Imperial College London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

How good are approximations in Bayesian Deep Learning?

Author:
Matthew Collins

Supervisor:
Dr. Mark van der Wilk

Second Marker:
Dr. Yingzhen Li

Submitted in partial fulfillment of the requirements for the MSc degree in Artificial Intelligence of Imperial College London

June 2022

Abstract

Bayesian Deep Learning has show promising results in quantifying the uncertainty of predictions. However, these models are also sensitive to their hyperparameters, which can hamper performance, and as a consequence methods of model selection for Bayesian models are valuable to the field. Bayesian Neural Networks estimate uncertainty by approximating the posterior over weights and biases. Many of these methods also provide approximations to the marginal likelihood, which is shown to be the Bayesian way to preform model selection. Then if the marginal likelihood approximations of these methods are strong, we should be able to use them to automatically select hyperparameters, including Neural Network architecture. In particular, we investigate Mean Field Variational Inference and Laplace Approximations on Bayesian Neural Networks.

We use a regression dataset to find a variety of marginal likelihood estimations for several Bayesian Deep Learning methods by testing their ability to select good models using their marginal likelihood approximations. We compare these estimations to simpler models, namely Gaussian Processes, where the exact marginal likelihood can be computed, considering their test performances and comparability in their model structures. Useful comparisons are made with the Neural Network Gaussian Process equivalence to infinite width Bayesian Neural Networks.

We conclude that while the variational inference methods can be used to find good models, they generally make poor marginal likelihood approximations which are not suitable for model selection over Neural Network architecture, and that although the Laplace marginal likelihood approximations are tighter and better at selecting models, their uncertainty estimates are unsatisfactory.

Acknowledgements

I would like to express my deepest appreciation to Dr. Mark van der Wilk for his original proposal and continued direction throughout the project, as well as his extensive knowledge on the subject and his excellent original lectures in Probabilistic Inference.

Secondly, I would like to sincerely thank Seth Nabarro for his attentiveness to my project, and his thorough and quick responses to my questions.

I am also very grateful to Yingzhen Li for her Bayesian Neural Network tutorial, and corresponding code that inspired my own implementations.

Finally, I would like to thank my family and friends for their support throughout this journey.

Contents

Chapter 1

Introduction

In the past decade Deep Learning models have revolutionised our ability to make predictions from data. However as these networks have grown in complexity, so has their lack of explainability. Bayesian Deep Learning offer a persuasive alternative, as it can be used to capture the uncertainty over our data, which gives us a measure of when to trust a prediction. As with all Bayesian methods, this is achieved by calculating the posterior probability distribution, however BNNs rely on approximations to this posterior. Many of these approximations also provide approximations to the marginal likelihood, which has a surprising benefit: Comparing marginal likelihoods has been shown to be the Bayesian method of model selection [1]. Despite this relationship being well-known, there has been little success in utilising these approximations for automatic model selection in Bayesian Deep Learning.

The two main paradigms in parametric Bayesian Deep Learning are the Mean Field Variational Inference Bayesian Neural Network (MFVI-BNN) and the Laplace Approximation over Neural Network weights. In the original MFVI-BNN paper [2], the authors apply variational inference to Neural Networks to develop their MFVI-BNN, utilising a loss objective known as the Evidence Lower Bound (ELBO). As the name suggests, this bound is a lower bound on the true marginal likelihood. If the ELBO is tight to the true marginal likelihood, it should be useful for marginal likelihood based model selection, however the authors empirically found the ELBO to make a poor tool for hyperparameter optimisation. We investigate the original ELBO, along with Collapsed Variational Bounds from [3], which claims to tighten the ELBO. Laplace approximations also provide marginal likelihood estimations, however they are computationally taxing, and require approximations to be scalable for Deep Learning. To this end, we investigate scalable Laplace approximations from [4].

To test how close these marginal likelihood approximations are to the true marginal likelihood, and whether their corresponding surfaces can be used for automatic model selection, we would like to be able to use the true marginal likelihood on these models. Though simpler and less scalable than Bayesian Neural Networks, Gaussian Processes offer non-parametric modelling with well studied marginal likelihoods estimations, and for many models the exact marginal likelihood can be computed. Then if we can make good comparisons between Bayesian Neural Networks and Gaussian Processes, we can make strong statements on the accuracy of the marginal likelihood approximations. In particular, we utilise the equivalence between Neural Network Gaussian Processes, and the infinite width limit of Bayesian Neural Networks.

1.1 Contributions

The main contributions of this report can be summarised as follows:

- We create an explainable regression problem, with good targets for epistemic uncertainty estimates in interpolation and extrapolation, and implement a range of Bayesian models on this problem, including recent methods involving marginal likelihood approximations for hyperparameter selection.
- We preform rigorous hyperparameters searches over different testing metrics to create good comparisons between models and explore the approximate marginal likelihood surfaces for these different marginal likelihood approximation methods, being careful not to introduce bias, and carefully differentiating the performance of models and their marginal likelihood approximations.
- We make strong comparisons between Gaussian Processes and Bayesian Neural Networks, allowing for absolute comparisons of marginal likelihood estimates.
- We use these experiments and comparisons to conclude strong assertions on the efficacy of approximating the marginal likelihood with our investigated methods.

1.2 Outline

The report is organised as follows: Chapter 2 motivates Bayesian Inference as a learning procedure, and marginal likelihood as a tool for automatic model selection. Chapter 3 explains the Bayesian models we will be studying, and Chapter 4 details the different marginal likelihood estimations for these models. Then Chapter 5 forms the main body of the report, describing and discussing the experiments we chose to preform and their results. Chapter 6 takes a brief look at our experimental implementation, and evaluates the results of the experiments, and finally Chapter 7 summarises the results, and discusses future work and ethical discussion.

1.3 Related Work

Much research has been done on the original MFVI-BNN from [2] to improve its performance and extend the theory to different architectures and models. Many of these papers demonstrate issues with its ability to predict, for example [5] which shows the MFVI-BNN gives poor posterior uncertainty estimates in-between separated areas of observations (and utilised the Laplace approximation to improve them), and [6] which shows how better prior choice for different models can improve performance.

Equally model selection using the marginal likelihood is well-explored and justified. However, much less work has been done to take advantage of marginal likelihood based model selection for Bayesian Deep Learning.

The work in this area of particular interest to us is of course the models which we explore, namely [4] using Laplace approximations, and [3] using collapsed variational bounds. There are no papers comparing these methods for model selection, the later paper has only had a

few citations, none of which focus on this model's marginal likelihood approximations.

Again, the connection being Gaussian Processes and Bayesian Neural Networks is well studied, and much work has been done in comparing and combining their capabilities, which will be using to our advantage. However, I am not aware of any papers studying the absolute values of marginal likelihood approximations between GPs and BNNs, and moreover there are no paper comparing [4] or [3] to GPs.

Chapter 2

Bayesian Inference

To start we would like to motivate the hypothesis of this project, and introduce the notation we will use throughout this report.

First we am going to motivate the use of Bayes rule from a machine learning perspective. Perhaps this order is unnatural in that Bayes rule explains the machine learning methodology, and not the other way around, but many are far more familiar with methods like Maximum Likelihood Estimation than Bayesian Inference. The fundamental axiom is that Bayes rule is a rational way of updating our beliefs given data, as laid out in [7].

$$\textbf{Bayesian Inference using Bayes Rule:} \qquad p(H|E) = \frac{p(E|H)p(H)}{p(E)} \tag{2.1}$$

H is the hypothesis, typically this will be a model (or the parameters of a model), and E is the evidence, some observed data we did not use to form our prior probability of the hypothesis, $p(H)$. Then we update our beliefs by using the probability of the evidence to find a posterior probability of our hypothesis, i.e the probability of our model including the new data.

2.1 Probabilistic Supervised Learning

In supervised machine learning, we have a dataset $\mathcal{D} = \{\mathbf{X}, \mathbf{Y}\}$, where $\boldsymbol{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$ are inputs, and $\boldsymbol{Y} = \{\boldsymbol{y}_1, ..., \boldsymbol{y}_N\}$ are corresponding targets. This dataset has been created from some underlying function (or more likely, distribution) which maps any $\boldsymbol{x}$ in the domain to a $\boldsymbol{y}$ (or distribution over $\boldsymbol{y}$), and we would like to learn a model, $\mathcal{M}$, which describes this distribution, and we can use for predictions. Typically $\mathcal{M}$ specifies a function, $\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{y}$, which is parameterised by latent variable $\boldsymbol{\theta}$. Specifically we want to optimise over $\boldsymbol{\theta}$ to maximise $p(\boldsymbol{y}_*|\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_*), \mathcal{D})$, where $(\boldsymbol{x}_*, \boldsymbol{y}_*)$ is a datapoint not necessarily in our dataset. Our model represents all our assumptions (parameters and hyperparameters), for now I am going to just discuss optimising parameters.

Typically learning is done by maximum (log) likelihood estimation (MLE), where we use an optimisation method to find:

$$\boldsymbol{\theta}_{ml} = \arg\max_{\boldsymbol{\theta}} \log p(\mathcal{D}|\boldsymbol{\theta})^1 \tag{2.2}$$

[1] In literature it is common to see this distribution written $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})$, however since y is determined completely

For regression, we commonly use a normal distribution to model the probability of each target equalling the function value, given an input. Then the likelihood is modelled as the joint of n i.i.d. distributions:

$$p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{i=1}^{N} \mathcal{N}(\boldsymbol{y}_i|\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \sigma_l^2 \boldsymbol{I}) \tag{2.3}$$

From this we can directly derive mean squared error, the most common loss function for regression:

$$\arg\max_{\boldsymbol{\theta}} \log p(\mathcal{D}|\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} (\boldsymbol{y}_i - \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_i))^2 \tag{2.4}$$

Alternatively for classification we use a discrete likelihood, normally softmax.

Naturally MLE can lead to overfitting, as it is easy to fit a meaningless function exactly to the datapoints, optimising our training loss. An obvious solution is to restrict our function space to simpler, more generalisable models. More generally, we can enforce a prior distribution on our parameters, and preform maximum a posteriori estimation (MAP):

$$\boldsymbol{\theta}_{ml} = \arg\max_{\boldsymbol{\theta}} \log p(\mathcal{D}|\boldsymbol{\theta}) p(\boldsymbol{\theta}) \tag{2.5}$$

Then in the case of a neural network, where $\boldsymbol{\theta}$ is a vector containing the weights of our network. If $\boldsymbol{\theta}$ is given a standard normal prior, i.e. we model our weights to come from a standard normal distribution, before we observe any data, we derive L2 regularisation:

$$\begin{aligned} \arg\max_{\boldsymbol{\theta}} \log p(\mathcal{D}|\boldsymbol{\theta}) p(\boldsymbol{\theta}) &= \arg\max_{\boldsymbol{\theta}} \left[\log p(\mathcal{D}|\boldsymbol{\theta}) + \log \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{0}, \boldsymbol{I})\right] \\ &= \arg\min_{\boldsymbol{\theta}} \left[-\log p(\mathcal{D}|\boldsymbol{\theta}) + \boldsymbol{\theta}^2\right] \end{aligned} \tag{2.6}$$

This is an example of an uninformative prior. The hypothesis we have made about our model before observing new data is not based on any previous data, only around the principle of Occam's razor, that we should prefer simpler models.

Maximum a posteriori estimation is named as such, because we are actually find the $\arg\max$ of the posterior distribution. If we apply Bayes rule:

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(\mathcal{D})} \tag{2.7}$$

We see the marginal likelihood $p(\mathcal{D})$ is constant with respect to $\boldsymbol{\theta}$, and so can be ignored if we are only interested in the $\boldsymbol{\theta}$ which maximises the posterior. MLE is actually a special case of MAP, with the choice of prior being a uniform distribution.

Hence both MLE and MAP are applications of Bayesian inference. They are both attempts to update our beliefs via Bayes rule. However they do not compute the posterior, only the parameters which maximise it, which are the parameters most probable given the data.

by x this notation is also correct and I find it easier to read.

2.2 Full Posterior

If we apply Bayes rule fully, we can compute the posterior, which is a complete distribution over our parameters give our observed data. To approximate the posterior it is not always necessary to involve Bayes rule directly, which will be important later. However if we want to apply Bayes rule as is, we will need to compute the marginal likelihood, the denominator of Bayes rule, which decomposes to:

$$p(\mathcal{D}) = \int p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta} \tag{2.8}$$

We are effectively weighting the likelihood of every parameter by its prior. In the case of a neural network, this is equivalent to an infinite ensemble of networks (testing a network with every possible set of parameters). If we wish to you flexible models it is unlikely this integral will be possible to calculate analytically, and this is where approximations are necessary for Bayesian learning. Estimation of the marginal likelihood forms the fundamental object of this project.

Then rather than using one model with the best parameters from our $\arg\max$ to make predictions, like in MLE and MAP, we can now use the predictive posterior distribution:

$$p(\boldsymbol{y}_*|\boldsymbol{x}_*, \mathcal{D}) = \int p(\boldsymbol{y}_*|\boldsymbol{x}_*, \boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta} \tag{2.9}$$

This is the likelihood of our parameterised model applied to new data, multiplied by our posterior, marginalised (integrated) over all possible parameters. Again it is using an ensemble of all possible models. Then if we test a new datapoint, if this point's inputs $\boldsymbol{x}^*$ are distant from our training data, it is expected that the probability of $\boldsymbol{y}^*$ given our original dataset will have higher variance, and from there we have our epistemic uncertainty, the uncertainty due to the lack of data.

The predictive posterior is an expectation over the original posterior, so we can use a Monte Carlo estimate to calculate it, provided we can sample from the posterior:

$$\mathbf{E}_{p(\boldsymbol{\theta}|\mathcal{D})}[p(\boldsymbol{y}_*|\boldsymbol{x}_*, \boldsymbol{\theta})] = \frac{1}{S}\sum_{s=1}^{S} p(\boldsymbol{y}_*|\boldsymbol{x}_*, \boldsymbol{\theta}_s), \quad \boldsymbol{\theta}_s \sim p(\boldsymbol{\theta}|\mathcal{D}) \tag{2.10}$$

We can also find the epistemic uncertainty by using a Monte Carlo estimate of the variance, against sampling from the posterior, and then approximate this uncertainty with a Gaussian distribution using the estimated mean and variance. In our experiments, this is how we will display uncertainty on our plots, and calculate Gaussian Likelihood test metrics.

2.3 Model Selection

The posterior gives us an inference over parameters, but we made a number of inductive biases to get to the point of having a model with parameters. These are the hyperparameter of the model, which we collect into $\boldsymbol{\eta}$:

1. Our choice of parameterised function $\boldsymbol{f}_{\boldsymbol{\theta}}(.)$. Traditionally we would use the simplest function which explains our data adequately, however later we will argue for using more flexible models.

2. Any hyperparameter of the likelihood distribution, for example, the likelihood variance for regression, σ_l^2. When only taking the in the $\arg\max$, in the case of MSE, this term is dropped, and in the case of MAP, it becomes a constant weighting on the likelihood term, which is equivalent to a weighting on the prior (e.g. weight decay for L_n regularisation). We will call these likelihood parameters, and collect them into $\boldsymbol{\beta}$.
3. Our choice of prior $p(\boldsymbol{\theta})$. This prior can include any knowledge we already have about our model before we observe the data, but it does not have to. Like the standard normal prior which results in L_2 regularisation, it can still be meaningful to use uninformative priors.
4. Any hyperparameters of the prior, for example a more general prior than the standard normal, would be a normal distribution with the mean and variance as hyperparameters. We will call these prior parameters, and collect them into vector $\boldsymbol{\alpha}$.

Optimising hyperparameters is traditionally done using a validation set, a held-out section of the training data which allows testing the model's ability to generalise to unseen data. Often cross-validation is used to make best use of the training data, and to diminish any overfitting on the validation set. However we are again going to argue for a more Bayesian approach.

Arguably all these hyperparameters can be seen as a choice of prior, we only set the hyperparameters out like this so we apply inference to our parameterised model. This is an important observation, as it answers the question, how do we find the best hyperparameters using Bayesian inference? Apply Bayes rule again:

$$p(\boldsymbol{\eta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\eta})p(\boldsymbol{\eta})}{p(\mathcal{D})} \tag{2.11}$$

We can use Bayesian inference to find our best hyperparameters, in exactly the same way as we found our parameters, again inducing this rational method of learning from data.

Here is a more appropriate place to apply MLE or MAP. We plan to use this equation for model comparison (hyperparameter selection), so we are only interested in the $\arg\max$, and can ignore $p(D)$ as constant to the data. Generally we will assume $p(\boldsymbol{\eta})$ is also a uniform prior (MLE approach), as we have no particular reason to prefer some hyperparameters to others, although this is still an assumption and could cause problems (see Sure Thing model [1] p421). Then we can choose between different sets of hyperparameters, by comparing the $p(\mathcal{D}|\boldsymbol{\eta})$. We have actually seen this value already, it is the marginal likelihood from Equation 2.7, since every distribution in that equation is also conditioned on our hyperparameters, $\boldsymbol{\eta}$. Then if we have a way of approximating the marginal likelihood[2], not only can we find the posterior, but we can also use that marginal likelihood for model selection.

The marginal likelihood does not require the use of any held-out test-sets, we can (and should) evaluate it directly on the training data, and through Bayes rule it generalises to future data. To understand why that is, we see that the marginal likelihood can be decomposed over all the datapoints in the training set:

$$p(\mathcal{D}|\boldsymbol{\eta}) = \prod_{n=1}^{N} p(\mathcal{D}_n|\boldsymbol{\eta}, \{\mathcal{D}_i\}_{i=1}^{n-1}), \quad \text{where } \mathcal{D}_i = (\boldsymbol{x}_i, \boldsymbol{y}_i) \tag{2.12}$$

[2]A marginal likelihood is the denominator of Bayes rule, which is a marginalisation over a likelihood, however in this report when we refer to *the marginal likelihood* we will be specifically be discussing $p(\mathcal{D}|\boldsymbol{\eta})$.

At each datapoint, we are taking the probability of our previous datapoints as our prior, and multiply it by the likelihood of observing this new datapoint. Moreover, the marginal likelihood is invariant to the order in which we observe the training datapoints. Therefore, we could see this method as equivalent to a complete cross-validation, using every subset of the training dataset as a held-out test-set.

2.4 Empirical Bayes

As discussed, all are hyperparameters are prior assumptions on our model, and can be seen as prior parameters. The method of estimating the prior distribution from the data is called Empirical Bayes, and is well-studied [8]. While it may at first seem in contradiction to the definition of a prior, which we did define to represent our beliefs before we observe new data, it should be viewed as an approximation to a fully Bayesian treatment of a hierarchical Bayes model, which is what we have set out above. We are lifting the original method of learning from any poor inductive biases enforced by the original prior, assuming a far more general prior (the uniform distribution we put on hyperparameters, over whichever hyperparameters we choose to evaluate with our model, in an attempt to find the best model).

To this end, marginal likelihood comparisons can be made between any two models, under the assumption that we are applying no preference to either model before observing its performance. That being said, models with different structures, will give differing marginal likelihoods, because of the supports they give to different models. For example, a SVM puts a far high probability on linear fits, than a deep neural network. Therefore, if we want to compare the exact marginal likelihood values for different models (using different methods), we would like these models to be similar in structure and support.

Chapter 3

Choice of Bayesian Models

We now introduce the two models which form the focus of this project.

3.1 Bayesian Neural Networks

Bayesian Neural Networks (BBNs) form the central focus of this project. They are valuable because they retain the same high flexibility of non-Bayesian Neural Networks (NNs), while having all the benefits we have described of Bayesian methods.

We will look at two different types of Bayesian Neural Network:

3.1.1 Mean-Field Variational Inference Bayesian Neural Network

The MVFI-BNN was originally proposed in [2], and forms the focus of this project. It makes use of variational inference, a technique not limited to BBNs, to approximate the posterior distribution without the need to calculate the marginal likelihood. We use a tractable distribution $q(\boldsymbol{\theta})$ to approximate the true posterior $p(\boldsymbol{\theta}|\mathcal{D})$, and minimise the KL divergence between the two:

$$D_{KL}[q(\boldsymbol{\theta}) \parallel p(\boldsymbol{\theta}|\mathcal{D})] = \log P(\mathcal{D}) - \mathbf{E}_{q(\boldsymbol{\theta})}[\log p(\boldsymbol{\theta}, \mathcal{D}) - \log q(\boldsymbol{\theta})] \tag{3.1}$$

The divergence consists of the log marginal likelihood and a (negative) expectation term called the Evidence Lower Bound (ELBO). The marginal likelihood does not depend on the approximate posterior parameters, so optimising these parameters equates to minimising the ELBO. Moreover, minimising the ELBO has another benefit, it gives us a lower bound to the marginal likelihood. This is obvious from rearranging the terms of the equation:

$$\log P(\mathcal{D}) = D_{KL}[(]q(\boldsymbol{\theta}) \parallel p(\boldsymbol{\theta}|\mathcal{D})] + ELBO(\boldsymbol{\theta}, \mathcal{D}) \geq ELBO(\boldsymbol{\theta}, \mathcal{D}) \tag{3.2}$$

If the true posterior can be represented precisely by the approximate posterior, then the divergence between the two posteriors will equal zero, the bound will be tight, and the ELBO will equal the marginal likelihood. The hope is then that the ELBO can be used to approximate the marginal likelihood, which can in turn be used for model comparison, to find good hyperparameters.

MFVI-BNNs allow us to use a NN to learn the approximate posterior by preforming gradient descent on the ELBO. NNs are functions which have weights and biases as their parameters.

We can turn a Neural Network into a Bayesian Neural Network by drawing these weights and biases from a distribution instead, and the parameters of the BNN are the parameters of the distribution, rather than the weights and biases themselves. Specifically our weights and biases are drawn from a diagonal multivariate Gaussian distribution, $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{\mu}_q, \boldsymbol{\sigma}_q^2 I)$, where $\boldsymbol{w}$ is a vector containing all the weights and biases of the model and $\boldsymbol{\mu}_q, \boldsymbol{\sigma}_q^2$ are vectors containing our Gaussian parameters, we call these the approximate posterior mean and variance. We see a BNN has twice as many parameters as a NN. As the Gaussian distribution is diagonal, the distribution on each weight and bias is independent, i.e. $w_i \sim \mathcal{N}(\mu_{q,i}, \sigma_{q,i}^2)$, which is the mean-field assumption.

The ELBO decomposes into a likelihood term and a complexity term, which for our BNN looks like:

$$ELBO(\boldsymbol{w}, \mathcal{D}) = \mathbf{E}_{q(\boldsymbol{w})}[\log p(\mathcal{D}|\boldsymbol{w})] - KL[q(\boldsymbol{w}))||p(\boldsymbol{w})]. \tag{3.3}$$

The complexity term is the KL divergence between the approximate posterior and the prior. If we take our prior to be diagonal Gaussian, we can continue our mean-field assumption, and moreover if we are using an uninformative prior, it would not make much sense for it to have more parameters than the posterior. Generally we will assume our prior to have mean zero, and only one prior variance parameter for the diagonal, similar to the standard normal prior from equation (2.6). The KL divergence between two Gaussians is easily computed, and forms a convex surface to compute gradients on.

For the likelihood term we can use a Monte Carlo estimate for the expectation as in (2.9), by sampling from our approximate posterior. However we cannot compute gradients by sampling directly. The fundamental insight of [2] is that we can reparameterise this sampling process, so that we are instead sampling from a standard normal distribution, while allows us to compute gradients of our approximate posterior parameters, through backpropagation. The authors call this "Bayes by Backprop".

3.1.2 Laplace Approximation

Laplace's method is a simple method for approximating the posterior with a Gaussian distribution. As we know the issue with computing the posterior is computing the marginal likelihood, which forms a normalising constant on our joint distribution, i.e. $p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D},\boldsymbol{\theta})}{Z}$. We can use the original MAP from (2.5) to find the mode of our posterior, which we could use for our approximate Gaussian mean (equal to mode for a Gaussian), but we still a way of approximating the variance. If we make a Taylor expansion of the log of our MAP joint distribution (likelihood multiplied by prior) around the MAP parameters, we see that the first derivative vanishes as it is a stationary point, leaving us with:

$$\log p(\mathcal{D}, \boldsymbol{\theta}) \approx \log p(\mathcal{D}, \boldsymbol{\theta}^*) - \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^*)^T \mathbf{H}_{\boldsymbol{\theta}^*}(\boldsymbol{\theta} - \boldsymbol{\theta}^*) \tag{3.4}$$

where $\mathbf{H}_{\boldsymbol{\theta}} = -\nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} \log p(\mathcal{D}, \boldsymbol{\theta})$ is the Hessian describing the local curvature. Using Laplace's method our posterior is approximated by $p(\boldsymbol{\theta}|\mathcal{D}) \approx \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}^*, \mathbf{H}_{\boldsymbol{\theta}^*}^{-1})$, and we have the following approximation for our marginal likelihood:

$$\log p(\mathcal{D}) \approx \log p(\mathcal{D}, \boldsymbol{\theta}^*) - \frac{1}{2} \log |\frac{1}{2\pi} \mathbf{H}_{\boldsymbol{\theta}^*}| \tag{3.5}$$

As before we can condition all the above distributions on the model choice, and use the above quantity for model selection, as first proposed by Mackay [9].

Then in the case of a Neural Network, we can compute the MAP through regular gradient descent, exactly as described in (2.5), with $\boldsymbol{\theta}$ as the weights and biases of the network, and make our Laplace approximation with the trained neural network. We cover this in more detail in 4.3.

3.2 Gaussian Processes

Gaussian Processes will be of interest to us because they allow for exact (or very tight approximations to) the marginal likelihood. The standard way to train the hyperparameters of Gaussian Processes is through the use of the marginal likelihood. We can compute metrics like Mean Squared Error (MSE) and Gaussian negative Log Likelihood (GnLL) on Gaussian Processes and make comparisons to the marginal likelihood with confidence. Moreover we can use well-known comparisons between Gaussian Processes and Neural Networks.

Gaussian processes are non-parametric models, our (GP) prior is directly over functions, rather than over any parameters:

$$p(\boldsymbol{f}(\boldsymbol{x})) = \mathcal{N}(\mu(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')) \tag{3.6}$$

A Gaussian Process is completely specified by its mean function (μ) and covariance function (k). A common choice is a zero mean function, $\mu(\boldsymbol{x}) = 0 \ \forall \boldsymbol{x}$, and the squared exponential kernel, $k_{SE}(\boldsymbol{x}, \boldsymbol{x}') = \sigma^2 \exp(\frac{-(\boldsymbol{x}-\boldsymbol{x}')^2}{2l^2})$, which has the hyperparameters $\{\sigma^2, l\}$.

With small datasets and conjugate likelihoods (e.g. Gaussian likelihood for regression), it is possible to compute the marginal likelihood exactly for GPs. However with larger datasets and non-conjugate likelihoods (e.g. discrete likelihoods for classification) this computation becomes infeasible, so we need to turn to approximations again.

3.2.1 Gaussian Process Regression

We model the homogeneous noise independently of our prior distribution, using a Gaussian likelihood as in Equation 2.3. The predictive distribution conditioned on the observed training targets can then be found analytically as a Gaussian distribution. Moreover so can the (negative log) marginal likelihood:

$$\log p(\mathcal{D}|\boldsymbol{\eta}) = \frac{1}{2}\boldsymbol{y}^T(k(\boldsymbol{x}, \boldsymbol{x}') + \sigma_l^2 I)^{-1}\boldsymbol{y} + \log|k(\boldsymbol{x}, \boldsymbol{x}')| + \frac{N}{2}\log 2\pi \tag{3.7}$$

Where $\mathcal{D} = \{\boldsymbol{x}, \boldsymbol{y}\}$, $N = |\mathcal{D}|$, and we are assuming a zero mean function. Our hyperparameters ($\boldsymbol{\eta}$) for a Gaussian Process are exactly as described in section 2.3, with likelihood parameter σ_l^2, and prior parameters as any parameters of the chosen kernel function.

3.2.2 ArcCosine Kernel

In [10], several cosine kernels are derived, which are equivalent to NNs with an infinite width, where the weights are Gaussian distributed with zero mean and unit variance. With ReLU activations the kernel is:

$$J_1(\theta) = \sin\theta + (\pi - \theta)\cos\theta, \quad \theta = \cos^{-1}\left(\frac{\boldsymbol{x}\cdot\boldsymbol{y}}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|}\right) \tag{3.8}$$

This is a simple kernel which we can easily preform exact inference on. When we preform experiments on non-Bayesian Neural Networks (NNs), we will be able to see how well they approach this limit, and although we cannot compute the marginal likelihood of a NN, there is a clear connection between them and Bayesian Neural Networks.

3.2.3 NNGP

In [11], the exact equivalence between infinitely wide BNNs and GPs is derived. In order to make this equivalence, our BNN must take the form for each layer:

$$z_i^{l+1} = \frac{\sigma_w}{\sqrt{N_{in}}}\sum_j W_{ij} z_i^l + \sigma_b b_i \tag{3.9}$$

Where $W_{ij}, b_i \sim \mathcal{N}(0,1)$ at initialisation, and σ_w, σ_b are the prior weight and bias stds. As you can see we need to apply square-root width scaling to the weight std, which is equivalent to dividing the variance by the width of the layer, and ensures the equivalence is true as the width tends to infinity.

Again we can compute the exact marginal likelihood with this kernel, which should allow us to make direct comparisons to BNNs with correct marginal likelihoods.

Chapter 4

Marginal Likelihood Estimation Methods

So far we have discussed how the marginal likelihood is useful for model selection, and several Bayesian models, which allow us to approximate the posterior and marginal likelihood over a supervised learning problem. We now consider practical methods to learn parameters using marginal likelihood approximations.

4.1 Empirical Bayes by Backprop

In subsection 3.1.1 we introduced the MFVI-BNN, which uses gradients of the ELBO to optimise its approximate posterior, and whose ELBO is a lower bound to the marginal likelihood. If this lower bound is tight to the true marginal likelihood then it follows that we could make the hyperparameters of our MVFI-BNN learnable parameters, rather than fixed hyperparameters, and include their gradients in the gradient of the ELBO, thus allowing us to learn these parameters by gradient descent. In the original paper [2], the authors try optimising the prior parameters by taking derivatives and find it yields worse results, which suggests that the ELBO is not a tight bound to the marginal likelihood.

4.2 Collapsed Variational Bounds

In this paper Tomczak et al. propose training the regular MFVI-BNN with a collapsed variational bound. By collapsing the variational bound, we can marginalise out the prior parameters from our loss function, which is claimed to create a tighter bound on the marginal likelihood, and also make learning more efficient. The main limitation of this method is we still need to define distributions for the prior parameters we are marginalising out, so we are effectively adding another layer of inference to our hierarchical Bayes. However in the paper, the proposed "hyperpriors" of the distributions on the prior parameters, are claimed to be robust to many problems, and in some cases can also be including in the learning objective without weakening the bound.

The full derivations of the collapsed variational bounds we will experiment with, are included in appendices in their paper. The general steps for creating a bound are:

1. Choose distributions over the prior parameters, $p(\boldsymbol{\mu}_p), p(\boldsymbol{\sigma}_p^2)$ (along with a prior and approximate posterior distribution).

2. Calculate the prior parameters which maximise the posterior using:

$$\log q^*(\boldsymbol{\mu}_p, \boldsymbol{\sigma}_p^2) \propto \log p(\boldsymbol{\mu}_p) + \log p(\boldsymbol{\sigma}_p^2) + E_{q(\boldsymbol{W}|\boldsymbol{\mu}_q, \boldsymbol{\sigma}_q^2)} \log p(\boldsymbol{W}|\boldsymbol{\mu}_p, \boldsymbol{\sigma}_p^2) \tag{4.1}$$

3. Substitute $q^*(\boldsymbol{\mu}_p, \boldsymbol{\sigma}_p^2)$ back into the variational bound, analytically integrating out the prior parameters, to form the final collapsed bound, which we use as our learning objective.

As with the original mean-field BNN, we assume a mean-field Gaussian prior distribution, $\mathcal{N}(\boldsymbol{W}|\boldsymbol{\mu}_p, \boldsymbol{\sigma}_p^2 I)$ (and mean-field Gaussian posterior), not least because it is important we can analytically integrate out the prior parameters. Then the first proposed bound marginalises out the prior mean parameter, giving it a Gaussian hyperprior (the authors just call this the prior, since we have collapsed the bound anyway), $\mathcal{N}(\boldsymbol{\mu}_p|\mathbf{0}, \alpha I)$. Then the hyperparameters of the resulting model are α[1] and γ, where $\gamma\mathbf{1} = \boldsymbol{\sigma}_p^2$ keeping the prior variance constant across all the weights (and biases). Since we are making the mean-field assumption for the prior and posterior, we can write the resulting bound for one weight, and sum over all weights. We will refer to this bound as the Collapsed Mean ELBO (CM-ELBO):

$$\frac{1}{2}\left(\frac{\alpha_{reg}\mu_{q,w}^2 + \sigma_{q,w}^2}{\gamma} + \log\gamma - \log\alpha_{reg} - \log\sigma_{q,w}^2 - 1\right) \tag{4.2}$$

Where $\alpha_{reg} = \frac{\gamma}{\gamma+\alpha}$ controls the regularisation to the prior. The authors found $\alpha_{reg} = 0.05$ to be robust over many problems. They also suggest letting α_{reg} be trainable, by including its gradient with the gradient of the CM-ELBO bound.

For the second bound, we marginalise out the prior mean and variance, referring to it as the Collapsed Mean Variance ELBO (CMV-ELBO). Our prior is then $p(w) = \mathcal{N}(w|\mu_p, \frac{1}{\tau_p})$, where the mean hyperprior is drawn from $p(\mu_p|\tau_p) = \mathcal{N}(\mu_p|0, \frac{1}{t\tau_p})$, and the (inverse) std hyperprior is drawn from $p(\tau_p) = \Gamma(\tau_p|\alpha, \beta)$. Then with $\delta = \frac{t}{1+t}$, the bound becomes:

$$(\alpha + \frac{1}{2})\log[\beta + \frac{\delta\mu_{q,w}^2 + \sigma_{q,w}^2}{2}] - \frac{1}{2}\log\sigma_{q,w}^2 - \log\Gamma(\alpha + \frac{1}{2}) + \Gamma(\alpha) - \alpha\log\beta - \log\delta \tag{4.3}$$

4.3 Scalable Marginal Likelihood Estimation

In [12], Immmer et al. builds upon the Laplace approximation we introduced in 3.1.2. They point out that the Hessian necessary to compute the marginal likelihood has length and width equal to the number of weights of the model, and thus it is generally infeasible to compute the determinant or inverse for large models. The paper then explores scalable approximations to the Hessian. In this report we experiment with approximations based on the Generalized Gauss–Newton method (GGN). GGN approximates our Hessian using the Jacobian matrix over the model parameters, as well as the Hessian of the log-likelihood and the log-prior. We then use this approximation method (we refer to this method as the Full GGN Laplace), and the Kronecker-factored GGN Approximation, which uses a block-diagonal approximation to the GGN Hessian (we refer to this method as the Kron GGN Laplace).

[1] Earlier we defined α as a vector containing all prior parameters, while here it only represents the hyperprior variance on the prior mean. Similarly other symbols are reused in this section, I thought it best to agree with the reference paper than define new symbols.

Chapter 5

Experiments

This section will lead the reader through the experiments preformed to make comparisons for the models and methods described in the previous section. The format is generally chronological, to justify the rationale for each experiment/measurement as they are made.

5.1 Toy Regression Dataset

We test these methods on a simple regression problem of my creation. The aim is to compare these methods of marginal likelihood estimation and training on an explainable problem, and one in which we can compute exact or tight Gaussian processes on. The hope is that we can find evidence of the strengths and weaknesses of the proposed empirical methods. If they preform badly on a simple problem, there is little hope they will be robust on more complex problems.

To demonstrate the ability of Bayesian Neural Networks to compute uncertainty, we want this dataset to have open ends for extrapolation, and gaps in the data for interpolation. We would also like this function to be reasonably smooth, so that it is possible to make good predictions to the test set, given that it will have limited data.

To this end, we generated our dataset (Figure 5.1) by picking a dozen points in what we imagined to describe a roughly cubic graph, and normalised these points to have mean $y = 0$, and x in $[0, 1]$. We then fitted a Gaussian Process with a Matérn$^{5/2}$ kernel to this data, and sampled a single function from this GP, which will be our ground truth for testing. The resulting function is roughly septic. Finally, for the training data, we picked the x coordinates 0.2, 0.5 and 0.8, and generated 40, 20 and 40 points respectively, from $\mathcal{N}(x, 0.04^2)$ around those x, to create clusters of training points. Then for the corresponding y's, we added homogeneous noise from $\mathcal{N}(y, 0.1^2)$. We experimented with these parameters a little by seeing if GPs predicted interesting uncertainty estimates on the test set. In particular. more training points, or higher stds on the distribution of points, made the problem too easy.

Notably the training data does not include the minima around $x = 0.7$ inside (interpolation) the data, and the minima around 0.95 outside (extrapolation) the data. The hope is that the Gaussian likelihood function will correctly identify the homogeneous noise, while the approximate or exact posterior of our models will capture the underlying distribution the data is drawn from, free of noise, and the epistemic uncertainty caused by the lack of data in the interpolation and extrapolation areas of our graph.

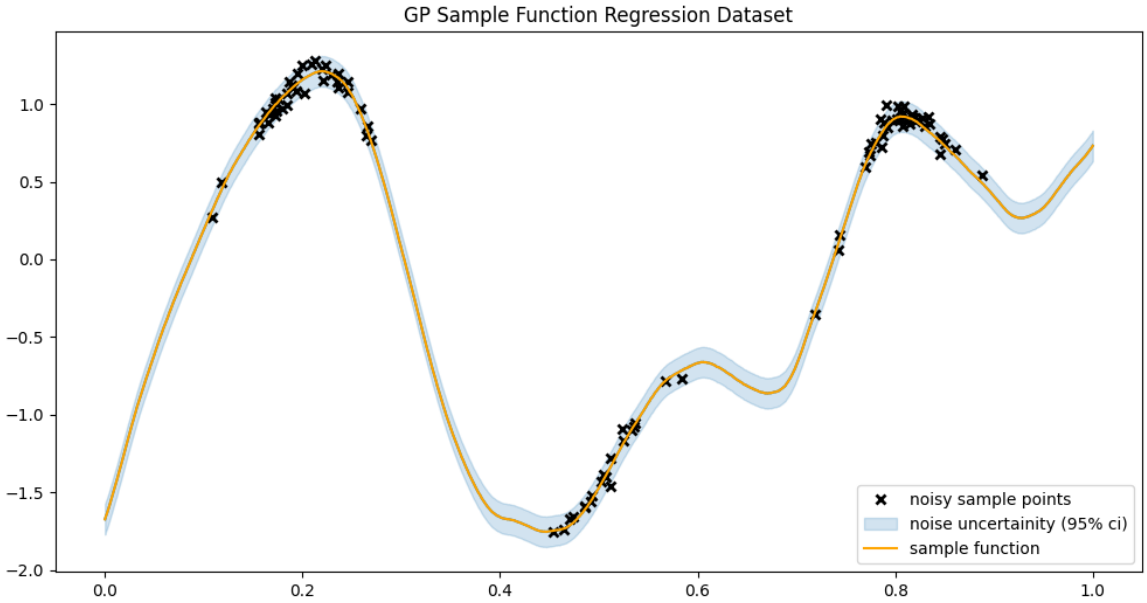


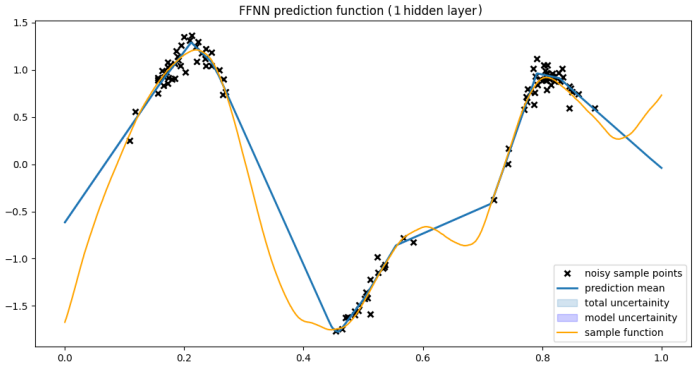
Figure 5.1: Regression dataset generated by sampling a function from a Gaussian Process. The training points are generated with homogeneous Gaussian noise, where the 95% confidence interval of the noise is included in the plot.

5.2 Non-Bayesian Neural Network

First, we fit a common fully connected feed forward (non-Bayesian) neural network to the dataset. By fitting a regular NN we can ensure our BNNs also have the capacity to overfit, as a BNN has double the parameters of an equivalent NN. As described in the last section, we want to make good comparisons to Gaussian Processes by utilising infinite width limits of (Bayesian and non-Bayesian) neural networks, so to this end the layer width over each layer will be fixed at 50. We find that increasing the layer width above 50 did not significantly improve the performance of NNs or BNNs on this problem. We use ReLU activations, and the root mean squared error (RMSE) loss function.

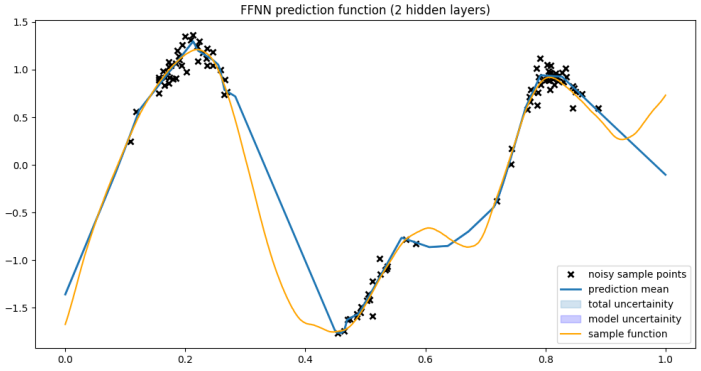
To start with we train NNs without regularisation, varying the number of layers. We see it is easy for deeper neural networks to overfit on this noisy dataset (Figure 5.2). In training the NN with 3 hidden layers, the low minima in the test MSE (Figure 5.2(e)) suggests a NN with at least 3 hidden layers is capable of being a very good fit (less hidden layers may also be capable), which is to say the function space includes a very close approximation to the true function. However we need to apply the correct regularisation to find this function, which is the essence of hyperparameter tuning on NNs, and requires use of validation sets.

((a)) Learning curve for 1 hl NN.



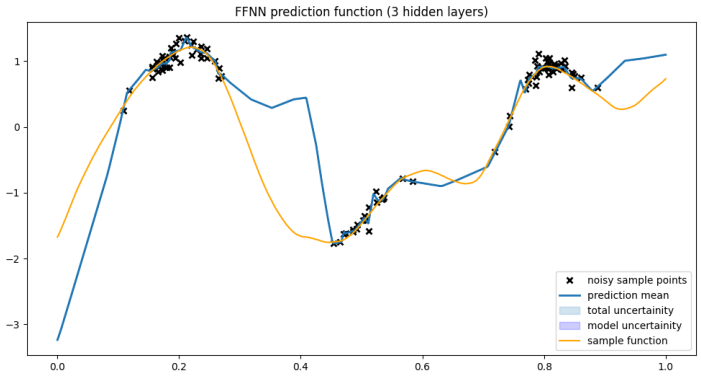
((b)) 1 hl NN output on test set and true function.

((c)) Learning curve for 2 hl NN.



((d)) 2 hl NN output on test set and true function.

((e)) Learning curve for 3 hl NN.



((f)) 3 hl NN output on test set and true function.

Figure 5.2: Fitting a NN on regression dataset, with no regularisation. The learning curves on the left show the test MSE as the network trains, demonstrating how they overfit.

To this end, we preform 5-fold cross-validation over a grid search of weight decay (L2 regularisation) and number of layers (and width). For a selection of hyperparameters, we then train over the whole training set, and find the test RMSE. The hyperparameters, average RMSE over the (cross) validation sets, the RMSE on the training set, and the RMSE on the test set are included in Table 5.6. The average CV RMSEs are all close and there is large variability in the cross-validation due to the small and clustered training sets. Although the best CV RMSE does predict the best Test RMSE here, in repeats of this experiment this was not always true.

Cross Validation for Non-Bayesian Neural Network				
Num. Layers	Weight Decay	Avg. CV RMSE	Train RMSE	Test RMSE
1	1e-4	0.128	0.132	0.316
2	1e-4	0.122	0.140	0.298
3	1e-4	0.123	0.131	0.301
4	1e-4	0.121	0.125	0.261
1	1e-6	0.129	0.138	0.337
2	1e-6	0.126	0.126	0.320
3	1e-6	0.137	0.114	0.375

Table 5.1: Results of NN Cross Validation over number of layers and weight decay. Note: the Train RMSE includes weight decay, so the values are higher than that of the Avg. CV RMSE.

The best model found has a Test RMSE of 0.261, using a NN with 4 hidden layers and weight decay of 1e-4. Observing this fit on over the test set, we see that the model is not smooth, and is definitely overfitting to the training data.

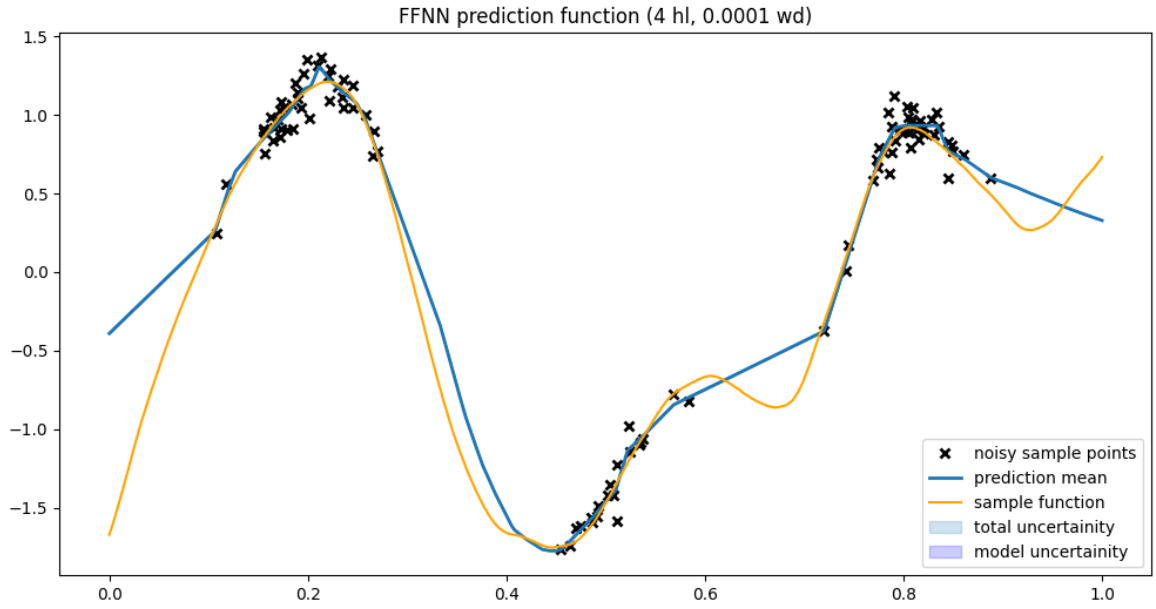


Figure 5.3: Training NN on the best hyperparameters found via cross validation: 1 hidden layer and 1e-6 weight decay.

The main takeaway from these experiments on NNs is that this regression problem is difficult. The hope is that when our models understand uncertainty, they will be able to better differentiate noise in the training data, and epistemic uncertainty, which should prevent the models from overfitting (along with regularisation to their priors).

5.3 Gaussian Processes

Next we fit several Gaussian Processes to the data. Gaussian Processes preform excellently on simple regression problems, and since the training data was sampled from a function which was sampled from a Gaussian process, is is expected that these will fit well. Moreover as this is a regression problem, we can use a conjugate Gaussian likelihood, which allows us to calculate an exact Gaussian process. As such Gaussian processes will form our gold standard for marginal likelihood estimations.

5.3.1 Simple Kernels

Our Gaussian Process kernels automatically learn their hyperparameters from calculating their exact marginal likelihoods, so we can be confident in their hyperparameter selection, and need not compare their estimates. Instead we look at a few different kernels. I chose the Matern$\frac{5}{2}$ kernel, since this is the kernel we used to generate the dataset, the Squared Exponential kernel, as one of the most common Gaussian Process kernel for regression, and then the Arc-Cosine kernel as described in subsection 3.2.2, which is equivalent to an infinite width NN with 1 hidden layer.

Then in Table 5.2, we observe the negative log marginal likelihood (nLML) on the train-set, which is our learning objective (and central point of discussion for this report), the RMSE on the train-set, which is the traditional learning objective for maximum likelihood regression models, the RMSE on the test set, and the mean Gaussian negative log likelihood (GnLL), which for each test point uses the std predicted by our as the std of the Gaussian likelihood, the idea being to test the GP's ability to give accurate uncertainty measures. We will use this last metric with other models, so to be completely clear we will include the equation:

$$GnLL(y, \mu_y, \sigma_y) = \frac{1}{2}\left(2\log\sigma + \log 2\pi + \left(\frac{y-\mu_y}{\sigma_y}\right)^2\right) \tag{5.1}$$

This is the GnLL for point x, with target y, and predicted mean μ_y and predicted std σ_y. The Test GnLL is the the mean of this metric over all test points.

Gaussian Process Model Results on Toy Regression					
Kernel	Lik std	nLMargLik	Train RMSE	Test RMSE	Test GnLL
Matern52	0.102	-56.220	0.095	0.274	-0.610
SqExp	0.102	-57.198	0.096	0.310	-0.563
ArcCos (ReLU)	0.102	-47.803	0.094	0.211	-0.624
ArcCos (Tanh)	0.103	-39.769	0.097	0.258	-0.642

Table 5.2: Loss Metrics for different GP kernels. The best (lowest) metrics are shaded blue.

After training the Gaussian Processes, we see that a likelihood std very close to 0.1 is found with every kernel using the nLML as the learning objective, which is the original std of the homogeneous noise added to the training data. This is a good demonstration of the value of the marginal likelihood as a learning objective. Relative to other models we will use, the nLML estimates for each kernel are all very close. The lowest MSE on the test set is achieved with the ArcCosine kernel using the ReLU activation (Figure 5.5), and the Test GnLLs are all

very close, but also generally preferred the ArcCosine kernel. However this is not predicted by the nLML, which is lowest for the Squared Exponential kernel (Figure 5.6). The marginal likelihood generally embodied Occam's Razor, so it is not a surprise that the simplest model is preferred. However this is a good demonstration of why more parametric models like Bayesian Neural Networks are useful. The assumption is that given our uninformative prior (or hyperprior considering we are learning prior parameters here), the most probable model given the data is a simpler model than the actual model, hence why the nLML prefers that model, however are actual test function is more parametric.

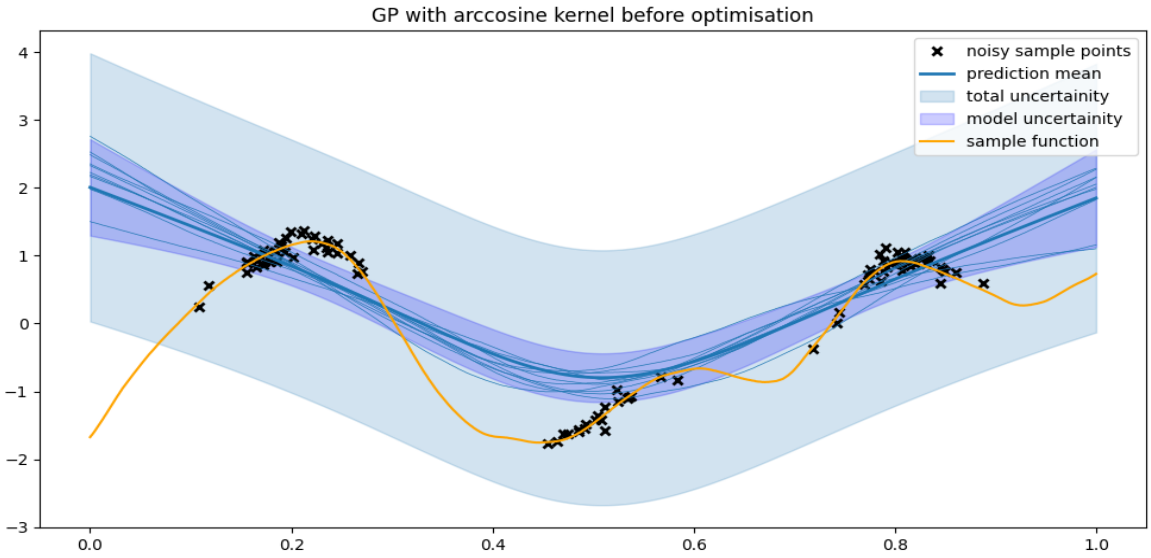


Figure 5.4: Exact Gaussian Process on Toy Regression using ArcCosine Kernel (equivalent to 1 hidden layer NN with ReLU activations), before hyperparameter optimisation.

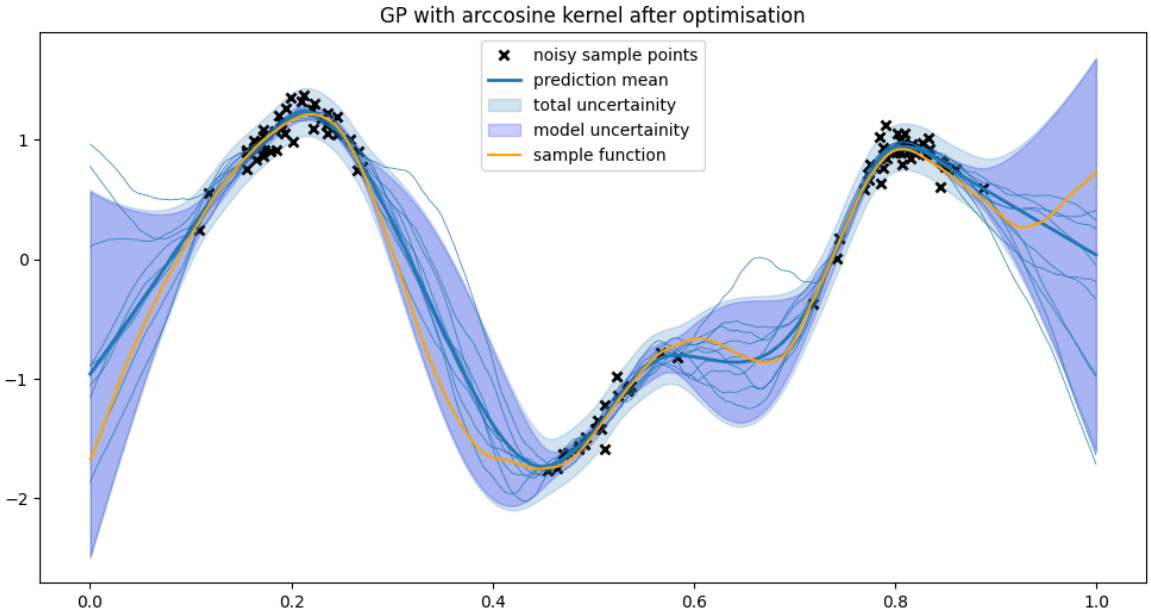


Figure 5.5: Exact Gaussian Process on Toy Regression using ArcCosine Kernel (equivalent to 1 hidden layer NN with ReLU activations), after hyperparameter optimisation.

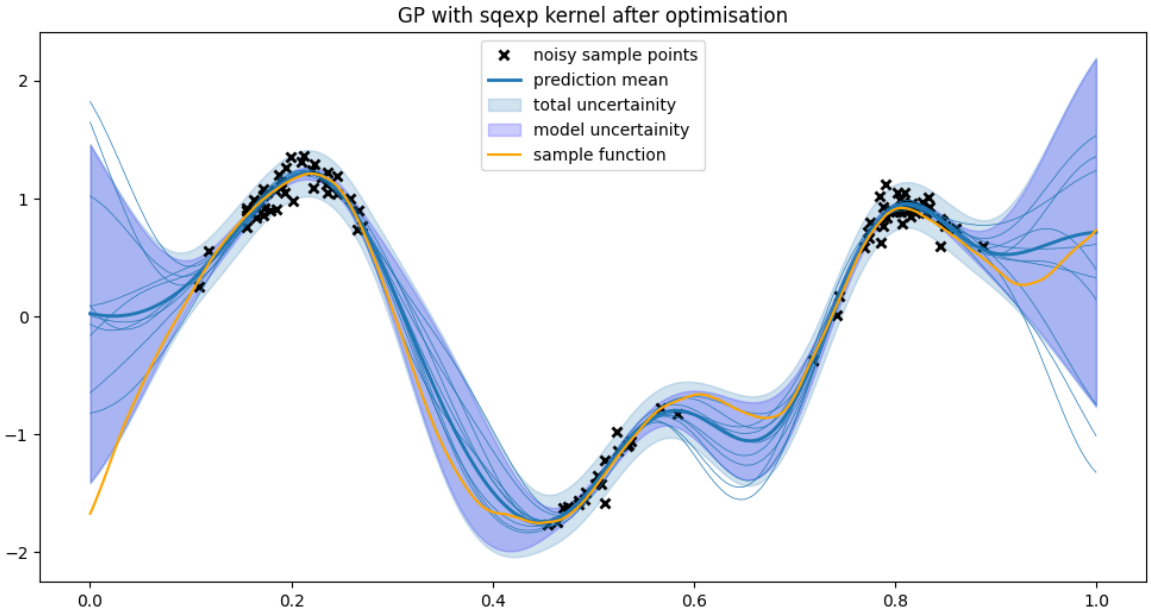


Figure 5.6: Exact Gaussian Process on Toy Regression using Squared Exponential Kernel, after hyperparameter optimisation.

In our later BNN experiments we will see that lower likelihood stds than the std of the homogeneous noise, preform better on test set metrics. To this end, we preform further experiments on these Gaussian Process kernels, with the likelihood stds fixed to a reduced number. When finding hyperparameters for BNNs using cross-validation over the MSE, a likelihood std of 0.05 preforms well (section 5.4), and when fixing the same likelihood std on our GPs, we see similar results (Table 5.3). On inspection of the plots, the higher likelihood std learnt by the GP causes it to fit less tightly to the curvature of the training data (Figure 5.5). In particular this causes the data to under-predict the gradient when extrapolating to the left (Figure 5.8), compared to when the likelihood std is lower. This is perhaps an unfortunate result of our regression problem being in the form of clusters and not a cause for alarm as the test metrics are all still relatively close. The lowest nLML here does successfully predict the best Test GnLL.

If we decrease the likelihood std more, the Gaussian Process collapses, completely overfitting the model (Figure 5.9), hence the low Train RMSE, but very high nLML (and Test MSE).

Gaussian Process Model Results on Toy Regression					
Kernel	Lik std	nLMargLik	Train RMSE	Test RMSE	Test GnLL
Matern52	0.05(fixed)	17.666	0.093	0.187	-0.969
SqExp	0.05(fixed)	19.971	0.096	0.208	-0.888
ArcCos (ReLU)	0.05(fixed)	22.992	0.091	0.151	-0.920
ArcCos (ReLU)	0.02(fixed)	230.392	0.030	0.711	-0.552

Table 5.3: Loss Metrics for different GP kernels with learnable and fixed likelihoods. The lowest metrics for learnable and fixed likelihoods are shaded blue and red respectively.

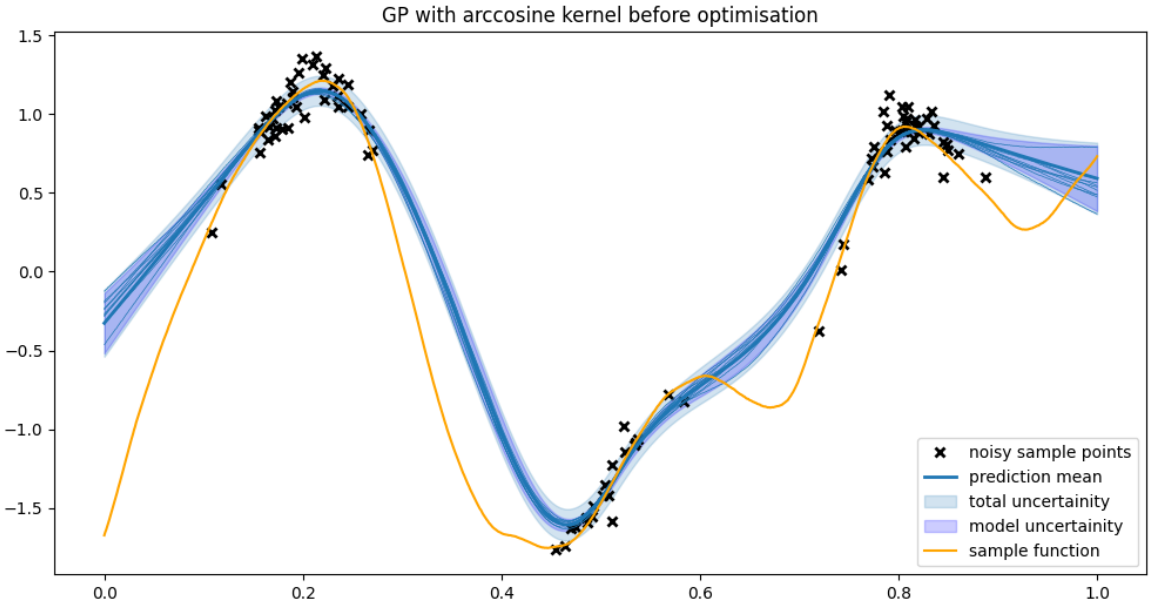


Figure 5.7: Exact Gaussian Process on Toy Regression using ArcCosine Kernel and a fixed likelihood std of 0.05, before hyperparameter optimisation.

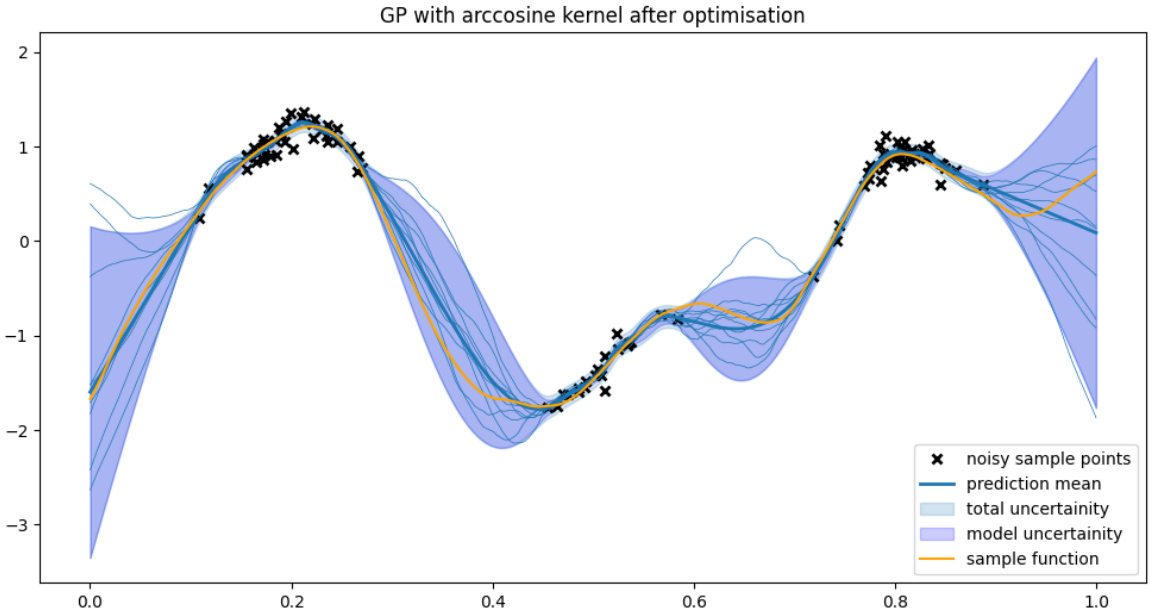


Figure 5.8: Exact Gaussian Process on Toy Regression using ArcCosine Kernel and a fixed likelihood std of 0.05, after hyperparameter optimisation.

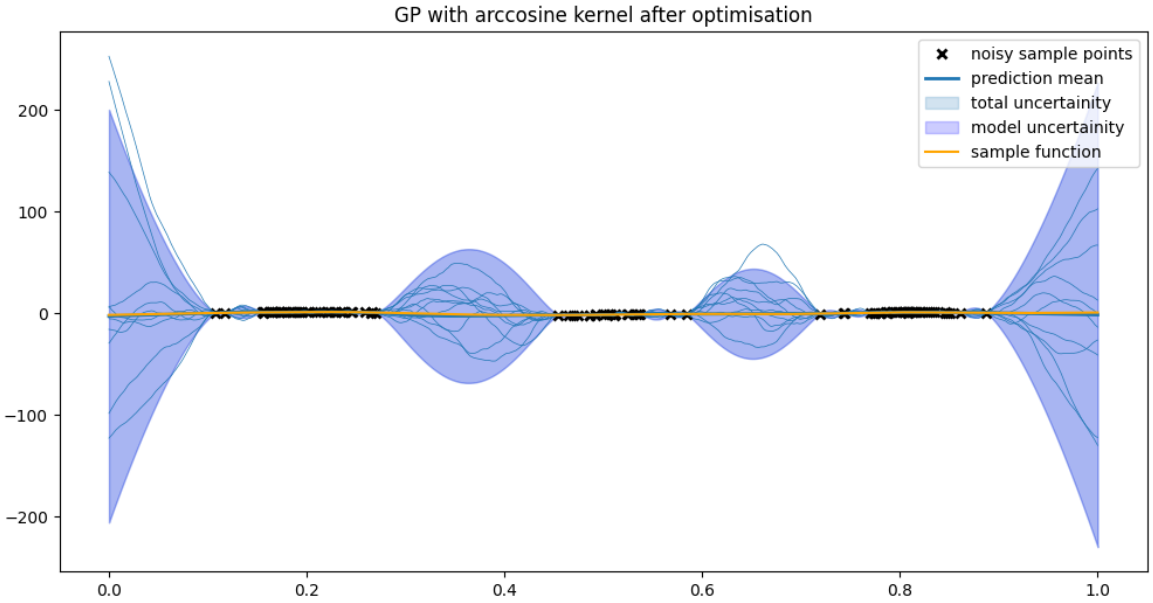


Figure 5.9: Exact Gaussian Process on Toy Regression using ArcCosine Kernel and a fixed likelihood std of 0.02, after hyperparameter optimisation.

5.3.2 Neural Network Gaussian Process

We now turn to our Neural Network Gaussian Process (NNGP) kernel, which should allow us to make exact comparisons to Bayesian Neural Networks.

As explained in subsection 6.1.2, we need to train our NNGPs by preforming a discrete search over hyperparameters using the marginal likelihood as the training objective. We set one parameter for the weight std and a second for the bias std for the whole NNGP, which means every weight and every bias across all the layers has the same std respectively. We also search over the likelihood std, and leave the number of layers as a hyperparameter to make comparisons with as a defacto for model complexity, returning the best results in Table 5.5.

NNGP Results on Toy Regression						
Num. Layers	nLML	lik std	weight std	bias std	Test RMSE	Test GnLL
1	-90.1	0.092	2.01	1.16	0.291	2.120
2	-88.0	0.102	2.51	1.69	0.251	0.383
3	-92.2	0.095	1.79	1.44	0.233	0.329
4	-90.1	0.076	1.26	0.93	0.243	2.182
5	-92.7	0.094	1.76	0.52	0.284	0.602

Table 5.4: Loss Metrics for NNGP based on NLML on train varying number of layers.

The results are generally what we expected, all the nLML are low, in fact lower than we saw with simpler kernels. There is more variation in the hyperparameters, as see when we vary the number of layers. This is expected as we are not using a gradient based learning method for hyperparameters, and all are marginal likelihoods are very close together (after finding good hyperparameters). We also see good results on the test metrics after training, although

not as good as our simpler kernels. Our best model chosen from the nLML (Figure 5.12) is not the best model seen on test metrics, but the best model (Figure 5.11) has a very similar nLML. We will keep this in mind when we make absolute comparisons at the end of the experiment. Observing the plots, the NNGPs are generally a little overconfident with their uncertainty predictions between training data clusters, although our test curve is generally within the 95% confidence interval.

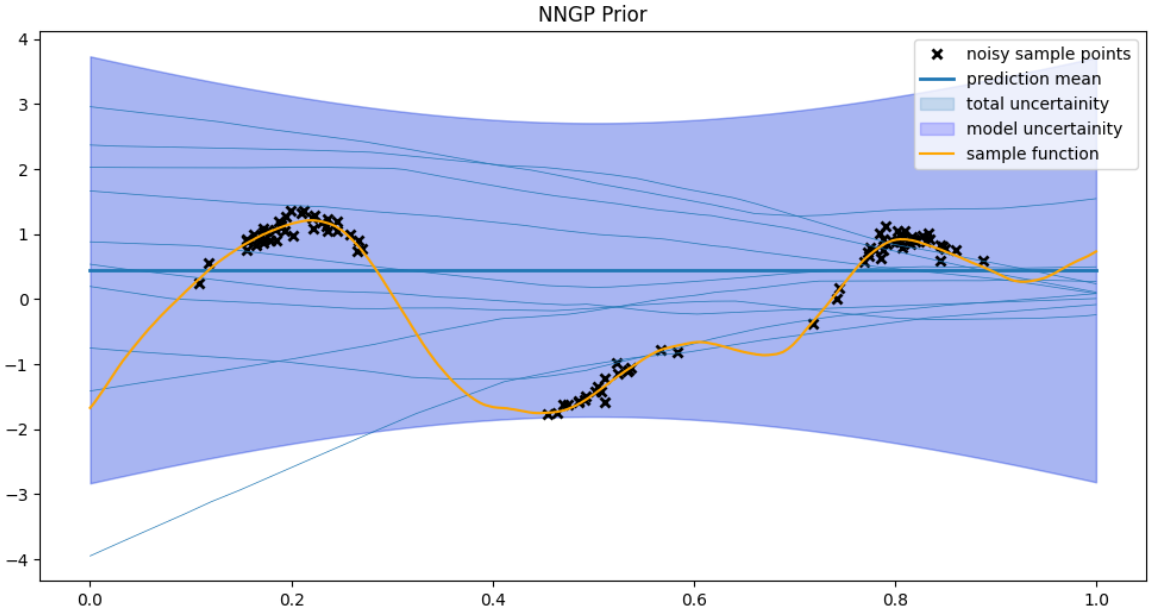


Figure 5.10: NNGP intialised at Standard Normal prior.

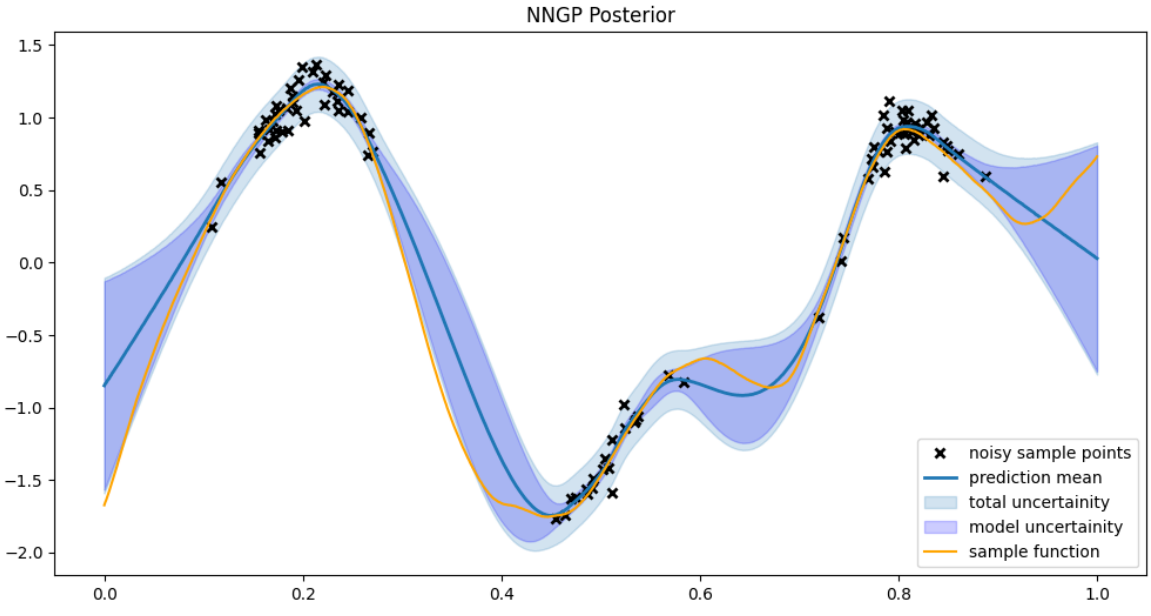


Figure 5.11: Best NNGP found on nLML with 3 hidden layers.

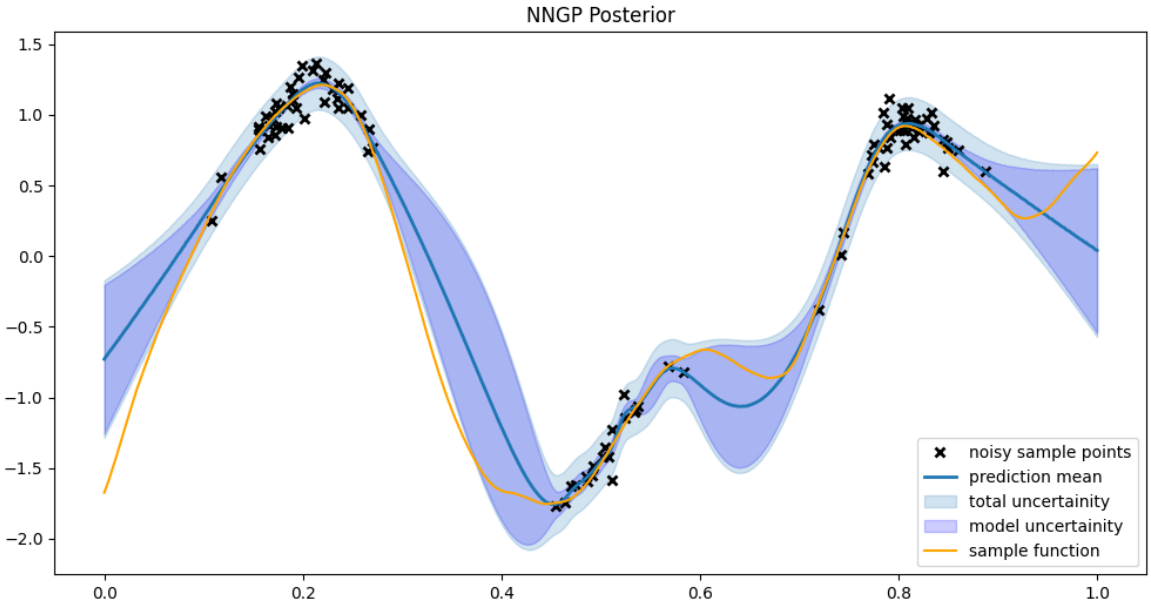


Figure 5.12: Best NNGP found on nLML with 5 hidden layers.

To see the best model NNGPs are capable of achieving in theory, we did two more searches using the MSE and GnLL respectively, on the **test set**. Naturally in practice we cannot preform model selection over test metrics. The best model over MSE (Figure 5.13) achieves a RMSE of 0.211, which is only 0.073 lower than the RMSE from the model with the lowest nLML. Expectedly, this model has bad uncertainty predictions, resulting in a GnLL of 5.010 and a NaN (or infinite) nLML, as the model was not trained on a metric which involves uncertainty (RMSE only uses the mean prediction, not the predicted std). The best model on GnLL (Figure 5.14) achieves a GnLL of 0.060, which is significantly lower than the GnLL from nLML. Our NNGPs give overconfident uncertainty predictions when using the nLML on training data, which is a theme we will also see with our BNNs. The best model on GnLL under-predicts the likelihood, which is not surprising as the test data does not contain noise.

Num. Layers	nLML	lik std	weight std	bias std	Test RMSE	Test GnLL
2	49.6	0.038	2.26	2.51	0.234	0.060
5	NaN	n/a	1.76	0.52	0.211	5.010

Table 5.5: Best NNGP models possible based on test metrics

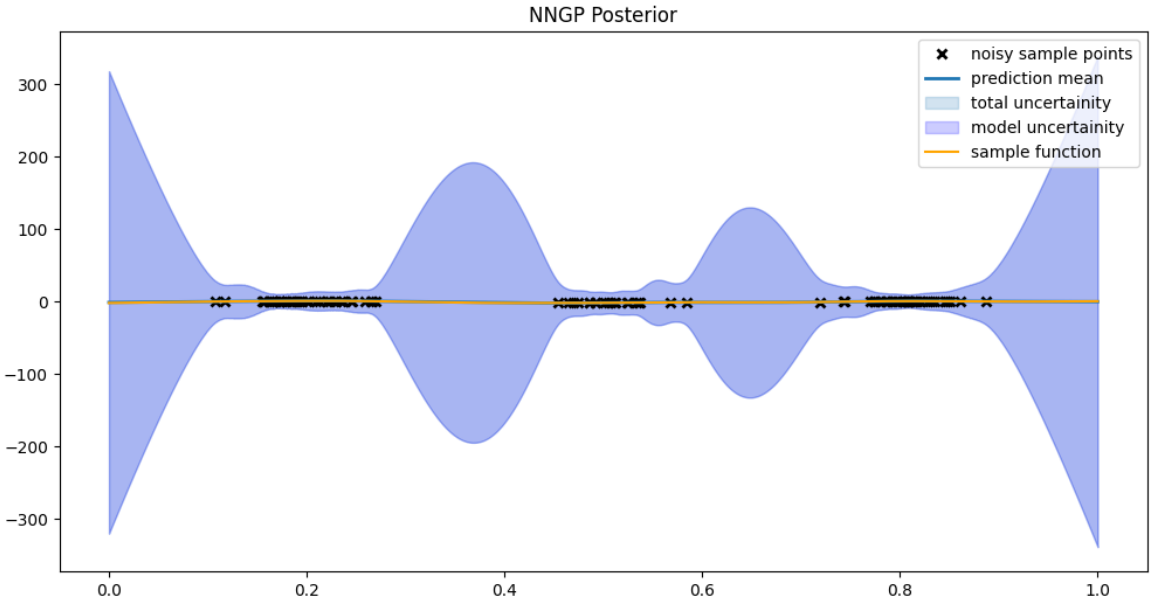


Figure 5.13: Best NNGP on Test MSE.

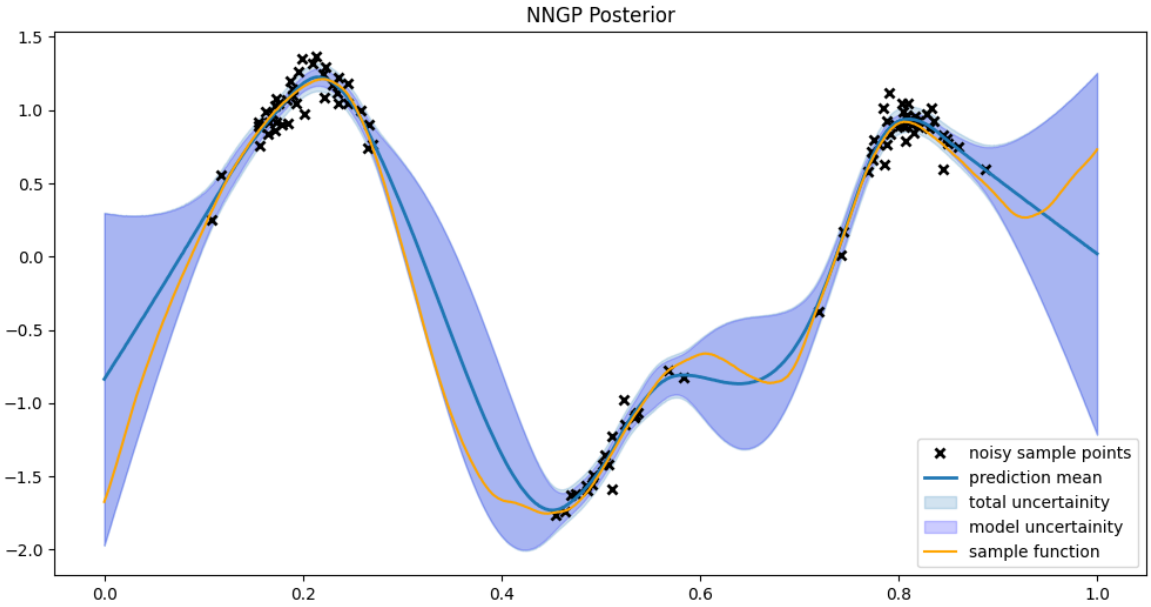


Figure 5.14: Best NNGP on Test GnLL.

5.4 First Inference Bayesian Neural Network

Our aim with experiments for the first inference Bayesian Neural Network, is to create a strong benchmark for the capabilities of BNNs on this particular problem, without learning hyperparameters using marginal likelihood approximations. The first observation we make is the sensitivity the BNN has to its hyperparameters, especially the prior weights, and initial conditions.

We start by fitting a BNN with a fixed uninformative prior. The training data is normalised to have mean 0 and std 1 before it enters the model, so we choose a standard normal prior with width scaling for the prior weight std, to make the model comparable to a NNGP (and because it generally improves the performance of BNNs). Then initialising the network from the prior gives the distribution $\mathcal{N}(\mu, \sigma)$ over the data, where μ, σ are the mean and std of the data respectively, as we see in Figure 5.15. Observing the plot of this prior, it seems reasonable that it would be suitable for training a BNN, but that a vaguer prior might preform better, as this prior will assign lower probability to data on the edge of the confidence interval, than data near the mean.

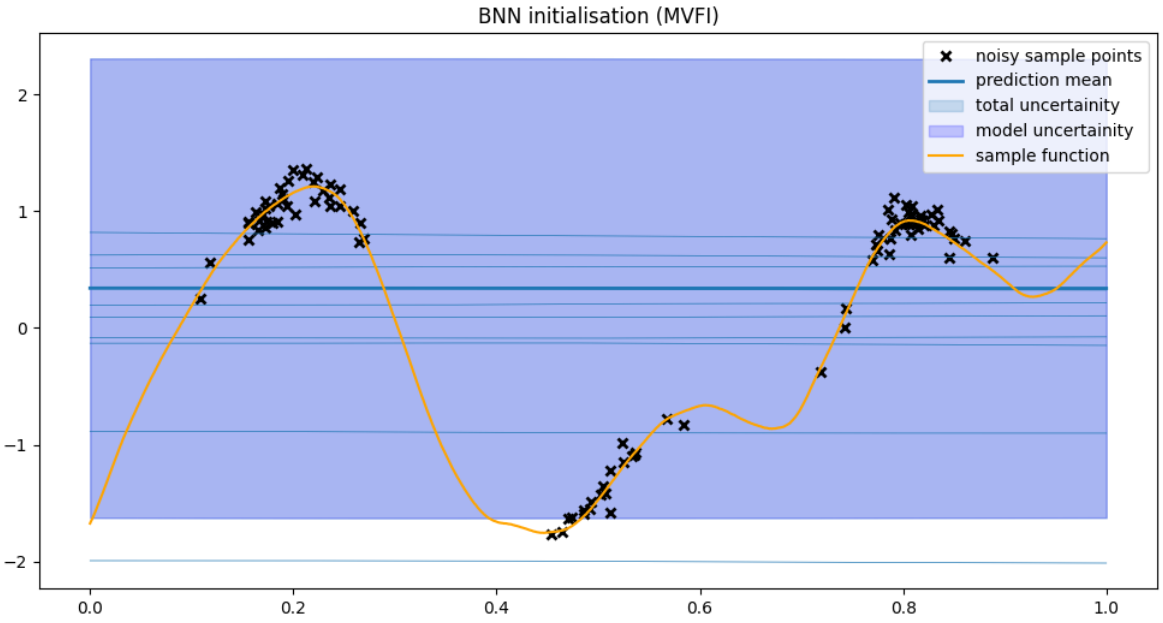


Figure 5.15: Standard BNN prior (uncertainty plotted is 1.96σ, explaining 95% of the data).

As with previous experiments, we vary the number of layers to see how the ELBO preforms as a training metric with respect to the complexity of the model, and include metrics in Table 5.6. The metrics on the train set are the likelihood cost (negative Gaussian log likelihood) and complexity cost (prior regulation) whose sum equals the negative ELBO, and the RMSE on the Train set. The Train RMSE and likelihood cost are proportional to each other, however the RMSE is included as it allows us to make comparisons to non-Bayesian models. The GnLL on the test set again uses the std predicted by the BNN as the std of the Gaussian likelihood for each test point, as with our Gaussian Processes.

BNN Results on Toy Regression with standard normal prior						
Num. Layers	nLikeli-hood	Compl-exity	nELBO	Train RMSE	Test RMSE	Test GnLL
1	2238.2	221.3	2459.5	0.519	1.124	6.273
2	916.8	414.4	1331.2	0.322	0.806	2.427
3	1014.5	488.6	1503.1	0.335	0.840	2.366
4	1313.5	511.6	1825.1	0.377	0.894	2.937
5	1402.8	586.7	1989.5	0.391	0.875	2.677

Table 5.6: Loss Metrics for standard BNN initialised at prior.

The first observation is that all the nELBOs are magnitudes higher than the nLMLs found with NNGPs, or indeed regular GPs. This is already good evidence than the ELBO is not a tight bound on the marginal likelihood, however the Test RMSEs and GnLLs are also all far higher than that of the GPs, so we cannot yet rule out that these are good marginal likelihood approximations.

The ELBO does give us the model with the best MSE on the test set (Figure 5.16). However, so does the train MSE/likelihood. The standard normal prior is over regulating the model, causing it to underfit, as we predicted when observing the prior. As we increase the number of layers of the model, we increase the number of parameters (linearly), and thus the effect of the complexity cost, causing the model to underfit more severely (see Figure 5.17). Interestingly, the model with the best test GnLL has 3 layers, even though this is not predicted by any training metric. I would hypothesise this is because the model does not fit well (test RMSE far above that of a regular NN) on any number of layers, but the increased regulation to the prior with more complexity, increases the predicted std, and thus the test GnLL penalises the bad fit less.

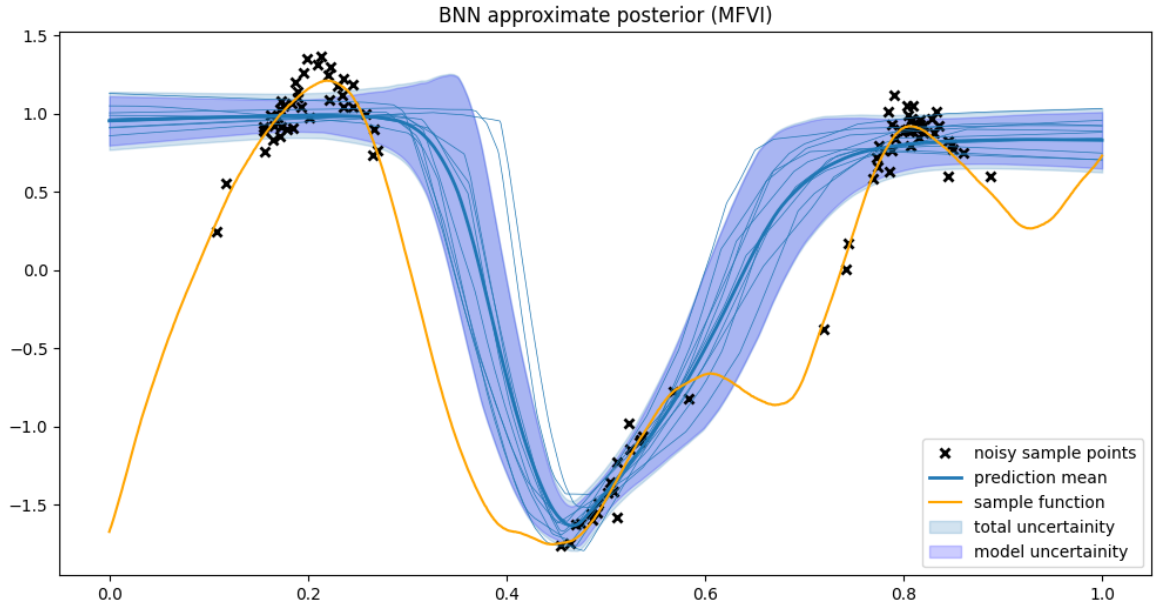


Figure 5.16: Standard BNN after training with 2 layers

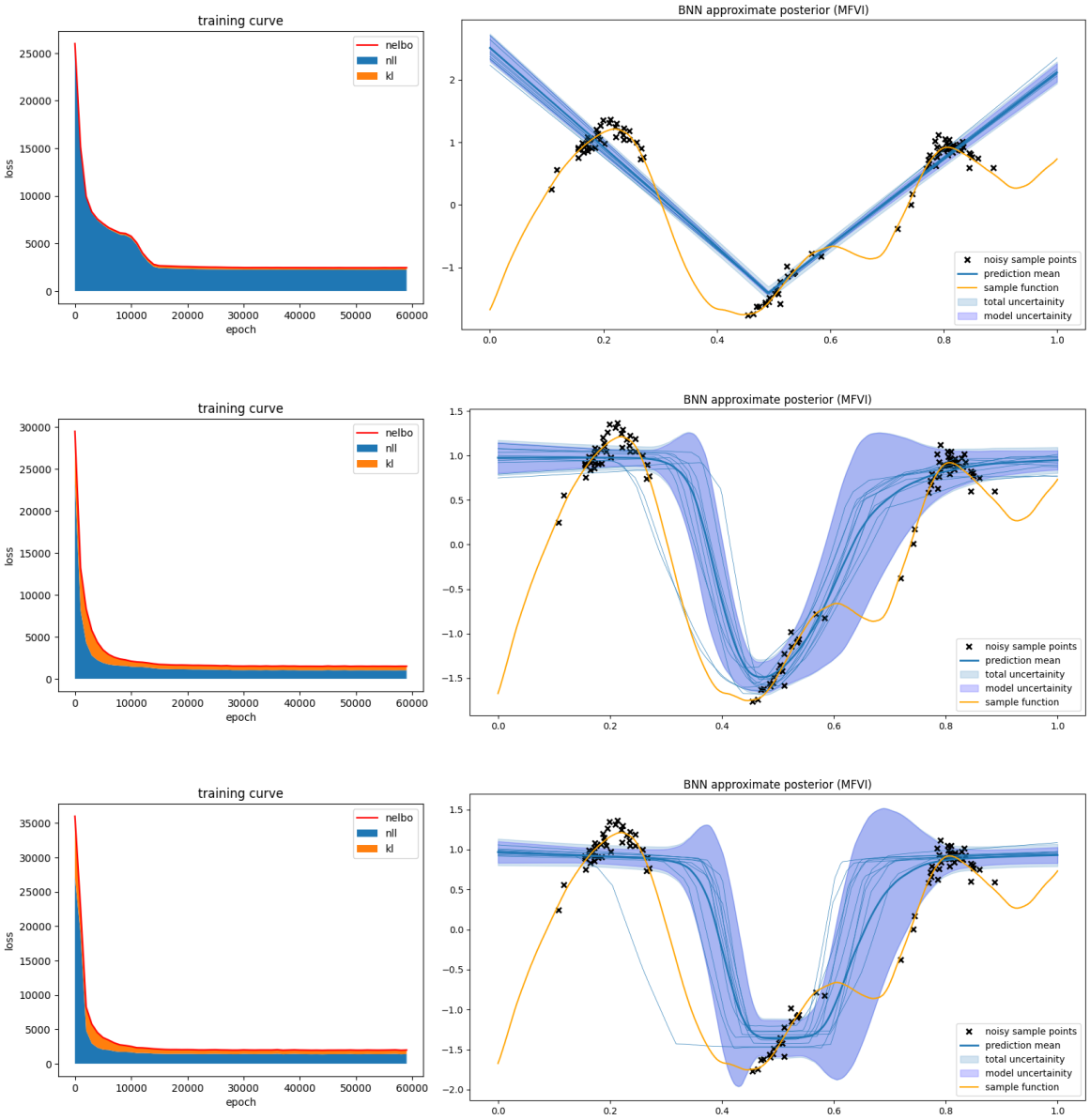


Figure 5.17: Standard BNN initialised at prior with 1, 3 and 5 hidden layers respectively.

There is another reason the BNNs do not fit well. We find BNNs on this problem to be very sensitive to the initialisation of their approximate posterior, or more specifically, when the approximate posterior is initialised at a vague prior it does not converge to a good fit, settling presumably to a local minima. If the ELBO is not tight to the true marginal likelihood, we might expect an uneven training (ELBO) surface. Initialising instead at the likelihood std gives better results for each number of layers, and the best model (Figure 5.19) now has 3 layers, which gives an ELBO of 671.9, and a Test RMSE of 0.303. This is again promising, as the lowest ELBO we have seen aligns with the lowest RMSE we have seen. The results of initialising at the likelihood and varying over number of layers are very similar to the results from choosing parameters via cross-validation, so for the sake of brevity we have emitted them.

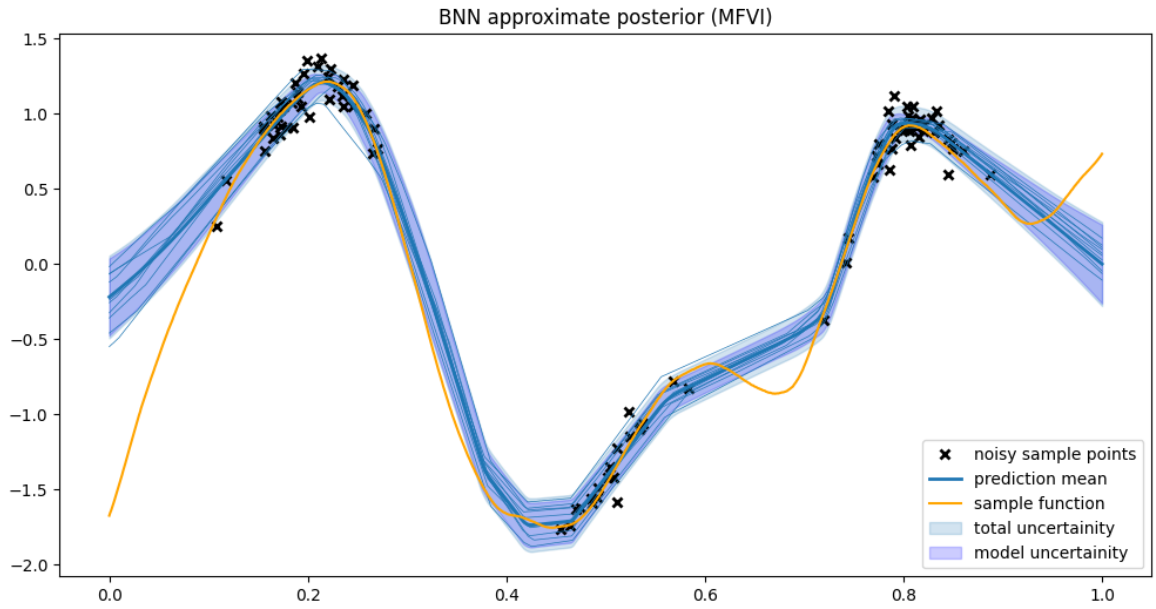


Figure 5.18: BNN initialised at likelihood with standard normal prior.

The results of the last experiment, show over regulation to the prior, resulting in bad fits to the test data. To find better priors without using the marginal likelihood, we preform 5-fold cross-validation over the training set, using RMSE as the test metric, and varying the initial std on the weights and biases, the likelihood std, the number of layers, the prior weight std, and the prior bias std, with several grid-searches. These models take a while to train, so we cannot claim the search was exhaustive, but it did result in better hyperparameters than with the standard prior, which should allow us to make good comparisons. Then we train a BNN over the whole training set using the best hyperparameters found via cross-validation, and varying the number of layers (Table 5.7).

BNN Results on Toy Regression with standard normal prior						
Num. Layers	nLikeli-hood	Compl-exity	nELBO	Train RMSE	Test RMSE	Test GnLL
1	922.0	791.0	1713.0	0.145	0.342	0.842
2	783.3	1026.2	1809.4	0.134	0.391	0.608
3	779.5	1310.7	2090.2	0.130	0.217	-0.238
4	844.8	1593.5	2438.4	0.130	0.318	-0.072
5	860.8	1859.9	2438.4	0.134	0.375	-0.018

Table 5.7: Loss Metrics for best BNN found via cross-validation

The best hyperparameters found are a likelihood std of 0.02, a prior weight std of 2.0, a prior bias std of 5.0, and 3 hidden layers. As we discussed with our Gaussian processes, it not generally expected that a likelihood std lower than the noise of the model would result in a better fit, but it should also be noted that we are maximising MSE here and not GnLL. These results are clearly not predicted by the ELBO. The nELBOs are higher than for the BNNs we trained on a standard prior, despite the RMSEs being lower. Cross-validation using RMSE as a metric did predict the best RMSE on the test set of 0.217, which is lower than what we achieved with a standard NN, but not as low as the ArcCosine (ReLU) Kernel.

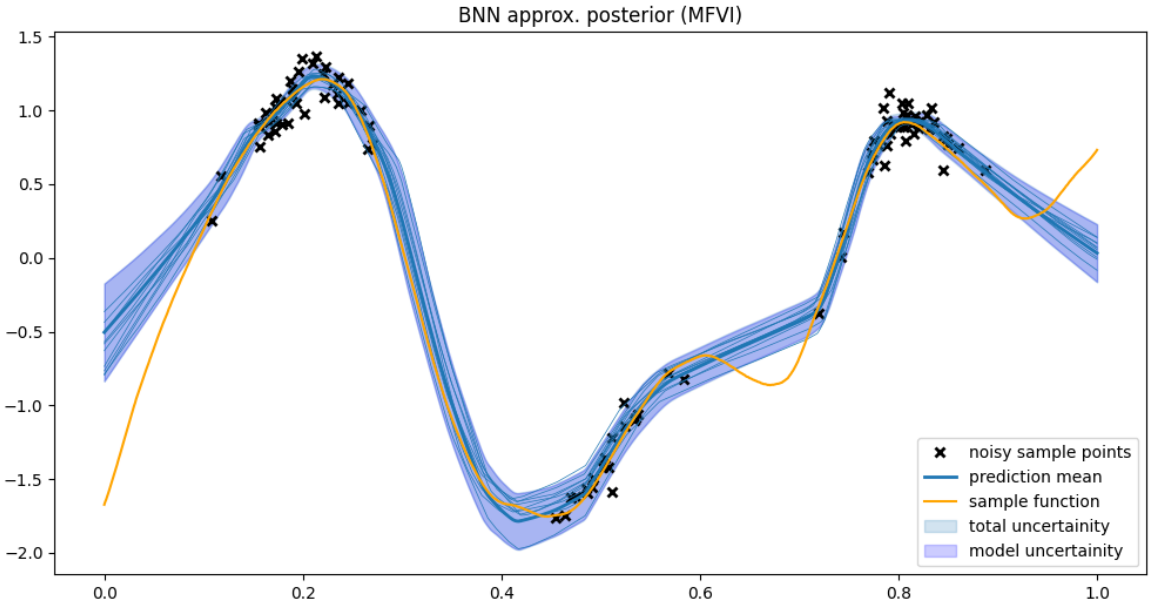


Figure 5.19: Best BNN found from cross-validation. The fit is not bad, but the uncertainty estimates are over confident, as seen with our NNGPs.

5.5 Empirical Bayesian Neural Network

Next we train the hyperparameters of our BNN with the same loss objective as before (ELBO), as described in section 4.1. We fix the prior weight and bias means to zero, and create a single learnable parameter to use as the prior std for all the prior weights and biases.

We first observe that initialising the approximate posterior std at the initial prior std of 1.0, and leaving the likelihood std learnable, causes the model to collapse, whereby the likelihood std increases rapidly to explain the data, rather than trying to fit the model (Figure 5.25). Although this does allow for low ELBO terms as the complexity term equals zero, it does not result in a good fit to the test function.

Initialising the model at the initial likelihood std of 0.1, prevents collapse in simpler models (less layers), but the likelihood std still takes over in larger models, with the nELBO collapsing to the prior (Table 5.8). We should really expect the BNN with 2 layers to also collapse, as it could achieve a lower ELBO, but it does not, perhaps settling at a local minima.

EmpBNN Results on Toy Regression with learnable prior and likelihood							
Num. Layers	nLikelihood	Complexity	nELBO	prior std	Train RMSE	Test RMSE	Test GnLL
1	76.4	67.0	143.4	0.91	0.578	1.054	1.618
2	50.2	127.5	177.7	1.00	0.369	0.747	1.200
3	144.0	1.3	145.2	0.50	1.413	1.164	1.596
4	143.0	1.0	144.0	0.39	1.416	1.131	1.578
5	142.0	0.0	142.0	0.06	1.415	1.138	1.575

Table 5.8: Loss Metrics for best BNN found via cross-validation.

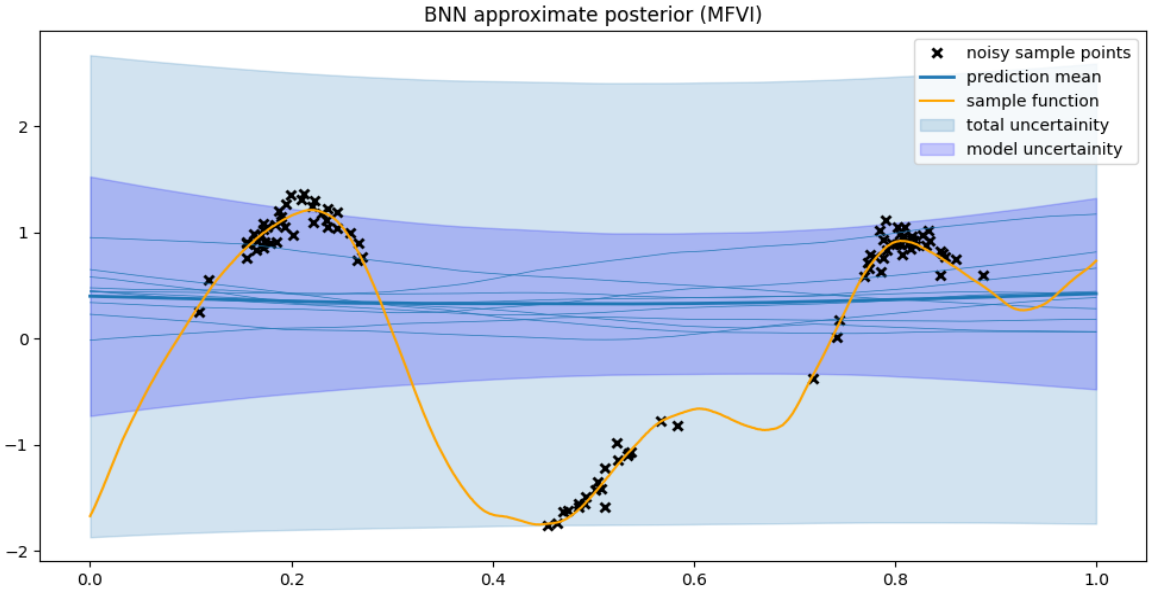


Figure 5.20: Collapsed empBNN when learning all its hyperparameters from prior initialisation.

Fixing the likelihood std partially prevents this issue (Table 5.9), however the model still collapses to the prior at 5 hidden layers (this isn't always the case, depending on the initiation of randomness in the learning, but it is clear the learning surface isn't consistent). The best Test MSE here is 0.197 with only 1 hidden layer, and for 2-4 hidden layers it is around 0.5. While this is a better performance than fixing a standard normal prior, it is worse than fixing a vaguer prior (higher std), found through cross-validation, which the ELBO fails to learn. This reinforces the statement in section 3.3 of [2], that the model learns to fit poor prior parameters quickly, as it is easier to adjust these parameters. Moreover, the ELBOs are all much lower for all these empirical experiments, than when we didn't learn priors, so this correlates with our hypothesis, that the ELBO is not a tight bound to the true marginal likelihood. In particular, the main problem here is that the complexity term of the ELBO increases disproportionately to the regularisation needed, as the number of parameters increases. In practice a β term is often introduced, which controls the strength of the complexity term. We do not experiment with a β term, as the ELBO would no longer be a lower bound to the marginal likelihood.

EmpBNN Results on Toy Regression with learnable prior initialised at 0.1							
Num. Layers	nLikelihood	Complexity	nELBO	prior std	Train RMSE	Test RMSE	Test GnLL
1	-2.8	308.0	305.2	1.61	0.200	0.444	0.644
2	61.5	279.3	340.8	0.99	0.255	0.669	1.325
3	100.5	347.3	447.8	0.91	0.276	0.700	1.532
4	135.7	415.1	550.8	0.87	0.303	0.727	1.709
5	4866.0	3.8	4869.8	3.83	1.415	1.144	6.263

Table 5.9: Loss metrics for empBNN with learnable prior, and fixed likelihood of 0.1.

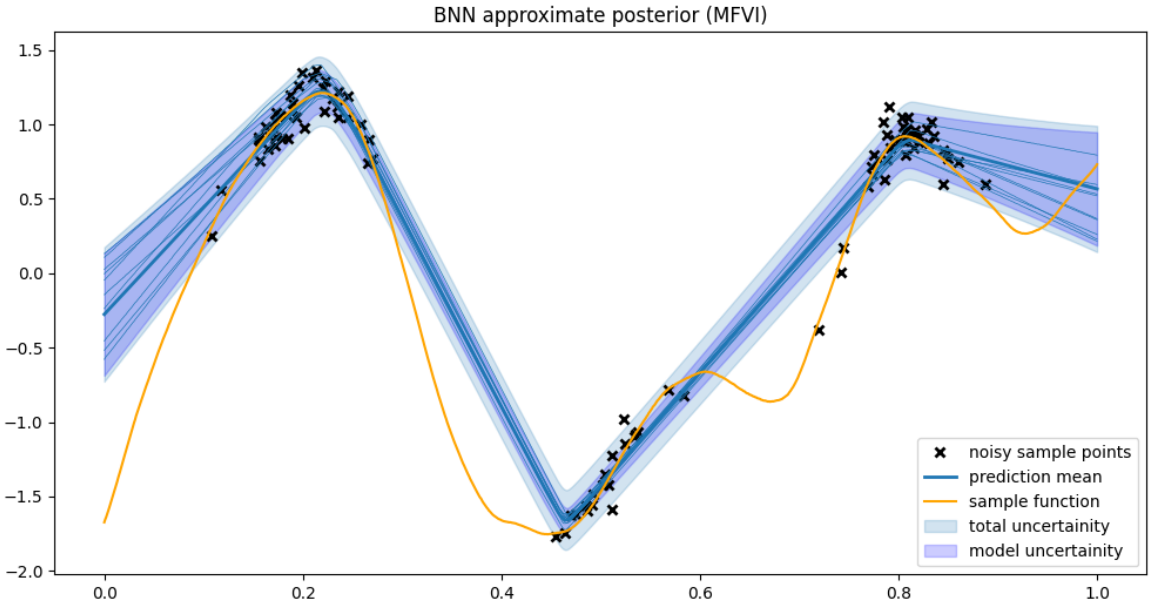


Figure 5.21: Best empBNN predicted by nELBO with learnable prior, but fixed likelihood std.

5.5.1 Collapsed Variational Bounds

We now move on from investigating the original ELBO and onto alternative bounds and methods for finding hyperparameters from marginal likelihood approximations.

We first test the Collapsed Mean Bound from section 4.2, where the only hyperparameter is the alpha hyperprior. As suggested in the paper, we set this hyperprior to 0.05 for our first experiment. Immediately I found that leaving the likelihood std to be learnable, creates the same issues as earlier with empirical BNNs, with the likelihood std quickly increasing to encompass the data, and the resulting model collapsing to the prior.

Collapsed Mean Bound Results on Toy Regression						
Num. Layers	nLikeli-hood	Compl-exity	CM-ELBO	Train RMSE	Test RMSE	Test GnLL
1	94.1	476.1	570.2	0.161	0.349	0.399
2	37.7	4537.2	4574.9	0.134	0.226	-0.056
3	58.2	8691.3	8749.5	0.138	0.192	-0.071
4	79.7	12848.4	12928.1	0.138	0.378	0.347
5	95.5	17015.2	17110.7	0.141	0.329	0.221

Table 5.10: Loss Metrics for CM-ELBO BNN with fixed likelihood.

With the likelihood std fixed at 0.1, varying over number of layers (Table 5.10), we do receive the best Test RMSE we have seen so far on BNN models, of 0.192 with 3 layers, however this is not predicted by the CM-ELBO, which dramatically prefers simpler models. I believe that the addition of hyper-priors helps to regularise the model, so it converges to better Test MSEs, but dramatically increases the complexity term, which incorrectly rewards simpler models. The model predicted to be best by the CM-ELBO thus only has 1 layer, and

does give a better Test MSE than we have seen with the original ELBO on the empirical BNN, but it is not as low as that predicted by the best hyperparameters for a BNN found via cross-validation.

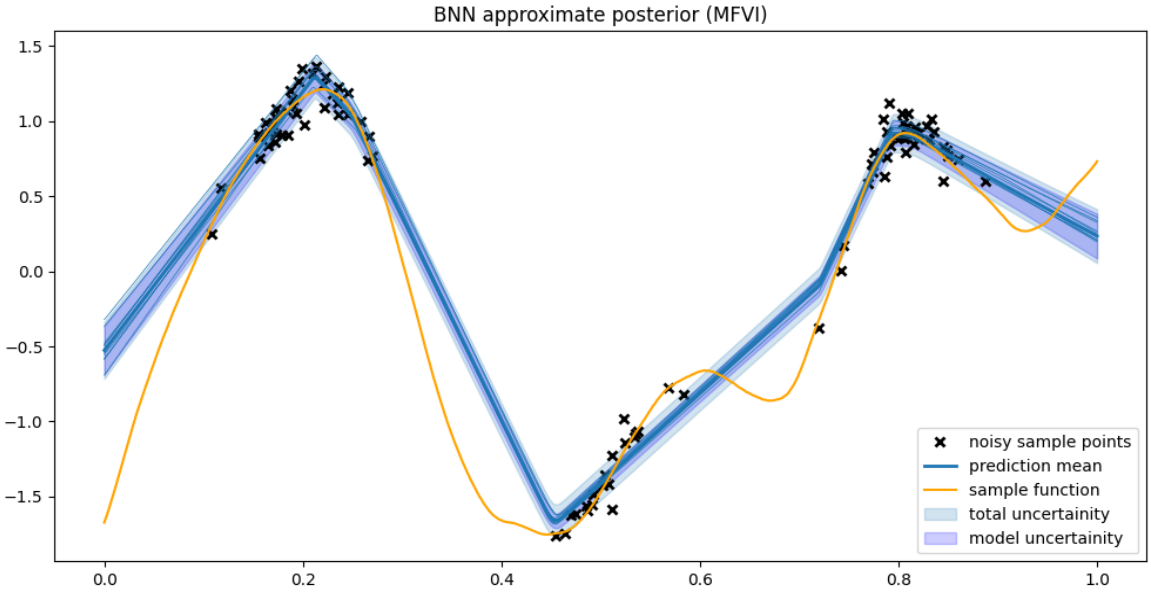


Figure 5.22: Best BNN predicted by CM-ELBO with fixed likelihood std.

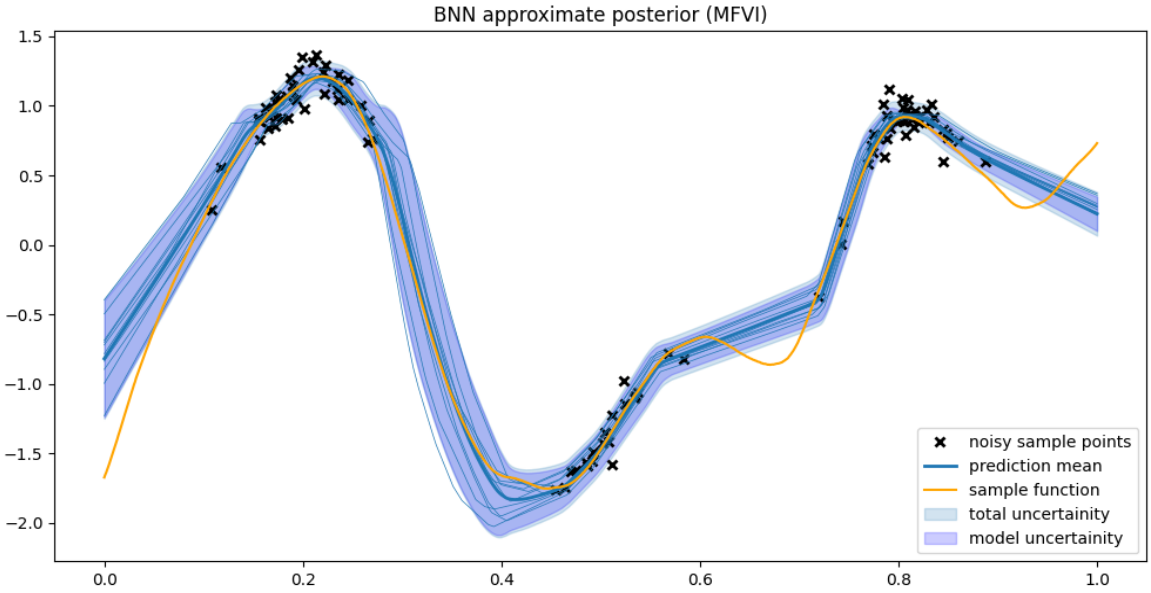


Figure 5.23: Best CM-ELBO BNN on testing.

Leaving the hyperprior to be learnable, results in the hyperprior quickly reducing to zero, which is equivalent to α_{reg} approaching it's limit of 1. The authors point out that this is an advantage of the method, as the regularisation can't increase further, enabling the model parameters to learn rather than collapse to the prior, however the regularisation is not learning anything meaningful, and the model preforms worse than fixing the hyperprior for any number of layers. The best model learning its hyperprior gives an CM-ELBO of 972 and a

RMSE of 0.349 with 4 hidden layers, and again this isn't predicted by the CM-ELBO, which prefer the simplest model of 1 hidden layer, with a CM-ELBO of 585, and a RMSE of 0.377.

The CMV-ELBO gives worse Test MSEs than the CM-ELBO as predicted by the original paper, but otherwise shows similar results. The CMV-ELBO rapidly grows as the number of layers increases, and the best models on testing do not correspond to the best CMV-ELBO.

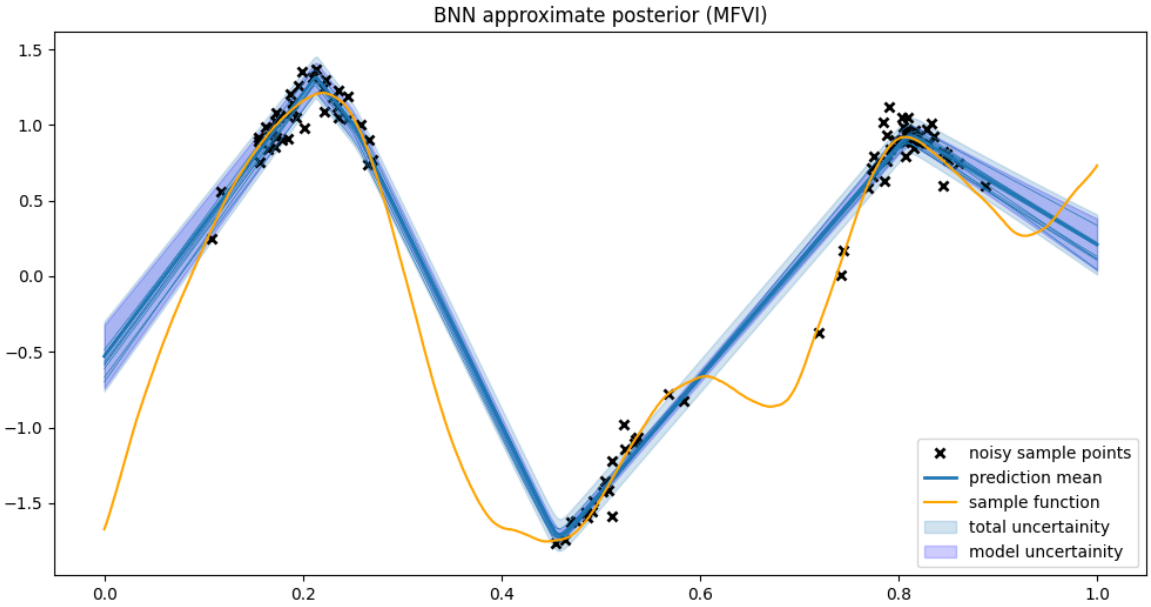


Figure 5.24: Best BNN predicted by CMV-ELBO with fixed likelihood std.

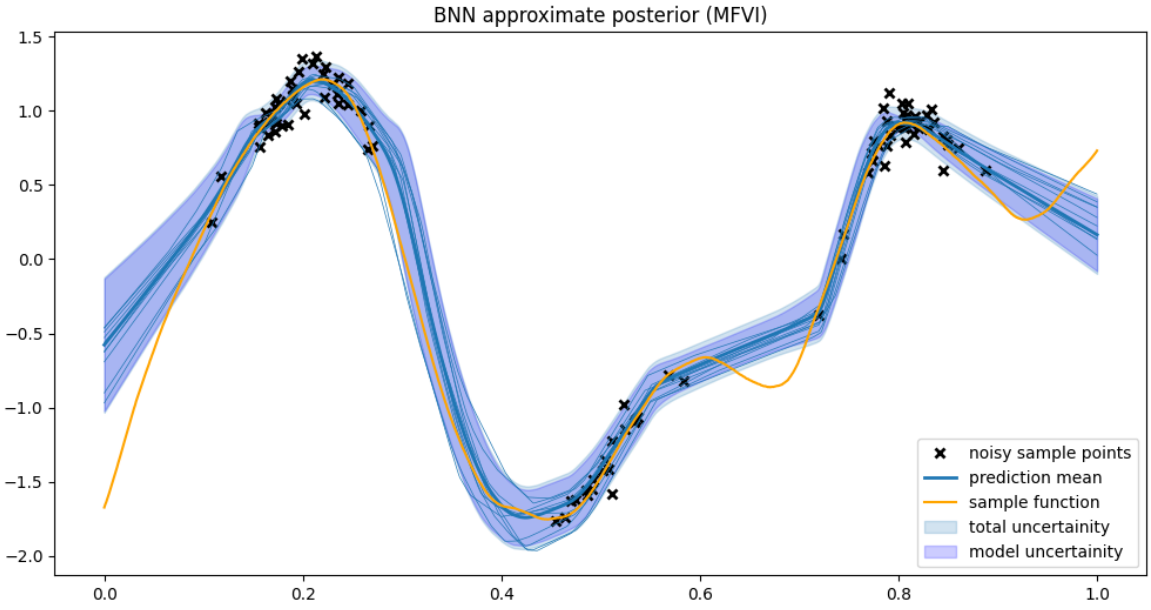


Figure 5.25: Best BNN using CMV-ELBO seen at testing.

Collapsed Mean Variance Bound Results on Toy Regression						
Num. Layers	nLikeli-hood	Compl-exity	CMV-ELBO	Train RMSE	Test RMSE	Test GnLL
1	203.5	729.1	932.6	0.190	0.392	0.728
2	57.1	8956.0	9013.0	0.138	0.365	0.219
3	82.5	17221.1	17303.6	0.138	0.239	-0.018
4	115.2	25471.9	25587.1	0.145	0.374	0.303
5	163.7	33718.3	33882.1	0.155	0.374	0.285

Table 5.11: Loss Metrics for CMV-ELBO BNN with fixed likelihood.

5.6 Laplace Approximations

We now move away from MFVI-BNNs to BNNs using Laplace Approximations.

We first test the Full GNN method varying the number of layers (Table 5.12). The method learns the likelihood std from its marginal likelihood estimates.

Laplace Approximation Results on Toy Regression						
Num. Layers	lik std	nLML	Train RMSE	Train GnLL	Test RMSE	Test GnLL
1	0.35	43.1	0.358	0.660	0.854	1.272
2	0.16	3.8	0.184	0.462	0.404	0.681
3	0.18	4.4	0.141	0.400	0.333	0.681
4	0.27	14.4	0.167	0.455	0.363	0.830

Table 5.12: Loss Metrics for Full GGN Laplace Approximation BNN

The nLML estimates given are small, in the same region as the Gaussian processes, which is partially expected as this is a simpler model then our MFVI-BNNs. The nLML estimates also generally align with the best number of layers according to the Test RMSE and GnLL, with the nLML being very close for 2 and 3 layers, the Test MSE being best with 3 layers, and the Test GnLL being equal for 2 and 3 layers.

However the Test RMSEs and GnLLs themselves are high, and when we observe the approximate predictive posterior, it's clearly unsatisfactory (Figure 5.26). Firstly the predicted likelihood stds are under-confident, all being larger than the noise std. Secondly, the uncertainty estimates are not smooth, with large uncertainties in odd places.

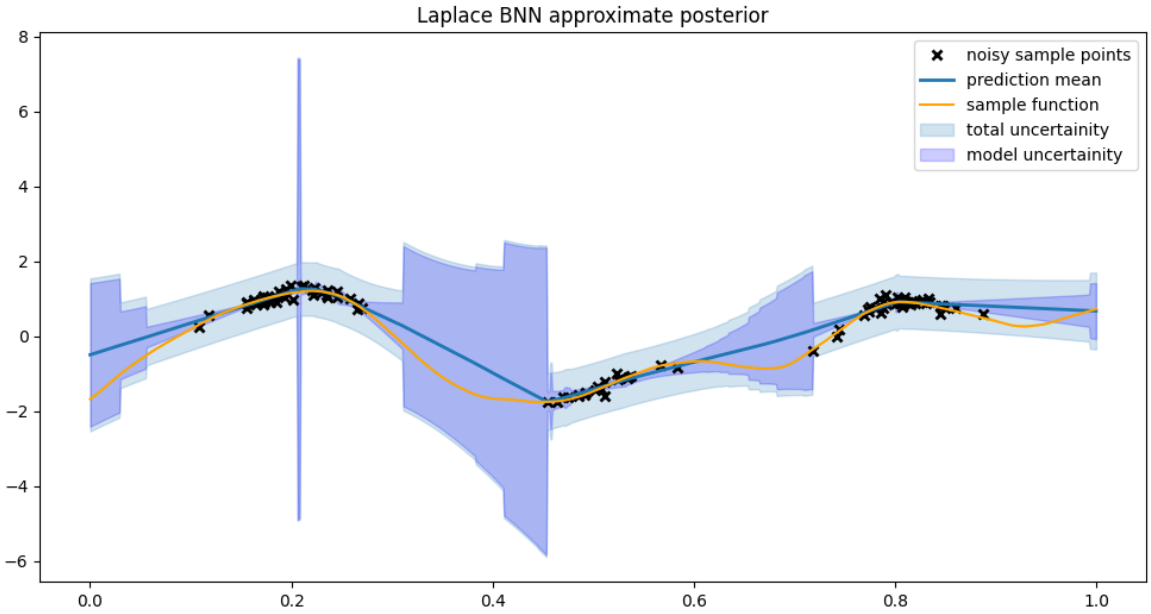


Figure 5.26: Full Laplace.

We repeat the same experiment with the Kron GGN Laplace, again varying the number of layers (Table 5.13). Since the Kron Laplace is an approximation to the Full Laplace, it is unlikely this method will preform better. It is however significantly faster.

Laplace Approximation Results on Toy Regression						
Num. Layers	lik std	nLML	Train RMSE	Train GnLL	Test RMSE	Test GnLL
1	0.27	65.4	0.382	0.670	0.914	1.338
2	0.15	87.9	0.207	0.483	0.479	0.742
3	0.22	151.5	0.310	0.616	0.789	1.120
4	0.89	175.1	1.358	1.385	1.151	1.582
5	0.91	162.0	1.395	1.407	1.3	1.579

Table 5.13: Loss Metrics for Kron GGN Laplace Approximation BNN

The Kron Laplace method does not achieve as low nLML estimates as the Full Laplace for any number of layers. The nLML estimates still seem to be good predictors of the models success, although arguably not predicting as well as the Full Laplace, especially in relation to uncertainty estimates, as the model selected by the nLML has a Test GnLL quite a lot larger than the best model, which has a higher nLML.

The Test RMSEs and GnLLs themselves are very high, significantly worse than the Full Laplace. The 1 layer model chosen by the nLML (Figure 5.27), does not fit the data well at all, especially around the areas of extrapolation. The uncertainty estimates are again not smooth, and have sharp peaks inside the training data, where we should be confident in our epistemic uncertainty.

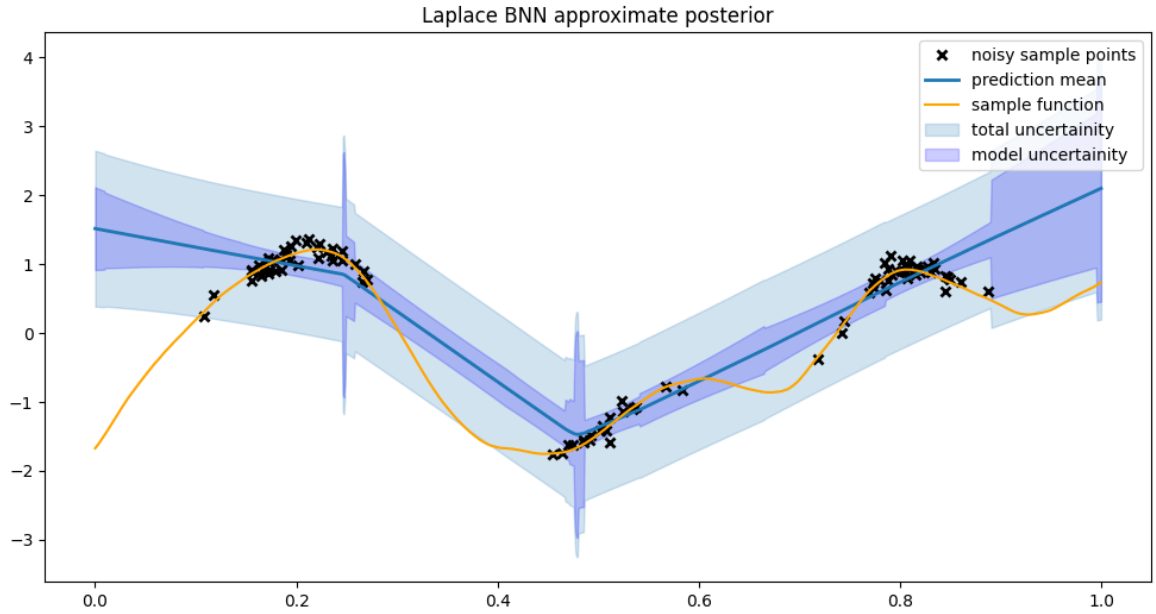


Figure 5.27: Kron Laplace.

5.7 Comparisons Summary

We have shown the relationships (or lack there of) between the marginal likelihood approximations of many models/methods, their hyperparameters, and their resulting test metrics. In Table 5.14, we now list only the models which were chosen by their respective marginal likelihood approximations, i.e. we exclude any model which we found using cross-validation RMSE or GnLL, or using intuition gained from said experiments. The only caveat is fixing the likelihood stds for some models, as this was essential for their learning.

From Figure 5.28 we see that no model achieves the negative nLMLs possible with Gaussian Processes, although the Laplace approximations come close. The Collapsed Mean Variational bound achieves the lowest nLML of any parametric Bayesian Deep Learning method, but as discussed its nLML estimate is dissatisfying.

Overall, we conclude that learning hyperparameters from the ELBO, even after collapsing the prior parameters, is not effective in training, and from our comparisons we show they do not produce good marginal likelihood estimations, with out lowest ELBO being obtained for a collapsed posterior with no regularisation, and the best preforming variational method, CM-BNN, producing an nLML estimate 4 times high than that of the collapsed posterior, and several magnitudes higher than the equivalent NNGP.

Model Results on Toy Regression					
Model/Method	Layers	lik std	nLML	Test RMSE	Test GnLL
SqExp GP	n/a	0.102	-57.2	0.310	-0.563
ArcCos (ReLU) GP	n/a	0.102	-47.8	0.211	-0.624
NNGP	5	0.094	-92.7	0.284	0.602
BNN	2	0.1 (fix)	916.8	0.806	2.427
empBNN	5	0.81	142.0	1.294	1.575
empBNN	1	0.1 (fix)	305.2	0.444	0.644
CM-BNN	1	0.1 (fix)	570.2	0.349	0.399
CMV-BNN	1	0.1 (fix)	932.6	0.392	0.728
Full GGN Lap	2	0.16	3.8	0.404	0.681
Kron GGN Lap	1	0.27	65.4	0.479	0.742

Table 5.14: Metrics for the models chosen using their respective nLML approximations.

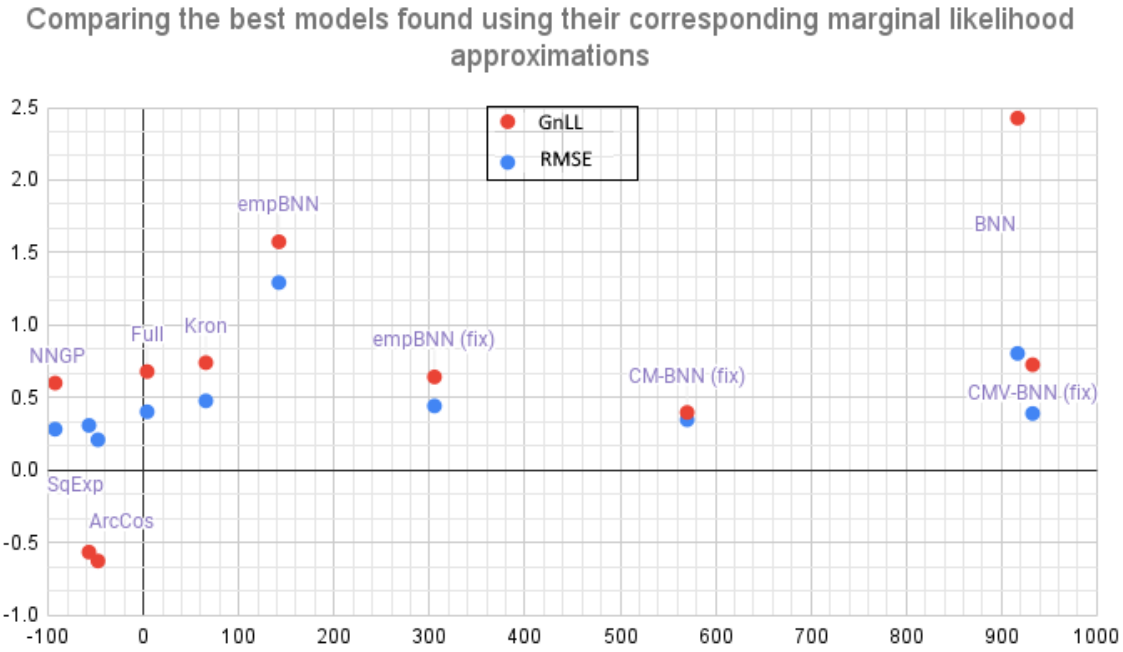


Figure 5.28: Plotting metrics of models chosen using their respective nLML approximations.

Chapter 6

Implementation and Evaluation

6.1 Implementation

Although the focus of this project was not implementation, most of the time spent on this project was understanding and implementing methods and learning procedures, and as such we include a brief summary of this implementation, including citations.

6.1.1 MFVI - Bayesian Neural Networks

The implementation of MFVI-BNNs began with Yingzhen Li's implementation of a BNN in PyTorch, and application of that BNN to a regression problem [13]. We refactored this code into a package, and then extended it by adding new loss function and layers for our new experimental marginal likelihood estimations, and extending the helper functions for regression to compute metrics on the problem and create useful plots.

The learning procedure includes mini-batching, whereby the ELBO is approximated stochastically, although as the training data only contained 100 points, we only utilised 2 batches.

6.1.2 Gaussian Processes

Simple GP kernels were implemented using GPFlow [14], which also handles regression, and computation of exact marginal likelihoods.

To implement NNGPs, we use the Neural Tangents package [15]. The package allows one to easily create NNGPs with specified hyperparameters, and runs a MSE ensemble training procedure to compute the approximate posterior for a NNGP. To learn hyperparameters using the marginal likelihood (or any other metric) we cannot compute gradients, so we need to use a discrete method. A grid search would have sufficed as we knew reasonable hyperparameter ranges to search over, but we chose to use a blackbox optimiser, namely an implementation of the Tree-structured Parzen Estimator Approach from [16], using the package 'benderopt' [17].

One observation I made here was that NNGPs are equally sensitive to hyperparameters as BNNs, so even though we can compute good marginal likelihoods, trained NNGPs often have infinite negative log marginal likelihoods (or rather marginal likelihoods we cannot compute due to the values of the covariance matrix being equivalent to an infinite negative marginal likelihood). When encountered I set these to be large but not infinite value so as not to upset

our black-box optimiser, and with enough evaluations the black-box optimiser still finds good hyperparameters.

6.2 Evaluation

Throughout these experiments, I attempted to be extremely diligent in ensuring the quality of the results. There are several factors, which often made this difficult:

- It was necessary to create and run a large number of different models, especially when preforming cross-validations or searches without gradient-descent.
- The dataset I created while small in size, was not a simple task by design. However this did occasionally lead to strange results, and added another element of uncertainty to the experiments.
- There were often many different avenues to take experiments, and this was naturally mostly guided by the results on my dataset.
- Bayesian models are uncertain by nature. This introduced more uncertainty into the learning process, and required more careful thought about which metrics to value.
- While I did make significant use of packages to preform this project, I also implemented several of the methods, and most of the learning procedures myself, which again introduced another element of uncertainty, and with the other elements, sometimes made bug-fixing difficult. Moreover sometimes packages do not work in the way you expect them too.

This often meant rerunning, and making small alterations, to experiments, while carefully recording many results from the multiple metrics used in training and testing different methods. It would still be untrue to say I have exhaustively explored these methods, and again many questions I attempted to answer were we a result of my dataset.

All that being said, these results would greatly be strengthened by applying my implementations to different regression datasets and comparing the results. This would also help to put this dataset into perspective. I would have especially liked to explore some of the datasets used in the papers I studied, in particular some of the UCI regression datasets for [4] and [3] (and also for [2], although they use a regression dataset as a bandit problem).
It would also be useful to include more uncertainty estimates in the results themselves, by rerunning experiments multiple times and recording the means and stds in the results, especially in cross-validation experiments, or experiments where the estimations are all very close, to see if the difference between them is statistically significant.

Chapter 7

Conclusions and future work

7.1 Summary

In our experiments, we have successfully shown the degree to which several Bayesian Deep Learning Methods can approximate the marginal likelihood for model selection. Specifically, we showed the following relationships:

- The ELBO of the original MVFI-BNN [2] is a loose upper bound to the true marginal likelihood, which does not accurately represent its true surface, and is not effective for model selection.
- The Collapsed Variational Bounds from [3] help to regularise the original ELBO, and ensure better prior parameters, but the values of these bounds are no more useful for model selection than the original ELBO, and is not effective for neural network architecture selection.
- The Scalable Laplace Approximations from [4] show tighter marginal likelihood approximations which are more effective for model selection than the explored variational bounds, however the models themselves make worse uncertainty estimates than MVFI-BNNs.

To make strong arguments for the efficacy of these marginal likelihood approximations, we implemented Gaussian Process models which produce exact marginal likelihoods, and compared the performances of the different models. In particular, we used the Neural Network Gaussian Process to make well-founded comparisons to our Bayesian Neural Networks, for which the exact equivalence has been shown.

To aid in these experiments, we implemented a package for manipulating Bayesian Neural Networks based on [13], which extends its capabilities to empirical BNNs and Collapsed Variational Bounds, as well as integrating Scalable Marginal Likelihood Estimation from [4], and Gaussian Process Regression from [14]. Included are helper classes for the regression experiment we preformed, which would be helpful for any other regression (or to a less extent, classification) experiments in the same domain.

Overall, our results show a necessity for continued research into marginal likelihood approximations in Bayesian Deep Learning, if we are to use these approximations for model selection. We hope this report will motivate more research into this topic, and aid in the construction of better marginal likelihood approximations.

7.2 Future Work

As mentioned in section 6.2, the results of the report would be assisted by preforming the same experiments on comparable datasets. They would also be furthered by preforming experiments on different applicable problems. We did preform some experiments on vectorised MNIST, including a SVGP (Sparse Variational Gaussian process) which we will include in the code repository, but we did not have time to preform a full empirical study which could be included in this report. MNIST was tested on in [2], [4] and [3], so it would be useful for affirming the results of these papers. This and other classification problems would also offer a different view point on uncertainty estimates. Although classification problems do not use Gaussian likelihoods, so one could not make use of conjugate distributions for exact Gaussian processes, the marginal likelihood estimates have been proven to be tight to the true marginal likelihood for discrete likelihood functions. The estimates have also been proven to be tight for large datasets, where exact Gaussian processes are no longer computationally feasible, and for convolutional networks, which opens up more possible problems to explore and compare marginal likelihood estimates on.

We had also hoped to explore Marcin B. Tomczak's earlier paper in detail, Marginal Likelihood Gradient for Bayesian Neural Networks [12], which combines local reparametrisation of BNNs w.r.t the prior distribution, with the self-normalized importance sampling estimator, for a claimed superior marginal likelihood gradient to the original MVFI-BNN ELBO. We attempted to implement his approximate scheme using one objective (Eq 33 in his appendix), but we could not get the model to learn on the target problem. An deeper exploration of this paper, especially into the advantages and disadvantages of the method with respect to the other methods we have explored would be of value to this field.

7.3 Ethical Discussion

The focus of this report is purely theoretical, so there are very few ethical considerations to make. The dataset used was from our creation, and involves no personal information or moreover any real observed data, so there is no risk of data protection issues. As with any extension of research, there is a possibility for conclusions made to be later used for military applications or malevolent potential, however considering Bayesian Learning is about quantifying risk, which is often used to reduce risk, by ensuring Machine Learning results are well-justified by the data, it is unlikely any results here could be abused.

Bibliography

[1] David J. C. MacKay. Bayesian methods for adaptive models. 1992. URL https://www.inference.org.uk/mackay/thesis.pdf. pages 2, 8

[2] Koray Kavukcuoglu Daan Wierstra Charles Blundell, Julien Cornebise. Weight uncertainty in neural networks. 2015. URL http://proceedings.mlr.press/v37/blundell15.pdf. pages 2, 3, 10, 11, 14, 33, 42, 43, 44

[3] Andrew Y. K. Foong Richard E. Turner Marcin B. Tomczak, Siddharth Swaroop. Collapsed variational bounds for bayesian neural networks. 2021. URL https://papers.nips.cc/paper/2021/file/d5b03d3acb580879f82271ab4885ee5e-Paper.pdf. pages 2, 3, 4, 42, 43, 44

[4] Vincent Fortuin Gunnar Rätsch Mohammad Emtiyaz Khan Alexander Immer, Matthias Bauer. Scalable marginal likelihood estimation for model selection in deep learning. 2021. URL https://arxiv.org/pdf/2104.04975.pdf. pages 2, 3, 4, 42, 43, 44

[5] Jose Miguel Hernandez-Lobato Richard E. Turner Andrew Y. K. Foong, Yingzhen Li. 'in-between' uncertainty in bayesian neural networks. 2019. URL https://arxiv.org/pdf/1906.11537.pdf. pages 3

[6] Sebastian W. Ober Florian Wenzel Gunnar Rätsch Richard E. Turner Mark van der Wilk Laurence Aitchison Vincent Fortuin, Adrià Garriga-Alonso. Bayesian neural network priors revisited. 2022. URL https://arxiv.org/pdf/2102.06571.pdf. pages 3

[7] Ian Hacking. Slightly more realistic personal probability. 1967. URL http://fitelson.org/probability/hacking_smrpp.pdf. pages 5

[8] Bradley P. Carlin and Thomas A. Louis. Empirical bayes: Past, present and future. 2000. URL https://doi.org/10.2307/2669771. pages 9

[9] David J. C. MacKayA. A practical bayesian framework for backpropagation networks. page 448–472, 1992. pages 12

[10] Lawrence K. Saul Youngmin Cho. Kernel methods for deep learning. pages 2–3, 2009. URL https://papers.nips.cc/paper/2009/file/5751ec3e9a4feab575962e78e006250d-Paper.pdf. pages 12

[11] Roman Novak Samuel S. Schoenholz Jeffrey Pennington Jascha Sohl-Dickstein Jaehoon Lee, Yasaman Bahri. Deep neural networks as gaussian processes. 2018. URL https://arxiv.org/pdf/1711.00165.pdf. pages 13

[12] Richard E. Turner Marcin B. Tomczak. Marginal likelihood gradient for bayesian neural networks. 2020. pages 15, 44

[13] Yingzhen Li. A hands-on tutorial for bayesian neural networks. 2022. URL `https://colab.research.google.com/drive/1yNnS-iWlWXRQibKHGKLf--g-U-F2P56_?usp=sharing`. pages 41, 43

[14] Alexander G. de G. Matthews, Mark van der Wilk, Tom Nickson, Keisuke. Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18 (40):1–6, apr 2017. URL `http://jmlr.org/papers/v18/16-537.html`. pages 41, 43

[15] Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A. Alemi, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. Neural tangents: Fast and easy infinite neural networks in python. 2020. URL `https://github.com/google/neural-tangents`. pages 41

[16] Yoshua Bengio Balazs Kegl James Bergstra, Remi Bardenet. Algorithms for hyper-parameter optimization. 2011. URL `https://papers.nips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf`. pages 41

[17] Valentin Thorey. URL `https://github.com/vthorey/benderopt`. pages 41