Master's Thesis

Gaussian Process Regression and Bayesian Deep Learning for Insurance Tariff Migration

Department of Statistics
Ludwig-Maximilians-Universität München

Kai Philipp Becker

Munich, Novemeber 11th, 2022

Submitted in partial fulfillment of the requirements for the degree of M.Sc. in Statistics
Supervised by Dr. Janek Thomas

Abstract

This thesis reviews the current state of Gaussian Processes and Bayesian Deep Learning hybrid models, and their applicability to the transfer of actuarial functionalities. I conduct a benchmark study on two insurance tariff datasets and four OpenML regression tasks and compare Deep Kernel Learning, Deep Gaussian Processes, and Deep Sigma Point Processes with several strong baselines including a bayesian deep learning baseline. The model classes are examined with respect to fit, scalability, stability, and uncertainty quantification capabilities. My results show that among the analyzed models Variational Deep Kernel Learning, Deep Gaussian Processes, and Deep Ensembles show the best results with respect to predictive performance and uncertainty estimation abilities on the insurance tariff datasets.

Contents

1 Introduction

In today's fast-paced business environment, aging policy administration systems (PAS) can make it difficult for life insurers to quickly adapt to changing market conditions. Therefore, modernizing PAS and moving away from legacy systems is key to coping with increased competition from traditional companies and startups, evolving customer expectations, and shifting regulatory environments. However, these tasks often require semi-manual migration of large policy portfolios with complex actuarial functions, which can not only take several years but also tie up valuable resources (msg life (2022)). This thesis is part of a research project between LMU Munich and msg life to leverage automation and explainable AI to achieve significant gains in efficiency, scalability, and cost reduction for PAS migration.

Due to regulatory requirements, Uncertainty Quantification (UQ) and the interpolation of training data are required for this type of data. Gaussian Processes (GPs) possess both of these properties, which makes them a suitable initial candidate. However, they do not scale well to large data settings, and life insurers and their PAS typically manage multiple millions of insurance tariffs. Deep neural networks (DNNs) on the other hand have emerged in recent years as state-of-the-art methods in areas such as speech recognition and object detection due to their ability to learn data representations and their capability to handle vast amounts of data (LeCun et al. (2015)). They do not possess inherent UQ capabilities, however.

Neal (1995) has shown an equivalence between these two approaches. Specifically, he showed that a single-layer neural network with infinitely many hidden units converges to a GP with a specific kernel. More recently, researchers have begun to combine ideas from GPs and neural networks, giving birth to a series of new models such as Deep Kernel Learning (Wilson et al. (2016)) and Deep Gaussian Processes (Damianou and Lawrence (2013)). Deep Kernel Learning uses a neural network to extract features and uses them as input to a GP. Deep Gaussian Processes are inspired by the compositional structure of a neural network and can be interpreted as a composition of multiple GPs. Variations of these models include Variational Deep Kernel Learning (Bradshaw et al. (2017)), which combines a sparse Gaussian Process with a neural network feature extractor, and Deep Sigma Point Process (Jankowiak et al. (2020a)), a parametric model that tries to target the Deep Gaussian Processes predictive distribution directly.

The goal of this thesis is to summarize the current state of Gaussian Process neural network hybrid models, conduct a benchmark study on insurance tariff data, and analyze the

approaches with respect to scalability, stability, and quality of their uncertainty estimates. My results show, that Variational Deep Kernel Learning showed great predictive results over all examined datasets. The Deep Sigma Point Process displayed good performance on small and medium-sized datasets. The Deep Gaussian Process showed competitive performance on the examined large insurance dataset.

This Thesis is structured as follows: I begin by highlighting the distinctive features of insurance tariff data and motivating the application of GP regression models. In Section 3, I briefly review GP regression. In section 4, I summarize the current state of GP and neural network hybrid models. In section 5, I conduct a benchmark study of the models reviewed in section 4 on insurance tariff and OpenML regression datasets. I will conclude this with section 6 where I discuss the strength and weaknesses of these models.

2 The Composition of Insurance Tariff Data

A PAS migration includes the transfer of insurance policies and the reimplementation of the associated actuarial functions. The approximation of these actuarial functions is an ideal regression use case. The fictitious endowment insurance tariff dataset K2204 is of central interest to this thesis. It contains benefit and premium present values for a random portfolio of K2204 life-insurance policies. In this section, I will explain the math behind the K2204 dataset and motivate why GPs are a promising model class for actuarial function approximation.

K2204 is a classical endowment insurance tariff that pays out a single benefit payment both in case of survival or decease of the policyholder. For a policy with a duration of n years, the survival benefit is paid out at time n contingent upon the survival of the policyholder. In the event of the passing of the policyholder, the death benefit is paid out at the end of the respective insurance year. For this dataset, both death and survival benefits are the same. The mortality table DAV_2008_T_090 (*Sterbetafeln-DAV* (2008)) given by the German Actuarial Society is used for all present value calculations. The interest rate is given by i. All calculations are made with respect to a benefit payment of $\mathbf{B} = 1$.

For the duration of an insurance policy n, the policyholder at age x pays a constant premium $\mathbf{P}$. The premium present value $\mathbf{PV_P}$ (PPV) is the discounted sum of all premiums over the entire policy duration:

$$\mathbf{P} \cdot \mathbf{PV_P} = \mathbf{P} \cdot \sum_{k=0}^{n-1} \nu^k \cdot {}_kp_x, \tag{1}$$

where $\nu = \frac{1}{1+i}$ is the discount factor and ${}_kp_x$ is the k-year survival rate, which represents the probability that a person of age x survives to reach the age $x + k$.

The benefit present value $\mathbf{PV_B}$ (BPV) for a n-year insurance policy for a person of age x is given by a combination of two different benefits, a pure endowment insurance that is paid out in case of survival, and a term life insurance that is paid out in case of death. The pure endowment insurance is given by

$${}_nE_x = \nu^n \cdot \prod_{k=0}^{n-1} p_{x+k}, \tag{2}$$

where p_{x+k} is the one-year survival rate, which describes the probability that a person of age $x + k$ is still alive in the next year.

The term life insurance can be calculated as

$$ {}_nA_x = \sum_{k=0}^{n-1} \nu^{k+1} \cdot {}_kp_x \cdot q_{x+k}, \tag{3} $$

where q_{x+k} is the one-year mortality rate, which describes the probability that a person of age $x + k$ dies in the next year.
The BPV can then be calculated as

$$ \mathbf{PV_B} = {}_nA_x + {}_nE_x. \tag{4} $$

The task for the K2204 now is to predict BPV and PPV given a policyholder's age x, the policy duration n, and the gender of the policyholder as features. Due to regulatory requirements, a given algorithm should not only be able to interpolate the data points, but also provide uncertainty estimates. GPs were chosen as the base model class for this thesis, as they have the following key advantages. First, as I will show in Section 3, GP predictions can interpolate observations. As there is no observation noise present in the K2204 dataset, GPs should be able to handle this task. Second, the prediction of a GP is probabilistic. As the above calculations allow for generating arbitrarily many new observations, the model can be refitted in areas of high uncertainty or poor performance. Third, GPs are versatile as specialized kernels can be specified to handle the given properties by the task at hand.
The R1_08 is the second insurance dataset that will be analyzed in this thesis. It contains annuity insurance tariffs and differs from K2204 in that the insurance benefit is paid in lifetime annuity payments and not in a one-time payment as in K2204. For more reference on annuity insurance and actuarial mathematics see Kahlenberg et al. (2018).

3 Gaussian Process Regression

Given a dataset $\mathcal{D}$ of size n with input vectors $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ each of dimension d and a vector of targets $\mathbf{y} = \{y_1, ..., y_n\}$, a Gaussian process is a collection of random variables, where any finite number of those random variables have a joint Gaussian Distribution (Rasmussen (2003)). Thus, if a function $f(x)$ is generated by a GP, $f(x) \sim GP(m(x), K(X, X))$, then for any finite collection of inputs $\{\mathbf{x}_1, ..., \mathbf{x}_n\}$, the associated vector of function values $\{f(\mathbf{x}_1), ..., f(\mathbf{x}_n)\}$ has a Gaussian distribution

$$\mathbf{f} = [f(\mathbf{x}_1), ..., f(\mathbf{x}_n)]^T \sim \mathcal{N}(m(x), K(\mathbf{x}, \mathbf{x}'), \tag{5}$$

where m(x) is defined as the mean function and $(K(\mathbf{x}, \mathbf{x}'))_{i,j} = k(\mathbf{x}_i, \mathbf{x}'_j)$ is a covariance function determined by the covariance kernel $k(.,.)$ of the Gaussian process. A GP is therefore completely specified by its mean and covariance function. For notational simplicity, the mean function will be set to 0.

For test points $\mathbf{X}_*$, the joint distribution over the function values $\mathbf{f}$ and $\mathbf{f}_*$ can be written as

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(\mathbf{X}, \mathbf{X}) & K(\mathbf{X}, \mathbf{X}_*) \\ K(\mathbf{X}_*, \mathbf{X}) & K(\mathbf{X}_*, \mathbf{X}_*) \end{bmatrix}\right) \tag{6}$$

By conditioning $\mathbf{f}_*$ on $\mathbf{X}_*$, $\mathbf{X}$ and $\mathbf{f}$ we get the following predictive distribution

$$\begin{aligned} p(\mathbf{f}_* \mid \mathbf{X}_*, \mathbf{X}, \mathbf{y}) \sim \mathcal{N}\big(&K(\mathbf{X}_*, \mathbf{X}) K(\mathbf{X}, \mathbf{X})^{-1} \mathbf{f}, \\ &K(\mathbf{X}_*, \mathbf{X}_*) - K(\mathbf{X}_*, \mathbf{X}) K(\mathbf{X}, \mathbf{X})^{-1} K(\mathbf{X}, \mathbf{X}_*)\big) \end{aligned} \tag{7}$$

The prediction for a training point $\mathbf{x}_i$ is the exact target value $\mathbf{y}_i$.

$$\begin{aligned} p(\mathbf{f} \mid \mathbf{X}, \mathbf{y}) \sim \mathcal{N}\big(&K(\mathbf{X}, \mathbf{X}) K(\mathbf{X}, \mathbf{X})^{-1} \mathbf{y}, \\ &K(\mathbf{X}, \mathbf{X}) - K(\mathbf{X}, \mathbf{X}) K(\mathbf{X}, \mathbf{X})^{-1} K(\mathbf{X}, \mathbf{X})\big) = \mathcal{N}(\mathbf{y}, 0) \end{aligned} \tag{8}$$

A GP is therefore a function interpolator. However, this property is not always warranted. For that reason, a small constant is often added to the diagonal of the covariance matrix, which can be estimated during training.

The choice of the kernel function encodes prior assumptions about the functions drawn from a Gaussian Process such as smoothness and periodicity. A popular choice for a kernel is the Squared Exponential kernel (Rasmussen (2003)), also known as Radial Basis Function (RBF) kernel

$$\mathbf{K}_{RBF}(\mathbf{x}, \mathbf{x}') = exp(-\frac{\|\mathbf{x} - \mathbf{x}'\|}{2 * l^2}). \tag{9}$$

The choice of the kernel also introduces learnable Hyperparameters. Here, the Hyperparameter lengthscale controls the smoothness of the underlying function f. A larger lengthscale will result in a smooth function, while a small lengthscale will produce a more wiggly function.

The Hyperparameters of a GP can be optimized using gradient methods using the marginal log-likelihood (MLL). The marginal likelihood is given as the integral of the likelihood times the prior

$$p(\mathbf{y} \mid \mathbf{X}) = \int \underbrace{p(\mathbf{y} \mid \mathbf{f}, \mathbf{X})}_{\text{likelihood}} \underbrace{p(\mathbf{f} \mid \mathbf{X})}_{\text{GP Prior}} d\mathbf{f}. \tag{10}$$

The integral in equation 10 can be solved analytically. The Hyperparameters of a GP θ can be discovered by minimizing the marginal log-likelihood

$$\log p(\mathbf{y} \mid \mathbf{X}, \theta) = -\frac{1}{2}\mathbf{y}^\top K_\theta^{-1}\mathbf{y} - \frac{1}{2}\log|K_\theta| - \frac{n}{2}\log 2\pi \tag{11}$$

where K_θ is short for K(X,X) given θ. The term MLL refers to the marginalization over the function values and can be viewed as a penalized fit measure, where the term $\frac{1}{2}\mathbf{y}^\top K_\theta^{-1}\mathbf{y}$ measures the data fit and $\frac{1}{2}\log|K_\theta|$ is a complexity penalization term. The final term $\frac{n}{2}\log 2\pi$ is a normalization constant.

Limitations of GPs include their computational and storage cost. To calculate the term $K_\theta^{-1}\mathbf{y}$ in equation 11, one needs to invert the kernel matrix, which is of size $N \times N$ where N is the number of data points in the training set. The most common approach, to compute the Cholesky decomposition, requires $O(N^3)$ operations (Rasmussen (2003)) and $O(N^2)$ storage costs due to keeping the whole dataset in memory. This is infeasible for large-scale datasets.

In addition to the scalability issues, the performance and generalization behavior of a GP is highly dependent on the choice of the covariance function (Wilson and Adams (2013)). Hence, specifying an appropriate kernel for a given task is crucial. Moreover, Bengio et al. (2005) found that popular choices such as the squared exponential kernel suffer from the curse of dimensionality.

4 The Current State of Gaussian Process Neural Network Hybrid Models

Gaussian Processes are limited by the size of the training data and choice of an appropriate kernel. To overcome these limitations, different ideas have been developed to combine the expressive power of Neural Networks with the uncertainty estimation capabilities of Gaussian Processes. In the following section, I am going to outline the main ideas for Gaussian Process and Neural Network Hybrid Models for Regression.

4.1 Deep Kernel Learning

As we have seen in Chapter 3, the choice of a suitable covariance kernel is a crucial component of a Gaussian Process as it encodes the structure and assumptions about the function which we wish to learn. However, the choice may depend on the given dataset and prior knowledge of the problem at hand. Furthermore, popular kernels such as the RBF kernel are unable to learn effective representations from data to improve predictions and instead just provide smoothing (Ober et al. (2021)). This limits GPs from fully utilizing the information in high-dimensional datasets or when dealing with highly structured data, such as images (Ober et al. (2021)). Deep neural networks (DNN), on the other hand, are known for their great representational power, and ability to learn feature representations that aid prediction and scalability in large data settings (LeCun et al. (2015)).
To obtain the best of both worlds, Calandra et al. (2016) and Wilson et al. (2016) combine the uncertainty representation advantages of GPs with the representation-learning advantages of DNNs. The idea is to jointly learn a mapping by a DNN into a latent dimension k, with $d \gg k$, and a GP that uses the latent feature representation as its input space in an end-to-end fashion.
In DKL, the input $\mathbf{x}$ is forwarded through a DNN with L hidden Layers. The output of the DNN is then fed into a base kernel, which is subsequently used as the covariance function of a GP.

$$k_\theta(\mathbf{x}, \mathbf{x}'|\boldsymbol{\theta}) \rightarrow k^{deep}(g(\mathbf{x}, \mathbf{w}), g(\mathbf{x}', \mathbf{w})|\boldsymbol{\theta}, \mathbf{w}) \tag{12}$$

$k_\theta(\mathbf{x}, \mathbf{x}'|\boldsymbol{\theta})$ is the base kernel with Hyperparameters $\boldsymbol{\theta}$ and $g(\mathbf{x}, \mathbf{w})$ is the non-linear mapping given by a DNN with weights $\mathbf{w}$. I denote k^{deep} as the base kernel, which takes the DNN mapping $g(\mathbf{x}, \mathbf{w})$ as its input. As a sensible choice for the base kernel, Wilson et al. (2016) suggest using the RBF Kernel. For added flexibility, Wilson et al. (2016) propose

to use the spectral mixture (SM) kernel as a base kernel as it can discover quasi-periodic stationary structures.

$$k_{\mathrm{SM}}\left(\mathrm{x}, \mathrm{x}' \mid \boldsymbol{\theta}\right) = \sum_{q=1}^{Q} a_q \frac{\left|\Sigma_q\right|^{\frac{1}{2}}}{(2\pi)^{\frac{D}{2}}} \exp\left(-\frac{1}{2}\left\|\Sigma_q^{\frac{1}{2}}\left(\mathrm{x}-\mathrm{x}'\right)\right\|^2\right) \cos\left\langle \mathrm{x}-\mathrm{x}', 2\pi\boldsymbol{\mu}_q\right\rangle \tag{13}$$

The complete model is shown in figure 1. This model applies a GP with base kernel k_θ to the final layer of a DNN conditioned on all kernel Hyperparameters. As a GP with an RBF or SM kernel correspond to a representation with infinite basis functions, the DNN can be viewed as having a hidden layer with an infinite number of hidden units.

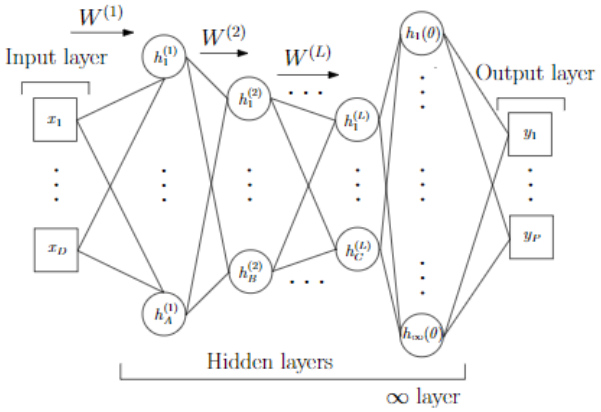


Figure 1: In Deep Kernel Learning, the input $\mathbf{x}$ is mapped through a L-layered DNN followed by a hidden layer with an infinite number of hidden units. The mapping of the GP is parameterized by the DNN weights $\mathbf{w}$ and base kernel Hyperparameters $\boldsymbol{\theta}$. $\{\mathbf{w}, \boldsymbol{\theta}\}$ are learned via the MLL of the GP.
Source: Wilson et al. (2016)

Similar to a standard GP, the parameters of the DNN can be treated as Hyperparameters of the kernel. Thus to train the model, all Hyperparameters $\{\mathbf{w}, \boldsymbol{\theta}\}$ are jointly learned by maximizing the MLL of the GP from Equation 10, where $\mathbf{w}$ are the weights of the DNN and $\boldsymbol{\theta}$ are the kernel parameters, resulting in an end-to-end training scheme. The chain rule is used to calculate the derivatives of the MLL with respect to the kernel Hyperparameters:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{L}}{\partial K^{deep}} \frac{\partial K^{deep}}{\partial \boldsymbol{\theta}}, \quad \frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{\partial \mathcal{L}}{\partial K^{deep}} \frac{\partial K^{deep}}{\partial g(\mathbf{x}, \mathbf{w})} \frac{\partial g(\mathbf{x}, \mathbf{w})}{\partial \mathbf{w}} \tag{14}$$

The derivative of the MLL with respect to the covariance matrix K^{deep} is given by

$$\frac{\partial \mathcal{L}}{\partial K^{deep}} = \frac{1}{2}(K^{deep-1}\mathbf{y}\mathbf{y}^{\mathbf{T}}K^{deep-1} - K^{deep-1}). \tag{15}$$

$\frac{\partial K^{deep}}{\partial \boldsymbol{\theta}}$ refers to the derivates of the kernel with respect to the base kernel Hyperparameters, conditioned on the transformation of the inputs $g(\mathbf{x}, \mathbf{w})$. The base kernel Hyperparameters could include the lengthscale of the RBF kernel or mixture weights, bandwidths, and frequencies for the Spectral Mixture kernel. Likewise, $\frac{\partial K^{deep}}{\partial g(\mathbf{x},\mathbf{w})}$ are the derivatives of the kernel with respect to the transformation g while holding $\boldsymbol{\theta}$ constant. The derivatives with respect to the DNN weights $\mathbf{w}$ can be computed using backpropagation.
Looking at equation 15, you can see immediately that this learning training scheme still suffers from the same problem as regular GP training. The covariance matrix K^{deep} needs to be inverted, resulting in a complexity of $\mathcal{O}(N^3)$. Without any changes, this would make DKL inapplicable to larger regression problems where the benefits of a feature extractor were to be needed. To circumvent this issue, Wilson et al. (2016) leverage structured kernel interpolation and replace every occurrence of K^{deep} with the KISS-GP covariance matrix (Wilson and Nickisch (2015)).
The idea of KISS-GP is to construct kernel approximations for fast computations through kernel interpolation. Popular inducing points methods such as subset of regressors (SoR), deterministic training conditional (DTC), and fully independent training conditional (FITC) used to scale up GPs to large datasets cost $\mathcal{O}(M^2N)$ for M inducing points (Quinonero-Candela and Rasmussen (2005)). Subset of regressors uses the approximate kernel

$$\tilde{k}_{\text{SoR}}(\mathbf{x}, \mathbf{z}) = K_{\mathbf{x},U} K_{U,U}^{-1} K_{U,\mathbf{z}} \tag{16}$$

over a set of M inducing points $U = [\mathbf{u}_{i=1,\dots,M}]$. To improve over standard GP regression, one is constricted to choose $M \ll N$. However, this can lead to a decrease in predictive performance (Wilson et al. (2014)). *Structured Kernel Interpolation* (SKI) approximates the $N \times M$ matrix $K_{\mathbf{x},U}$ of cross covariances evaluated at training points $\mathbf{x}$ and inducing points U, by interpolating on the $M \times M$ covariance matrix $K_{U,U}$:

$$K_{\mathbf{x},U} \approx W K_{U,U}, \tag{17}$$

where W is a sparse $N \times M$ matrix of interpolation weights. SKI uses local cubic interpolation resulting in a sparse matrix containing only four non-zero entries per row. Thus, K^{deep} can be rewritten as

$$\begin{aligned} K^{deep} &\overset{\text{SoR}}{\approx} K_{\mathbf{x},U} K_{U,U}^{deep-1} K_{U,\mathbf{x}} \overset{\text{Eq(17)}}{\approx} W K_{U,U}^{deep} K_{U,U}^{deep-1} K_{U,U}^{deep} W^\top \\ &= W K_{U,U}^{deep} W^\top \coloneqq K_{KISS}. \end{aligned} \tag{18}$$

where $K_{U,U}^{deep}$ is the covariance matrix in Eq. 12 evaluated over the inducing points U.

This procedure costs $\mathcal{O}(M^2 + N)$ computations and $\mathcal{O}(M^2 + N)$ storage. By placing the inducing points over a regular multidimensional lattice and exploiting the resulting decomposition of $K_{U,U}$ into a Kronecker product of Toeplitz matrices, this can be improved to $\mathcal{O}(N + PM^{1+1/P})$ cost for computations and $\mathcal{O}(N + PM^{\frac{2}{P}})$ cost for storage with P grid points. In this manner, this approach scales almost linearly in N. In contrast to the aforementioned scalable approximations techniques, KISS-GP allows having $M \approx N$ due to the linear scaling. Therefore, it yields a near-exact accuracy in its approximation (Wilson and Nickisch (2015)).

4.1.1 Related Methods

Kirstein et al. (2022) use Tensor-Train (TT) decompositions to parameterize the weights of a DNN. The Idea is to use a TT function to extract features from the data and use these as inputs into a GP. The parameters of the TT are treated as kernel-Hyperparameters and can be jointly trained with the GP parameters. Ober et al. (2021) showed the susceptibility of DKL to overfitting. Kirstein et al. (2022) state that overfitting is highly reduced for their method, due to inherent regularisation by the choice of basis functions and by imposing a low-rank structure. Furthermore, this low-rank structure allows for implicit feature selection on the data and thus alleviates problems in high-dimensional tasks. For scalability, the authors utilized variational inference for sparse GPs.

4.2 Deep Gaussian Processes

Inspired by the phenomenon in DNNs that increasing their depth through additional layers can lead to better predictive performance and greater capacity, DGPs are the multi-layer hierarchical composition of this idea to Gaussian processes. They are created by stacking multiple GPs in parallel and on top of each other, where each layer is modeled as the output of a multivariate GP and acts as the next GP layers input and thus resulting in a flexible, compositional function prior.
The resulting model is no longer a GP, but this compositional structure brings different advantages. First, it offers greater expressive power compared to standard GPs as it can learn more complex interactions between data. Many common kernel functions used for GP regression have simple similarity metrics. This can become insufficient for more complex datasets where different similarity metrics have to be used in different regions of the input space. The compositional structure of DGPs allows them to perform input warping, dimensionality reduction, or expansion and thus automatically constructs a kernel that works well for a given problem (Bui et al. (2016)). Second, even though they are a very rich model class, they can learn representations with only a few Hyperparameters to optimize. Third, it retains the advantages of GPs such as good uncertainty estimates. In contrast to DNNs, the outputs of a layer are probabilistic as they are governed by a GP resulting in the uncertainty being propagated through the network.

4.2.1 DGP Model

Damianou and Lawrence (2013) introduced the DGP architecture which corresponds to a graphical model consisting of L layers of latent variables $\{\mathbf{h}_l\}_{l=1}^{L}$. All latent variables $\{\mathbf{h}_l\}_{l=1}^{L}$ act as inputs for the layer below and outputs for the layer above. A Gaussian process controls the mapping between two layers. Each mapping is handled by a separate GP with covariance function k_l and Hyperparameters θ_l. The process then takes the form:

$$\begin{aligned} \mathbf{Y} &= \mathbf{f}_L\left(\mathbf{h}_{l-1}\right) + \epsilon_L, \quad \epsilon_L \sim \mathcal{N}\left(0, \sigma_L^2 \mathbf{I}\right) \\ \mathbf{h}_l &= \mathbf{f}_l\left(\mathbf{h}_{l-1}\right) + \epsilon_l, \quad \epsilon_l \sim \mathcal{N}\left(0, \sigma_l^2 \mathbf{I}\right), \quad l = 1 \ldots L-1 \end{aligned} \tag{19}$$

where the function $\mathbf{f}_l$ is drawn from a GP with covariance function k_l: $\mathbf{f}_l(x) \sim \mathcal{GP}(0, k_l(x, x^I))$. For notational simplicity, I reduced the hidden layers to be of a single dimension, but this can be generally extended to multiple dimensions. The noise between the layers is assumed to be i.i.d. Gaussian. I define $h_0 = \mathbf{X}$. For $L = 1$ this collapses back to a GP. Hidden variables in intermediate layers can and will generally have multiple dimensions.

The joint density can be written analogously as to the GP model:

$$p(\mathbf{y}, \{\mathbf{f}_l\}_{l=1}^{L}) = \underbrace{p(\mathbf{y} \mid \mathbf{f}_L)}_{\text{likelihood}} \underbrace{\prod_{l=1}^{L} p(\mathbf{f}_l \mid \mathbf{f}_{l-1})}_{\text{DGP Prior}} \tag{20}$$

4.2.2 Inference

Inference is used to marginalize out the latent variables and minimize the marginal likelihood. Unlike in the GP case, the Inference problem for DGPs can not be solved analytically. This comes as a result of the non-linearities introduced by the covariance function between two layers. To overcome this issue, Damianou and Lawrence (2013) use inducing points and variational inference to approximate the marginal likelihood. The inducing points for the layers are denoted by $Z^1, \dots Z^L$ with associated inducing outputs $\mathbf{u}^1 = \mathbf{f}_1(Z^1) \dots \mathbf{u}^L = \mathbf{f}_L(Z^L)$. The number of inducing points K does not need to be the same for every GP and can vary over the overall architecture. The joint density now becomes

$$p(\mathbf{y}, \{\mathbf{f}^l\}_{l=1}^{L}, \{\mathbf{u}^l\}_{l=1}^{L}) = p(\mathbf{y} \mid \mathbf{f}^L) \prod_{l=1}^{L} p(\mathbf{f}^l \mid \mathbf{u}^l) p(\mathbf{u}^l) \tag{21}$$

The variational inference scheme introduced by Damianou and Lawrence (2013) uses a variational posterior that maintains the exact model conditioned on $\mathbf{u}^l$ and allows for a tractable lower bound. They use a mean-field variational posterior which makes strong independence and Gaussian assumptions. However, Salimbeni and Deisenroth (2017) have identified two key problems with their approach. First, the DGP method by Damianou and Lawrence (2013) forces the inputs to each layer to be independent of the outputs of the previous layer. Furthermore, the noisy corruptions in Eq. 19 are modeled separately and are factorized by a fully Gaussian variational distribution. Second, the output is a single GP with independent Gaussian inputs. This causes the posterior to lose the correlations between the layers, thus limiting the expressiveness of the model and resulting in the variance likely being underestimated. Therefore, Salimbeni and Deisenroth (2017) propose an inference method where they lose analytical tractability but retain a posterior with the full conditional structure of the full model.

4.2.3 Doubly Stochastic Variational Inference

In DGP modeling, the difficulty lies within correctly modeling the within and between layer correlations. In contrast to the original DGP inference scheme where independence between layers is assumed and which allows for analytic tractability, Salimbeni and Deisenroth (2017) propose a variational inference scheme that simplifies the correlation within layers, but maintains the correlation between layers. However, the resulting model can no longer be evaluated analytically. The authors circumvent this by drawing samples using univariate Gaussians and thus updating the bound stochastically.
Their proposed variational posterior takes the following factorized form with three assumptions:

$$q\left(\left\{\mathbf{f}^l, \mathbf{u}^l\right\}_{l=1}^L\right) = \prod_{l=1}^L p\left(\mathbf{f}^l \mid \mathbf{u}^l; \mathbf{f}^{l-1}, \mathbf{Z}^{l-1}\right) q\left(\mathbf{u}^l\right) \tag{22}$$

First, the posterior, conditioned on $\mathbf{u}^l$, maintains the exact model. Second, the posterior distribution of $\{\mathbf{u}^l\}_{l=1}$ is factorized between layers. And third, $q(\mathbf{u}^l)$ is Gaussian with mean $\mathbf{m}^l$ and variance $\mathbf{S}^l$.
For each layer, as both terms in the variational posterior are Gaussian, the inducing variables can be marginalized out analytically.

$$\begin{aligned} q\left(\left\{\mathbf{f}^l\right\}_{l=1}^L\right) &= \int \prod_{l=1}^L p\left(\mathbf{f}^l \mid \mathbf{u}^l; \mathbf{f}^{l-1}, \mathbf{Z}^{l-1}\right) q\left(\mathbf{u}^l\right) d\mathbf{u}^l \\ &= \prod_{l=1}^L q\left(\mathbf{f}^l \mid \mathbf{m}^l, \mathbf{S}^l; \mathbf{f}^{l-1}, \mathbf{Z}^{l-1}\right) \\ &= \prod_{l=1}^L \mathcal{N}\left(\mathbf{f}^l \mid \tilde{\mu}^{\mathbf{l}}, \tilde{\mathbf{\Sigma}}^{\mathbf{l}}\right) \end{aligned} \tag{23}$$

with $\left[\tilde{\boldsymbol{\mu}}^l\right]_i = \mu_{\mathbf{m}^l, \mathbf{Z}^{l-1}}\left(\mathbf{f}_i^l\right)$ and $\left[\tilde{\mathbf{\Sigma}}^l\right]_{ij} = \Sigma_{\mathbf{S}^l, \mathbf{Z}^{l-1}}\left(\mathbf{f}_i^l, \mathbf{f}_j^l\right)$. Looking at this, we can see that the marginal of the final layer $\mathbf{f}_i^L$ only depends on $\mathbf{f}_i^L - 1$. More specifically, the i-th marginal of the final layer only depends on the i-th marginals of all previous layers:

$$q\left(\mathbf{f}_i^L\right) = \int \prod_{l=1}^{L-1} q\left(\mathbf{f}_i^l \mid \mathbf{m}^l, \mathbf{S}^l; \mathbf{f}_i^{l-1}, \mathbf{Z}^{l-1}\right) d\mathbf{f}_i^l \tag{24}$$

As the marginals only depend on its inputs conditioned on the previous layer, this allows for sampling from the variational posterior. Using the re-parameterization trick (Kingma et al. (2015)), we can draw samples using only univariate Gaussians by first drawing $\epsilon_i^l \sim \mathcal{N}(0, \mathbf{I}_l)$ and afterward recursively sampling $\hat{\mathbf{f}}_i^l \sim q(\mathbf{f}_i^l \mid \mathbf{m}^l, \mathbf{S}^l; \hat{\mathbf{f}}_{\mathbf{i}}^{\mathbf{l-1}}, \mathbf{Z}^{l-1})$ for $l = 1, \ldots, L-1$ as

$$\hat{\mathbf{f}}_i^l = \mu_{\mathbf{m}^l,\mathbf{Z}^{l-1}}\left(\hat{\mathbf{f}}_i^{l-1}\right) + \boldsymbol{\epsilon}_i^l \odot \sqrt{\Sigma_{\mathbf{S}^l,\mathbf{Z}^{l-1}}\left(\hat{\mathbf{f}}_i^{l-1}, \hat{\mathbf{f}}_i^{l-1}\right)} \tag{25}$$

The evidence lower bound of the DGP is given by

$$\mathcal{L}_{DGP} = \sum_{i=1}^{N} \mathbb{E}_{q\left(\mathbf{f}_i^L\right)}\left[\log p\left(\mathbf{y}_n \mid \mathbf{f}_n^L\right)\right] - \sum_{l=1}^{L} \mathrm{KL}\left[q\left(\mathbf{u}^l\right) \| p\left(\mathbf{u}^l; \mathbf{Z}^{l-1}\right)\right]. \tag{26}$$

This bound is evaluated using two sources of stochasticity. First by drawing Monte Carlo samples from the posterior according to Eq. 25. Second by using sub-sampling techniques, since the bound factorizes over the data, which allows for scalability. The computational complexity of a DGP is $\mathcal{O}(NM^2(D^1 + ... + D^L))$, where D^l is the dimension of output at layer l.

Predictions for DGPs are done by sampling from the variational posterior changing the input locations to test locations x_*. The function values at the test locations are denoted as $\mathbf{f}_*^l$. The density for $\mathbf{f}_*^L$ is obtained by using the Gaussian mixture

$$q\left(\mathbf{f}_*^L\right) \approx \frac{1}{S}\sum_{s=1}^{S} q\left(\mathbf{f}_*^L \mid \mathbf{m}^L, \mathbf{S}^L; \mathbf{f}_*^{(s)^{L-1}}, \mathbf{Z}^{L-1}\right) \tag{27}$$

and sampling S samples $\mathbf{f}_*^{(s)^{L-1}}$ using 25.

4.2.4 Model Design

Duvenaud et al. (2014) show for the DGP architecture by Damianou and Lawrence (2013) with zero mean, that the model's representational power decreases as the number of layers increases. This is the result of the GP mapping being non-injective resulting in the function values being clustered around the same few values as the depth increases. To resolve this, Duvenaud et al. (2014) propose additionally feeding the input $\mathbf{X}$ to each layer. Salimbeni and Deisenroth (2017) instead use a linear mean function for all inner layers and a constant mean function for the last layer.

4.2.5 Related Methods

Bui et al. (2016) use the fully independent training conditional inducing point approximation in each DGP layer, which results in a parametric model. As the marginal likelihood is not tractable for a DGP, the authors use an expectation propagation approximation scheme and combined this with probabilistic backpropagation to propagate uncertainly through the non-linear GP mapping. They demonstrate good results on various regression

tasks with up to 500.000 data points, but were outperformed later by Salimbeni and Deisenroth (2017) and their variational inference method.

Havasi et al. (2018) show for a variety of datasets, that the posterior distribution is non-Gaussian and thus that the Gaussian approximation to the posterior distribution employed by Salimbeni and Deisenroth (2017) can therefore be a poor approximation. To solve this, they apply the Stochastic Gradient Hamiltonian Monte Carlo (SGHMC) sampling method to efficiently estimate the posterior distribution. The authors compared their method to baseline using doubly stochastic variational inference (DSVI) and showed that the SGHMC can outperform DSVI on 9 UCI regression benchmark datasets in both average test Log-Likelihood and runtime. However, their method introduces its own parameters in addition to the DGP parameters, which makes them difficult to train without prior knowledge of the data.

Cutajar et al. (2017) propose a modification to the DGP model formulation based on random feature expansion at each hidden layer. In contrast to the aforementioned inference methods, the GP governing the mapping between two layers is replaced by a two-layer weight space approximation. First, random feature expansion is used to approximate the kernel function and followed by a linear transformation parameterized by a weight matrix. The authors show, that this yield a Bayesian DNN with low-rank weight matrices while the approximations on the covariance functions result in DNN activation functions, e.g. the Rectified Linear Unit function for the ARC-COSINE kernel. For inference, they use stochastic variational inference and exploit mini-batch-based stochastic gradient optimization for scalability. In their experiment, they demonstrate the scalability of their methods on datasets with up to 8 million data points and that their approach can outperform the methods by Bui et al. (2016).

4.3 Deep Sigma Point Process

Inspired by the multi-layer structure of Deep Gaussian Processes, Jankowiak et al. (2020a) introduce Deep Sigma Point Processes (DSPP), which are a multi-layer composition of parametric GPs. DSPPs possess many of the advantages of DGPs such as mini-batch training, while offering a much simpler inference problem. As shown in chapter 4.2, inference is a difficult task for DGPs, as it revolves around solving an L-dimensional integral. In contrast to that, DSPPs solve a simpler maximum likelihood inference problem. Before diving into detail on DSPPs, I'm going to introduce and motivate the underlying model class, the parametric Gaussian Process. This section is structured as follows. First, I will briefly motivate parametric GPs. Afterward, I will summarize stochastic variational inference (SVI) for GPs to introduce the model formulation for the parametric GP and lastly extend it to the DSPP.

4.3.1 Parametric Gaussian Processes

The main bottleneck of Gaussian Processes is the inversion of the kernel matrix. Different solutions exist to get around this hurdle. One popular idea which has enabled Gaussian process regression on large datasets is the combination of inducing points method with Variational Inference (Hensman et al. (2013)). However, Jankowiak et al. (2020b) show that the resulting predictive distribution often underestimates uncertainties, as the predictive variance is often dominated by the observation noise. The Idea for parametric GPs is to bypass the posterior approximations entirely and instead use a parametric model to target the predictive distribution directly.

4.3.2 Stochastic Variational Gaussian Process (SVGP)

SVI revolves around introducing a set of inducing variables $\mathbf{u}$ that depend on variation parameters $\mathbf{Z} = \{\mathbf{z}_m\}_{m=1}^{M}$, where $M = \dim(\mathbf{u}) \ll N$. Thus, the GP prior from Eq. 10 extends from $p(\mathbf{f} \mid \mathbf{X})$ to $p(\mathbf{f} \mid \mathbf{u}, \mathbf{X}, \mathbf{Z})p(\mathbf{u} \mid \mathbf{Z})$. Applying Jensen's inequality to the log joint density of the target and the inducing variables leads to the following lower bound:

$$\begin{aligned}\log p(\mathbf{y}, \mathbf{u} \mid \mathbf{X}, \mathbf{Z}) &= \log \int d\mathbf{f} p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{f} \mid \mathbf{u}) p(\mathbf{u}) \\ &\geq \mathbb{E}_{p(\mathbf{f}|\mathbf{u})}[\log p(\mathbf{y} \mid \mathbf{f}) + \log p(\mathbf{u})] \\ &= \sum_{i=1}^{N} \log \mathcal{N}\left(y_i \mid \mathbf{k}_i^T \mathbf{K}_{MM}^{-1} \mathbf{u}, \sigma_{\text{obs}}^2\right) \\ &- \frac{1}{2\sigma_{\text{obs}}^2} \operatorname{Tr} \tilde{\mathbf{K}}_{NN} + \log p(\mathbf{u})\end{aligned} \tag{28}$$

This approach cuts down the model complexity to just $O(M^3)$, as the expensive computation of inverting $\mathbf{K}_{NN}$ gets replaced by K_{MM}. Furthermore, the log-likelihood and the trace term factorizes as a sum over the datapoints (y_i, x_i) and thus allow for data subsampling.

Hensman et al. (2013) introduce a variational distribution $q(\mathbf{u}) = \mathcal{N}(\mathbf{m}, \mathbf{S})$, whose variational parameters m and S are optimized using the evidence lower bound (ELBO), which is the expectation over equation X w.r.t. to $q(\mathbf{u})$ plus an entropy term:

$$\begin{aligned}\mathcal{L}_{\text{svgp}} =& \mathbb{E}_{q(\mathbf{u})}[\log p(\mathbf{y}, \mathbf{u} \mid \mathbf{X}, \mathbf{Z})] + H[q(\mathbf{u})] \\ =& \sum_{i=1}^{N} \left\{ \log \mathcal{N}\left(y_i \mid \mu_{\mathbf{f}}\left(\mathbf{x}_i\right), \sigma_{\text{obs}}^2\right) - \frac{\sigma_{\mathbf{f}}\left(\mathbf{x}_i\right)^2}{2\sigma_{\text{obs}}^2} \right\} \\ & - \mathrm{KL}(q(\mathbf{u}) \mid p(\mathbf{u}))\end{aligned} \tag{29}$$

where $\mu_{\mathbf{f}}\left(\mathrm{x}_i\right)$ is the predictive mean function given by

$$\mu_{\mathbf{f}}\left(\mathbf{x}_i\right) = \mathbf{k}_i^T \mathbf{K}_{MM}^{-1} \boldsymbol{m}$$

and where $\sigma_{\mathbf{f}}\left(\mathbf{x}_i\right)^2 \equiv \operatorname{Var}\left[\mathbf{f}_{\mathrm{i}} \mid \mathbf{x}_{\mathrm{i}}\right]$ denotes the latent function variance

$$\sigma_{\mathbf{f}}\left(\mathbf{x}_i\right)^2 = \tilde{\mathbf{K}}_{ii} + \mathbf{k}_i^T \mathbf{K}_{MM}^{-1} \mathbf{S} \mathbf{K}_{MM}^{-1} \mathbf{k}_i$$

The predictive distribution for SVGP at a test location x_{star} is given by

$$p\left(\mathbf{y}^* \mid \mathbf{x}^*\right) = \mathcal{N}\left(\mathbf{y}^* \mid \mu_{\mathbf{f}}\left(\mathbf{x}^*\right), \sigma_{\mathbf{f}}\left(\mathbf{x}^*\right)^2 + \sigma_{\text{obs}}^2\right) \tag{30}$$

Note here that the predictive variance has two components: the input-dependent latent function variance $\sigma_{\mathbf{f}}(\mathbf{x}^*)^2$ and the input-independent observation noise σ_{obs}^2. However, these terms appear asymmetrically in the SVGP Loss function 29. Jankowiak et al. (2020b) assume that the absence of $\sigma_{\mathbf{f}}(\mathbf{x}^*)^2$ in the data fit term results in the domination of the predictive variance by the observation noise.

4.3.3 Predictive Parametric Gaussian Processes

To address this miss-match, the authors introduce a parametric GP regression model which directly targets the predictive distribution in 30.

$$\mathcal{L}_{\text{ppgpr}} = \sum_{i=1}^{N} \log \mathcal{N}\left(y_i \mid \mu_{\mathbf{f}}\left(\mathbf{x}_i\right), \sigma_{\text{obs}}^2 + \sigma_{\mathbf{f}}\left(\mathbf{x}_i\right)^2\right) - \beta_{\text{reg}} \text{KL}(q(\mathbf{u})||p(\mathbf{u})) \tag{31}$$

The parameters m, S, Z, the observation noise, and the kernel Hyperparameters can be optimized via gradient methods.

4.3.4 Deep Sigma Point Process

Deep Sigma Point Processes are inspired by DGPs and their flexible function priors through the composition of multiple GPs. The main challenge with DGPs is their inference scheme.

Analogous to the PPGPR, DSPPs try to target the predictive Distribution of DGPs directly. Unfortunately, the predictive distribution for a DGP is a continuous mixture of Normal distributions and cannot be computed in a closed form. Instead, the continuous mixture is replaced with a parametric finite mixture. Using Gauss-Hermite-Quadrature, the continuous mixture is approximated by an S-component mixture of Dirac delta distributions controlled by weights $w^{(s)}$ and quadrature points $\mathcal{E}^{(s)}$. For more details on the quadrature rules used to derive the DSPP, see the derivation in Jankowiak et al. (2020a). As for PPGPR, the objective function coincides to regularized maximum likelihood estimation

$$\mathcal{L}_{\text{dspp}} = \sum_{i=1}^{N} \log p_{\text{dspp}}\left(y_i \mid \mathbf{x}_i\right) - \beta_{\text{reg}} \sum \text{KL} \tag{32}$$

The main difference between DSPPs and DGPs is twofold. First, DGPs are trained by approximating the Evidence Lower Bound in Eq. 26 while DSPP is trained via a regularized maximum likelihood objective that targets the predictive distribution of DGPs. Second, in the DGPs the latent function values are sampled while for the DSPP they are parameterized via a learnable quadrature rule. Apart from these differences, the two models are very similar as they make use of the same parameters apart from the quadrature parameters. The total computational complexity of DSPP is given by $\mathcal{O}(DM^3)$, where D refers to the number of GPs in the hidden layers.

4.4 Other Methods

4.4.1 Neural Network Gaussian Process

Neal (1995) showed the equivalence between a single-layered infinitely wide neural network with an i.i.d prior to its parameters and a Gaussian Process. Williams (1996) demonstrated that the i.i.d prior over weights and biases can be substituted with a corresponding GP prior over functions and thus enabling exact Bayesian inference for regression. Lee et al. (2017) extend these works to multi-layer neural networks and create a covariance kernel for GPs that is equivalent to multi-layer infinitely wide neural networks. For a review of the correspondence between single-layer neural networks and GPs, see Neal (1995) and Williams (1996).

I'll briefly review the correspondence between multi-layer neural networks and GPs. Consider an L-layer infinitely wide neural network. The output for the ith component z_i^l at layer l will be

$$z_i^l(x) = b_i^l + \sum_{j=1}^{\infty} W_{ij}^l x_j^l(x), \quad x_j^l(x) = \phi\left(z_j^{l-1}(x)\right), \tag{33}$$

where ϕ is a non-linear function, x_i^l the post-nonlinearity transformation of the output of the previous layers and $W_{i,j}^l$ and b_i^l are the i.i.d weights and biases at layer l. If z_j^{l-1} is a GP, then z^l is a sum of i.i.d terms so that any finite collection $\{z_i^l(x^1), ..., z_i^l(x^k)\}$ will have a joint multivariate Gaussian distribution and $z_i^l \sim GP(0, K^l)$. The covariance function can be recursively calculated via a deterministic function F which only depends on the nonlinearity ϕ:

$$K^l(x, x') = \sigma_b^2 + \sigma_w^2 F_\phi\left(K^{l-1}(x, x'), K^{l-1}(x, x), K^{l-1}(x', x')\right) \tag{34}$$

This can be iteratively computed to obtain K^L for the GP describing the network's final output. The function F can be analytically computed for particular kernels, e.g. ReLU. For others, the authors provide a numerical approximation. In summary, Neural Network Gaussian Process (NNGP) refers to the GP prior induced by an infinitely wide neural network. Thus, a neural network with random initialization can be interpreted as prior over functions. As a deep infinitely wide neural network can be described as a GP, this allows for full exact Bayesian inference. The work by Lee et al. (2017) allows analyzing how neural networks work and revisiting this from a function space of view. Furthermore, this allows examining towards what functions neural networks are biased at initialization. Jacot et al. (2018) build on this and show that the behavior of neural networks during training via gradient descent can be explained by a kernel, the Neural Tangent Kernel. Both kernels can be examined in the neural tangents package (Novak et al. (2020)).

Even though NNGPs have interesting theoretical properties, I will not consider them in my experiments. As I have demonstrated in Chapter 3, scalability is an issue for fully Bayesian training. Lee et al. (2017) used 64 CPUs to analyze datasets with up to 50.000 data points. As I am interested in methods, that can scale to million data points, NNGP will be disregarded for the experiments. Lee et al. (2017) state that they are intending to look into scalability methods. Thus, NNGP may be considered at a future point in time.

4.4.2 Scalable Gaussian Process Regression Using Deep Neural Networks

Similar to DKL, Huang et al. (2015) proposes to use a feature-mapping function as input into a GP. Instead of a DNN, they use a stacked denoising auto-encoder (SDAE). The internal representation of the last layer is used as an explicit feature map for calculating the covariance function. The idea is, that the model can learn a much more meaningful representation of the data through the feature-mapping function of the SDAE. The model training can be divided into two steps. First, the SDAE is pretrained in an unsupervised manner. Afterward, the parameters of the SDAE can be treated as kernel Hyperparameters which are fine-tuned by maximizing the MLL. To reduce the computational complexity, Huang et al. (2015) use the FITC approximation (Quinonero-Candela and Rasmussen (2005)).

4.4.3 Improving Output Uncertainty Estimation and Generalization in Deep Learning via Neural Network Gaussian Processes

Iwata and Ghahramani (2017) propose to use a DNN for the mean function of a GP. The idea stems from GPs excelling at local generalization, due to their local interpolation properties. However, GPs fail to generalize in regions where there are no training data. A GP with zero mean predicts zero for test points far from training samples. DNNs on the other hand have good generalization behavior for previously unseen inputs by learning multiple levels of distributed representations. By combining GPs and DNNs in this way, the proposed method can improve generalization performance. The parameters of the GP are trained via SVI while the DNN parameters are optimized via stochastic gradient descent (SGD). The authors alternate between SVI and SGD for each minibatch.

5 Experiments

I will conduct a benchmark study on two insurance tariff datasets and 4 regression tasks taken from the OpenML repository (Vanschoren et al. (2014)) to compare the generalization performance and uncertainty estimation quality of the presented methods.

5.1 Experiment Setup

To evaluate the methods presented in chapter 4, I chose two small-sized datasets (<10k data points), two medium-sized datasets (10-100k data points), and two large-sized datasets (∼500k data points). See A.1 for more details.
Hyperparameter optimization (HPO) via a Tree-structured Parzen Estimator (TPE) was conducted to find the optimal configuration for each method and each dataset. To ensure unbiased results, nested resampling was used to find the optimal Hyperparameter configurations (Bischl et al. (2012)). I used 5-Fold cross-validation (CV) for the outer loop to estimate the generalization performance. Mean-squared error (MSE), mean-absolute error (MAE), maximum absolute error (MaAE), and negative log-likelihood (NLL) were chosen as evaluation criteria. For the insurance datasets, the most important metric is the MaAE as each insurance tariff prediction can not deviate more than a certain amount from its true value due to regulatory requirements. For the inner loop, I used 4-Fold cross-validation for the small and medium datasets for more stable results, while a simple train-test split was used for the large datasets to find an optimal Hyperparameter configuration. Additionally, I implmented early stopping with the MSE criterion. The MSE criterion was chosen over the MaAE criterion, as it considers the whole validation set while strongly punishing outliers. The fear was, as MaAE only considers one observation from the validation set, the TPE would lose out on valuable information. A time-based pruner was also used to speed up the HPO and to favor good and not too-costly architectures. Additionally, Hyperband was used for the large datasets to speed up the HPO even further.
For each of the 5 outer loop iterations, 100 HPO trials were conducted. This means that for each model and each dataset a total of 500 Hyperparameter configurations were evaluated. For the inner loop, all models were trained for a maximum of 400 epochs with 5 early stopping iterations based on the MSE. Once an optimal configuration was found, all models, except DKL, were retrained for a maximum of 1000 iterations with 10 early stopping iterations based on the MLL or ELBO. As DKL suffered from stability issues, the model was trained for the same number of epochs as the configuration found by the

inner HPO loop. This will be discussed in detail in Section 6. For more details on the HPO Study setup, see A.2.

Deep Kernel Learning: I slightly altered the architecture from the original architecture proposed by Wilson et al. (2016). The feature extractor search space was taken from the Auto-Pytorch Tabular paper (Zimmer et al. (2021)). Furthermore, Wilson et al. (2016) restricts the output size of the feature extractor to two neurons. I extend this up to four neurons, as more output neurons showed great potential during testing. However, the number of grid points for DKL grows exponentially with the number of output neurons, which is why I had to reduce the grid size for a larger output neuron size. In addition to the RBF and SM Kernel, I considered the Matern Kernel with $nu \in \{0.5, 1.5\}$ for my search space.

DKL training was limited by GPU memory constraints, as it does not allow for mini-batching. As at the beginning of this thesis, DKL was considered one of the more promising methods, I decided to not include the whole r1_08 dataset and instead only a subsample of up to 600.000 data points.

Variational Deep Kernel Learning (VDKL): To reduce the memory issues of DKL, I replaced the KISS-GP kernel approximation with the variational inference strategy proposed by Hensman et al. (2015). The inducing points are now learned, instead of being placed over a grid. This follows the framework proposed by Bradshaw et al. (2017). VDKL thus is a sparse variational GP with a DNN as a feature extractor. Up to 20 output neurons are considered for the DNN since I was no longer restricted by the memory issues induced by the grid size in DKL. As I ran into difficulties combining VDKL with the SM kernel, the SM kernel was removed from the search space. The rest remains as in DKL above.

Deep Gaussian Process: The search space for the DGP implementation differs slightly from the models evaluated in Salimbeni and Deisenroth (2017). The structure of my DGP architectures is governed by two variables: n_gp_layers refers to the number of hidden layers in the DGP model and n_gp_layers represents the number of GPs in each GP hidden layer. In this architecture, the number of GPs is the same in each layer.

Neural Network Deep Gaussian Process (NNDGP): This architecture was inspired by Jankowiak et al. (2020a), where they combined the idea of DKL and DGPs. A DNN was used as a feature extractor and then fed into a DGP architecture. The neural network parameters are jointly trained with the DGP parameters with the doubly stochastic variational inference framework presented in 4.2.3.

Deep Sigma Point Process: The search space design follows the one by DGPs, which

means that in every layer the number of GPs is the same. For the rest of the architecture, the search space of by Jankowiak et al. (2020a) was used.

Baselines: As baselines with strong generalization performance on tabular datasets Random Forest and XGBoost were chosen. As a bayesian deep learning baseline, Deep Ensembles (DE) by Lakshminarayanan et al. (2017) was chosen. Here, the network architecture search space follows, like DKL, the Auto-Pytorch Tabular search space. I use random initialization of the neural network parameters along with random shuffling of the data points.

For details about the Hyperparameter search space, see A.3.

All model implementations were based on PyTorch and GPyTorch (Gardner et al. (2018)). The code can be found on https://github.com/likai97/GP-NN-Hybrids. The Hyperparameter search was done using the Optuna Hyperparameter Optimization Framework (Akiba et al. (2019)). For all GP-based models, both features and outputs were scaled using normalization. For the Deep Ensemble, the features were scaled using standardization. All models were trained using the AdamW optimizer (Loshchilov and Hutter (2017)).

All experiments were performed on a Linux machine with one NVIDIA Tesla V100 GPU and 16 GB of memory.

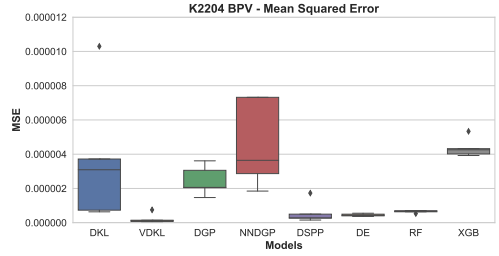
5.2 Benchmark Results

The Benchmark Results are summarized in Figures 2 - 4. Individual results can be found in Appendix A.4. NNDGP was only evaluated on small and medium-sized datasets, due to its extremely slow training speed and poor generalization performance.

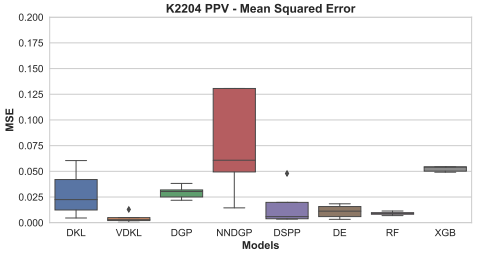
5.2.1 K2204 and R1_08 Results

My results in Figure 2 show that for the K2204 dataset, VDKL outperforms all other methods in terms of MSE and most importantly MaAE. As this dataset is fairly small, it is not too surprising that all Hyperparameter configurations found for VDKL have a feature extractor with just one hidden layer (see Appendix A.4.2). For K2204 BPV, DSPP and the DE baseline show almost equally good performance with regard to MSE and MaAE. On the K2204 PPV dataset, the second-best model classes are DSPPs with regard to MaAE. DE fall slightly behind and show equally good performance with DKL and RF. DKL and NNDGP show highly variable results over all 5 outer CV-splits, which are especially pronounced for NNDGP. This may be the result of the model classes' susceptibility to

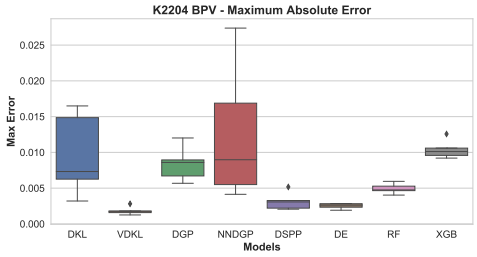
overfitting and stability issues, which I am going to explore in more detail in Section 6. The DGP results show less variability, but its performance falls behind VDKL, DSPP, and DE in both metrics. Surprisingly, the XGB baseline does not show good performance and falls behind all methods with regard to MSE.



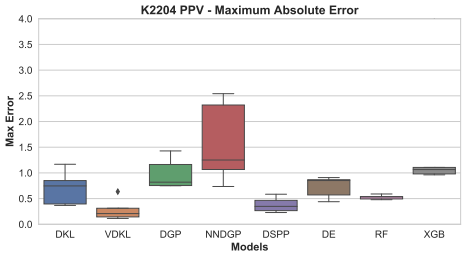
(a) K2204 BPV - Mean Squared Error



(b) K2204 PPV - Mean Squared Error



(c) K2204 BPV - Maximum Absolute Error

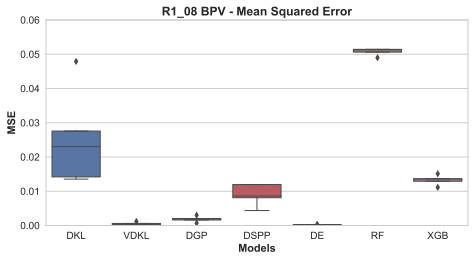


(d) K2204 PPV - Maximum Absolute Error

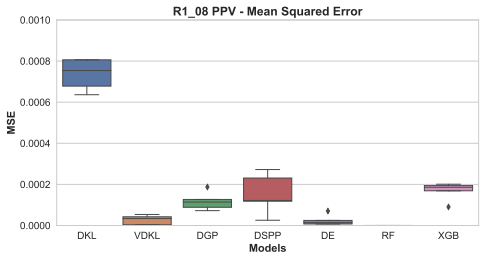
Figure 2: Benchmark Results for the K2204 datasets. Figure a) and b) display the Mean Squared Error over the 5 nested resampling splits. Figure c) and d) exhibit the Maximum Absolute Error (a lower score is better). VDKL, DSPP, and DE show great performance for both metrics.

The results differ slightly for the R1_08 dataset. Note, that the results in Figure 3 are not for the entire R1_08 dataset but instead a subset of 600K randomly sampled data points. The boxplots show that for the R1_08 BPV dataset the best-performing models are VDKL, DGP, and DE with respect to MSE and MaAE. DSPP, which showed great results on the K2204 datasets, falls short in predictive performance.
As VDKL, DGP, and DE showed great results over both BPV and PPV, these methods were applied to the entire R1_08 dataset. The results are displayed in table 1. Due to time constraints, only a single train/test split was conducted using the best configuration found over the subset of R1_08. Each model was trained for a maximum of 400 epochs with 3 early stopping iterations. The results for the full dataset however are very similar to the results on the subset of R1_08, indicating that the results and configurations are

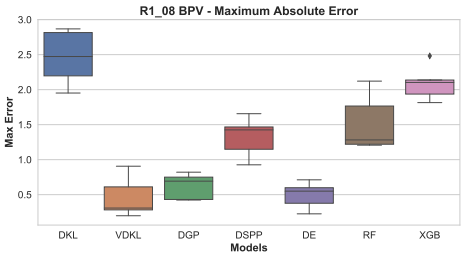
transferable. Even though DE achieved better predictive results compared to VDKL, they required upwards of 7 times the training time. Further investigation might go into training VDKL for more epochs with longer patience for early stopping to see whether better results can be achieved.



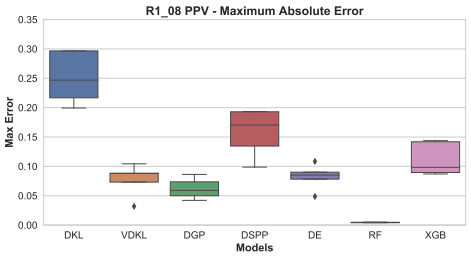
(a) R1_08 BPV - Mean Squared Error



(b) R1_08 PPV - Mean Squared Error



(c) R1_08 BPV - Maximum Absolute Error



(d) R1_08 PPV - Maximum Absolute Error

Figure 3: Benchmark Results for the R1_08 datasets. Figure a) and b) display the Mean Squared Error over the 5 nested resampling splits. Figure c) and d) exhibit the Maximum Absolute Error (a lower score is better). VDKL, DGP, and DE show great performance for both metrics.

	R1_08 BPV			**R1_08 PPV**		
	VDKL	**DGP**	**DE**	**VDKL**	**DGP**	**DE**
MSE	0.039	0.0011	**4e-05**	**8.6e-06**	9.3e-05	9.3e-06
MaAE	0.794	0.796	**0.729**	0.052	0.069	**0.041**
NLL	-0.25	-0.329	**-4.35**	-1.882	-1.845	**-4.49**
Train time in s	737	29347	15708	1595	3700	11461

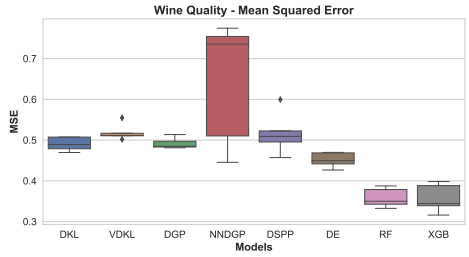
Table 1: Results for the entire R1_08 dataset with 7.6M observations. A single 75:25 train-test split was conducted. The best Hyperparameter configuration for each dataset from Appendix A.4 was chosen. The training time is displayed in seconds.

5.2.2 OpenML Results

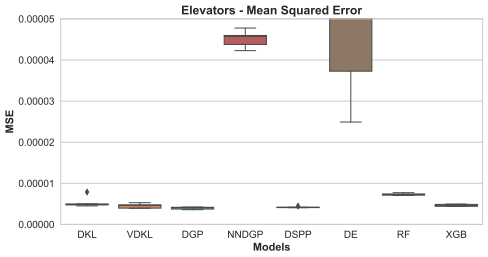
For the OpenML regression datasets, I display the results in Figure 4. XGB shows consistently strong performance with regard to MSE over all four datasets in contrast to its poor performance on insurance tariff data. RF produces almost equally good performance except on the Year dataset.

For the small Wine Quality dataset, all GP methods demonstrate similar performance except for NNDGP. Unsurprisingly, some architecture found for DGPs revert to a standard GP (see Appendix A.4.3). A similar picture arises for the Elevators dataset with the small exception of the poor performance by the DE baseline. For the Diamond dataset, DGPs and DSPPs exhibit fairly stable results, equalling the performance of RF and XGB. For the Year dataset, VDKL outperforms all GP methods and only falls behind XGB. DKL shows quite stable results here, exceeding DGP, DSSP, and DE.

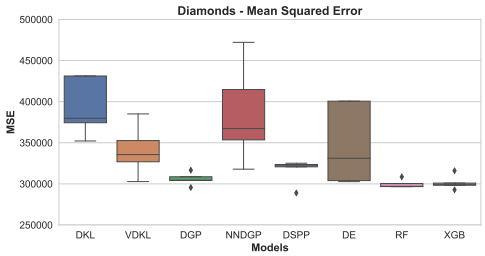
Comparing the GP-based models and DE for the Wine and Year dataset to the Deep Ensemble Results achieved by Lakshminarayanan et al. (2017), they fall behind slightly on the Wine dataset, but VDKL outperforms their results on the Year dataset.



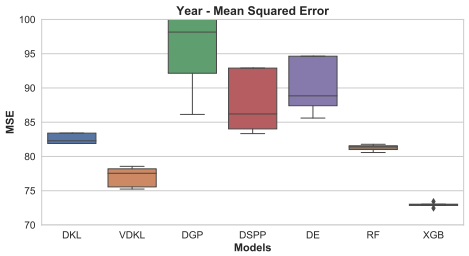
(a) Wine Quality (N=6497, D=11)



(b) Elevators (N=16599, D=18)



(c) Diamonds (N=53940, D=9)

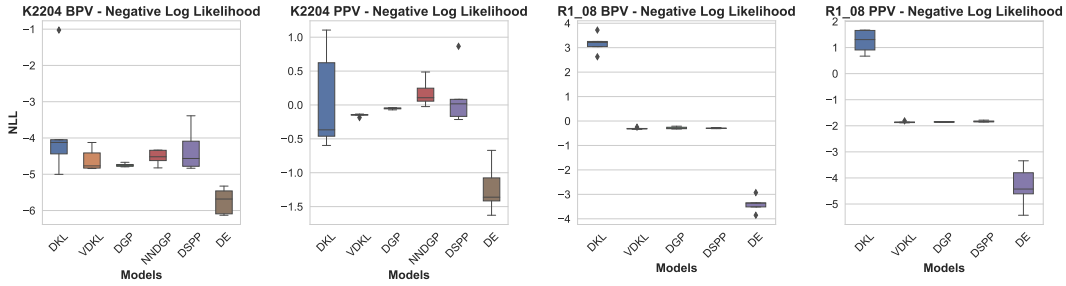


(d) Year (N=515345, D=90)

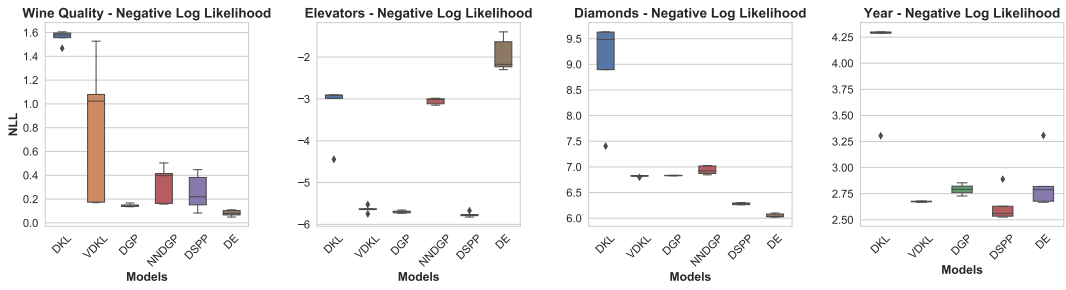
Figure 4: Mean Squared Error results over 5 nested resampling splits for the OpenML regression datasets Wine, Elevators, Diamonds, and Year (a lower score is better).

5.2.3 Negative Log Likelihood Comparison

Proper scoring rules assess the quality of predictive uncertainty (Gneiting and Raftery (2007)). I follow the example of Lakshminarayanan et al. (2017) and use the negative log-likelihood as UQ metric. The results are displayed in Figure 5.
DE show good performance over all datasets except for the Elevator datasets. They show especially good results on the insurance datasets, outperforming all other methods except for the R1_08 BPV dataset. DSPPs achieves lower NLL than DGPs on the Diamonds and Year datasets and show equal results for the other datasets. This is in line with expectation as the idea of DSPP was to directly target the predictive distribution because Havasi et al. (2018) have shown that DGP posterior approximations can degrade the calibration of the test time predictive distribution. VDKL exhibits comparable results to DGP. DKL demonstrates poor NLL performance likely due to overfitting behavior by the feature extractor as demonstrated by Ober et al. (2021) and van Amersfoort et al. (2021), which I am going to discuss in detail in Section 6.



(a) Insurance tariff datasets Negative log likelihood results

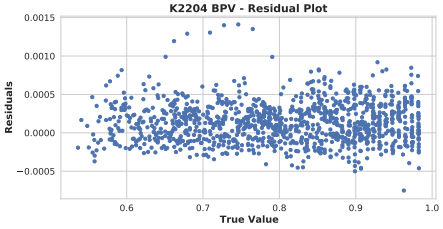


(b) OpenML datasets Negative log likelihood results

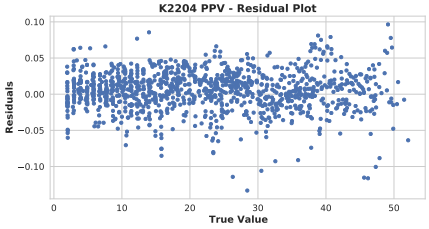
Figure 5: Negative Log-Likelihood results for the K2204, R1_08 and OpenML regression datasets. A lower NLL score is better.

5.3 Model Prediction Analysis

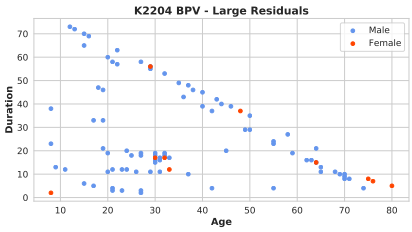
In this section, I will analyze the prediction accuracy of the three best-performing models, VDKL, DSPP, and DE, on the K2204 datasets. All visualizations will only display the results of the first outer nested resampling split.



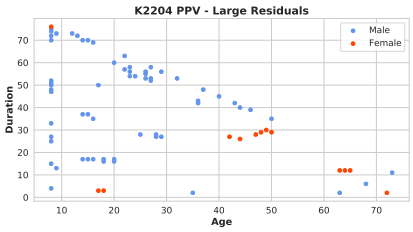
(a) K2204 BPV - Residual Plot



(b) K2204 PPV - Residual Plot



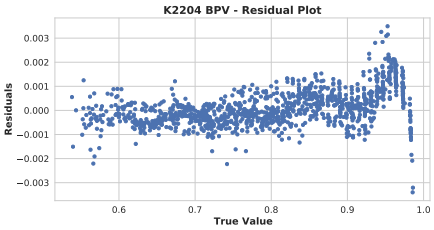
(c) K2204 BPV - Large Residuals



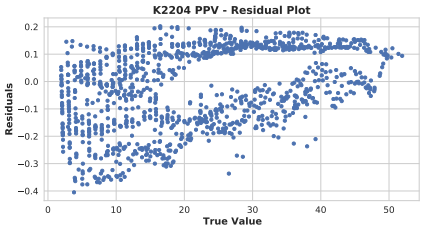
(d) K2204 PPV - Large Residuals

Figure 6: Prediction Analysis for the first HPO study of Variational Deep Kernel Learning for the K2204 BPV and K2204 PPV datasets. Plots a) and b) show the true values vs Residuals. Plot c) shows the data points of the BPV dataset, where the absolute values of the residuals exceeded the threshold of 0.0005. Plot d) shows the data points of the PPV dataset, where the absolute values of the residuals exceeded the threshold of 0.05.

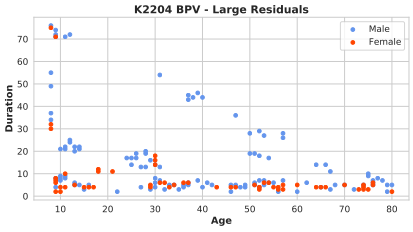
In Figure 6, the Residuals for VDKL are displayed. For both BPV and PPV the residuals seem to be normally distributed around 0. Looking at Figure 6 c) and d), VDKL seems to struggle accurately predict the tariff premium for Male insurance holders. Furthermore, it appears to perform not too well on tariffs, which lie on the edge of possible policyholder's age and contract duration combinations. As K2204 is a fictitious dataset, new data points can be created in these regions to further improve the model.
Inspecting the residual plots in Figure 7, DSPP displays larger residuals for short contract durations on the BPV dataset and for older male policyholders on the PPV dataset.
In Figure 8 c), no clear pattern can be observed for which data points DE is struggling. For PPV, DE seems to struggle on the ridge of possible age and contract durations.



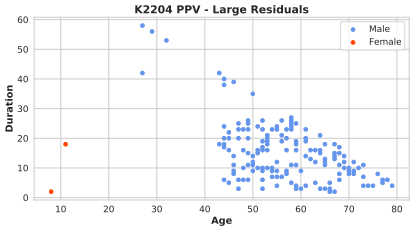
(a) K2204 BPV - Residual Plot



(b) K2204 PPV - Residual Plot

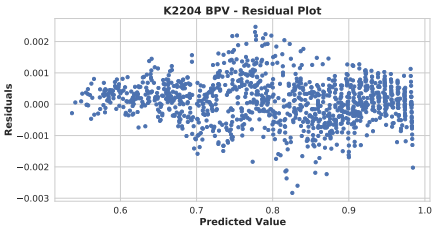


(c) K2204 BPV - Large Residuals

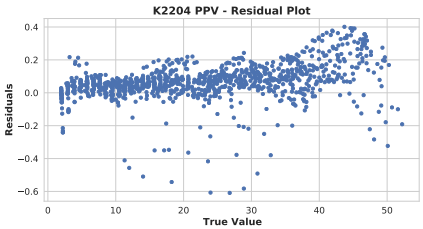


(d) K2204 PPV - Large Residuals

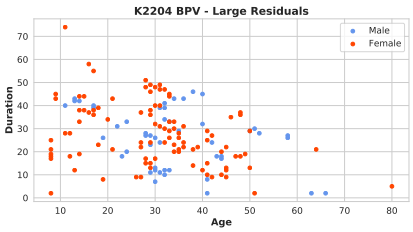
Figure 7: Prediction Analysis for Deep Sigma Point Process on the K2204 BPV and PPV datasets. Plots a) and b) show the true values vs Residuals. Plots c) and d) show the data points, where the absolute values of the residuals exceeded the thresholds 0.001 and 0.1.



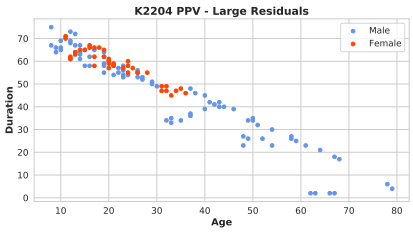
(a) K2204 BPV - Residual Plot



(b) K2204 PPV - Residual Plot



(c) K2204 BPV - Large Residuals



(d) K2204 PPV - Large Residuals

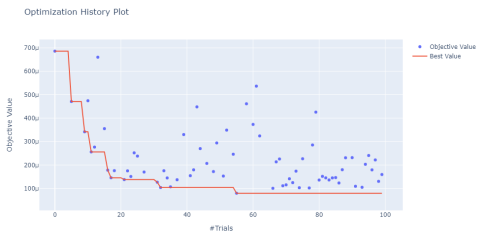
Figure 8: Prediction Analysis for Deep Ensembles on the K2204 BPV and PPV datasets. Plots a) and b) show the true values vs Residuals. Plots c) and d) show the data points, where the absolute values of the residuals exceeded the thresholds 0.001 and 0.2.

5.4 HPO Routine Analysis

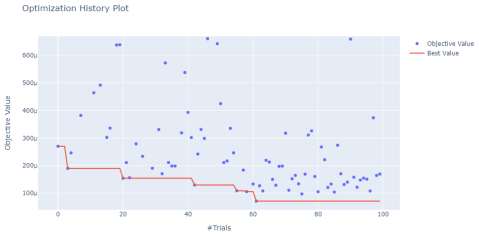
In this section, I will analyze the results of the conducted Hyperparameter Optimization study. I will focus on the K2204 dataset and the best-performing models VDKL, DSPP, and DE. All visualizations will only display the results of the first outer nested resampling split.

In Figure 9, I visualized the optimization history, the Hyperparameter feature importance, and the Hyperparameter search history for the most important Hyperparameters. Optuna calculates the feature importance using the normalized functional analysis of variance method by Hutter et al. (2014). Looking at the optimization history, you can see that the amounts of trials used seems to be sufficient. For both datasets, the minimum is found around 60 trials. An accurately chosen learning rate seems to be of high importance for VDKL. For both datasets, the learning rate search converges to be between 0.001-0.01. Also interesting in Figure 9 c) is that the number of output neurons parameter *n_out* converges to be between 3-7, which makes sense as the K2204 dataset is low-dimensional. Figure 10 displays the study results for the DSPP. An optimal configuration for the K2204 PPV dataset seems to be quickly found. Inspecting the optimization history for K2204 BPV however, it appears the search has not converged yet, indicating that the number of HPO trials might have been too low in this case. For both datasets, one hidden layer looks to give the best results.

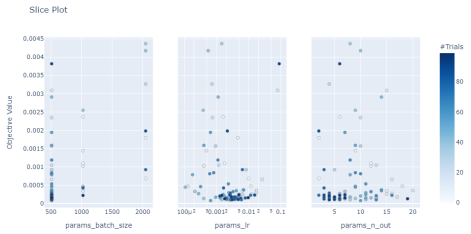
The Deep Ensemble baseline performs fairly well on both K2204 datasets. What is interesting here is that although the HPO converges to an approximately equal maximum number of neurons, for PPV the search converges to a higher number of layers. This is consistent over all nested resampling splits, see A.4.6. The models seem to need a higher capacity for PPV, even though the formula for PPV only consists of one component (see Section 2), while the formula for BPV consists of two separate components.



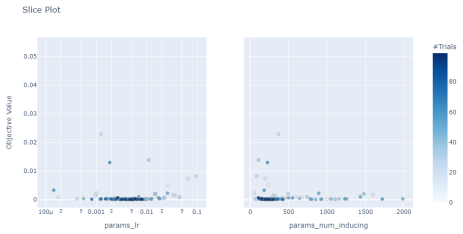
(a) K2204 BPV - Optimization History



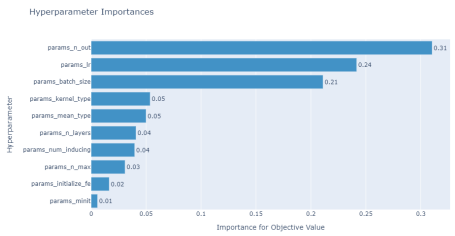
(b) K2204 PPV - Optimization History



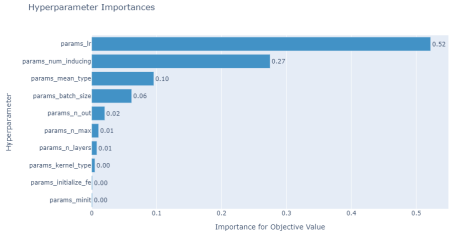
(c) K2204 BPV - Hyperparameter Search History



(d) K2204 PPV - Hyperparameter Search History

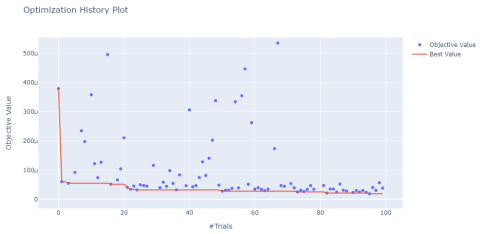


(e) K2204 BPV - Hyperparameter Feature Importance

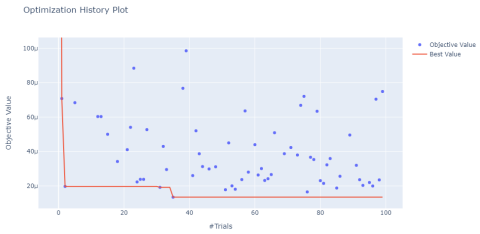


(f) K2204 PPV - Hyperparameter Feature Importance

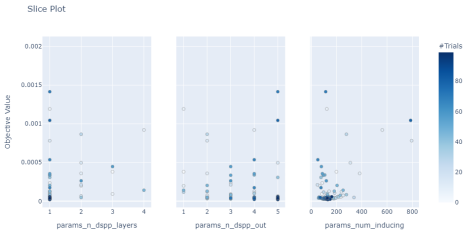
Figure 9: Results for the first HPO study of Variational Deep Kernel Learning on the K2204 BPV and PPV datasets. Figures a) and b) display the average Mean Squared Error over four inner CV splits. Figures c) and d) exhibit the Hyperparameters evaluated over time and the MSE the configuration scored. Figures e) and f) show the normalized feature importance of the Hyperparameters calculated using the normalized functional analysis of variance.



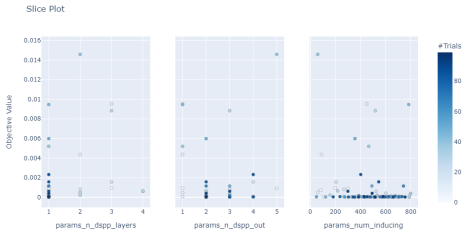
(a) K2204 BPV - Optimization History



(b) K2204 PPV - Optimization History

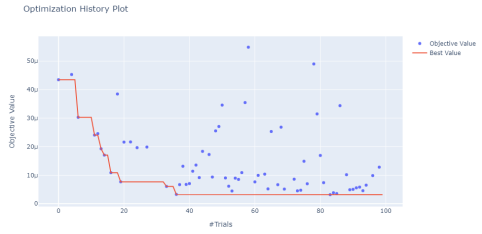


(c) K2204 BPV - Search History

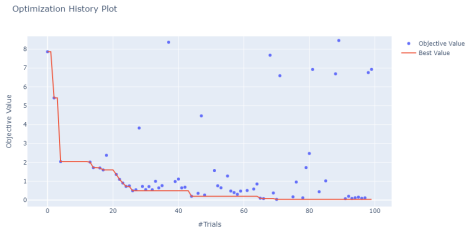


(d) K2204 PPV - Search History

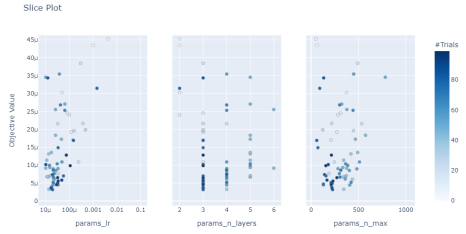
Figure 10: Results for the first HPO study of Deep Sigma Point Process on the K2204 BPV and PPV datasets. Figures a) and b) display the average MSE for the evaluated configurations. Figures c) and d) exhibit the evaluated Hyperparameters.



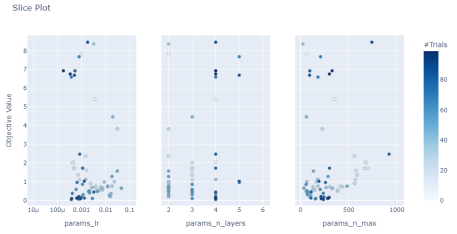
(a) K2204 BPV - Optimization History



(b) K2204 PPV - Optimization History



(c) K2204 BPV - Search History



(d) K2204 PPV - Search History

Figure 11: Results for the first HPO study of Deep Ensembles for the K2204 BPV and K2204 PPV datasets. Figures a) and b) display the average MSE for the evaluated configurations. Figures c) and d) exhibit the evaluated Hyperparameters.

6 Model Discussion

In this section, I am gonna share the lessons I learned from the benchmark study and highlight the advantages and disadvantages of the models concerning performance, stability, scalability, uncertainty estimation, and ease of use.

Deep Kernel Learning: The DKL benchmark results had higher variance and consistently displayed worse performance on the insurance tariff datasets compared to the other GP-based methods. Wilson et al. (2016) claim that DKL does not need regularization as it is guarded against overfitting due to the model complexity being automatically calibrated through the optimization of the MLL. However, looking at Figure 2, DKL training is rather volatile. Ober et al. (2021) agrees that the automatic calibration holds when selecting a small number of Hyperparameters but that for models such as DKL with many Hyperparameters MLL training can encourage overfitting. Strategies to address the overfitting issue include regularization of the feature extractor (van Amersfoort et al. (2021); Liu et al. (2020)) and fully bayesian training (Ober et al. (2021)).

Even with displaying competitive results on the OpenML datasets, scalability remains an issue for DKL due to memory constraints. The KISS-GP approximation costs about $\mathcal{O}(N)$ storage, thus limiting the applicability of DKL by the underlying GPU hardware. Training DKL on datasets with up to a million data points was possible, but going farther than that meant running into CUDA memory errors as DKL does not allow for mini-batch training. For this reason, the aforementioned experiments were not trained on the whole R1_08 dataset with 7.6M data points and instead on just a subsample of 600k data points.

Even though DKL was rather straightforward to implement using the GPytorch package, the models proved difficult to train with two potential points of failure. In contrast to the remaining methods, the calculation of the MLL could break down during training, see Figure 12. Near the minimum of the MLL, the in-built GPytorch MLL would once in a while return a NaN value. Training further would often lead to substantially worse performance, as can be seen by the oscillating test loss curve. Therefore, I implemented an extra early stopping criterion to stop training once a NaN value was reached.

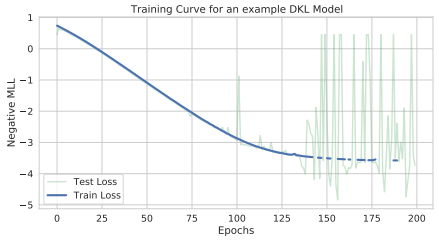


Figure 12: Example training curve for a DKL Model. The blanks in the training loss curve are areas where the calculation of the MLL failed for DKL.

The second issue also revolves around the computation of the MLL. Gardner et al. (2018) use a modified batched version of linear conjugate gradients for the computations of the MLL and its derivatives in the GPytorch package. Occasionally during my HPO loop, the conjugate gradient method would not converge, slowing down training and degrading results.

Variational Deep Kernel Learning: VDKL showed consistently good predictive results over all datasets and outperformed the other GP-based method on the insurance tariff datasets. The variability in the VDKL results is far less than compared to the DKL results. This most likely stems from using stochastic mini-batching as this can help mitigate the overfitting issue for DKL as found by Ober et al. (2021).
Examining the UQ capabilities of DKL and VDKL, it regularly performs worse than the DE baseline. The issue is not as pronounced for VDKL as for DKL most likely due to the stochastic mini-batch training. This is in line with research by Ober et al. (2021) and van Amersfoort et al. (2021) which show these models to underperform in uncertainty estimation. van Amersfoort et al. (2021) show that for certain feature extractors, out-of-distributions data points can get mapped close to representations of in-distribution data points. They propose to use a bi-Lipschitz constraint on the feature extractor as this results in a feature representation that is sensitive to changes in the input but also generalizes due to smoothness.
As VDKL relies on a sparse approximation of the kernel matrix through inducing points and thus allows for mini-batching, memory constraints were not as big of an issue compared to DKL. In addition to that, the predictive performance for the R1_08 dataset was competitive with the DNN-based DE, even with selecting a small number of inducing points compared with the number of observations. The training time in seconds is displayed in Figure 13. Surprisingly, the training speed for the R1_08 datasets where roughly the same as for the 10 times smaller Diamonds dataset. The Year dataset with a comparable number of observations to the R1_08, but with 10 times the variable took 2h hours in the maximum. Comparing the training speed on the

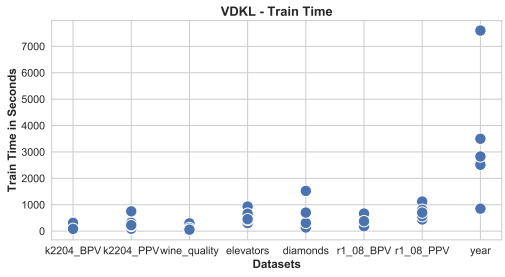


Figure 13: Train time of the final configurations for Variational Deep Kernel Learning.

full R1_08 dataset to DGP in Table 1, VDKL achieved better results concerning MaAE in a much shorter amount of time.
The implementation for VDKL was straightforward. Compared to DKL, VDKL proved much easier to train and did not suffer from the same stability issues as DKL. The architecture of the feature extractor is highly dependent on the task at hand and needs therefore to be chosen carefully. Pre-training of the feature extractor can improve results.

Deep Gaussian Process: The benchmark results for DGPs were highly competitive with VDKL on the R1_08 dataset and the OpenML Regression tasks. Only for the K2204 dataset, the performance lags behind. This supports the findings of Bui et al. (2016) that DPGs, through the stacking of the GPs, can automatically construct kernels that work well for a given task and can counteract the damage to the representational power through sparse approximations
The uncertainty estimation capabilities are on par with VDKL but are generally worse than DE. Havasi et al. (2018) attributes this to the uni-modal Gaussian approximation of the posterior distribution. In their research, they show that the posterior distribution is non-Gaussian for every dataset they examine and therefore propose a Hamiltonian Monte Carlo based sampling technique.
The training speed for DGPs is displayed in Figure 14. For the small and medium-sized datasets, the training speed can be as much as twice as high compared to VDKL. For the subset of the R1_08, the differences in training speed are even more pronounced with DGP taking multiple hours compared to VDKL which gets similar or better performance in a much shorter amount of time. The same holds true for the entire R1_08 dataset in Table 1.

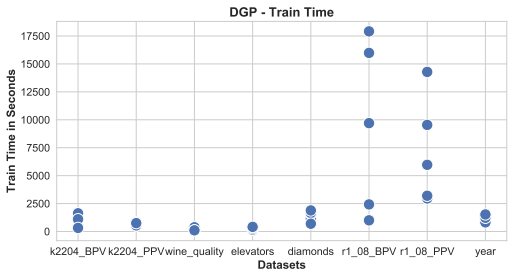


Figure 14: Train time of the final configurations for Deep Gaussian Processes.

The design of the search space for DGP required many iterations as some DGP architectures required a substantial amount of memory. The search space had to be fine-tuned with respect to the maximum depth, width, and number of inducing points per GP. Damianou and Lawrence (2013), Salimbeni and Deisenroth (2017), Bui et al. (2016) and Havasi et al. (2018) only used the RBF kernel in their experiments. The introduction of the Matern Kernel improved the

predictive results on the small and medium-sized datasets. All in all, the DGP was quite pleasant to train with regard to stability and predictive performance with the exception of training speed.

Neural Network Deep Gaussian Process: NNDGPs predictive performance has shown the highest variability of all models. Contrary to the experiment results found by Jankowiak et al. (2020a), that a feature extractor can improve the predictive performance of a DGP, my results show that it can even worsen it.
NNDGP has been highly difficult to train compared to the other methods. One reason could be the immense amount of parameters, as it combines DGP and VDKL parameters in one model. Glancing back at the benchmark results in Figure 2 and 4, the best NNDGP result can be on par or slightly better than the best DGP result, indicating the potential as found by Jankowiak et al. (2020a). The authors trained their model with multiple restarts and only fully trained the model with the best training NLL. I did not utilize multiple restarts. Instead, I used random initialization of inducing points as in DGPs and pretraining of the feature extractor as in VDKL. Less variable results might have been found with a better initialization strategy and also imploring the multiple restart strategy.
Due to the highly variable results and even slower training speed than DGPs, NNDGP was not benchmarked for the R1_08 and Year datasets.

Deep Sigma Point Process: DSPPs were introduced as a parametric model inspired by the compositional structure of DGPs. They showed competitive results with VDKL on the small and medium-sized datasets but fell behind DGPs and VDKL in predictive performance on the R1_08 and Year datasets.
As DSPP directly targets the predictive distribution of DGPs in the training objective, they show better-calibrated uncertainty estimates on the Diamonds and Year dataset and equal results for the other datasets. They still fall short of the estimates by the DE baseline.
Figure 15 exhibits the training speed for DSPP. For the small and medium-sized datasets, the training time is comparable with DGP. For the R1_08 and Year datasets, DSPPs have faster training speed but worse generalization performance compared to DGPs.
Due to the similarity to DGPs, the same search space was used at first. However, DSPP proved to be even more memory intensive than DGPs, causing CUDA memory in my early trials. By reducing the number of inducing points, the number of GP layers, and

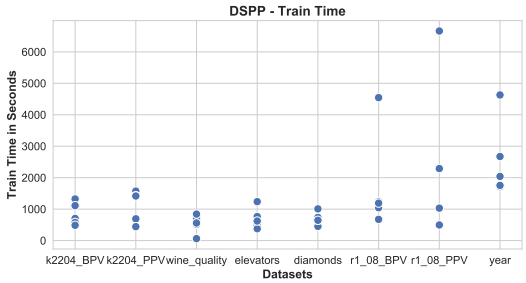


Figure 15: Train time of the final configurations for Deep Sigma Point Process.

batch size I landed on a suitable search space. Altogether, DSPP were stable in their training and proved to have a good performance on small and medium-sized datasets.

7 Conclusion

I reviewed the current state of Gaussian Process and neural network hybrid models for regression tasks and their application for PAS migration. I then conducted a benchmark study on two insurance tariff datasets and four OpenML regression tasks and compared the results to a Deep Ensemble baseline.

I found that VDKL showed great predictive performance on both the insurance datasets and OpenML regression tasks while being significantly faster compared to the other examined methods given the same number of epochs and patience. It showed good uncertainty estimation capabilities on the OpenML regression tasks but fell behind DE on the insurance datasets. DGP showed good performance on the R1_08 and OpenML datasets but struggled on the K2204 dataset. DSPP exhibited good performance in both predictive performance and uncertainty estimation on the small and medium-sized regression tasks but performed worse on R1_08 and the Year dataset. DE showed great predictive performance with superior uncertainty estimates compared to the GP-based methods on the insurance datasets.

All in all, Variational Deep Kernel Learning and Deep ensembles should be further explored in their application to insurance tariff migration due to their great predictive performance, scalability, and good uncertainty estimates. Further research may go into more recent ensemble methods, such as Neural Ensemble Search by Zaidi et al. (2021).

References

Akiba, T., Sano, S., Yanase, T., Ohta, T. and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework, *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Bengio, Y., Delalleau, O. and Roux, N. (2005). The curse of highly variable functions for local kernel machines, *Advances in neural information processing systems* **18**.

Bischl, B., Mersmann, O., Trautmann, H. and Weihs, C. (2012). Resampling methods for meta-model validation with recommendations for evolutionary computation, *Evolutionary computation* **20**(2): 249–275.

Bradshaw, J., Matthews, A. G. d. G. and Ghahramani, Z. (2017). Adversarial examples, uncertainty, and transfer testing robustness in gaussian process hybrid deep networks, *arXiv preprint arXiv:1707.02476* .

Bui, T., Hernández-Lobato, D., Hernandez-Lobato, J., Li, Y. and Turner, R. (2016). Deep gaussian processes for regression using approximate expectation propagation, *International conference on machine learning*, PMLR, pp. 1472–1481.

Calandra, R., Peters, J., Rasmussen, C. E. and Deisenroth, M. P. (2016). Manifold gaussian processes for regression, *2016 International Joint Conference on Neural Networks (IJCNN)*, IEEE, pp. 3338–3345.

Cutajar, K., Bonilla, E. V., Michiardi, P. and Filippone, M. (2017). Random feature expansions for deep gaussian processes, *International Conference on Machine Learning*, PMLR, pp. 884–893.

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

Duvenaud, D., Rippel, O., Adams, R. and Ghahramani, Z. (2014). Avoiding pathologies in very deep networks, *Artificial Intelligence and Statistics*, PMLR, pp. 202–210.

Gardner, J. R., Pleiss, G., Bindel, D., Weinberger, K. Q. and Wilson, A. G. (2018). Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration, *Advances in Neural Information Processing Systems*.

Gneiting, T. and Raftery, A. E. (2007). Strictly proper scoring rules, prediction, and estimation, *Journal of the American statistical Association* **102**(477): 359–378.

Havasi, M., Hernández-Lobato, J. M. and Murillo-Fuentes, J. J. (2018). Inference in deep gaussian processes using stochastic gradient hamiltonian monte carlo, *Advances in neural information processing systems* **31**.

Hensman, J., Fusi, N. and Lawrence, N. D. (2013). Gaussian processes for big data, *arXiv preprint arXiv:1309.6835* .

Hensman, J., Matthews, A. and Ghahramani, Z. (2015). Scalable variational gaussian process classification, *Artificial Intelligence and Statistics*, PMLR, pp. 351–360.

Huang, W., Zhao, D., Sun, F., Liu, H. and Chang, E. (2015). Scalable gaussian process regression using deep neural networks, *Twenty-fourth international joint conference on artificial intelligence*.

Hutter, F., Hoos, H. and Leyton-Brown, K. (2014). An efficient approach for assessing hyperparameter importance, *International conference on machine learning*, PMLR, pp. 754–762.

Iwata, T. and Ghahramani, Z. (2017). Improving output uncertainty estimation and generalization in deep learning via neural network gaussian processes, *arXiv preprint arXiv:1707.05922* .

Jacot, A., Gabriel, F. and Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks, *Advances in neural information processing systems* **31**.

Jankowiak, M., Pleiss, G. and Gardner, J. (2020a). Deep sigma point processes, *Conference on Uncertainty in Artificial Intelligence*, PMLR, pp. 789–798.

Jankowiak, M., Pleiss, G. and Gardner, J. (2020b). Parametric gaussian process regressors, *International Conference on Machine Learning*, PMLR, pp. 4702–4712.

Kahlenberg, J. et al. (2018). *Lebensversicherungsmathematik*, Springer.

Kingma, D. P., Salimans, T. and Welling, M. (2015). Variational dropout and the local reparameterization trick, *Advances in neural information processing systems* **28**.

Kirstein, M., Sommer, D. and Eigel, M. (2022). Tensor-train kernel learning for gaussian processes, *Conformal and Probabilistic Prediction with Applications*, PMLR, pp. 253–272.

Lakshminarayanan, B., Pritzel, A. and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles, *Advances in neural information processing systems* **30**.

LeCun, Y., Bengio, Y. and Hinton, G. (2015). Deep learning, *nature* **521**(7553): 436–444.

Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J. and Sohl-Dickstein, J. (2017). Deep neural networks as gaussian processes, *arXiv preprint arXiv:1711.00165* .

Liu, F., Xu, W., Lu, J., Zhang, G., Gretton, A. and Sutherland, D. J. (2020). Learning deep kernels for non-parametric two-sample tests, *International conference on machine learning*, PMLR, pp. 6316–6326.

Loshchilov, I. and Hutter, F. (2017). Decoupled weight decay regularization, *arXiv preprint arXiv:1711.05101* .

msg life (2022). Machine learning & migration in life insurance.
URL: *https://www.actuview.com/video/Machine-Learning-amp-Migration-in-Life-Insurance/73163dd39ec87a960565e7ff2bd43a82*

Neal, R. M. (1995). *BAYESIAN LEARNING FOR NEURAL NETWORKS*, PhD thesis, University of Toronto.

Novak, R., Xiao, L., Hron, J., Lee, J., Alemi, A. A., Sohl-Dickstein, J. and Schoenholz, S. S. (2020). Neural tangents: Fast and easy infinite neural networks in python, *International Conference on Learning Representations*.
URL: *https://github.com/google/neural-tangents*

Ober, S. W., Rasmussen, C. E. and van der Wilk, M. (2021). The promises and pitfalls of deep kernel learning, *Uncertainty in Artificial Intelligence*, PMLR, pp. 1206–1216.

Quinonero-Candela, J. and Rasmussen, C. E. (2005). A unifying view of sparse approximate gaussian process regression, *The Journal of Machine Learning Research* **6**: 1939–1959.

Rasmussen, C. E. (2003). Gaussian processes in machine learning, *Summer school on machine learning*, Springer, pp. 63–71.

Salimbeni, H. and Deisenroth, M. (2017). Doubly stochastic variational inference for deep gaussian processes, *Advances in neural information processing systems* **30**.

Sterbetafeln-DAV (2008).
URL: *https://aktuar.de/Dateien_extern/Aktuelle%20Veranstaltungen/PRF2927_Sterbetafeln.pdf*

van Amersfoort, J., Smith, L., Jesson, A., Key, O. and Gal, Y. (2021). On feature collapse and deep kernel learning for single forward pass uncertainty, *arXiv preprint arXiv:2102.11409* .

Vanschoren, J., Van Rijn, J. N., Bischl, B. and Torgo, L. (2014). Openml: networked science in machine learning, *ACM SIGKDD Explorations Newsletter* **15**(2): 49–60.

Williams, C. (1996). Computing with infinite networks, *Advances in neural information processing systems* **9**.

Wilson, A. and Adams, R. (2013). Gaussian process kernels for pattern discovery and extrapolation, *International conference on machine learning*, PMLR, pp. 1067–1075.

Wilson, A. G., Gilboa, E., Nehorai, A. and Cunningham, J. P. (2014). Fast kernel learning for multidimensional pattern extrapolation, *Advances in neural information processing systems* **27**.

Wilson, A. G., Hu, Z., Salakhutdinov, R. and Xing, E. P. (2016). Deep kernel learning, *Artificial intelligence and statistics*, PMLR, pp. 370–378.

Wilson, A. and Nickisch, H. (2015). Kernel interpolation for scalable structured gaussian processes (kiss-gp), *International conference on machine learning*, PMLR, pp. 1775–1784.

Zaidi, S., Zela, A., Elsken, T., Holmes, C. C., Hutter, F. and Teh, Y. (2021). Neural ensemble search for uncertainty estimation and dataset shift, *Advances in Neural Information Processing Systems* **34**: 7898–7911.

Zimmer, L., Lindauer, M. and Hutter, F. (2021). Auto-pytorch: Multi-fidelity metalearning for efficient and robust autodl, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **43**(9): 3079–3090.

A Appendix

A.1 Datasets

In this table, you can see all datasets used for the benchmark study, the number of observations n, and the number of variables p. K2204 and R1_08 are insurance tariff datasets. Wine, Elevators, Diamonds, and Year are taken from `openml.org`.

Dataset Name	OpenML Dataset ID	n	p
K2204	/	5839	3
Wine	287	6497	11
Elevators	216	16599	18
Diamonds	42225	53940	9
Year	44027	515345	90
R1_08_small	/	600000	9
R1_08	/	7496192	9

Table 2: HPO Study Datasets

A.2 Study Set Up

Below, you can see the HPO setup. Each method was evaluated 100 times. The time budget refers to the maximum time a Hyperparameter configuration can be trained before being pruned. For the dataset with less than 100K Observation, 4-Fold CV was used to get more stable results. All models were trained in parallel. For the large datasets, a simple train-test split was used with an additional Hyperband pruner.

Dataset Name	Time Budget	Inner Validation Loop Method	Pruner Type
K2204	10min	4-fold CV	None
Wine	10min	4-fold CV	None
Elevators	15min	4-fold CV	None
Diamonds	30min	4-fold CV	None
Year	45min	0.75/0.25 Train/Test Split	Hyperband
R1_08_small	45min	0.75/0.25 Train/Test Split	Hyperband

Table 3: Study Set Up

A.3 Model Search Space

In the following tables, you can see the Hyperparameter Search Space used in the optuna HPO study for the small and medium sized datasets. Note, that for the larger datasets with more than 500K observations, the search space was restricted slightly for some methods, usually a slightly smaller grid size or number of inducing points. This was done in order to avoid Memory Errors.

Variable Name	Description	Range	Log
n_max	Maximum number of neurons in feature extractor	$\mathcal{U}$(64, 1024)	
n_layer	Number of layers in feature extractor	$\mathcal{U}$(2, 5)	
n_out	Number of output neurons in feature extractor	$\mathcal{U}$(2, 4)	
mean_type	Mean Type	[Constant, Linear]	
kernel_type	Kernel Type	[RBF, Matern(nu=0.5), Matern(nu=1.5), Spectral Mixture]	
num_mixtures	Number of spectral mixtures for Spectral Mixture Kernel	$\mathcal{U}$(2, 8)	
lr	Learning Rate	$\mathcal{U}$(1e-04, 1e-01)	✓
grid_size	Grid Size	If n_out=2: $\mathcal{U}$(10, 400) If n_out=3: $\mathcal{U}$(10, 50) If n_out=4: $\mathcal{U}$(10, 20)	
initialize_fe	Initialize Feature Extractor	[True, False]	

Table 4: Deep Kernel Learning Search Space

Variable Name	Description	Range	Log
n_max	Maximum number of neurons in feature extractor	$\mathcal{U}$(64, 1024)	
n_layer	Number of layers in feature extractor	$\mathcal{U}$(2, 5)	
n_out	Number of output neurons in feature extractor	$\mathcal{U}$(2, 20)	
mean_type	Mean Type	[Constant, Linear]	
kernel_type	Kernel Type	[RBF, Matern(nu=0.5), Matern(nu=1.5)]	
lr	Learning Rate	$\mathcal{U}$(1e-04, 1e-01)	✓
batch_size	Batch Size	[512, 1024, 2048]	
num_inducing	Number of inducing points	$\mathcal{U}$(50, 2000)	✓
initialize_fe	Initialize Feature Extractor	[True, False]	
minit	Inducing point Initialization	[random, kmeans]	

Table 5: Variational Deep Kernel Learning Search Space

Variable Name	Description	Range	Log
n_gp_layers	Number of GP layers	$\mathcal{U}$(1, 5)	✓
n_gp_out	Number of GPs per layer	$\mathcal{U}$(1, 4)	✓
kernel_type	Kernel Type	[RBF, Matern(nu=0.5), Matern(nu=1.5)]	
lr	Learning Rate	$\mathcal{U}$(1e-04, 1e-01)	✓
batch_size	Batch Size	[256, 512, 1024]	
num_inducing	Number of inducing points in each hidden layer	$\mathcal{U}$(50, 1000)	✓
num_samples	Number of likelihood samples	$\mathcal{U}$(2, 15)	

Table 6: Deep Gaussian Process Search Space

Variable Name	Description	Range	Log
n_max	Maximum number of neurons in feature extractor	$\mathcal{U}$(64, 1024)	
n_layer	Number of layers in feature extractor	$\mathcal{U}$(2, 5)	
n_out	Number of output neurons in feature extractor	$\mathcal{U}$(2, 20)	
n_gp_layers	Number of GP layers	$\mathcal{U}(1, 5)$	✓
n_gp_out	Number of GPs per layer	$\mathcal{U}(1, 4)$	✓
kernel_type	Kernel Type	[RBF, Matern(nu=0.5), Matern(nu=1.5)]	
lr	Learning Rate	$\mathcal{U}$(1e-04, 1e-01)	✓
batch_size	Batch Size	[256, 512, 1024]	
num_inducing	Number of inducing points in each hidden layer	$\mathcal{U}$(50, 1000)	✓
num_samples	Number of likelihood samples	$\mathcal{U}$(2, 15)	
initialize_fe	Initialize Feature Extractor	[True, False]	

Table 7: Neural Network Deep Gaussian Process Search Space

Variable Name	Description	Range	Log
n_dspp_layers	Number of DSPP hidden layers	$\mathcal{U}$(1, 5)	✓
n_dspp_out	Number of GPs per hidden layer	$\mathcal{U}$(1, 5)	✓
lr	Learning Rate	$\mathcal{U}$(1e-04, 1e-01)	✓
batch_size	Batch Size	[256, 512, 1024]	
num_inducing	Number of inducing points in each hidden layer	$\mathcal{U}$(50, 800)	✓
num_quadrature_sites	Number of quadrature sites	$\mathcal{U}$(5, 10)	
beta	Beta	[0.01, 0.05, 0.2, 1.0]	

Table 8: Deep Sigma Point Process Search Space

Variable Name	Description	Range	Log
n_max	Maximum number of neurons in feature extractor	$\mathcal{U}$(64, 1024)	
n_layer	Number of layers in feature extractor	$\mathcal{U}$(2, 4)	
lr	Learning Rate	$\mathcal{U}$(1e-05, 1e-01)	✓
batch_size	Batch Size	[256, 512, 1024,2048]	

Table 9: Deep Ensemble Search Space

Variable Name	Description	Range	Log
n_estimators	Number of Trees	$\mathcal{U}(5, 500)$	✓
max_depth	Maximum Tree Depth	$\mathcal{U}(5, 100)$	✓
min_samples_split	Minimum number of samples to split an internal node	$\mathcal{U}(2, 25)$	✓
min_samples_leaf	Minimum number of samples to be at a leaf node	$\mathcal{U}(1, 25)$	
max_features	Number of features to consider when looking for a split	[sqrt, auto]	

Table 10: Random Forest Search Space

Variable Name	Description	Range	Log
num_boost_round	Maximum Number of Boosting Rounds	10000	
max_depth	Maximum Tree Depth	$\mathcal{U}(5, 75)$	
learning_rate	Learning Rate	$\mathcal{U}(0.001, 0.1)$	✓
colsample_bytree	Subsample ratio of columns when constructing each tree	$\mathcal{U}(0.2, 0.6)$	✓
subsample	Subsample ratio of the training instances	$\mathcal{U}(0.4, 0.8)$	✓
reg_alpha	L1 regularization term on weights	$\mathcal{U}(0.01, 10)$	✓
reg_lambda	L2 regularization term on weights	$\mathcal{U}(1e\text{-}08, 10)$	✓
gamma	Gamma	$\mathcal{U}(1e\text{-}08, 10)$	✓
min_child_weight	Minimum sum of instance weight needed in a child	$\mathcal{U}(2, 100)$	✓

Table 11: XGBoost Search Space

A.4 Configurations found by HPO

The best configurations as found by the Hyperparameter Optimization in each split.

A.4.1 Deep Kernel Learning

outer_run	**1**	**2**	**3**	**4**	**5**
n_max	706	166	625	650	225
n_layers	2	2	2	2	5
n_out	4	3	3	4	4
mean_type	linear	constant	linear	constant	linear
kernel_type	rbf	rbf	matern1.5	matern1.5	rbf
lr	0.07858	0.07444	0.07531	0.07798	0.00134
grid_size	13	30	14	19	15
initialize_fe	True	False	False	False	True
MSE	7.3e-07	3.1e-06	6.4e-07	1.03e-05	1.6e-07
MAE	0.0007	0.0013	0.0006	0.002	0.0003
Max Error	0.0032	0.0073	0.0063	0.0149	0.0024
NLL	-4.438	-4.048	-5.003	-4.122	-1.025

Table 12: DKL - K2204 BPV

outer_run	**1**	**2**	**3**	**4**	**5**
n_max	143	354	788	403	717
n_layers	2	2	2	2	2
n_out	3	3	3	3	3
mean_type	linear	constant	constant	linear	linear
kernel_type	matern0.5	matern0.5	matern0.5	matern1.5	matern1.5
lr	0.0926	0.0754	0.0861	0.0708	0.0711
grid_size	37	33	14	16	29
initialize_fe	False	True	False	True	True
MSE	0.022	0.042	0.06	0 0.012	0.005
MAE	0.109	0.158	0.211	0.099	0.055
Max Error	1.168	0.851	0.747	0.367	0.396
NLL	-0.461	0.623	1.105	-0.367	-0.598

Table 13: DKL - K2204 PPV

outer_run	1	2	3	4	5
n_max	940	793	782	554	148
n_layers	3	5	2	5	5
n_out	4	3	4	4	3
mean_type	constant	linear	linear	linear	linear
kernel_type	rbf	rbf	rbf	matern1.5	rbf
lr	0.0008	0.0005	0.0044	0.0017	0.0001
grid_size	17	27	15	20	45
initialize_fe	True	True	True	True	False
MSE	0.489	0.507	0.469	0.507	0.479
MAE	0.538	0.557	0.539	0.544	0.542
Max Error	3.17	3.475	3.293	2.953	3.198
NLL	1.58	1.596	1.468	1.557	1.605

Table 14: DKL - Wine

outer_run	1	2	3	4	5
n_max	645	846	870	504	693
n_layers	4	4	3	5	2
n_out	4	4	3	4	3
mean_type	constant	linear	linear	constant	linear
kernel_type	rbf	rbf	rbf	rbf	rbf
lr	0.0001	0.0001	0.0001	0.0001	0.0752
grid_size	18	11	20	19	42
initialize_fe	True	True	True	True	True
MSE	5e-06	5e-06	5e-06	5e-06	8e-06
MAE	0.0017	0.0016	0.0016	0.0017	0.0019
Max Error	0.019	0.017	0.029	0.012	0.023
NLL	-2.912	-2.904	-2.987	-2.909	-4.446

Table 15: DKL - Elevators

outer_run	1	2	3	4	5
n_max	705	266	711	562	193
n_layers	2	2	2	3	4
n_out	3	3	2	3	3
mean_type	linear	constant	linear	linear	linear
kernel_type	matern0.5	matern1.5	rbf	matern1.5	matern1.5
lr	0.0009	0.0031	0.0305	0.0005	0.0225
grid_size	23	38	347	28	27
initialize_fe	False	True	True	True	False
MSE	553329.75	379790.81	352204.31	374353.68	431235.94
MAE	453.66	346.57	326.19	358.32	345.49
Max Error	12292.01	7269.53	10183.71	5718.19	16212.13
NLL	9.633	9.484	7.406	9.629	8.897

Table 16: DKL - Diamonds

outer_run	**1**	**2**	**3**	**4**	**5**
n_max	724	969	995	571	429
n_layers	4	2	2	4	4
n_out	2	2	2	2	2
mean_type	constant	constant	constant	constant	linear
kernel_type	rbf	matern1.5	matern1.5	matern1.5	rbf
lr	0.0001	0.0488	0.0012	0.0002	0.0001
grid_size	103	48	31	145	186
initialize_fe	True	False	False	True	True
MSE	82.29	152.07	83.41	81.89	81.90
MAE	6.43	10.54	6.40	6.24	6.32
Max Error	69.79	69.12	80.22	70.93	71.05
NLL	4.29	3.31	4.30	4.30	4.29

Table 17: DKL - Year

outer_run	**1**	**2**	**3**	**4**	**5**
n_max	808	748	663	606	343
n_layers	3	5	3	4	5
n_out	2	2	2	2	2
mean_type	linear	constant	constant	linear	constant
kernel_type	matern1.5	matern1.5	rbf	rbf	rbf
lr	0.0393	0.0121	0.0114	0.0003	0.0415
grid_size	95	221	34	54	259
initialize_fe	False	False	True	True	True
MSE	0.014	0.023	0.0028	0.014	0.048
MAE	0.085	0.112	0.129	0.088	0.162
Max Error	2.815	2.473	1.952	2.868	2.197
NLL	2.629	3.255	3.214	3.720	3.044

Table 18: DKL - R1_08 BPV

outer_run	**1**	**2**	**3**	**4**	**5**
n_max	182	1024	1024	511	915
n_layers	2	4	4	2	4
n_out	2	2	2	2	2
mean_type	linear	constant	constant	linear	constant
kernel_type	rbf	rbf	rbf	matern1.5	matern1.5
lr	0.025	0.01	0.01	0.025	0.032
grid_size	141	300	300	116	152
initialize_fe	False	False	False	False	False
MSE	0.0007	0.0007	0.051	0.0008	0.0008
MAE	0.019	0.019	0.137	0.022	0.23
Max Error	0.297	0.199	1.558	0.247	0.217
NLL	0.908	1.656	1.670	1..305	0.671

Table 19: DKL - R1_08 PPV

A.4.2 Variational Deep Kernel Learning

outer_run	**1**	**2**	**3**	**4**	**5**
n_max	645	930	171	963	503
n_layers	2	2	2	2	2
n_out	10	6	13	9	6
mean_type	linear	constant	linear	constant	constant
kernel_type	rbf	rbf	matern1.5	rbf	rbf
lr	0.0006	0.0029	0.0004	0.003	0.002
batch_size	512	512	512	512	512
num_inducing	175	949	52	811	95
initialize_fe	True	True	True	False	True
minit	kmeans	kmeans	random	kmeans	random
MSE	3.49e-08	1.17e-07	7.5e-07	7.46e-08	1.46e-07
MAE	0.00013	0.00024	0.00068	0.0002	0.00029
Max Error	0.0013	0.0018	0.0028	0.0016	0.0018
NLL	-4.412	-4.829	-4.124	-4.769	-4.842

Table 20: VDKL - K2204 BPV

outer_run	**1**	**2**	**3**	**4**	**5**
n_max	945	369	778	236	82
n_layers	2	2	2	2	3
n_out	10	11	15	9	15
mean_type	constant	constant	constant	constant	constant
kernel_type	matern1.5	rbf	matern1.5	matern1.5	rbf
lr	0.004	0.002	0.004	0.004	0.001
batch_size	512	512	512	512	512
num_inducing	299	664	897	1487	396
initialize_fe	True	False	True	False	True
minit	kmeans	random	kmeans	kmeans	kmeans
MSE	0.013	0.003	0.0004	0.002	0.005
MAE	0.072	0.037	0.016	0.037	0.058
Max Error	0.64	0.31	0.11	0.14	0.21
NLL	-0.141	-0.134	-0.148	-0.186	-0.148

Table 21: VDKL - K2204 PPV

outer_run	1	2	3	4	5
n_max	687	860	641	242	723
n_layers	4	3	4	3	3
n_out	19	5	16	10	19
mean_type	constant	constant	constant	constant	constant
kernel_type	rbf	matern0.5	matern1.5	matern1.5	matern1.5
lr	0.0015	0.0001	0.0014	0.003	0.002
batch_size	512	512	512	512	512
num_inducing	916	1419	792	264	58
initialize_fe	True	True	True	False	True
minit	random	random	random	kmeans	random
MSE	0.51	0.55	0.52	0.50	0.51
MAE	0.49	0.54	0.51	0.54	0.54
Max Error	4.08	3.19	3.83	3.19	5.05
NLL	1.023	1.080	1.527	0.170	0.175

Table 22: VDKL - Wine

outer_run	1	2	3	4	5
n_max	854	748	535	350	805
n_layers	3	5	4	4	4
n_out	8	10	4	18	8
mean_type	constant	constant	constant	constant	constant
kernel_type	rbf	rbf	matern1.5	matern1.5	matern1.5
lr	0.0009	0.0005	0.0004	0.0002	0.0006
batch_size	512	512	512	512	512
num_inducing	826	164	1707	55	233
initialize_fe	False	True	True	True	True
minit	kmeans	random	kmeans	kmeans	random
MSE	4e-06	5e-06	5e-06	4e-06	5e-06
MAE	0.0015	0.0017	0.0016	0.0015	0.0016
Max Error	0.014	0.013	0.0196	0.0119	0.0114
NLL	-5.75	-5.64	-5.63	-5.53	-5.65

Table 23: VDKL - Elevators

outer_run	1	2	3	4	5
n_max	827	721	856	368	767
n_layers	3	3	5	3	4
n_out	12	5	14	8	20
mean_type	constant	constant	constant	constant	constant
kernel_type	matern0.5	rbf	rbf	matern0.5	matern0.5
lr	0.0035	0.004	0.0007	0.0008	0.0001
batch_size	512	512	512	512	512
num_inducing	946	99	392	1401	687
initialize_fe	True	True	False	False	False
minit	random	kmeans	kmeans	random	random
MSE	352684.94	326883.94	335607.09	302789.16	385089.13
MAE	311.53	316.42	304.56	293.14	314.08
Max Error	16702.43	5394.03	6976.53	6122.79	12513.64
NLL	6.83	6.83	6.83	6.80	6.82

Table 24: VDKL - Diamonds

outer_run	1	2	3	4	5
n_max	910	765	634	549	306
n_layers	3	2	3	3	2
n_out	12	10	9	11	3
mean_type	constant	constant	linear	constant	linear
kernel_type	matern1.5	matern1.5	matern0.5	matern1.5	matern0.5
lr	0.00026	0.0011	0.00025	0.0001	0.0002
batch_size	2048	1024	1024	1024	1024
num_inducing	1029	1194	494	395	914
initialize_fe	False	False	True	True	False
minit	random	random	random	kmeans	random
MSE	75.24	72.13	79.70	80.88	76.60
MAE	5.92	6.11	6.07	5.96	6.10
Max Error	74.99	72.13	79.70	80.88	76.60
NLL	2.67	2.68	2.68	2.67	2.67

Table 25: VDKL - Year

outer_run	1	2	3	4	5
n_max	1024	141	675	259	688
n_layers	4	3	5	3	4
n_out	2	9	10	14	2
mean_type	constant	constant	constant	constant	constant
kernel_type	rbf	matern1.5	rbf	rbf	rbf
lr	0.01	0.0028	0.017	0.0037	0.021
batch_size	8192	2048	2048	2048	8192
num_inducing	1000	690	375	1052	931
initialize_fe	False	True	False	False	False
minit	random	kmeans	kmeans	random	kmeans
MSE	0.0003	0.0012	0.0005	0.0002	0.0006
MAE	0.012	0.025	0.017	0.009	0.019
Max Error	0.61	0.91	0.31	0.28	0.20
NLL	-0.25	-0.32	-0.32	-0.34	-0.31

Table 26: VDKL - R1_08 BPV

outer_run	1	2	3	4	5
n_max	1024	553	1024	757	366
n_layers	4	4	4	2	2
n_out	8	6	8	8	13
mean_type	constant	constant	constant	linear	linear
kernel_type	rbf	rbf	rbf	matern1.5	matern0.5
lr	0.01	0.0042	0.01	0.032	0.002
batch_size	1024	1024	1024	2048	1024
num_inducing	1000	1889	1000	1708	56
initialize_fe	False	False	False	True	True
minit	random	random	random	kmeans	random
MSE	0.00003	0.000004	0.000004	0.00005	0.00004
MAE	0.005	0.001	0.001	0.006	0.005
Max Error	0.073	0.032	0.104	0.088	0.088
NLL	-1.858	-1.886	-1.887	-1.811	-1.864

Table 27: VDKL - R1_08 PPV

A.4.3 Deep Gaussian Process

outer_run	1	2	3	4	5
n_gp_layers	2	1	2	2	2
n_gp_out	1	4	3	3	1
kernel_type	matern0.5	matern0.5	matern0.5	matern0.5	matern0.5
batch_size	256	256	256	256	512
lr	0.0046	0.0038	0.0049	0.0077	0.0065
num_inducing	571	349	664	413	204
num_samples	7	8	6	15	3
MSE	2e-06	4e-06	1e-06	2e-06	3e-06
MAE	0.001	0.0013	0.0009	0.0011	0.0013
Max Error	0.008	0.012	0.006	0.007	0.009
NLL	-4.78	-4.73	-4.8	-4.76	-4.67

Table 28: DGP - K2204 BPV

outer_run	1	2	3	4	5
n_gp_layers	1	1	2	1	2
n_gp_out	2	1	2	1	4
kernel_type	matern0.5	matern0.5	matern0.5	matern0.5	matern0.5
batch_size	256	256	256	256	512
lr	0.0072	0.0043	0.0089	0.0039	0.0066
num_inducing	673	522	568	755	378
num_samples	13	8	9	7	7
MSE	0.038	0.032	0.022	0.03	0.025
MAE	0.15	0.13	0.11	0.13	0.12
Max Error	1.16	1.43	0.75	0.75	0.82
NLL	-0.057	-0.044	-0.05	-0.07	-0.04

Table 29: DGP - K2204 PPV

outer_run	1	2	3	4	5
n_gp_layers	1	1	1	1	1
n_gp_out	3	1	2	4	4
kernel_type	rbf	matern1.5	rbf	matern1.5	matern1.5
batch_size	256	512	256	256	512
lr	0.0061	0.0019	0.0135	0.0065	0.0083
num_inducing	329	976	51	169	83
num_samples	10	13	12	4	14
MSE	0.497	0.483	0.485	0.513	0.481
MAE	0.556	0.547	0.537	0.552	0.541
Max Error	3.22	3.16	2.81	3.12	3.04
NLL	0.15	0.14	0.14	0.17	0.14

Table 30: DGP - Wine

outer_run	**1**	**2**	**3**	**4**	**5**
n_gp_layers	2	1	1	3	2
n_gp_out	3	3	3	4	3
kernel_type	matern1.5	matern0.5	rbf	matern1.5	matern1.5
batch_size	256	256	256	256	256
lr	0.0038	0.0073	0.0022	0.0077	0.003
num_inducing	152	334	427	104	375
num_samples	12	4	4	4	7
MSE	4e-06	4e-06	4e-06	4e-06	4e-06
MAE	0.0014	0.0015	0.0015	0.0016	0.0014
Max Error	0.01	0.014	0.019	0.011	0.014
NLL	-5.72	-5.66	-5.69	-5.69	-5.73

Table 31: DGP - Elevators

outer_run	**1**	**2**	**3**	**4**	**5**
n_gp_layers	1	2	1	1	2
n_gp_out	3	3	3	2	4
kernel_type	matern1.5	matern1.5	matern1.5	matern1.5	matern0.5
batch_size	512	256	256	512	512
lr	0.0015	0.0019	0.0016	0.0031	0.0085
num_inducing	624	514	722	581	574
num_samples	9	4	6	15	12
MSE	308664.65	304443.78	316624.72	304119.41	295596.94
MAE	298.61	299.84	299.44	309.58	295.48
Max Error	6101.09	5098.9	6514.66	5131.26	6672.86
NLL	6.83	6.83	6.84	6.84	6.83

Table 32: DGP - Diamonds

outer_run	**1**	**2**	**3**	**4**	**5**
n_gp_layers	1	2	2	2	1
n_gp_out	3	3	3	1	3
kernel_type	matern1.5	matern1.5	rbf	rbf	matern1.5
batch_size	512	1024	512	2048	1024
lr	0.011	0.007	0.008	0.013	0.008
num_inducing	59	138	185	297	380
num_samples	15	6	10	12	15
MSE	110.17	86.14	86.87	86.58	86.12
MAE	8.52 6.61	6.48	6.46	6.36	
Max Error	67.99	70.47	76.86	71.62	70.11
NLL	2.85	2.73	2.73	2.73	2.73

Table 33: DGP - Year

outer_run	**1**	**2**	**3**	**4**	**5**
n_gp_layers	3	1	2	1	1
n_gp_out	3	1	2	3	3
kernel_type	rbf	rbf	rbf	rbf	rbf
batch_size	1024	1024	1024	1024	1024
lr	0.0042	0.0022	0.0017	0.0044	0.0025
num_inducing	300	53	769	99	246
num_samples	15	5	7	2	14
MSE	0.002	0.003	0.002	0.002	0.001
MAE	0.033	0.041	0.031	0.033	0.020
Max Error	0.69	0.82	0.43	0.75	0.42
NLL	-0.299	-0.212	-0.326	-0.255	-0.314

Table 34: DGP - R1_08 BPV

outer_run	**1**	**2**	**3**	**4**	**5**
n_gp_layers	1	1	1	1	2
n_gp_out	1	2	2	3	2
kernel_type	rbf	rbf	rbf	rbf	rbf
batch_size	1024	1024	512	512	1024
lr	0.0024	0.0039	0.0017	0.0052	0.0029
num_inducing	478	704	698	406	386
num_samples	7	11	13	10	7
MSE	0.0001	0.00007	0.0001	0.0002	0.00009
MAE	0.009	0.007	0.0088	0.011	0.007
Max Error	0.049	0.042	0.059	0.086	0.074
NLL	-1.851	-1.859	-1.864	-1.842	-1.848

Table 35: DGP - R1_08 PPV

A.4.4 Neural Network Deep Gaussian Process

outer_run	**1**	**2**	**3**	**4**	**5**
n_gp_layers	1	4	1	4	1
n_gp_out	2	3	1	4	1
n_max	189	609	885	555	577
n_layers	4	4	4	2	2
n_out	15	6	20	14	17
batch_size	256	256	256	256	512
lr	0.0029	0.0055	0.002	0.003	0.0029
num_inducing	113	197	490	50	50
num_samples	4	10	12	6	2
MSE	6.9e-06	4.9e-05	3e-06	4e-06	2e-06
MAE	0.0023	0.0054	0.0014	0.0016	0.0009
Max Error	0.009	0.027	0.004	0.006	0.017
NLL	-4.52	-4.33	-4.82	-4.62	-4.34

Table 36: NNDGP - K2204 BPV

outer_run	**1**	**2**	**3**	**4**	**5**
n_gp_layers	3	2	2	2	3
n_gp_out	1	4	2	2	2
n_max	885	119	600	285	258
n_layers	2	4	4	3	5
n_out	9	10	4	9	7
batch_size	512	256	256	256	256
lr	0.0019	0.0072	0.0092	0.0084	0.0043
num_inducing	358	233	202	391	357
num_samples	4	9	11	13	15
MSE	0.0144	0.2957	0.1306	0.0608	0.0494
MAE	0.0931	0.4566	0.2747	0.2032	0.1786
Max Error	1.251	2.542	2.321	1.066	0.736
NLL	-0.02	0.47	0.25	0.11	006

Table 37: NNDGP - K2204 PPV

outer_run	**1**	**2**	**3**	**4**	**5**
n_gp_layers	1	4	1	4	1
n_gp_out	2	3	1	4	1
n_max	189	609	885	555	577
n_layers	4	4	4	2	2
n_out	15	6	20	14	17
batch_size	256	256	256	256	512
lr	0.0029	0.0055	0.002	0.003	0.0029
num_inducing	113	197	490	50	50
num_samples	4	10	12	6	2
MSE	6.9e-06	3.9e-05	7.4e-06	1.05e-05	3.8e-07
MAE	0.0017	0.0054	0.0021	0.0028	0.0005
Max Error	0.02	0.026	0.014	0.008	0.0042
NLL	0.16	0.16	0.41	0.40	0.50

Table 38: NNDGP - Wine

outer_run	**1**	**2**	**3**	**4**	**5**
n_gp_layers	5	4	4	5	4
n_gp_out	2	4	4	4	1
n_max	653	489	490	707	852
n_layers	3	2	3	5	5
n_out	4	10	5	11	2
batch_size	512	256	512	512	256
lr	0.0006	0.0002	0.0001	0.0001	0.0006
num_inducing	423	440	77	52	341
num_samples	11	9	15	9	13
MSE	0.00005	0.00004	0.00004	0.00005	0.00005
MAE	0.005	0.005	0.004	0.005	0.005
Max Error	0.05	0.05	0.06	0.05	0.05
NLL	-3.01	-3.00	-3.15	-2.98	-3.12

Table 39: NNDGP - Elevators

outer_run	**1**	**2**	**3**	**4**	**5**
n_gp_layers	2	2	2	3	3
n_gp_out	2	2	2	3	3
n_max	431	587	972	771	558
n_layers	2	3	3	4	4
n_out	7	9	15	5	7
batch_size	256	256	256	256	256
lr	0.008	0.004	0.037	0.004	0.001
num_inducing	73	138	199	345	505
num_samples	9	4	8	6	12
MSE	353533.1	414757.1	4721145.3	317888	367269.6
MAE	337.5	373.9	380.3	321.2	352.9
Max Error	12899.9	5776.4	7283.2	5379.2	14410.0
NLL	6.87	7.03	7.02	6.85	6.92

Table 40: NNDGP - Diamonds

A.4.5 Deep Sigma Point Process

outer_run	1	2	3	4	5
n_dspp_layers	1	1	1	1	1
n_dspp_out	5	3	2	2	3
batch_size	256	256	512	256	256
lr	0.0002	0.0003	0.0003	0.0003	0.0002
num_inducing	131	71	353	691	620
num_quadrature_sites	8	5	8	6	5
beta	0.01	0.05	0.01	0.01	0.01
MSE	1.6e-07	2.7e-07	5e-07	1.7e-06	2.7e-07
MAE	0.00029	0.00039	0.00056	0.00117	0.00041
Max Error	0.002	0.005	0.003	0.003	0.002
NLL	-4.565	-4.833	-3.388	-4.78	-4.089

Table 41: DSPP - K2204 BPV

outer_run	1	2	3	4	5
n_dspp_layers	1	1	2	1	1
n_dspp_out	2	2	1	3	2
batch_size	256	256	256	256	256
lr	0.0002	0.0002	0.0003	0.0003	0.0005
num_inducing	788	744	479	141	155
num_quadrature_sites	7	9	9	7	10
beta	0.01	0.01	0.01	0.01	0.01
MSE	0.0035	0.0059	0.0197	0.0039	0.0479
MAE	0.0467	0.0629	0.1334	0.0512	0.1914
Max Error	0.266	0.466	0.349	0.229	0.585
NLL	0.0828	0.8656	-0.2147	0.0164	-0.1687

Table 42: DSPP - K2204 PPV

outer_run	1	2	3	4	5
n_dspp_layers	3	2	2	1	1
n_dspp_out	3	4	4	4	5
batch_size	256	256	512	256	256
lr	0.006	0.0034	0.008	0.0002	0.0077
num_inducing	474	432	667	662	61
num_quadrature_sites	7	9	8	5	7
beta	0.01	0.05	0.01	0.01	0.20
MSE	0.6	0.495	0.522	0.51	0.457
MAE	0.64	0.564	0.588	0.544	0.531
Max Error	3.389	2.921	3.257	2.909	3.394
NLL	0.22	0.151	0.382	0.447	0.083

Table 43: DSPP - Wine

outer_run	**1**	**2**	**3**	**4**	**5**
n_dspp_layers	1	1	1	1	1
n_dspp_out	2	2	2	2	1
batch_size	256	256	256	256	512
lr	0.0014	0.003	0.0023	0.0007	0.0006
num_inducing	650	712	441	652	189
num_quadrature_sites	9	7	6	5	5
beta	1	1	0.01	0.01	0.20
MSE	4e-06	4e-06	4e-06	4e-06	4e-06
MAE	0.0015	0.0014	0.0015	0.0015	0.0015
Max Error	0.014	0.026	0.024	0.013	0.017
NLL	-5.777	-5.792	-5.767	-5.825	-5.672

Table 44: DSPP - Elevators

outer_run	**1**	**2**	**3**	**4**	**5**
n_dspp_layers	1	3	2	2	1
n_dspp_out	5	4	5	5	5
batch_size	256	256	256	256	512
lr	0.011	0.0135	0.01	0.017	0.008
num_inducing	194	198	163	236	692
num_quadrature_sites	6	5	5	5	5
beta	0.2	0.2	0.01	0.05	0.01
MSE	325110.28	323600.59	320756.91	288972.84	322821.47
MAE	295.92	296.24	284.46	284.75	297.33
Max Error	5845.91	5105.99	6677.17	4922.98	13589.05
NLL	6.3	6.3	6.28	6.26	6.27

Table 45: DSPP - Diamonds

outer_run	**1**	**2**	**3**	**4**	**5**
n_dspp_layers	3	2	1	2	3
n_dspp_out	3	4	3	4	4
batch_size	2048	512	2048	1024	512
lr	0.016	0.004	0.004	0.007	0.008
num_inducing	50	217	108	300	186
num_quadrature_sites	8	5	7	5	8
beta	0.05	1.0	0.20	1.0	1.0
MSE	92.9	84.02	86.2	83.3	118.9
MAE	7.3	6.5	6.4	6.3	8.1
Max Error	67.4	67.4	79.58	69.6	76.4
NLL	2.6	2.5	2.6	2.5	2.9

Table 46: DSPP - Year

outer_run	**1**	**2**	**3**	**4**	**5**
n_dspp_layers	2	3	2	3	2
n_dspp_out	2	2	3	4	4
batch_size	2048	1024	1024	1024	1024
lr	0.0088	0.0138	0.0122	0.0144	0.0045
num_inducing	150	241	146	547	91
num_quadrature_sites	5	10	10	7	6
beta	1	0.2	0.01	0.05	0.01
MSE	0.004	0.009	0.008	0.072	0.012
MAE	0.054	0.078	0.078	0.258	0.094
Max Error	0.926	1.656	1.467	1.148	1.425
NLL	-0.31	-0.31	-0.30	-0.27	-0.29

Table 47: DSPP - R1_08 BPV

outer_run	**1**	**2**	**3**	**4**	**5**
n_dspp_layers	2	2	1	2	2
n_dspp_out	2	4	1	3	2
batch_size	2048	2048	2048	512	2048
lr	0.019	0.006	0.033	0.011	0.006
num_inducing	163	776	472	66	396
num_quadrature_sites	9	10	5	8	10
beta	0.2	1.0	0.05	0.01	1.0
MSE	0.0003	0.00003	0.0001	0.0002	0.0001
MAE	0.014	0.004	0.008	0.013	0.008
Max Error	0.17	0.099	0.193	0.135	0.193
NLL	-1.82	-1.86	-1.84	-1.78	-1.85

Table 48: DSPP - R1_08 PPV

A.4.6 Deep Ensemble

outer_run	1	2	3	4	5
n_max	227	304	393	542	348
n_layers	3	3	3	4	3
lr	2e-05	2e-05	1e-05	1e-05	1e-05
batch_size	512	1024	512	512	512
MSE	4e-07	5.5e-07	3.7e-07	5e-07	4.3e-07
MAE	0.0005	0.0005	0.0004	0.0005	0.0005
Max Error	0.0026	0.0029	0.0028	0.0023	0.0019
NLL	-6.09	-5.45	-5.68	-5.33	-6.13

Table 49: DE - K2204 BPV

outer_run	1	2	3	4	5
n_max	239	876	375	904	201
n_layers	4	4	4	4	4
lr	0.0004	9e-05	0.0003	5e-05	0.0005
batch_size	512	512	512	512	512
MSE	0.018	0.006	0.011	0.003	0.016
MAE	0.083	0.047	0.061	0.035	0.078
Max Error	0.85	0.57	0.91	0.44	0.87
NLL	-1.42	-1.36	-0.67	-1.63	-1.07

Table 50: DE - K2204 PPV

outer_run	1	2	3	4	5
n_max	668	996	943	900	963
n_layers	3	4	2	2	2
lr	0.0028	0.0004	0.0021	0.0033	0.0012
batch_size	512	512	512	512	512
MSE	0.47	0.45	0.44	0.47	0.43
MAE	0.53	0.53	0.52	0.52	0.51
Max Error	3.36	2.99	3.29	2.82	3.2
NLL	0.104	0.079	0.067	0.109	0.049

Table 51: DE - Wine

outer_run	1	2	3	4	5
n_max	1017	287	554	821	932
n_layers	5	6	6	4	6
lr	0.000021	0.000097	0.000022	0.000023	0.000020
batch_size	8124	8124	8124	512	2048
MSE	0.000122	0.000056	0.000237	0.000134	0.000187
MAE	0.008688	0.005595	0.012509	0.008660	0.012281
Max Error	0.057	0.042	0.246	0.197	0.05
NLL	-1.82	-1.38	-0.63	-1.45	-1.41

Table 52: DE - Elevators

outer_run	1	2	3	4	5
n_max	924	968	686	52	920
n_layers	2	2	2	4	2
lr	0.032	0.019	0.013	0.0115	0.013
batch_size	512	512	512	512	512
MSE	849109.56	303990.69	331165.84	400592.34	302748.63
MAE	289.60	282.74	290.71	307.19	283.63
Max Error	75791.84	4974.049	9417.63	13011.88	8744.75
NLL	6.03	6.03	6.1	6.02	6.08

Table 53: DE - Diamonds

outer_run	1	2	3	4	5
n_max	877	335	696	101	283
n_layers	2	2	2	3	4
lr	0.0016	0.0027	0.029	0.0035	0.0020
batch_size	512	512	1024	512	512
MSE	86.6	148.8	88.8	87.4	94.6
MAE	6.42	9.07	6.67	6.38	7.29
Max Error	245.5	209.2	104.3	437.7	144.9
NLL	2.68	3.31	2.79	2.67	2.82

Table 54: DE - Year

outer_run	1	2	3	4	5
n_max	745	847	992	695	873
n_layers	6	5	5	5	6
lr	4e-05	4e-05	1e-05	7e-05	4e-05
batch_size	512	512	512	512	1024
MSE	0.00011	0.00011	0.00006	0.00019	0.00036
MAE	0.008	0.007	0.006	0.011	0.015
Max Error	0.55	0.71	0.23	0.38	0.60
NLL	-3.35	-3.51	-3.85	-3.36	-2.93

Table 55: DE - R1_08 BPV

outer_run	1	2	3	4	5
n_max	40	53	531	138	361
n_layers	5	6	2	5	3
lr	0.0002	0.0004	0.0003	0.0001	0.0001
batch_size	512	512	1024	1024	512
MSE	0.000014	0.00007	0.000007	0.000025	0.000007
MAE	0.0028	0.0067	0.0013	0.0041	0.0021
Max Error	0.078	0.085	0.11	0.09	0.05
NLL	-4.43	-3.34	-5.4	-3.8	-4.61

Table 56: DE - R1_08 PPV

A.4.7 Random Forest

outer_run	1	2	3	4	5
n_estimators	110	163	75	169	162
max_depth	21	36	36	41	16
min_samples_leaf	1	1	1	1	1
min_samples_split	2	2	2	2	2
max_features	all	sqrt	all	all	sqrt
MSE	5.33e-07	6.98e-07	6.38e-07	7.07e-07	6.77e-07
MAE	0.0004	0.0006	0.0005	0.0005	0.0006
Max Error	0.005	0.004	0.005	0.005	0.006

Table 57: RF - K2204 BPV

outer_run	1	2	3	4	5
n_estimators	197	151	146	156	134
max_depth	47	29	38	12	47
min_samples_leaf	1	1	1	1	1
min_samples_split	2	2	3	3	2
max_features	all	sqrt	all	all	all
MSE	0.007	0.01	0.009	0.01	0.008
MAE	0.05	0.07	0.06	0.07	0.05
Max Error	0.54	0.49	0.48	0.59	0.54

Table 58: RF - K2204 PPV

outer_run	1	2	3	4	5
n_estimators	372	402	73	296	167
max_depth	34	41	41	43	21
min_samples_leaf	1	1	2	1	1
min_samples_split	2	2	2	2	2
max_features	sqrt	all	sqrt	sqrt	sqrt
MSE	0.39	0.35	0.34	0.38	0.33
MAE	0.44	0.42	0.42	0.43	0.42
Max Error	3.41	3.54	2.54	2.84	2.95

Table 59: RF - Wine

outer_run	1	2	3	4	5
n_estimators	397	230	456	361	481
max_depth	47	38	28	40	78
min_samples_leaf	1	1	2	1	1
min_samples_split	2	2	4	3	3
max_features	all	all	all	all	all
MSE	8e-06	7e-06	7e-06	7e-06	7e-06
MAE	0.002	0.002	0.002	0.002	0.002
Max Error	0.03	0.02	0.04	0.02	0.02

Table 60: RF - Elevators

outer_run	**1**	**2**	**3**	**4**	**5**
n_estimators	380	440	341	304	459
max_depth	65	50	20	19	97
min_samples_leaf	1	1	1	1	1
min_samples_split	9	5	5	9	16
max_features	all	sqrt	all	all	sqrt
MSE	296984.19	300438.14	308578.82	296715.98	296594.85
MAE	270.3	269.43	273.49	271.51	270.63
Max Error	5790.27	6076.36	6566.03	12583.93	9961.32

Table 61: RF - Diamonds

outer_run	**1**	**2**	**3**	**4**	**5**
n_estimators	225	248	250	249	249
max_depth	32	28	39	50	39
min_samples_leaf	8	8	6	6	8
min_samples_split	8	10	6	4	6
max_features	all	all	all	all	all
MSE	81.8	81.0	81.4	81.6	80.6
MAE	6.35	6.33	6.33	6.33	6.30
Max Error	70.15	74.46	76.53	66.73	74.15

Table 62: RF - Year

outer_run	**1**	**2**	**3**	**4**	**5**
n_estimators	128	160	172	176	99
max_depth	13	13	12	13	13
min_samples_leaf	15	15	10	15	15
min_samples_split	15	9	14	11	4
max_features	all	all	all	all	all
MSE	0.05	0.05	0.12	0.05	0.05
MAE	0.17	0.17	0.27	0.17	0.17
Max Error	1.28	1.77	2.12	1.22	1.21

Table 63: RF - R1_08 BPV

outer_run	**1**	**2**	**3**	**4**	**5**
n_estimators	30	150	10	240	11
max_depth	37	29	40	35	28
min_samples_leaf	1	1	1	1	2
min_samples_split	8	4	5	3	3
max_features	all	all	all	all	all
MSE	1.4e-07	1.4e-07	1.6e-07	1.4e-07	1.7e-07
MAE	7e-05	7e-05	8e-05	7e-05	9e-05
Max Error	0.005	0.004	0.005	0.004	0.004

Table 64: RF - R1_08 PPV

A.4.8 XGBoost

outer_run	1	2	3	4	5
n_estimators	664	709	584	1355	490
max_depth	9	11	7	4	7
learning_rate	0.086	0.071	0.088	0.07	0.1
colsample_bytree	0.56	0.51	0.6	0.54	0.53
subsample	0.57	0.53	0.66	0.64	0.64
reg_alpha	0.18	0.39	0.018	0.04	0.02
reg_lambda	6.21	0.0004	0.001	0.002	0.004
gamma	9.38e-08	1.42e-04	6.13e-06	5.05e-07	1.05e-07
min_child_weight	13.58	24.68	15.25	11.36	11.34
MSE	4e-06	5e-06	4e-06	4e-06	4e-06
MAE	0.002	0.002	0.002	0.002	0.002
Max Error	0.01	0.01	0.01	0.01	0.01

Table 65: XGB - K2204 BPV

outer_run	1	2	3	4	5
n_estimators	664	709	584	1355	490
max_depth	8	4	6	4	11
learning_rate	0.08	0.08	0.08	0.09	0.08
colsample_bytree	0.57	0.53	0.6	0.6	0.22
subsample	0.53	0.64	0.47	0.69	0.71
reg_alpha	0.02	0.01	0.02	0.01	0.09
reg_lambda	0.01	0.18	0.001	0.16	0.00004
gamma	7.36e-07	6.62e-06	1.36e-06	4.88e-06	2.29e-05
min_child_weight	12.89	16.92	14.97	16.46	20.34
MSE	0.05	0.05	0.05	0.05	0.86
MAE	0.18	0.18	0.17	0.18	0.72
Max Error	1.11	0.96	1.06	0.98	4.06

Table 66: XGB - K2204 PPV

outer_run	1	2	3	4	5
n_estimators	299	183	261	172	566
max_depth	47	36	30	12	25
learning_rate	0.03	0.05	0.02	0.05	0.01
colsample_bytree	0.59	0.48	0.6	0.6	0.59
subsample	0.69	0.59	0.55	0.47	0.8
reg_alpha	0.05	0.05	0.02	0.02	0.01
reg_lambda	0.86	4.38e-06	1.22e-08	2.34e-02	8.87e-07
gamma	3.29e-04	2.09e-05	1.29e-06	7.22e-04	1.13e-08
min_child_weight	3.83	3.4	5.41	5.48	4.77
MSE	0.39	0.34	0.34	0.40	0.32
MAE	0.42	0.4	0.41	0.44	0.38
Max Error	3.7	3.13	2.71	2.68	2.89

Table 67: XGB - Wine

outer_run	1	2	3	4	5
n_estimators	3308	1573	1089	681	2541
max_depth	40	4	56	37	5
learning_rate	0.054	0.069	0.063	0.088	0.96
colsample_bytree	0.38	0.41	0.55	0.58	0.49
subsample	0.66	0.43	0.45	0.52	0.74
reg_alpha	1.1	0.38	0.54	0.4	1.28
reg_lambda	1.97e-04	2.54	0.046	1.81e-08	4.4e-08
gamma	7.48e-05	7.7e-08	5.69e-05	2.11e-06	1.72e-06
min_child_weight	18.62	12.38	28.31	16.5	3.08
MSE	5e-06	4e-06	5e-06	5e-06	4e-06
MAE	0.002	0.002	0.002	0.002	0.002
Max Error	0.02	0.02	0.03	0.02	0.02

Table 68: XGB - Elevators

outer_run	1	2	3	4	5
n_estimators	266	562	511	261	522
max_depth	72	49	16	28	52
learning_rate	0.065	0.036	0.078	0.87	0.084
colsample_bytree	0.57	0.51	0.6	0.47	0.48
subsample	0.54	0.72	0.59	0.72	0.8
reg_alpha	0.33	0.43	0.96	0.44	0.99
reg_lambda	9.7e-08	1.13e-05	2.19e-08	1.12e-04	1.55e-06
gamma	2.6e-04	7.06e-04	1.84e-08	1.53e-04	1.71e-057
min_child_weight	4.81	4.54	3.5	33.91	2.44
MSE	298920.73	300966.62	315999.61	292805.28	298428.54
MAE	275.33	277.04	282.61	281.54	280.42
Max Error	5257.6	6201.39	8100.32	8754.91	8892.31

Table 69: XGB - Diamonds

outer_run	1	2	3	4	5
n_estimators	840	469	742	494	519
max_depth	44	46	42	46	46
learning_rate	0.0557	0.0552	0.069	0.0552	0.0552
colsample_bytree	0.59	0.59	0.48	0.59	0.59
subsample	0.65	0.63	0.80	0.63	0.63
reg_alpha	2.37	1.47	1.64	1.47	1.47
reg_lambda	2.6e-08	2.7e-08	7.9e-06	2.7e-08	2.7e-08
gamma	2.3e-06	2.6e-05	7.8e-07	2.6e-05	2.6e-05
min_child_weight	84.1	63.1	75.4	63.05	63.05
MSE	73.03	72.98	72.85	73.42	72.46
MAE	5.95	5.96	5.94	5.95	5.93
Max Error	70.33	69.94	73.18	68.52	70.43

Table 70: XGB - Year

outer_run	**1**	**2**	**3**	**4**	**5**
n_estimators	664	709	584	1355	490
max_depth	43	6	69	39	29
learning_rate	0.086	0.084	0.099	0.81	0.071
colsample_bytree	0.53	0.46	0.46	0.57	0.58
subsample	0.75	0.56	0.45	0.4	0.56
reg_alpha	1.4	1.1	0.43	1.14	2.21
reg_lambda	1.68e-04	0.0018	3.74e-07	1.84e-06	0.18
gamma	1.86e-08	8.67e-06	2.8e-06	1.72e-04	2.18e-06
min_child_weight	2.81	33.26	10.49	2.41	3.46
MSE	0.011	0.013	0.013	0.015	0.014
MAE	0.07	0.08	0.07	0.09	0.08
Max Error	1.94	2.10	1.81	2.14	2.48

Table 71: XGB - R1_08 BPV

outer_run	**1**	**2**	**3**	**4**	**5**
n_estimators	888	621	1376	578	526
max_depth	48	4	60	11	40
learning_rate	0.065	0.081	0.023	0.0997	0.087
colsample_bytree	0.326	0.392	0.4427	0.247	0.434
subsample	0.409	0.722	0.634	0.621	0.792
reg_alpha	0.018	0.102	0.432	0.035	0.59
reg_lambda	4.08e-02	2.5e-08	4.6e-03	1.8e-05	2.5e-05
gamma	1.0e-06	1.47e-07	1.2e-04	1.1e-07	2.5e-7
min_child_weight	8.32	2.92	18.22	50.19	37.75
MSE	0.0002	0.0001	0.0002	0.0002	0.0002
MAE	0.009	0.007	0.010	0.009	0.011
Max Error	0.15	0.09	0.09	0.15	0.09

Table 72: XGB - R1_08 PPV

Declaration of authorship

I hereby declare that the report submitted is my own unaided work. All direct or indirect sources used are acknowledged as references. I am aware that the Thesis in digital form can be examined for the use of unauthorized aid and in order to determine whether the report as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future Theses submitted. Further rights of reproduction and usage, however, are not granted here. This paper was not previously presented to another examination board and has not been published.

Location, date

Name