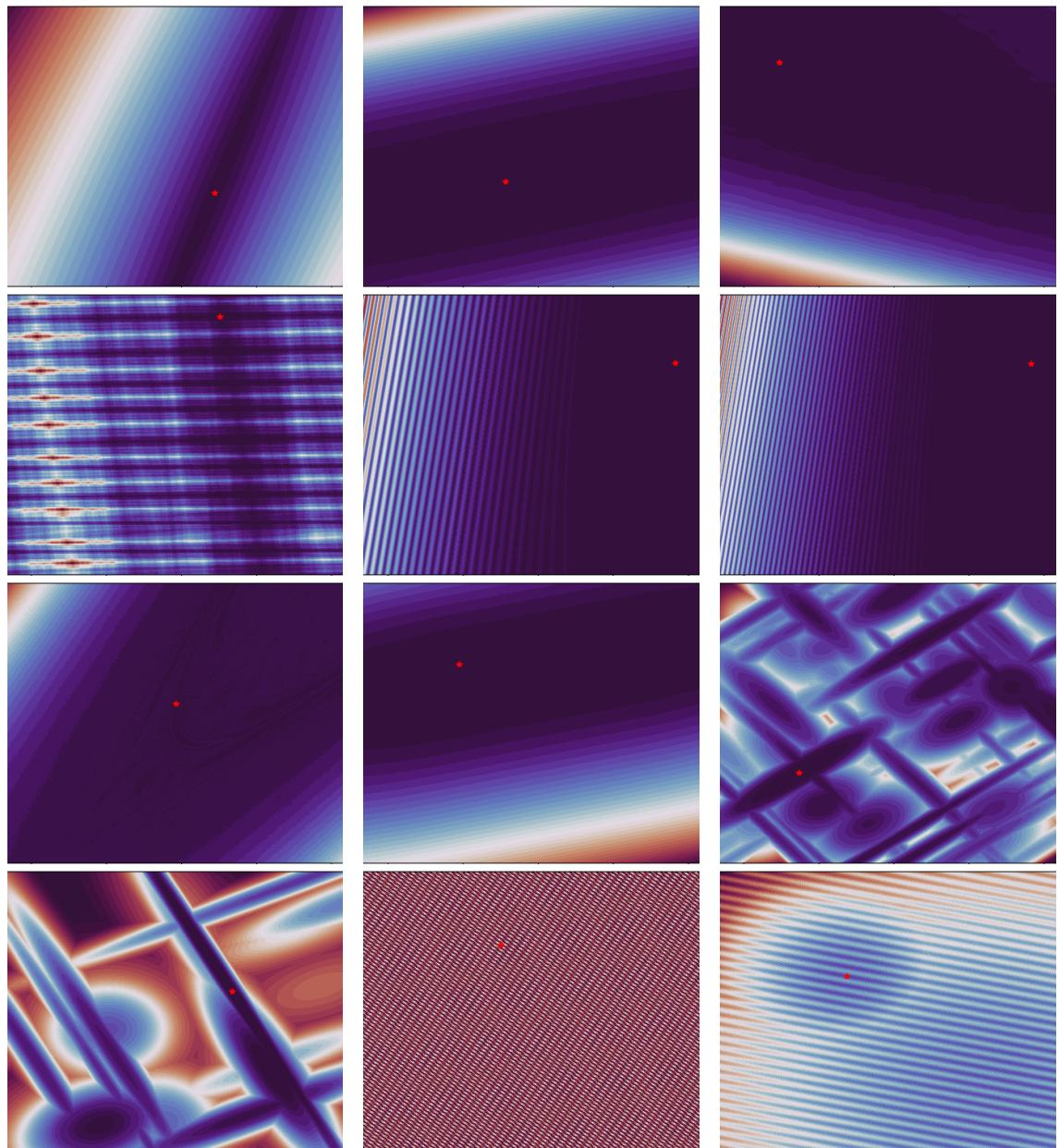


Figure 5.11: Test functions f13-24

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] 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.
- [3] 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>.
- [4] Roman Garnett. *Bayesian Optimization*. in preparation. Cambridge University Press, 2022.
- [5] Christopher M Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.
- [6] 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>.
- [7] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.
- [8] Jasper Snoek et al. *Scalable Bayesian Optimization Using Deep Neural Networks*. 2015. arXiv: 1502.05700 [stat.ML].
- [9] Jost Tobias Springenberg et al. “Bayesian Optimization with Robust Bayesian Neural Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016. url: <https://proceedings.neurips.cc/paper/2016/file/a96d3afec184766bfeca7a9f989fc7e7-Paper.pdf>.

A Implementation

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 [2][8][9] (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