Bayesian Neural Networks

Purna Vindhya, Kota

1 Introduction

Given some input–output (x $\rightarrow$ y) data, we wish wishes to learn a regressor $f(\cdot)$ for prediction $y^\star = f(x^\star)$ given a new input $x^\star$. Neural networks are a type of regressor that consists of multiple layers of simple nonlinear functions, parameterized by a set of weights (and biases), ω. Training the neural networks refers to calibrating the value of these weights using the available training data $\mathscr{D} = \{x, y\}$.
To represent the aleatoric uncertainties, adopt a probability model over the data, say normal distribution:

$$p(y|x,\omega) = \mathcal{N}(y; NN_\omega(x), \sigma_n^2) \tag{1}$$

In this study, the aim is to quantify the epistemic uncertainties, thus the variation of measurement noise σ_n^2 can be assumed apriori. Assuming training data points are i.i.d., the likelihood over the whole dataset $\mathscr{D}$ is computed as:

$$p(\mathscr{D}|\omega) = \Pi_{i=1}^{n} \mathcal{N}(y_i; NN_\omega(\omega), \sigma_n^2) \tag{2}$$

Epistemic uncertainties are quantified by learning a probability model over the parameters of the model ω. In a Bayesian setting, the prior $p_0(\omega)$ can be assumed apriori. With the data, the posterior on the weights is updated as:

$$p(\omega|\mathscr{D}) = \frac{p(\mathscr{D}|\omega)p_0(\omega)}{p(\mathscr{D})} \tag{3}$$

Therefore, during prediction, given a new input $x^\star$,

$$p(y^\star|x^\star,\mathscr{D}) = \int p(y^\star|x^\star,\omega)p(\omega|\mathscr{D})d\omega \tag{4}$$

The epistemic uncertainty can be quantified as the variance of $y^\star|x^\star,\mathscr{D}$. In a deterministic setting, the optimum weights ω can be found by:

$$\omega^{opt} = argmx_\omega \log(p(\mathscr{D}|\omega)p_0(\omega)) \tag{5}$$

and is equivalent to the standard training procedures of a neural network. To accurately quantify the epistemic uncertainties within the NN, the full posterior pdf must be inferred. A Bayesian Neural Network (BNN) is a probabilistic model that allows us to estimate uncertainty in predictions by representing the weights and biases of the network as probability distributions rather than fixed values. This allows us to incorporate prior knowledge about the weights and biases into the model, and update our beliefs about them as we observe data. BNNs are a handy tool to quantify the uncertainty over the posterior. Moreover, they provide estimates around predictions without the need for separate training, validation, and test data sets. To estimate the (intractable) posterior distribution $p(\omega|\mathscr{D})$, we can use

- Variational methods
- Markov Chain Monte Carlo sampling

2 Models

2.1 Stochastic Variational Inference

The normalized posterior probability density $p(\omega|\mathscr{D})$ is intractable, so we approximate it with a tractable parametrized density $q_\phi(\omega)$ from a family of probability densities $\mathscr{Q}$. The variational parameters are denoted by ϕ. The goal is to find the variational probability density that best approximates the posterior by minimizing the KL divergence
$KL(q_\phi(\omega)||p(\omega|\mathscr{D}))$
with respect to the variational parameters ϕ. However, directly minimizing the KL divergence is not tractable because we assume that the posterior density is intractable. To solve this, we use Bayes theorem to obtain

$$\log p(\mathscr{D}|\omega) = KL\big(q_\phi(\omega)||p(\omega|\mathscr{D})\big) + ELBO(q_\phi(\omega)) \tag{6}$$

$$\mathrm{ELBO}(q_\phi(\omega)) = E_{\omega\sim q_\phi(\omega)}[p(y|x,\omega)] - \mathrm{KL}(q_\phi(\omega)||p(\omega)) \tag{7}$$

By maximizing the ELBO, we indirectly minimize the KL divergence between the variational probability density and the posterior density. Therefore, the ELBO loss function:

$$\mathscr{L} = \mathrm{KL}(q_\phi(\omega)||p(\omega)) - E_{\omega\sim q_\phi(\omega)}[\log p(\mathscr{D},\omega)] \tag{8}$$

For the full-covariance Gaussian approximate posterior, the model weights for each layer ω_i are distributed according to the multivariate Gaussian distribution $\mathscr{N}(\mu_i,\Sigma_i)$. The mean-field approximation restricts Σ_i to be a diagonal covariance matrix, or equivalently assumes that the probability distribution can be expressed as a product of individual weight distributions:

$$\mathscr{N}(\mu_i,\Sigma_i) = \Pi_j \mathscr{N}(\mu_{ij},\sigma_{ij}) \tag{9}$$

The mean-field approximation greatly reduces the computational cost of both forward and backwards propagation, and reduces the number of parameters required to store the model from order n^2 in the number of weights to order n.

The *Stochastic Variational Inference (SVI)* is a stochastic optimization algorithm for mean-field variational inference. The algorithm approximates the posterior distribution of a probabilistic model with hidden variables. SVI by using a normal probability density with a diagonal covariance matrix. The algorithm uses stochastic gradient ascent to optimize the ELBO with respect to the global variational parameters. The SVI algorithm is explained in detail in [1]

2.1.1 Problem Description

The data that we are trying to fit to the BNN is:

$$y = x + 0.3\sin(2\pi x) + 0.3\sin(4\pi x) \tag{10}$$

where, $x = [-0.5, 1.5] + \varepsilon$ and, $\varepsilon \sim \mathcal{N}(0, 0.02)$

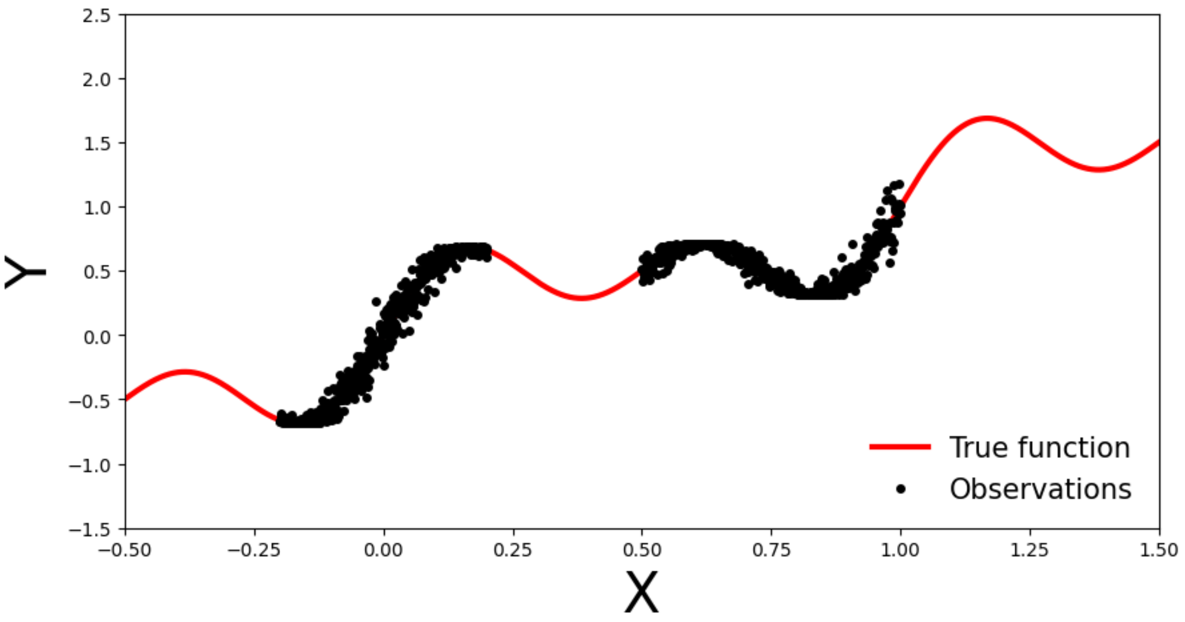


Figure 1: True function and data points

The histogram of the training data is given in Fig. 2. The plot of the data is given in Fig. 1

2.1.2 BNN Architecture

The BNN is designed to have 2 hidden layers, each with 20 neurons. The prior on the weights and biases of each neuron at all layer is $\mathcal{N}(0,1)$. The σ_n^2 (standard deviation on the posterior of y, $p(y|x,\omega)$) from Eq. 1 is set to random, i.e., $\sigma_n^2 \sim \mathcal{U}(0,1)$. Therefore,

$y|x,\mathcal{D} \sim \mathcal{N}(NN_\omega(x), \sigma_n^2)$

$\sigma_n^2 \sim \mathcal{U}(0,1)$

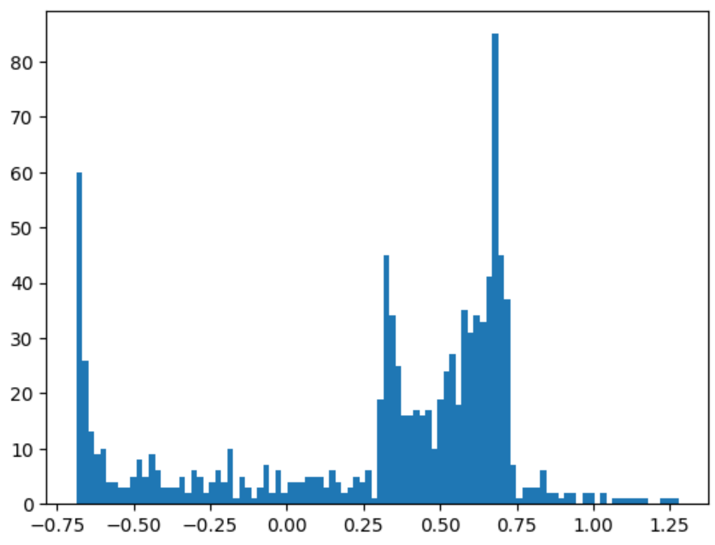


Figure 2: Histogram of training data

$\omega_j \sim \mathcal{N}(0,1)$
ADAM optimizer is used and the model is run for 25000 epochs while minimizing the loss function (a trace implementation of the minimization of ELBO). The parameters of the ADAM optimizer are set to be:

- learning rate: 0.001
- β_1: 0.999
- β_2: 0.999
- clipping normal: 2.0

To check the predictions (both interpolation and extrapolation are performed simultaneously), 500 samples are taken from the VI posterior approximate. 3000 points are in the range $x^\star = [-0.5, 1.5]$ are predicted on the trained BNN, and the predictions are given in Fig. 3. The loss function of the SVI training epochs is given in Fig. 4.

It can observed that the predictions in the interpolated regions perform fairly well. But in the extrapolated regions, $\{[-0.25, -0.5] \cup [0.5, 1.0]\}$, the prediction performance is poor.

2.2 HMC (No U-Turn Sampler)

As mentioned before, MCMC can be used to sample from the posterior $p(y,x|\mathscr{D})$. We can use MCMC sampler to get an unbiased estimate of $p(y,x|\mathscr{D})$.

$$p(y,x|\mathscr{D}) = E_{\omega \sim p(\omega|\mathscr{D})}[p(y|x,\omega)] \tag{11}$$

We can approximate $E_{\omega \sim p(\omega|\mathscr{D})}[p(y|x,\omega)]$ through Monte Carlo sampling as

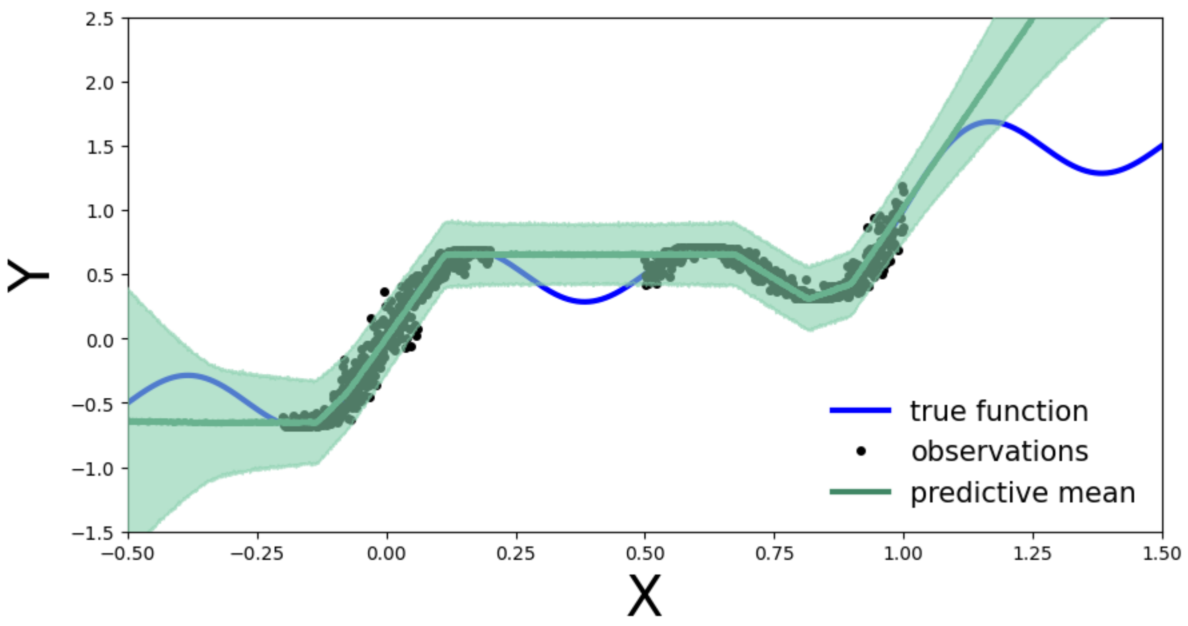


Figure 3: SVI Predictions

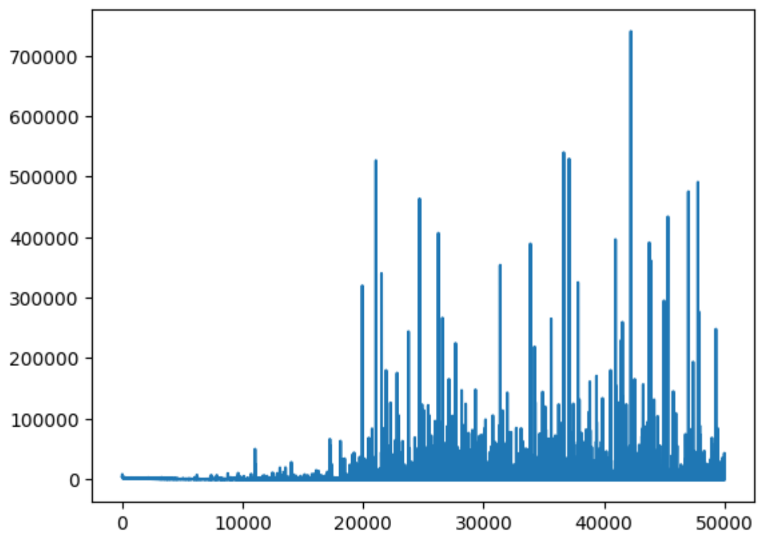


Figure 4: SVI loss function

$$E_{\omega \sim p(\omega|\mathscr{D})}[p(y|x,\omega)] \approx \frac{1}{N}\sum_{i=1}^{N} p(y|x,\omega_i) \tag{12}$$

where, $\omega_i \sim p(\omega_i|\mathscr{D})$. From Bayes theorem,

$$p(\omega|\mathscr{D}) = \frac{p(\mathscr{D}|\omega)p(\omega)}{p(\mathscr{D})} \tag{13}$$

So, $\omega_i \sim p(\omega_i|\mathscr{D}) \propto p(\mathscr{D}|\omega)p(\omega)$ are drawn from the posterior distribution. Because the normalizing constant is intractable, we require MCMC methods like Hamiltonian Monte Carlo to draw samples from the non-normalized posterior. Specifically, the No U-Turn Sampler (NUTS), a variant of the HMC sampler is used to sample from the posterior. The number of steps taken by the integrator is dynamically adjusted on each sample to ensure an optimal length for the Hamiltonian trajectory. As such, the samples generated will typically have lower autocorrelation than those generated by the HMC.
HMC's performance depends strongly on the tuning parameters M (momentum covariance/ mass matrix), ε (step size in chain), and L (number of steps per iteration/ length of chain). NUTS is an extension of HMC that adaptively tunes M and ε during burn-in, and adapts L throughout the MCMC run. NUTS eliminates the need to select the tuning parameters. The basic idea of NUTS is that the MC propagates the Hamiltonian dynamics until the trajectory starts going back towards where it started. The intuition is that we want to make as big a move as possible, so going back toward where we started is undesirable. NUTS is a way of constructing trajectories to avoid going back (momentum vector pointing in the direction of initial position), while still satisfying detailed balance. This is implemented via a procedure for choosing L adaptively. NUTS also employs a method of adapting ε and M during the burn-in period.

In addition, the vanilla HMC algorithm is used to sample from the posterior. The parameters of the HMC sampler are:

- $\varepsilon = 0.0001$
- $L = 5$

The HMC is run to sample from the posterior, with the BNN trained on $\{x_{train}, y_{train}\}$ as the potential function, and the results are plotted in Fig. $\boxed{5}$

The potential function for the NUTS sampler is the neural network and 550 samples from the posterior are sampled, with a burn in of 50 samples. The parameters of the HMC chains are adaptive and don't need to be specified explicitly. The predictions and the uncertainty is plotted in Fig. $\boxed{6}$.

2.2.1 Comparing posteriors

Inspecting the posteriors of the weights of all neurons in the first hidden layer (20 distributions), a plot of the posteriors from SVI and NUTS are compared in Fig. $\boxed{7}$.

2.3 Deterministic NN

To compare the BNN with deterministic NN, a point estimate of the NN (MAP estimate) is created using a fully connected MLP. The prediction of the single NN is plotted in Fig. $\boxed{8}$.

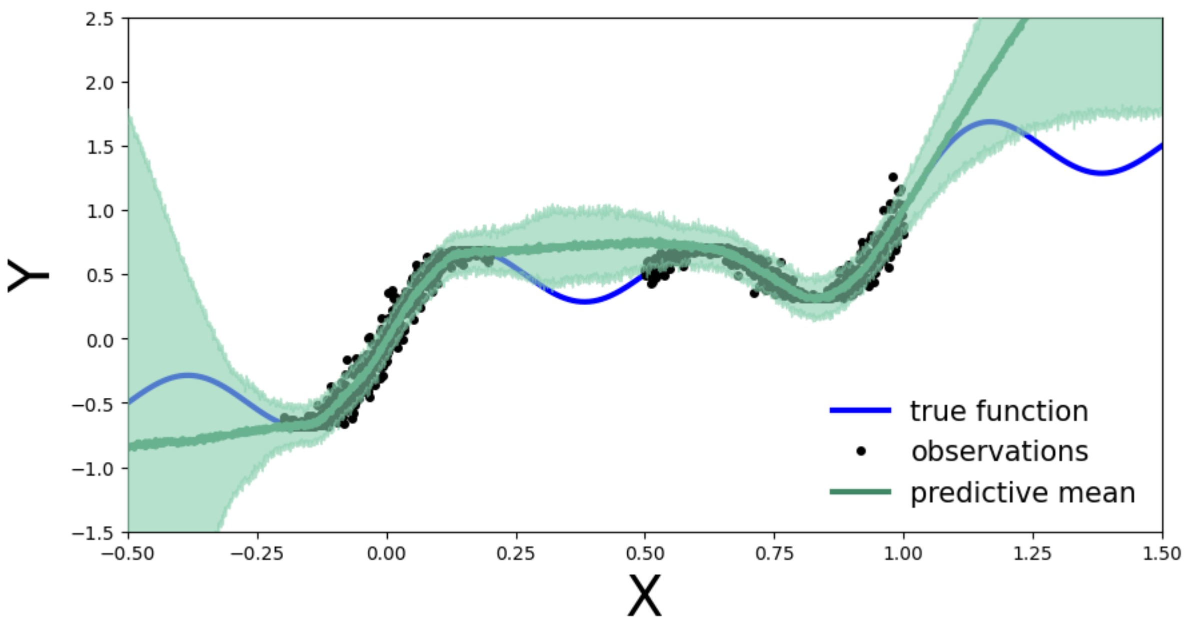


Figure 5: Vanilla HMC sampler for BNN

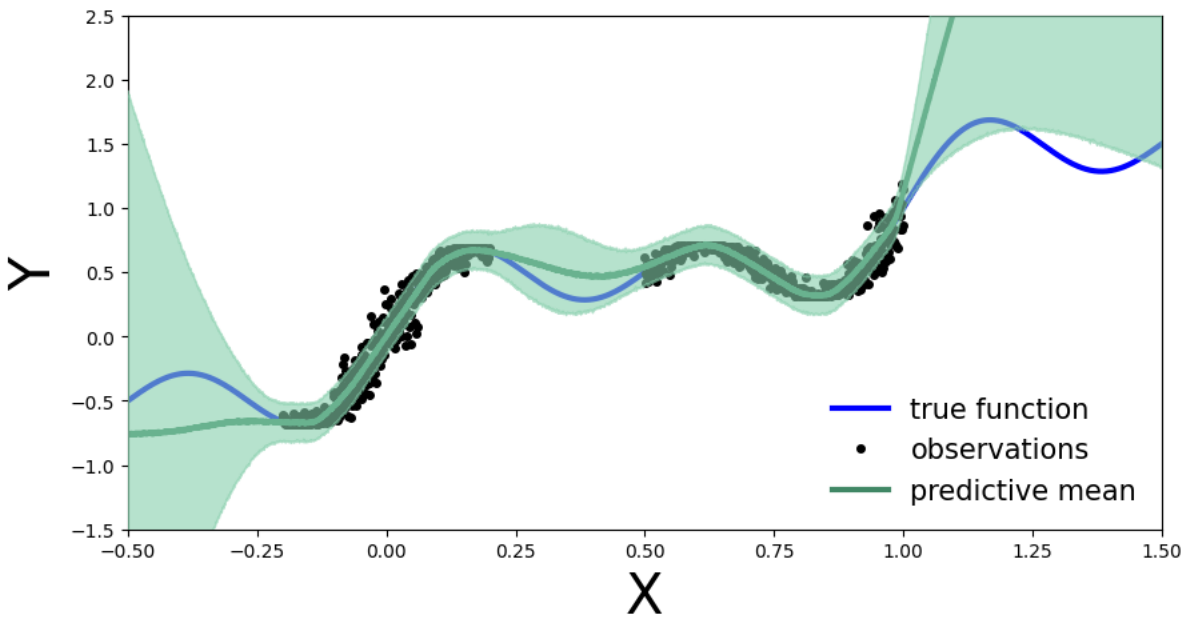


Figure 6: NUTS Sampler for BNN

2.4 Deep Ensemble

In deep ensemble, multiple point estimates of the NN are trained and the final prediction is computed as an average across the models. From a Bayesian perspective, the different point estimates correspond to samples of a Bayesian posterior. This can be interpreted as approximating the posterior with a distribution parametrized as multiple Dirac deltas.

$$q_\phi(\omega|\mathscr{D}) = \sum_{\omega_i} \alpha_{\omega_i} \delta_{\omega_i}(\omega) \tag{14}$$

where α_{ω_i} are positive constants such that their sum is equal to one. It is assumed that $\alpha_{\omega_i} = 1/N_{ensemble}$ for all the networks in the ensemble. 10 networks are considered to be an ensemble. The ensemble's predictive functions are plotted in Fig. 9.

The uncertainty bands of the ensemble NN is plotted in 10

2.5 Gaussian Process Regression

Gaussian Process Regression is trained to quantify the epistemic uncertainty. The mathematics of the GPR with noisy data is adapted from the class notes. The GP is assumed to have an RBF kernel with $length - scale = 0.05$ and $variance = 0.02$ (initial parameters). The optimum hyperparameters for the GP are obtained by optimizing the maximum likelihood function using L-BFGS optimizer. The GP before maximum likelihood estimation on the training data is plotted in Fig. 11. The performance of the GPR on the training data is plotted in Fig. 12. The performance of the GPR on the test data, $x_{\text{test}} = [-0.5, 1.5]$ is shown in Fig. 13

3 Conclusions

- **SVI:** The SVI does not capture the interpolation and extrapolation predictions very well. The training data is chosen such that the gradients at the beginning and end of the data are close to 0. Hence, the gradient update at the ends of the dataset remains in the same direction as that of the last points. Hence, the interpolation and extrapolation standard deviations are high. This behavior does not seem to improve with an increase in training epochs. As SVI is an approximation of the posterior, we used a normal distribution with a learnable mean and a diagonal covariance matrix. It is clear that for this specific example, this method leads to a very poor approximation of the actual posterior since it can only fit an unimodal distribution. This highlights the major limitations of simple VI methods for BNNs.

- **HMC:** The HMC sampler was used to generate samples from the posterior. It is observed that HMC has a lesser probability of acceptance (70%) compared to that of NUTS (91%). The posterior samples of HMC have high variance and hence the boundaries of the standard deviation and mean are jagged compared to NUTS sampler. Vanilla HMC is very difficult to adapt and takes a very long time to train even for the small dataset.

- **NUTS:** The NUTS sampler was used mainly as a way of generating samples from the true posterior. The MCMC sampler provides a very good approximation of samples from the exact posterior. The NUTS sampler for BNN posterior captures the interpolation better than the SVI model does. The uncertainty, as measured by the standard deviation a posteriori, is low where the data points lie and increases with distance away from the observations.

- **Deterministic NN:** The deterministic neural network gives only prediction and no information about the uncertainty.

- **Deep Ensemble:** Ensembling is the most straightforward way to perform ensemble learning with an NN and is done by just restarting the learning procedure multiple times. Using this strategy to learn the posterior gives quite good results, even better than naive VI as the samples can belong to different modes of the posterior. The main drawback of this approach is that it provides no estimation of the local uncertainty around the different modes of the posterior.

- **Gaussian Process Regression:** GPR has a nice mechanism for incorporating prior beliefs about the underlying function by specifying mean and covariance functions. This trains well compared to BNNs where training depends on a lot of hyperparameters and requires fine tuning. In BNNs, it is difficult to incorporate intuitive prior information about the functions. Rather, in BNNs priors over the network parameters are specified and are interpreted as a bias towards less complex functions (posterior) via smaller weights (generally normal distributions). For this particular small-scale example, GP performs significantly better in predicting the data in both interpolation and extrapolation. The variance at the points where observations are present is less and the mean of the GPR prediction fits the true mixture of sines best compared to the BNNs. However, Gaussian Processes have scalability limitations making BNNs a more practical model in large data settings, although this has not been demonstrated in this work.

References

[1] https://jmlr.org/papers/volume14/hoffman13a/hoffman13a.pdf

[2] https://andrewcharlesjones.github.io/journal/svi.html

[3] Olivier, A., Shields, M.D. and Graham-Brady, L., 2021. Bayesian neural networks for uncertainty quantification in data-driven materials modeling. Computer methods in applied mechanics and engineering, 386, p.114079.

[4] Jospin, L.V., Laga, H., Boussaid, F., Buntine, W. and Bennamoun, M., 2022. Hands-on Bayesian neural networks—A tutorial for deep learning users. IEEE Computational Intelligence Magazine, 17(2), pp.29-48.

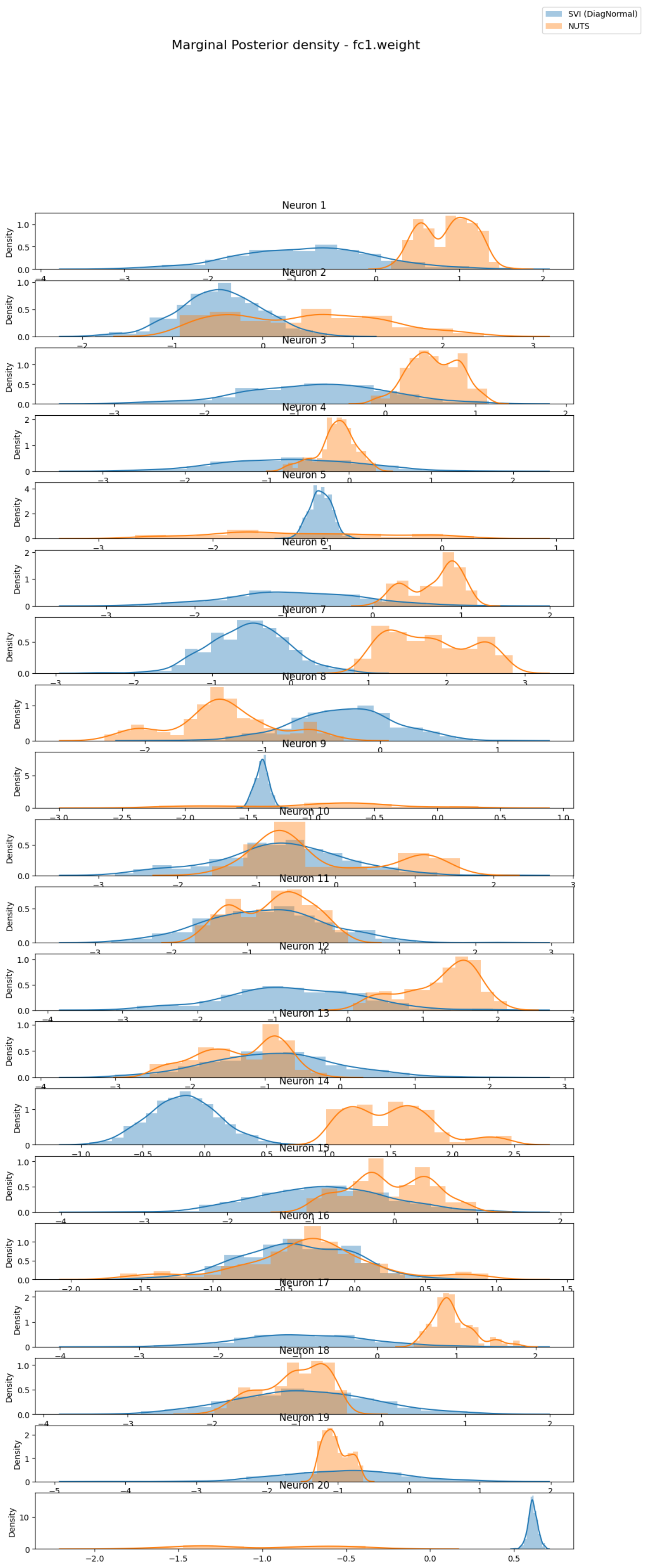


Figure 7: Posteriors of SVI vs NUTS

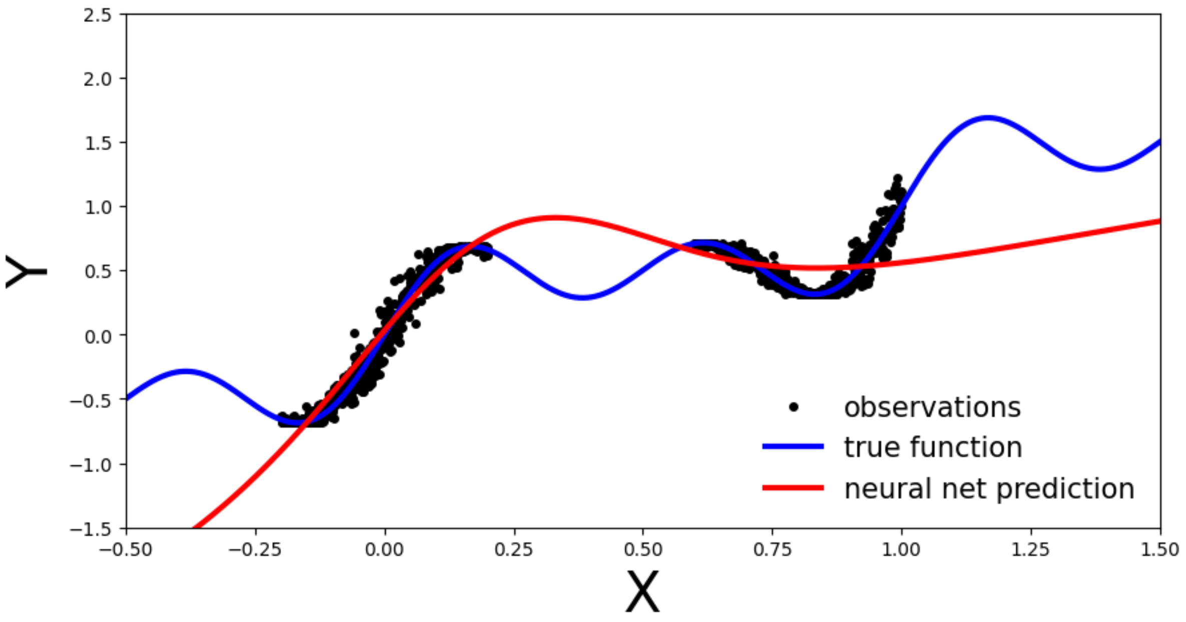


Figure 8: Neural Network with 2 layers, 20 neurons each

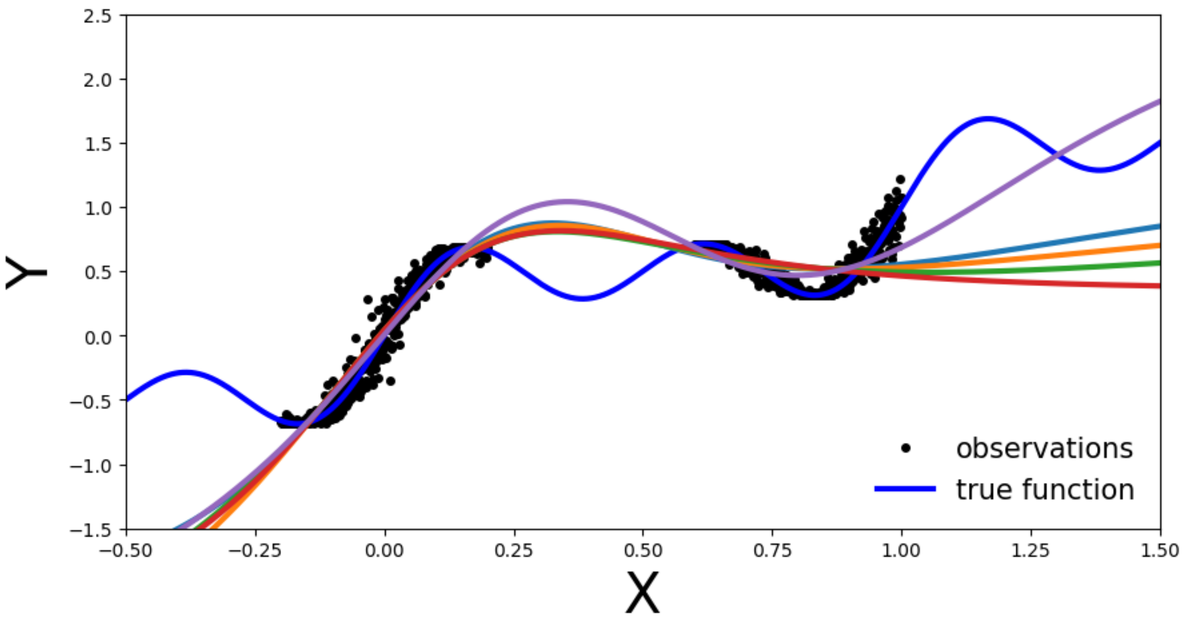


Figure 9: Ensemble

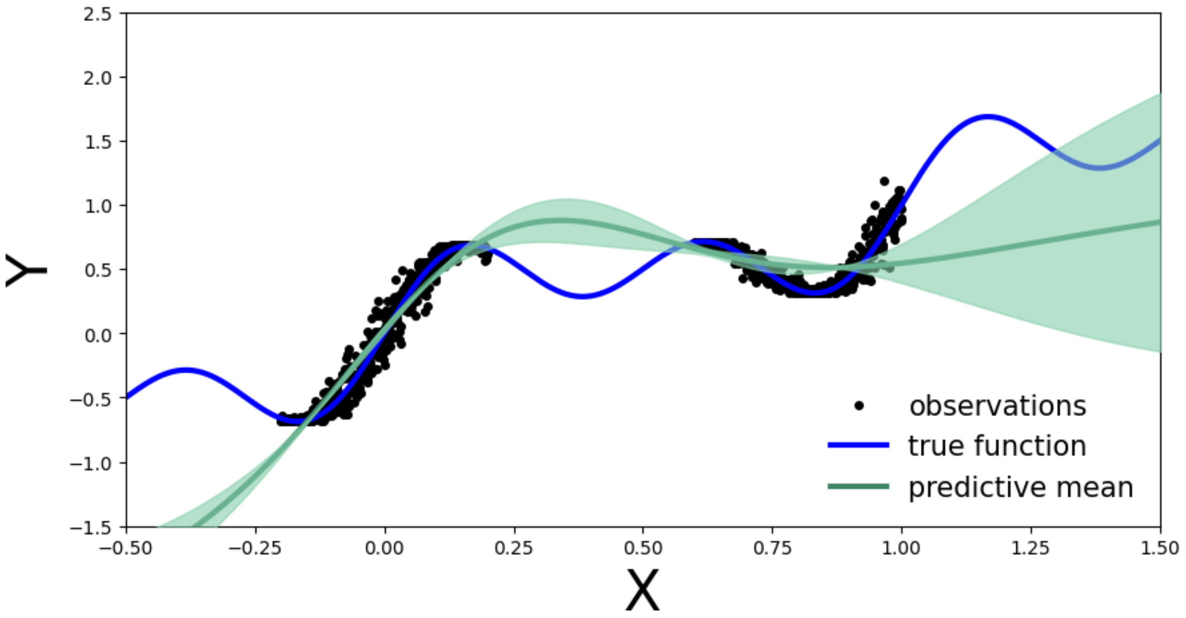


Figure 10: Uncertainty bands of ensemble NN

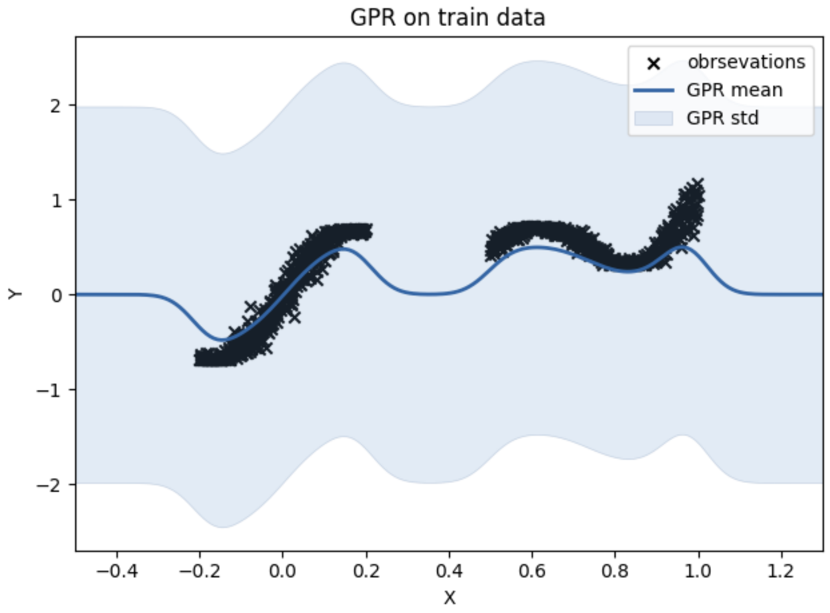
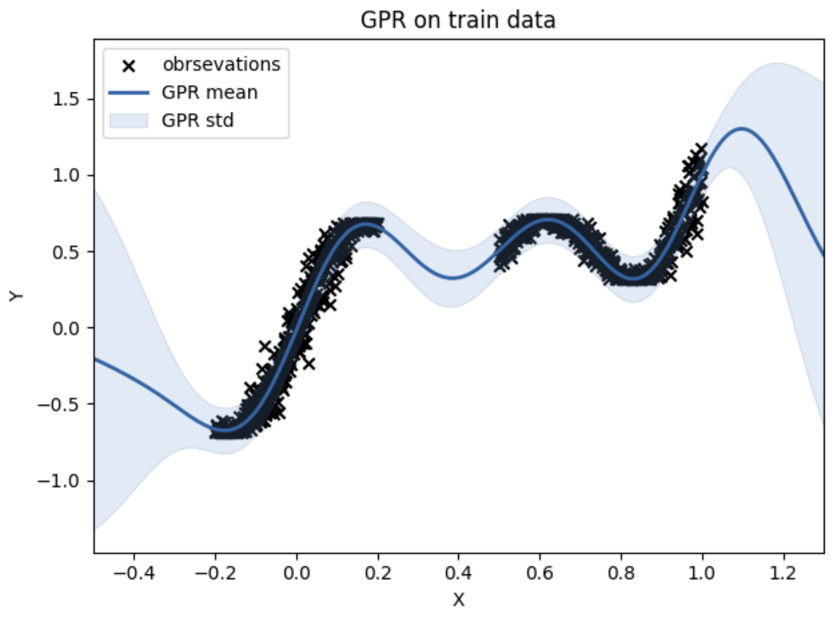


Figure 11: GPR without optimizing hyperparameters



[h!]

Figure 12: GPR with optimized hyperparameters on training data

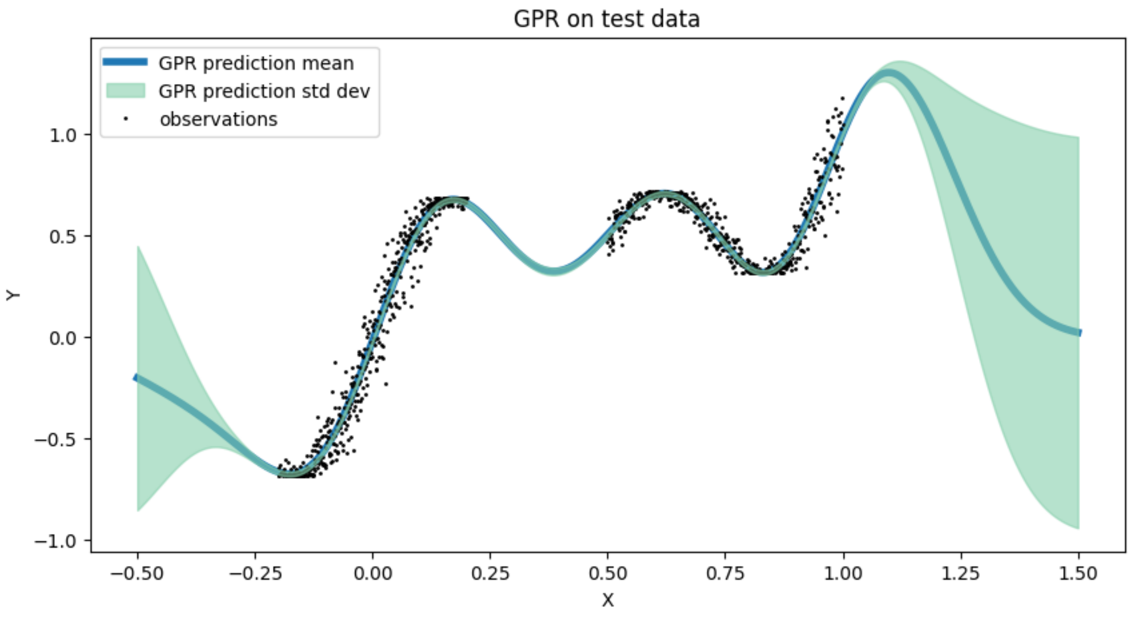


Figure 13: GPR with MLE on test data