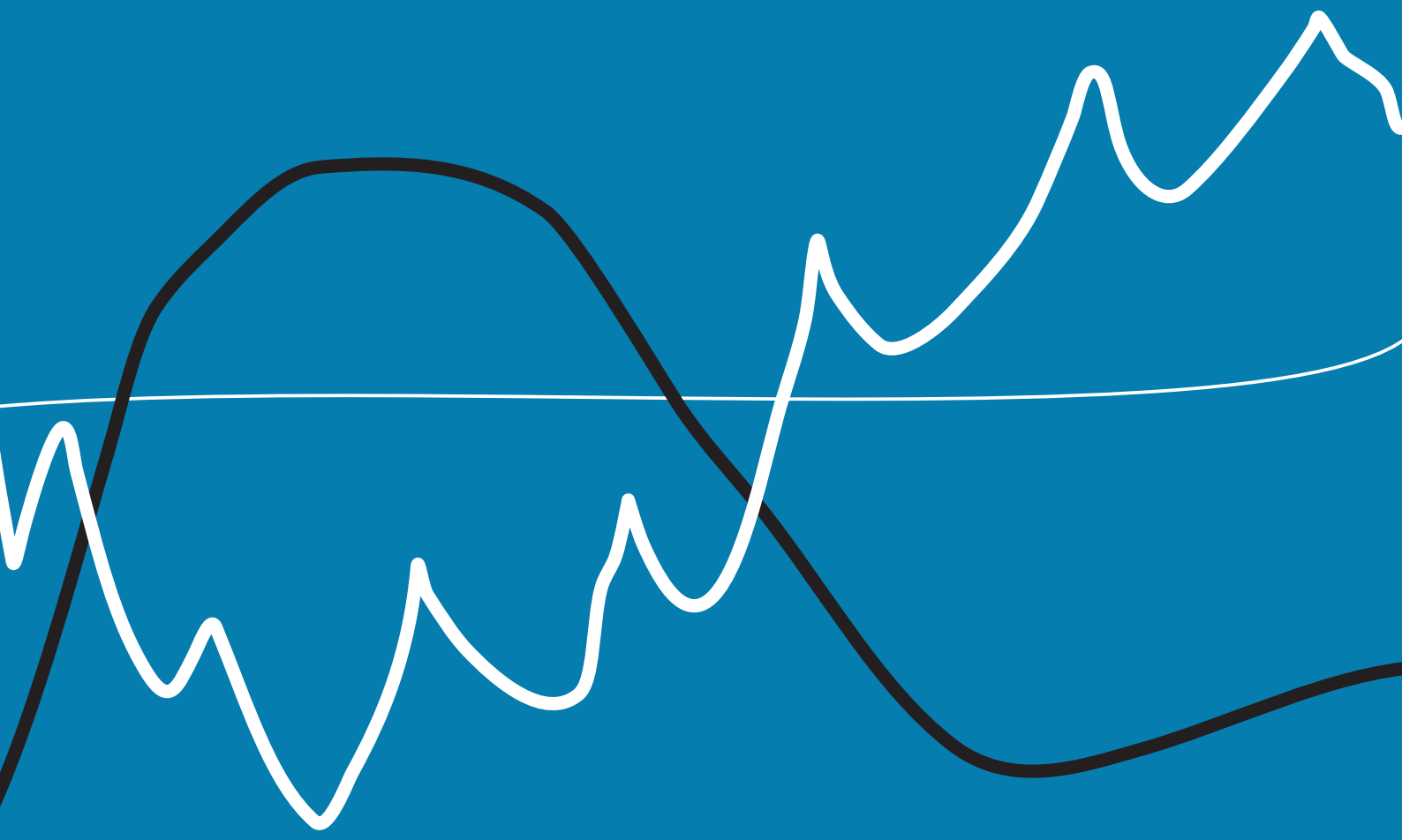
DTU Compute
Section of Cognitive Systems

Investigation of Deep Probibalistic Surrogate-models in Bayesian Optimization
in Bayesian Optimization

Master Thesis



Investigation of Deep Probibalistic Surrogate-models in Bayesian Optimization
in Bayesian Optimization

Master Thesis
June, 2022

By
Simon Saugmandsgaard Kruse

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

Optimization plays an important part in our everyday life, science development, and product design. What different transportation should you choose to get fast from A to B, what songs should land on your playlist, what is the optimal bridge construction. Mathematical optimization problems are all problems on the form,

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

where $f : \mathcal{X} \to \mathcal{R}$ is a functional. I.e if it is possible to set up an objective function. e.g. what is the cost of the bridge given a specific design , or how pleasant you think some music, and some constraints such that you keep in the domain of interest. Evaluation of the objective function can be cheap e.g. if it just requires summing and multiplieing numbers or highly expensive if it involves human rating or large simulation and physical experiments. Bayesian optimization is a preferred <ref> framework for the expensive objective functions. And is also referred to as *sample efficient* optimization.

Bayesian optimization is a probabilistic surrogate-based optimization: Assuming some samples from a highly expensive objective 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 regression model. The most common choice is a Gaussian Process, as it encapsel the uncertainty very well, but also because its inference procedure (computing answers to probability quries like $p(y|x)$) is exact.

Even though GP has proven good for many cases, there will be problems where the assumptions do not hold. In the thesis ... PhD from 2020 he tries out using Bayesian neural networks as surrogate models. This thesis will investigate GP and Bayesian neural networks and try out mixture regression, which might be a even better surrogate model.

<Make a figure of how the parts are all connected>

1.1 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** $P(Y = y^2|X = x)$

2 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 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 [**bayesoptbook**] and presented here as Algorithm 1.

Algorithm 1 Sequencial Optimization [**bayesoptbook**]

Input: Initial dataset $\mathcal{D}$ ▷ can be empty
while Temination is not reached **do**
  $x \leftarrow$ policy($\mathcal{D}$) ▷ select next observation location
  $y \leftarrow$ 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 probibalitic, 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 to the available data $\mathcal{D}$. Finaly, a stopping criterion decides whether to continue or terminate.

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

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

Example: Grid search
In grid search values along each dimention in $\mathcal{X}$ is seleced and combined with each other, which thereby defines a grid in the parameter space. All points are ordered and systematicly selected. In the content of algorhtim 1 we define the grid search policy as

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

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

Example: Random search
In random search a uniform distribtuion is layed over the domain space $\mathcal{X}$ and a random point is selected from the distribtuion.

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

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 descenting directiong. 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
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 the data.

2.1 When to use Bayesian optmization

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

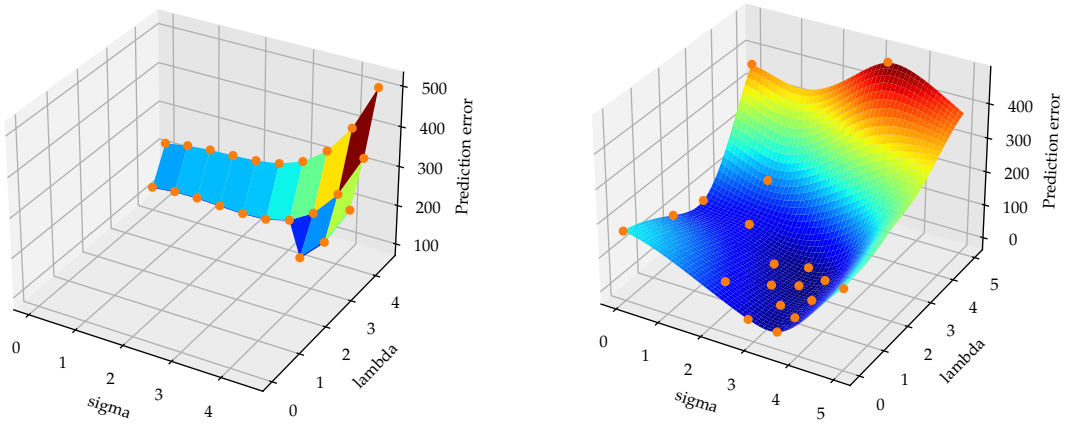


Figure 2.1: Hyper parameter tuning of a model $M(\lambda, \sigma)$, 23 evaluation in grid search vs 23 evaluations using Bayesian optimization

additionally, Bayesian Optimization allow for observation noise,

2.2 Observation model

Many optimization algorhtims 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 $y = f(x)$

3 Bayesian Optimization

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,

$$p(y|x, \mathcal{D}) \tag{3.1}$$

Before bringing the parameters/unknown quantities into play, we can ask: What quantities can we play with? This question is answered in two different ways in Gaussian process regression and Bayesian Neural network regression.

3.1 Discriminative model as surrogate model

Discriminative modeling

$$p(y|x)$$

3.2 Using a generative model as surrogate model

Given a generative model over x and y paramitised with θ, we are dealing with the joint distribution

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

and we are interested in the condtional distribution of y given x,

$$p(y|x, \theta_{y|x})$$

where we have put subscript on θ in order to jump up a level of abstraction since, in fact there is just a mapping between them $\theta_{y|x} := g(\theta, y, x)$

$$\begin{aligned} p(y|x, \mathcal{D}) &= \int p(y|x, \theta_{y|x}) p(\theta_{y|x}|\mathcal{D}) d\theta_{y|x} \\ &= p(y|x, \hat{\theta}_{y|x}) \end{aligned}$$

Where the last equation holds as we assume that $p(\theta_{y|x}|\mathcal{D})$ is a delta function i.e. a point estimate with value $\hat{\theta}_{y|x}$. In the case of our Gaussian mixture model, we obtain a point estiamte from the EM algorithm for the variance $\Sigma_{y|k}$, mean value $\mu_{y|k}$ and proportion $\pi_{y|k}$ for each component $k = 1, 2, \ldots, K$

$$\hat{\theta}_{y|x} = (\hat{\Sigma}_{y|k}, \hat{\mu}_{y|k}, \hat{\pi}_{y|k})_{k=1}^{K}$$

However, we are not satisfied with the variance estimate for the regression, as it is way too small for areas with no observed data. It is therefore necessary to manipulate the variance estimate accoring to that observation. We multiply the variance obtained using expectation-maximization on the joint distribution with the inverse of the probability of the data x, and control that the scaling factor is not going wild!

$$\hat{\Sigma}_{y|k} = \Sigma_{y|k}^{GMM} \frac{1}{\max(p(x), 0.01)}$$

In a way this is a manipulation in a Bayesian spirit, as we let prior and subjective knowledge influence the varience prediction.

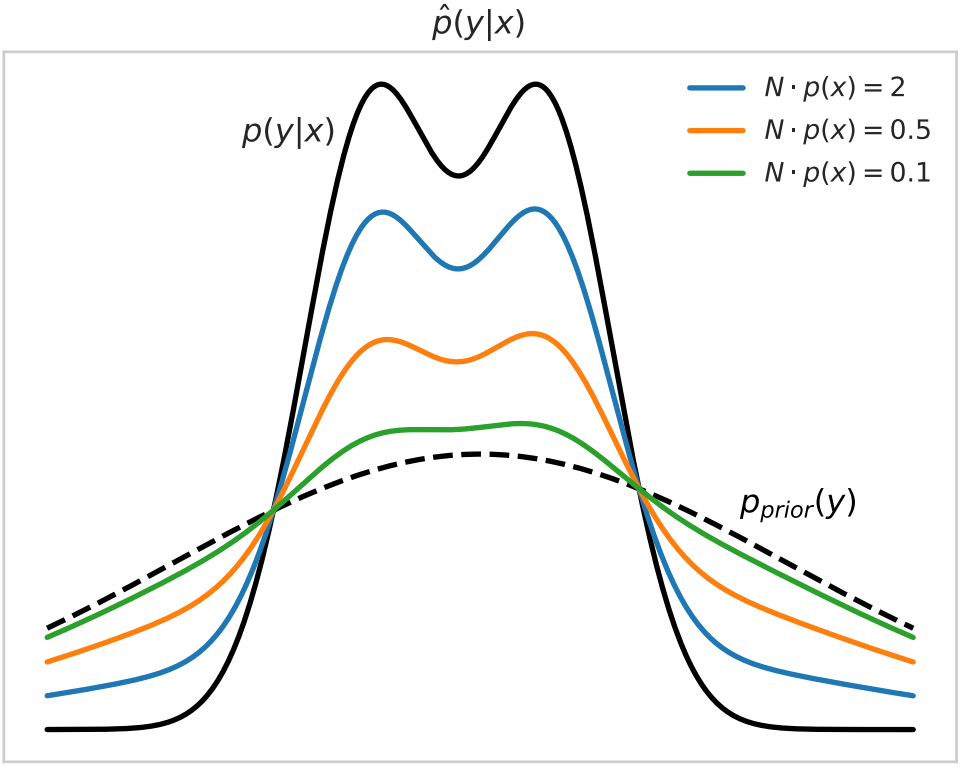
3.2.1 Inclusion of prior distribution

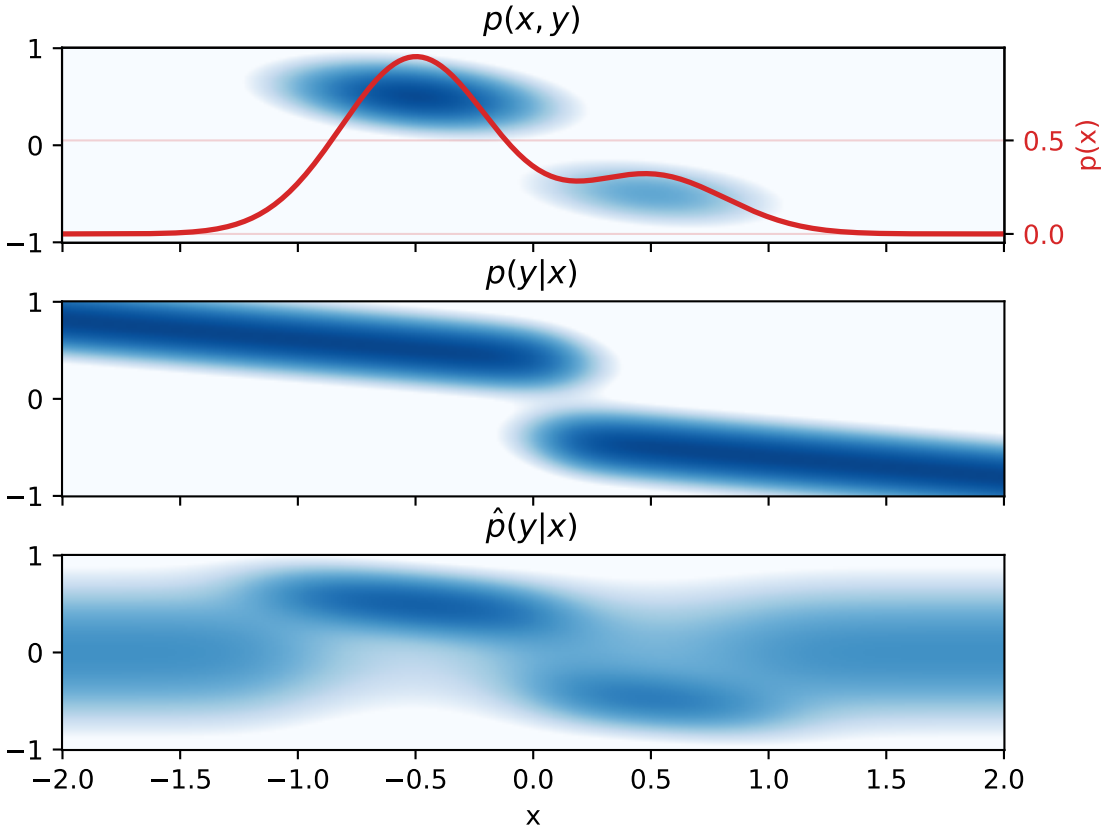
The conditional distribution $p(y|x)$ is without a prior. Therefore its beliefs is not really ok.. In the flavour of a Poission point process, we introduce the prior..

$$p_{bayes}(y|x) = \frac{p(x) \cdot N \cdot p(y|x) + \lambda p_{prior}(y)}{Z}$$

where the normalization constant Z makes sure the manipulated density integrates to 1, i.e.

$$Z = \int p(x) \cdot N \cdot p(y|x) + \lambda p_{prior}(y) dy = p(x) \cdot N + \lambda$$





mean of $p_{bayes}(y|x)$

$$\begin{aligned} E_{p_{bayes}(y|x)}[y] &= \int y \cdot \frac{f(n, p(x)) \cdot p(y|x) + \lambda p_{prior}(y)}{Z} dy \\ &= \left(p(x) \cdot N \cdot E_{p(y|x)}[y] + \lambda \cdot E_{p_{prior}(y)}[y]\right) \frac{1}{Z} \\ &= \frac{p(x) \cdot N \cdot E_{p(y|x)}[y]}{p(x) \cdot N + \lambda} \end{aligned}$$

variance of $p_{bayes}(y|x)$
We first calculate the second moment,

$$\begin{aligned} E_{p_{bayes}(y|x)}[y^2] &= \int y^2 \cdot \frac{p(x) \cdot N \cdot p(y|x) + \lambda p_{prior}(y)}{Z} dy \\ &= (p(x) \cdot N \cdot E_{p(y|x)}[y^2] + \lambda E_{p_{prior}(y)}[y^2]) \frac{1}{Z} \\ &= \frac{p(x) \cdot N \cdot (Var_{p(y|x)}[y] + E_{p(y|x)}[y]^2) + \lambda Var_{p_{prior}(y)}[y]}{p(x) \cdot N + \lambda} \end{aligned}$$

So the variance is calculated as,

$$V_{p_{bayes}(y|x)}[y] = E_{p_{bayes}(y|x)}[y^2] - E_{p(x,y)}[y]^2$$

implementation
It is not necessary to calculate the conditional distribution, since, using Bayes rule

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

so gives

$$p_{bayes}(y|x) = \frac{p(x) \cdot N \cdot p(y|x) + \lambda p_{prior}(y)}{Z} = \frac{N \cdot p(x,y) + \lambda p_{prior}(y)}{Z}$$

$$E_{p(x,y)}[y] = \int_y \int_x y p(x,y) dx dy = \int_y y p(y) dy = E_{p(y)}[y]$$

3.3 Acquisition function

A popular choice of acquisition function is expected improvement:

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

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

4 Bayesian regression models - probabilistic 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 probility of prediction for a arbitrary point x given datapoints $\mathcal{D} = \{x_1, y_1, \ldots, x_n, y_n\}$,

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

The surrogate model of choise 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.

4.1 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 $p(x, y)$, now, since the conditonal of a Gaussain mixture again is Gaussain mixture - i.e. closed form expression, we can exactly calculate $p(y|x) = GMM_{y|x}$

Assuming iid data the likelihood is given as

$$p(\mathcal{D}|\mu_1, \ldots, \mu_K, \Sigma_1, \ldots, \Sigma_K, \pi_1, \ldots, \pi_K) = \prod_{i=1}^{n} \sum_{k=1}^{K} \pi_k \mathcal{N}(z_i|\mu_k, \Sigma_k)$$

And the log likelihood,

$$\log p(\mathcal{D}|..) = \sum_{i=1}^{n} \log \sum_{k=1}^{K} \pi_k \mathcal{N}(z_i|\mu_k, \Sigma_k)$$

4.2 Mixture regression in a Bayesian setting

As seen in examples. The uncertainty of conditional distribution is way too certain in areas with no data points, therefore we need to enhance the model with some bayesian flavour.

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

Expetation-maximization algorithm

A way to find local maxima in the likehood function is using the EM algorithm.

If we define a latent/hiddem random variabel $Z_i \in \{1, \ldots, K\}$ for each data point, then the likelihood function becomes,

$$L(\theta|\mathcal{D}, Z) = \prod_{i=1}^{n} \sum_{k=1}^{K} 1(Z_i = k)\pi_k \mathcal{N}(z_i|\mu_k, \Sigma_k)$$

Now the expectation wrt. the current value $p(Z|\mathcal{D}, \theta^k)$ is given as

$$Q(\theta|\theta^k) = \mathcal{E}_{p(Z|\mathcal{D},\theta^k)} = L(\theta|\mathcal{D}, Z)$$

And then update the next parameter estimate with

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

This is repeated untill convergence.

4.3 Sum product networks

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

Mixture model

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 $i \in \mathcal{L}eaf(S)$ **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 exponential 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,*

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

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 $p_z(x, y)$,

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

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 $\lambda_z = \prod_{(q,j)\in\mathcal{E}(z)} w_{q,j}$. 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 equavalent to the derivative of the log-likehood of the SPN, which is easily obtained by automatic differentiation.

calculation of $\gamma(x)$

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 likehood,

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

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.

Mean and variance of $p(y|x)$

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,

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

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

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

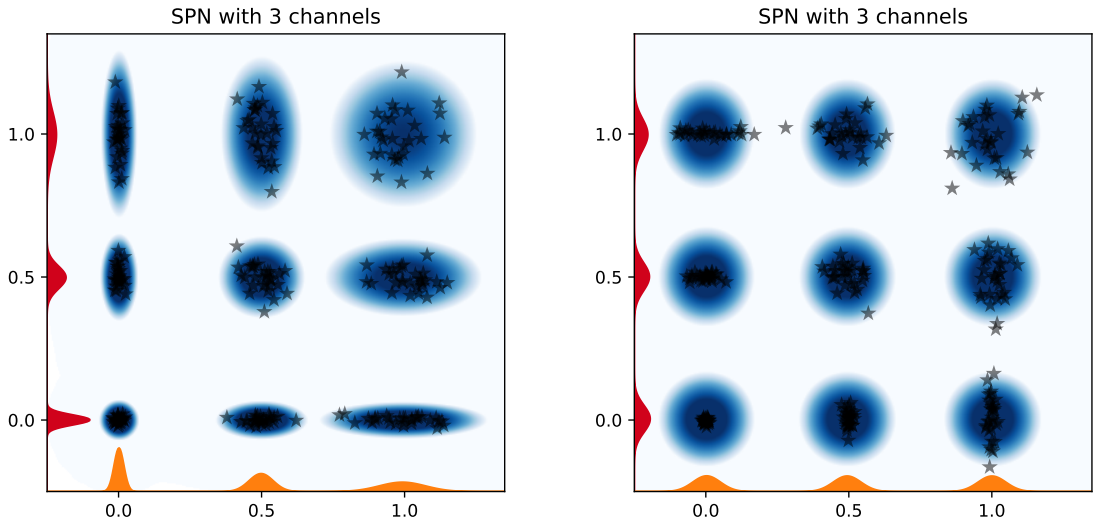


Figure 4.1: SPN

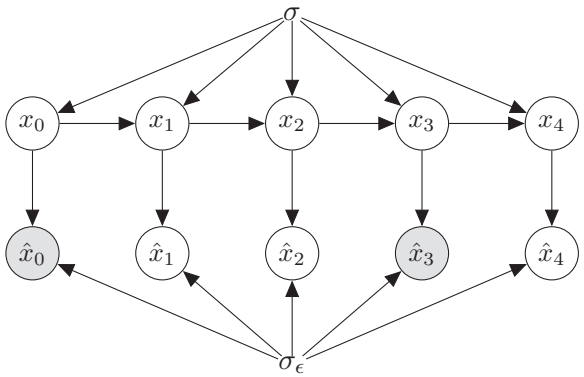
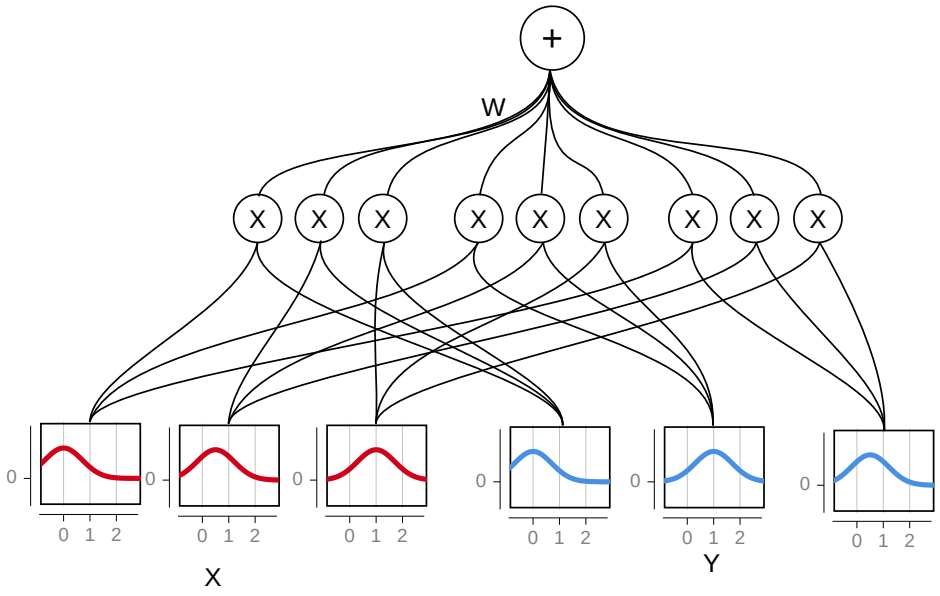


Figure 4.2: Model including wifi information



5 Inference: Prediction and learning

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 $E_{p(z|\theta^{(k)})}[z|\theta]$.

5.1 Exact and approximate inference

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 infered 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 stochatic HMC	
Numpyro BNN	No U-Turn Sampler	

Table 5.1: 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.

5.2 SPN

Sum-product networks are generative models, i.e. statistical models of the joint distribution $p(x, y)$. We need, however, a disciminative model for regression, i.e. a model of the conditional distribution $p(y|x)$. SPNs allow for exact inference of the joint distribution and any marginalized distribution. These combined queries is sufficient for the exact predictive posterior.

5.2.1 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, $j \in \mathcal{L}eaf(S) := \{j \in \mathcal{V}(S) | pa(j) = \emptyset\}$ and weights $w_{i,j}$ for the connections between every sum nodes $i \in \mathcal{S}$ and its children, $j \in ch(i)$.

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 $i \in \mathcal{L}eaf(S)$ and weigts $w_{i,j}$ for $(i, j) \in \{(i, j) | i \in \mathcal{S}um(S), j \in ch(i)\}$

function Eval(node i)

  if $i \in \mathcal{L}eaf(S)$ **then**

    return: $p_i(x, y)$ ▷ evaluate leaf distributions at point (x, y)

  if $i \in \mathcal{S}um(S)$ **then**

    return: $\sum_{j \in ch(i)} w_{i,j}$Eval$(j)$

  if $i \in \mathcal{P}rod(S)$ **then**

    return: $\prod_{j \in ch(i)}$ Eval(j)

$p(x) =$ Eval(root)

Calculation of $p(x)$

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

function Eval(node i)

  if $i \in \mathcal{L}eaf(S)$ **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 $i \in \mathcal{S}um(S)$ **then**

    return: $\sum_{j \in ch(i)} w_{i,j}$Eval$(j)$

  if $i \in \mathcal{P}rod(S)$ **then**

    return: $\prod_{j \in ch(i)}$ Eval(j)

$p(x) =$ Eval(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.

5.2.2 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 $i \in \mathcal{L}eaf(S)$ **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 exponential 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 2 *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,*

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

5.3 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,

$$p(x) = \sum_{k=1}^{K} w_k p_k(x)$$
$$\iff \qquad p(x) = p_z(x), \qquad z \sim Cat(w_i, \dots, w_K).$$

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 Bishops book>

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**
  $Q(\cdot, \theta^{(t)}) \leftarrow E_{p(\mathbf{Z}|\mathcal{D},\theta^{(t)})}[\log p(\mathcal{D}, \mathbf{Z}|\theta)]$ ▷ 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.

then we can marginalise over z in order to recover $p(x)$,

$$p(\mathbf{x}|\pi) = \sum_{z=1}^{Z} p(z, \mathbf{x}|\pi)$$

assuming iid data, then the complete likehood can be decompose as the product

$$p(z, \mathbf{x}|\pi) = \prod_{i=1}^{n} p(z, \mathbf{x}_i|\pi)$$

We wil for convinience transform the likelihood using a log transform, as it will not influence the maximum.

$$\log p(\mathbf{x}|\pi) = \sum_{z=1}^{Z} \sum_{i=1}^{n} \log p(z, \mathbf{x}_i|\pi)$$

<EM for SPN > <EM for GMM >

Her er en titel

- Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin euismod finibus enim vel tincidunt. Aliquam a placerat risus. Donec lobortis consequat massa et rhoncus. Cras a quam nec ante porta consequat at in nulla. Praesent sagittis, tortor id iaculis pharetra, dui eros gravida
- Cras euismod mauris ut magna porta egestas. Ut non nisl leo. In hac habitasse platea dictumst. Pellentesque sed diam hendrerit tellus sagittis bibendum. Donec lorem augue, aliquet
- In at interdum lacus. Ut purus arcu, consequat a leo at, viverra tincidunt eros. Sed non mi fringilla, ornare velit eget, feugiat tortor. Donec blandit orci at dapibus vehicula. Pellentesque non cursus tellus. Ut et molestie quam. Sed convallis laoreet odio at dignissim. Maecenas condimentum felis eu laoreet pretium. Fusce lacinia ligula purus, non

5.4 Gaussian Mixture Regression

The Gaussian mixture is a generative model of the joint probability of x and y given as,

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

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

5.4.1 GMR - prediction

We need the conditional distribution of the Gaussian mixture in order to get the predictive distribution. We will now formulate the conditial 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, How??!

$$p(x) = \sum_{k=1}^{K} \mathcal{N}(x|\mu_x^{(k)}, \Sigma_{xx}^{(k)})$$

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

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

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

$$p(y|x) = \frac{p(y,x)}{p(x)} \tag{5.1}$$
$$= \sum_{k=1}^{K} \frac{\pi^{(k)}}{p(x)} \mathcal{N}(x, y|\mu^{(k)}, \Sigma^{(k)}) \tag{5.2}$$
$$= \sum_{k=1}^{K} \frac{\pi^{(k)} \mathcal{N}(x|\mu_x^{(k)}, \Sigma_{xx}^{(k)})}{p(x)} \mathcal{N}(y|\mu_{y|x}^{(k)}, \Sigma_{y|x}^{(k)}) \tag{5.3}$$
$$= \sum_{k=1}^{K} \pi_{y|x}^{(k)} p(y|x, \mu_{y|x}^{(k)}, \Sigma_{y|x}^{(k)}) \tag{5.4}$$

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 conditonal of a Gaussian mixture model is again a Gaussian mixture model.

Background: Conditional og multivariate Gaussian
The conditional distribution is defined as <ref bishop>

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

where

$$\mu_{y|x} = \mu_y + \Sigma_{yx} \Sigma_{xx}^{-1} (x - \mu_x) \tag{5.5}$$
$$\Sigma_{y|x} = \Sigma_{yy} - \Sigma_{yx} \Sigma_{xx}^{-1} \Sigma_{xy} \tag{5.6}$$

5.4.2 GMR - Leaning
EM-algorithm, classic example.

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

$$p(y|x, \mathcal{D}) = \int \mathcal{N}(y|f(x), \sigma^2) p(f(x)|\mathcal{D}) df(x) \tag{5.7}$$

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 (5.8).

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,

$$p(y) = \mathcal{N}(y|A\mu + b, L^{-1} + A\Lambda^{-1}A^T) \tag{5.8}$$
$$p(x|y) = \mathcal{N}(x|\Gamma\mu + \Gamma[A^T L(y-b)], \Gamma) \tag{5.9}$$
$$\Gamma := (\Lambda + A^T LA)^{-1} \tag{5.10}$$

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*

$$p(f(\cdot)|\mathcal{D}) = \int p(f(\cdot)|\mathbf{x}, \mathbf{f}) p(\mathbf{f}|\mathcal{D}) d\mathbf{f}. \tag{5.11}$$

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,

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

We now calculate the first term in the integral (5.11), $p(f(\cdot)|\mathbf{x}, \mathbf{f})$ using that we have the joint prior distribution,

$$p(f(\cdot), \mathbf{f}|\mathbf{x}) = \mathcal{N}\left(\begin{bmatrix} f(\cdot) \\ \mathbf{f} \end{bmatrix} \middle| \begin{bmatrix} 0 \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} c(\cdot,\cdot) & c(\cdot, \mathbf{x}) \\ c(\mathbf{x}, \cdot) & c(\mathbf{x}, \mathbf{x}) \end{bmatrix} \right) \tag{5.12}$$

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 (5.11), $p(\mathbf{f}|\mathcal{D})$, 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 (5.11), and they are related such that it is possible to use (5.8) 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 (5.7) also is related in a simlar way, and we use (5.8), 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. ??! Forstår ikke

5.6 Deep Network Global Optimization - ?

?? Kernel regression :-)) Should I not work on this?

5.7 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, $\theta^{(1)},\ldots,\theta^{(K)}$ 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,

$$\sum_x p(x)p(x \to y) = p(y)\sum_x p(y \to x) \tag{5.13}$$

$$\implies \quad p(y) = \sum_x p(x)p(x \to y) \tag{5.14}$$

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

$$p(x,q) = \frac{1}{Z_H}\exp(-H(x,q)) \tag{5.15}$$
$$= \frac{1}{Z_E}\exp(-E(x))\frac{1}{Z_K}\exp(-K(x)) \tag{5.16}$$
$$= p(x)p(q) \tag{5.17}$$

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,

5.7.1 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.

5.7.2 Adaptive stochatic HMC

...

5.8 Appendix - SPN

Normalization of SPN

Calculation of Z

Input: node i
function Eval(node i)
  if i is leaf **then** ▷ leaf node
    return: 1 ▷ set node equal 1 at point (x, y)
  if $i \in \mathcal{S}$ **then**
    return: $\sum_{j \in ch(i)} w_{i,j}$Eval$(j)$
  if $i \in \mathcal{P}$ **then**
    return: $\prod_{j \in ch(i)}$ Eval(j)
$Z =$ Eval(root node)

6 Results

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

6.0.1 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.

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 [**PhDthesis**][**snoek2015scalable**][**NIPS2016_a96d3afe**] (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