Lightning UQ Box: Theory Guide

Contents

Preprint. Under review.

1 Description of UQ Methods

`Lightning UQ Box` provides the most comprehensive collection of the extensive and versatile landscape of UQ methods for DL. The following section gives an overview of these different UQ methods, which are listed in Table 1. For comprehensive explanations, we refer to existing reviews [1, 15].

For **regression tasks** DNNs $f_\theta : X \to Y$ are trained to predict a continuous 1-dimensional target $y^\star \in Y$. Currently, the toolbox supports six classes of UQ methods for 1d regression: deterministic, quantile, ensemble, Bayesian, Gaussian Process, and diffusion-based methods.

1. Deterministic methods: use a DNN, $f_\theta : X \to \mathcal{P}(Y)$, that map inputs x to the parameters of a probability distribution $f_\theta(x^\star) = p_\theta(x^\star) \in \mathcal{P}(Y)$, and include methods like Deep Evidential Regression (**DER**) [2] and Mean Variance Estimation (**MVE**) [44]. The latter, for example, outputs the mean and standard deviation of a Gaussian distribution $f_\theta^{\text{MVE}}(x^\star) = (\mu_\theta(x^\star), \sigma_\theta(x^\star))$.
2. Quantile based models: use a DNN, $f_\theta : X \to Y^n$, that map to n quantiles, $f_\theta(x^\star) = (q_1(x^\star), ..., q_n(x^\star)) \in Y^n$, and include Quantile Regression [30] (**Quantile Regression**) and the conformalized version thereof (**ConformalQR**) [50].
3. Ensembles: Deep Ensembles [33], which utilize an ensemble over MVE networks.
4. Bayesian methods: aim to model a distribution over the network parameters and are commonly used to approximate the first and second moment of a marginalized distribution. Here the network parameters are modelled as random variables. Multiple principles and techniques to approximate BNNs have been introduced. We include BNNs with Variational Inference (VI) (**BNN VI ELBO**) [6], BNNs with VI and alpha divergence (**BNN VI**) [11], Variational Bayesian Last Layers (**VBLL**) [21], MC-Dropout (**MCDropout**) [13], the Laplace Approximation (**Laplace**) [49][8] and **SWAG** [37] with partially stochastic variants [54].
5. Gaussian Process-based methods: these model a joint distribution over a set of functions in a data-driven manner that approximates the first and second moment of the marginalized distribution. These include Deep Kernel Learning (**DKL**) [61], an extension thereof Deterministic Uncertainty Estimation (**DUE**) [57, 58], and Spectral Normalized Gaussian Process (**SNGP**) [34].
6. Conditional Generative model: Classification and Regression Diffusion (**CARD**) [20].

For **classification**, the toolbox currently supports six classes of UQ methods. The DNN $f_\theta^c : X \to \mathcal{CAT}(c)$ is trained to predict a c class categorical distribution. Vanilla softmax probabilities can be directly used to obtain predictive uncertainties. However, they are often miscalibrated and have lead to post-hoc recalibration methods being proposed [18]. For most of the classification UQ methods the entropy of the softmax values or predicted class wise probabilities is commonly used as a measure of predictive uncertainty. Note that the entropy is maximal for a uniform distribution, which means that each class is equally likely to be predicted.

1. Deep Ensembles (**DeepEnsembles**) [33]: utilize an ensemble over independent standard classification networks.
2. Bayesian methods: **BNN VI ELBO** [6], **VBLL** [21], **MCDropout** [13], **Laplace** [49][8], **SWAG** [37].
3. Gaussian Process based methods: model a distribution over functions that also approximate the fist and second moment of the marginalized predictive distribution. **DKL** [61], **DUE** [57] and Spectral-normalized Neural Gaussian Processes (**SNGP**) [34].
4. Conformal Prediction: [50], Regularized Adaptive Prediction Sets (**RAPS**) [3] based on conformal prediction.
5. Other: Test-time Augmentation (**TTA**) [35], Temperature Scaling [18] which is based on a post-hoc calibration of classifiers.

Additionally to the general purpose tasks of regression and classification, `Lightning UQ Box` supports UQ methods for vision-specific tasks. These include segmentation and pixel-wise regression, where an extensive overview of supported UQ methods can be found on our documentation page.

Table 1: The methods provided with `Lightning UQ Box` and other available frameworks and reviews. The table represents the status at the time of publication and will be extended in the future. All currently available methods can be found in the provided repository.

		Publication								
Category	Method	[19]	[51]	[12]	[26]	[53]	[46]	[43]	[32]	`Lightning UQ Box`
Deterministic Methods	Gaussian (MVE)	✓							✓	✓
	Deep Evidential Networks (DER)								✓	✓
Neural Network Ensembles	Neural Network Ensembles	✓	✓	✓	✓	✓	✓	✓	✓	✓
Bayesian Neural Networks	MC Dropout (GMM)		✓	✓	✓	✓	✓	✓	✓	✓
	BNN with VI ELBO				✓	✓	✓	✓		✓
	BNN with VI (alpha divergence)									✓
	VBLL									✓
	Laplace Approximation					✓				✓
	SWAG				✓	✓				✓
	DVI, SI				✓					
	HMC				✓					
	Radial BNN							✓		
	Rank-1 BNN							✓		
Gaussian Process based	Deep Kernel Learning (DKL)									✓
	Det. Unc. Estimation (DUE)									✓
	Spectral Normalized GPs (SNGP)					✓		✓		✓
Quantile based	Quantile Regression (QR)	✓		✓						✓
	Conformal Prediction (CQR)	✓		✓						✓
Diffusion Model	CARD									✓
Post-hoc Calibration	RAPS									✓
	TempScaling						✓		✓	✓

2 UQ Methods Theory Guide

We define neural networks as a function from an input space X to a target space Y,

$$f_\theta : X \to Y \tag{1}$$

where θ represent the network parameters that are optimized during the training procedure. These parameters are optimized on a training dataset, that consists of n input-target pairs,

$$\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^n, \tag{2}$$

and with respect to a given loss function

$$\mathcal{L} : Y \times Y \to \mathbb{R}. \tag{3}$$

The loss function usually is some score, measure or quantity describing how well the network's predictions fit the given targets.

Baselines:

For *regression* tasks simple baseline models give a prediction of the 1-dimensional target value for a given input x^*, evaluated based with the squared error loss,

$$\mathcal{L}(\theta, (x^\star, y^\star)) = (f_\theta(x^\star) - y^\star)^2, \tag{4}$$

which is aggregated by the mean over batches.

For standard multi-class *classification*, baseline models predict vanilla soft-max probabilities $f_\theta^c(x^\star) \in \mathcal{CAT}(c)$, where $\mathcal{CAT}(c)$ describes the space of categorical distributions over c classes. Throughout this theory guide and where needed for clarification, we state networks that output a class probability by f_θ^c. Further, we reference to the network output before the soft-max transformation as the network logits, $l_\theta(x^\star)$, i.e. $f_\theta^c(x^\star) = \text{softmax}\left(l_\theta(x^\star)\right)$.

The standard loss for classification is given by the standard Cross-Entropy,

$$\mathcal{L}(\theta, (x^\star, y^\star)) = -\sum_{q=1}^{c} \mathbb{1}_{y_q^\star} \log\left(f_\theta^c(x^\star)_q\right), \tag{5}$$

where this applies to one-hot labels and the function $1_{y_q^\star}$ is given by

$$\mathbb{1}_{y_q^\star} = \begin{cases} 1 & \text{if} \quad y_q^\star = 1 \\ 0 & \text{else.} \end{cases} \tag{6}$$

Moreover, $f_\theta^c(x^\star)_q$ denotes the q-th component of $f_\theta^c(x^\star) \in \mathcal{CAT}(c)$, i.e., the predicted probability for class q. Similar, $y_q^\star$ denotes the q-th component of $y^\star \in \mathcal{CAT}(c)$.

2.1 Deterministic UQ Methods

In the following we list the deterministic UQ methods considered in this work. These methods provide UQ estimates within a single forward pass by predicting the parameters of a probability distribution.

Gaussian/MVE: The Gaussian model for *regression*, also referred to as Mean Variance Estimation, first studied in [44] and further used in [55], is a deterministic model that predicts the parameters of a Gaussian distribution

$$f_\theta(x^\star) = (\mu_\theta(x^\star), \sigma_\theta(x^\star)) \tag{7}$$

in a single forward pass, where standard deviations $\sigma_\theta(x^\star)$ can be used as a measure of data uncertainty. To this end, the network now outputs two parameters and is trained with the Gaussian negative log-likelihood (NLL) as a loss objective [28], that is given by

$$\mathcal{L}(\theta, (x^\star, y^\star)) = \frac{1}{2}\ln\left(2\pi\sigma_\theta(x^\star)^2\right) + \frac{1}{2\sigma_\theta(x^\star)^2}\left(\mu_\theta(x^\star) - y^\star\right)^2. \tag{8}$$

Correspondingly, the model prediction consists of a predictive mean, $\mu_\theta(x^\star)$, and the predictive uncertainty, in this case the standard deviation $\sigma_\theta(x^\star)$.

Deep Evidential Regression (DER): DER [2] is a single forward pass UQ method that aims to disentangle aleatoric and epistemic uncertainty. DER entails a four headed network output

$$f_\theta(x^\star) = (\gamma_\theta(x^\star), \nu_\theta(x^\star), \alpha_\theta(x^\star), \beta_\theta(x^\star)). \tag{9}$$

These four outputs are used to compute the predictive t-distribution with $2\alpha(x^\star)$ degrees of freedom, [2]:

$$p(y(x^\star)|f_\theta(x^\star)) = \mathrm{St}_{2\alpha_\theta(x^\star)}\left(y^\star \middle| \gamma_\theta(x^\star), \frac{\beta_\theta(x^\star)(1+\nu_\theta(x^\star))}{\nu_\theta(x^\star)\alpha_\theta(x^\star)}\right). \tag{10}$$

In [2] the network weights are obtained by minimizing the loss objective that is the negative log-likelihood of the predictive distribution and a regularization term. However, due to several drawbacks of DER, [40] propose the following adapted loss objective that we also utilise,

$$\mathcal{L}(\theta, (x^\star, y^\star)) = \log\sigma_\theta^2(x^\star) + (1+\lambda\nu_\theta(x^\star))\frac{(y^\star - \gamma_\theta(x^\star))^2}{\sigma_\theta^2(x^\star)} \tag{11}$$

where $\sigma_\theta^2(x^\star) = \beta_\theta(x^\star)/\nu_\theta(x^\star)$. The mean prediction is given as,

$$\mu_\theta(x^\star) = \gamma_\theta(x^\star). \tag{12}$$

Further following [40], we use their reformulation of the uncertainty decomposition. The aleatoric uncertainty is given by

$$u_{\text{aleatoric}}(x^\star) = \sqrt{\frac{\beta(x^\star)}{\alpha(x^\star)-1}}, \tag{13}$$

and the epistemic uncertainy by,

$$u_{\text{epistemic}}(x^\star) = \frac{1}{\sqrt{\nu(x^\star)}}. \tag{14}$$

The predictive uncertainty is then, given by

$$u(x^\star) = \sqrt{u_{\text{epistemic}}(x^\star)^2 + u_{\text{aleatoric}}(x^\star)^2}. \tag{15}$$

2.2 Ensemble Based UQ Methods

Deep Ensembles: As introduced in [33], Deep Ensembles approximate a posterior distribution over the model weights. For Deep Ensembles GMM, this is done with a Gaussian mixture model over the output of separately initialized and trained networks. In [60] the authors showed that Deep Ensembles can be interpreted as a Bayesian method.

Deep Ensembles Regression: For *regression* the Deep Ensembles model predictive mean is given by the mean taken over the outputs $f_{\theta_i}(x^\star)$ of $N \in \mathbb{N}$ baseline models with different weights $\{\theta_i\}_{i=1}^N$. The weights are obtained by individually training N networks with the MSE aggregated over batches. The weights of the ensemble members are in general different, as the loss objective is non-convex with respect to the network parameters and due the stochasticity of gradient descent methods. The ensemble prediction is given by,

$$\mu(x^\star) = \frac{1}{N}\sum_{i=1}^{N} f_{\theta_i}(x^\star). \tag{16}$$

The predictive uncertainty is given by the standard deviation of the predictions of the N different networks, the so called ensemble members,

$$\sigma(x^\star) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(f_{\theta_i}(x^\star) - \mu(x^\star)\right)^2}. \tag{17}$$

Deep Ensembles GMM: For *regression* the Deep Ensembles GMM model predictive mean is given by the mean taken over $N \in \mathbb{N}$ models $f_{\theta_i}(x^\star) = (\mu_{\theta_i}(x^\star), \sigma_{\theta_i}(x^\star))$ with different weights $\{\theta_i\}_{i=1}^N$,

$$\mu_g(x^\star) = \frac{1}{N}\sum_{i=1}^{N} \mu_{\theta_i}(x^\star). \tag{18}$$

The predictive uncertainty is given by the standard deviation of the Gaussian mixture model consisting of the N different networks, Gaussian ensemble members,

$$\sigma_g(x^\star) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\mu_{\theta_i}(x^\star) - \mu_g(x^\star)\right)^2 + \frac{1}{N}\sum_{i=1}^{N}\sigma_{\theta_i}^2(x^\star)}. \tag{19}$$

Note that the difference between "Deep Ensembles" and "Deep Ensembles GMM" is that in the latter we also consider the predictive uncertainty output of each individual ensemble member, whereas in the former we only consider the means and the variance of the mean predictions of the ensemble members.

Because each ensemble member has a probabilistic predictive distribution $f_{\theta_i}(x^\star) = (\mu_{\theta_i}(x^\star), \sigma_{\theta_i}(x^\star))$, we can also perform a decomposition into epistemic and aleatoric components:

$$u_{\text{epistemic}}(x^\star) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\mu_g(x^\star) - \mu_{\theta_i}(x^\star)\right)^2}\,, \tag{20}$$

$$u_{\text{aleatoric}}(x^\star) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\sigma^2_{\theta_i}(x^\star)}\,. \tag{21}$$

Deep Ensembles Classification: As introduced in [33], Deep Ensembles approximate a posterior distribution over the model weights with a mixture model over the output of separately initialized and trained networks. In particular, for *classification* the Deep Ensembles model prediction is chosen to be the softmax of the class-wise mean over the $N \in \mathbb{N}$ models' logit predictions $l_{\theta_i}(x^\star) \in \mathbb{R}^c$ (we refer to logits as the model output before the soft-max activation), with different weights $\{\theta_i\}_{i=1}^N$,

$$\mu(x^\star) = \text{softmax}\left(\frac{1}{N}\sum_{i=1}^{N} l_{\theta_i}(x^\star)\right) \in \mathcal{CAT}(c). \tag{22}$$

The loss is the standard Cross-Entropy as in (5). The predictive uncertainty is given by the standard Cross-Entropy, yet the true labels in (5) are substituted by the network predictions.

Hyperparameter Overview for Deep Ensembles:

Summary of hyperparameters for the Deep Ensembles models

Hyperparameter	value range	hints
Number of ensemble members	$N \approx [5, 20]$	do an ablation study on N.

2.3 Bayesian UQ Methods

The general aim of Bayesian UQ methods is to obtain the predictive distribution by marginalization over the model weights θ,

$$p(y^\star|x^\star, D) = \int p(y^\star|x^\star, \theta)p(\theta|D)d\theta. \tag{23}$$

The posterior distribution over the weights $p(\theta|D)$ can be approximated by utilizing Bayes' theorem for

$$p(\theta|\mathcal{D}) = \frac{p(Y|\theta, X)p(\theta)}{p(Y|X)}, \tag{24}$$

or, for example, by a variational approach. However, the predictive distribution, (23), is usually intractable and, in the following various approaches of approximation are presented (most of which rely on sampling over the posterior).

MC-Dropout: Is an approximate Bayesian method with sampling. A fixed dropout rate $p \in [0, 1)$ is used, meaning that random weights are set to zero during each forward pass with the probability p. This models the network weights and biases as a Bernoulli distribution with dropout probability p. While commonly used as a regularization method, [13] showed that activating dropout during inference over multiple forward passes yields an approximation to the posterior over the network weights. Due to its simplicity it is widely adopted in practical applications, but MC-Dropout and variants thereof have also been criticized for their theoretical shortcomings [24], [45].

MC-Dropout Regression: the MC Dropout model predicts a target value and a predictive uncertainty. The target is predicted by the mean of $m \in \mathbb{N}$ forward passes through the network $f_{p,\theta}$ with a fixed dropout rate p, resulting in masked weights $\{\theta_i\}_{i=1}^m = \{\theta \circ r\}_{i=1}^m$ with Bernoulli distributed mask r with each entry samples form $\text{Ber}(1-p)$. The mean prediction is given by

$$f_p(x^\star) = \frac{1}{m}\sum_{i=1}^{m} f_{p,\theta_i}(x^\star). \tag{25}$$

The predictive uncertainty is given by the standard deviation of the predictions over m forward passes,

$$\sigma_p(x^\star) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(f_{p,\theta_i}(x^\star) - f_p(x^\star)\right)^2}. \tag{26}$$

MC Dropout GMM: We also consider combining this method with the previous model Gaussian network, as in [28], aiming at disentangling the data and model uncertainties, abbreviated as MC Dropout GMM. For the MC Dropout GMM model, the prediction again consists of a predictive mean and a predictive uncertainty $f_{p,\theta}(x^\star) = (\mu_{p,\theta}(x^\star), \sigma_{p,\theta}(x^\star))$. Here the predictive mean is given by the mean taken over m forward passes through the Gaussian network mean predictions $\mu_{p,\theta}$ with a fixed dropout rate p, resulting in different weights $\{\theta_i\}_{i=1}^m$, given by

$$\mu_p(x^\star) = \frac{1}{m}\sum_{i=1}^{m} \mu_{p,\theta_i}(x^\star). \tag{27}$$

The predictive uncertainty is given by the standard deviation of the Gaussian mixture model obtained by the predictions over m forward passes,

$$\sigma_p(x^\star) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(\mu_{p,\theta_i}(x^\star) - \mu_p(x^\star)\right)^2 + \frac{1}{m}\sum_{i=1}^{m}\sigma^2_{p,\theta_i}(x^\star)}. \tag{28}$$

A decomposition of uncertainty then can be performed in a similar way as to with deep ensembles.

MC Dropout Classification: For *classification* the MC Dropout model prediction is given by the softmax of the class-wise mean over $m \in \mathbb{N}$ forward passes through the network up until the the logit output $l^c_{\theta_i}$ with a fixed dropout rate p, resulting in different (masked) weights $\{\theta_i\}_{i=1}^m$, given by,

$$\mu(x^\star) = \text{softmax}\left(\frac{1}{N}\sum_{i=1}^{N} l_{\theta_i}(x^\star)\right) \in \mathcal{CAT}(c). \tag{29}$$

The loss is the standard Cross-Entropy as in (5). The predictive uncertainty is given by the standard Cross-Entropy, yet computed with respect to the predictions only that are substituted for the true labels in (5).

Hyperparameter Overview for MC Dropout models:

Summary of hyperparameters for the MC Dropout models. The HP burn-in-epochs is only available for MC Dropout GMM.

Hyperparameter	value range	hints
Number of burn-in-epochs	$\approx [0, 100]$	after burn-in-epochs train variance and mean outputs.
Drop out rate	$p \in [0, 1)$	start with $p = 0.2$.

BNN with VI ELBO: Bayesian Neural Networks (BNNs) with variational inference (VI) are an approximate Bayesian method. Here, we follow the mean-field assumption, meaning that the variational distribution is factorized as a product of individual Gaussian distributions. This results in a diagonal Gaussian approximation of the posterior distribution over the model parameters

The most common approach is to maximize the evidence lower bound (ELBO). We note that there are other, alternative approaches for variational inference, such as α-divergence minimization [23].

Uitilizing standard stochastic gradient descent by using the reparameterization trick [29] one can backpropagate with a necessary sampling procedure, a process called Monte Carlo variational Bayes [48].

The predictive likelihood is given by a factorized as a product of individual Gaussian distributions per weight,

$$p(Y|\theta, X) = \prod_{i=1}^{N} p(y_i|\theta, x_i) = \prod_{i=1}^{N} \mathcal{N}(y_i|f_\theta(x_i), \Sigma). \tag{30}$$

The prior on the weights is given by,

$$p(\theta) = \prod_{l=1}^{L} \prod_{h=1}^{V_l} \prod_{j=1}^{V_{l-1}+1} \mathcal{N}(w_{hj,l}|0, \lambda) \tag{31}$$

where $w_{hj,l}$ is the h-th row and the j-th column of weight matrix θ_L at layer index L and λ is the prior variance. Note that as we use partially stochastic networks, (31) may contain less factors $\mathcal{N}(w_{hj,l}|0, \lambda)$ depending on how many layers are stochastic. Then, the posterior distribution of the weights is obtained by Bayes' rule as

$$p(\theta|\mathcal{D}) = \frac{p(Y|\theta, X)p(\theta)}{p(Y|X)}. \tag{32}$$

As the posterior distribution over the weights is intractable a variational approximation is used,

$$q(\theta) \approx p(\theta|\mathcal{D}), \tag{33}$$

that is a diagonal Gaussian. Now given an input $x^\star$, the predictive distribution can be obtained as

$$p(y^\star|x^\star, \mathcal{D}) = \int p(y^\star|\theta, x^\star)p(\theta|\mathcal{D})d\theta. \tag{34}$$

As (34) is intractable it is approximated by sampling form the approximation $q(\theta)$ in (33) to the posterior distribution in (32). The parameters of $q(\theta)$ are obtained by maximizing the evidence lower bound (ELBO) on the Kullback-Leibler (KL) divergence between the variational approximation and the posterior distribution over the weights.

BNN with VI ELBO Regression: For regression, the negative ELBO is given by,

$$\mathcal{L}(\theta, (x^\star, y^\star)) = \beta D_{KL}(q(\theta)||p(\theta)) + \frac{1}{2}\ln\left(2\pi\sigma^2\right) + \frac{1}{2\sigma^2}\left(f_\theta(x^\star) - y^\star\right)^2. \tag{35}$$

The KL divergence can be computed analytically in the case of a Gaussian prior. The hyperparameter β can be used to weight the influence of the variational parameters relative to that of the data. Alternatively, in the case of a fixed dataset of size N this parameter is automatically set to $\frac{1}{N}$. The hyperparameter σ can be either fixed or set to be an additional parameter to be tuned by including it in the objective function Eq. (35), a process called type-II maximum likelihood.

The predictive mean is obtained as the mean of the network output f_θ with S weight samples from the variational approximation $\theta_s \sim q(\theta)$,

$$f_m(x^\star) = \frac{1}{S}\sum_{i=1}^{S} f_{\theta_s}(x^\star). \tag{36}$$

The predictive uncertainty is given by the standard deviation thereof, including the (possibly estimated) constant output noise σ:

$$\sigma_p(x^\star) = \sqrt{\frac{1}{S}\sum_{i=1}^{S}\left(f_{\theta_s}(x^\star) - f_m(x^\star)\right)^2 + \sigma^2}. \tag{37}$$

If one uses the NLL and adapts the BNN to output a mean and standard deviation of a Gaussian $f_{\theta_s}(x^\star) = (\mu_{\theta_s}(x^\star), \sigma_{\theta_s}(x^\star))$, the mean prediction is given by

$$f_m(x^\star) = \frac{1}{S}\sum_{s=1}^{S}\mu_{\theta_s}(x^\star). \tag{38}$$

and the predictive uncertainty is obtained as the standard deviation of the corresponding Gaussian mixture model obtained by the weight samples,

$$\sigma_p(x^\star) = \sqrt{\frac{1}{S}\sum_{s=1}^{S}\left(\mu_{\theta_s}(x^\star) - f_m(x^\star)\right)^2 + \sum_{s=1}^{S}\sigma^2_{\theta_s}(x^\star)}. \tag{39}$$

BNN with VI ELBO Classification: For classification the negative loss objective is given by,

$$\mathcal{L}(\theta, (x^\star, y^\star)) = \beta D_{KL}(q(\theta)||p(\theta)) - \sum_{q=1}^{c} 1_{y_q^\star}\log\left(f_\theta^c(x^\star)_q\right). \tag{40}$$

The KL divergence can be computed analytically in the case of a Gaussian prior. The hyperparameter β can be used to weight the influence of the variational parameters relative to that of the data. Alternatively, in the case of a fixed dataset of size N this parameter is automatically set to $\frac{1}{N}$.

The prediction is obtained as the softmax of the mean of logit outputs l_θ with S weight samples from the variational approximation $\theta_s \sim q(\theta)$,

$$\mu_m^c(x^\star) = \text{softmax}\left(\frac{1}{S}\sum_{i=1}^{S} l_{\theta_s}(x^\star)\right) \in \mathcal{CAT}(c). \tag{41}$$

The predictive uncertainty is given by the standard cross entropy:

$$\sigma_p(x^\star) = -\sum_{q=1}^{c} \mathbb{1}_{y_q^\star}\log\left(\mu_m^c(x^\star)_q\right). \tag{42}$$

Hyper Parameter Overview for BNN with VI ELBO models:

Summary of hyperparameters for the BNN with VI ELBO models

Hyperparameter	value range	hints
Number burn-in-epochs	$\approx [0, n]$	after burn-in-epochs train variance and mean outputs.
Loss scale factor β	$\beta \approx [100, 500]$	should depend on parameter and train set size.
Samples during training S_{tr}	$S_{tr} \approx [5, 20]$	depending on network size and computing resources.
Samples during tests and prediction S_{te}	$S_{te} \approx [5, 50]$	depending on network size and computing resources.
Output noise scale σ	$\sigma \approx [1.0, 5.0]$	depending on label noise.
Prior mean μ_p for stochastic parameters	$\mu_p \approx [0, 1.0]$	start with 0.
Prior variance σ_p for stochastic parameters	$\sigma_p \approx [0, 3.0]$	start with 1.0.
Mean initialization for posterior μ_{pr}	$\mu_{pr} \approx [0, 1.0]$	approximate posterior over parameters
Variance initialization for posterior ρ_{pr}	$\rho_{pr} \approx [-6.0, 0.0]$	variance through $\sigma = \log(1 + \exp(\rho))$, approximate posterior over parameters
Bayesian layer type	"flipout" or "reparameterization"	
Stochastic module names	list of module names or a list of module numbers	Transform module to be stochastic.

BNN+LV: BNNs with latent variables (LVs) extend BNNs with VI to encompass LVs that model aleatoric uncertainty. The BNN+LV model is proposed in [11].

The likelihood is given by

$$p(Y|\theta, z, X) = \prod_{i=1}^{K} p(y_i|\theta, z_i, x_i) = \prod_{i=1}^{K} \mathcal{N}(y_i|f_\theta(x_i, z_i), \Sigma). \tag{43}$$

The prior on the weights by (31) as for BNNs with VI. The prior distribution of the latent variables z is given by

$$p(z) = \prod_{i=1}^{K} \mathcal{N}(z_i|0, \gamma) \tag{44}$$

where γ is the prior variance.

Then, with the assumed likelihood function and prior, a posterior over the weights θ and latent variables z is obtained via Bayes' rule:

$$p(\theta, z|\mathcal{D}) = \frac{p(Y|\theta, z, X)p(\theta)p(z)}{p(Y|X)} \tag{45}$$

The approximate the posterior is given by

$$q(\theta, z) = \underbrace{\left[\prod_{l=1}^{L}\prod_{h=1}^{V_l}\prod_{j=1}^{V_{l-1}+1} \mathcal{N}(w_{hj,l}|m^w_{hj,l}, v^w_{hj,l})\right]}_{q(\theta)} \times \underbrace{\left[\prod_{i=1}^{K} \mathcal{N}(z_i|m^z_i, v^z_i)\right]}_{q(\mathbf{z})}. \tag{46}$$

Now the parameters $m^w_{hj,l}$,$v^w_{hj,l}$ and m^z_i, v^z_i can be obtained by minimizing a divergence between $p(\theta, z|\mathcal{D})$. Here the following approximation of the α divergence, as proposed in [23] and [10], is used,

$$E_\alpha(q) = -\log Z_q - \frac{1}{\alpha}\sum_{n=1}^{N} \log \mathbf{E}_{\Theta, z_n \sim q}\left[\left(\frac{p(\mathbf{y}_n|\Theta, \mathbf{x}_n, z_n, \mathbf{\Sigma})}{f(\Theta)f_n(z_n)}\right)^\alpha\right], \tag{47}$$

where Z_n is the normalising constant of the exponential form of (46) and $f(\Theta)$ and $f_n(z_n)$ are functions depending on the parameters of the distributions (31) and (44), see [10] for details. In order to make this optimization problem scalable, SGD is used with mini-batches, and the expectation over q is approximated with an average over K samples drawn from q.

The posterior predictive distribution is given by,

$$p(y_\star|x_\star, \mathcal{D}) = \int \left[\int \mathcal{N}(y_\star|f_\theta(x_\star, z_\star), \Sigma)\mathcal{N}(z_\star|0, \gamma)dz_\star\right] p(\theta, z|\mathcal{D})d\theta dz. \tag{48}$$

The network prediction $f_\theta(x_\star, z_\star)$ uses $z_\star$ sampled from the prior distribution $\mathcal{N}(z_\star|0, \gamma)$ because this is the only evidence we have about the latent variable for a new data point since all data points are assumed to be independent. However, the above posterior predictive distribution is intractable in this form. So instead we use sampling from the posterior distribution of the weights. The mean prediction is then given by the mean prediction of samples and the predictive uncertainty is obtained as standard deviation of samples from the approximation to (48).

BNN VI Regression: The predictive mean is obtained as the mean of the network output f_θ with S weight samples from the variational approximation $\theta_s \sim q(\theta)$ and samples of latent variables $z \sim \mathcal{N}(0, \gamma)$,

$$f_m(x^\star) = \frac{1}{SK}\sum_{i=1}^{S}\sum_{k=1}^{K} f_{\theta_s}(x^\star, z_k). \tag{49}$$

The predictive uncertainty is given by the standard deviation thereof, including the (possibly estimated) constant output noise σ:

$$\sigma_p(x^\star) = \sqrt{\frac{1}{SK}\sum_{i=1}^{S}\sum_{k=1}^{K}\left(f_{\theta_s}(x^\star, z_k) - f_m(x^\star)\right)^2 + \sigma^2}. \tag{50}$$

If one uses the NLL and adapts the BNN LV to output a mean and standard deviation of a Gaussian $f_{\theta_s}(x^\star, z) = (\mu_{\theta_s}(x^\star, z), \sigma_{\theta_s}(x^\star))$, the mean prediction is given by

$$f_m(x^\star) = \frac{1}{SK}\sum_{i=1}^{S}\sum_{k=1}^{K}\mu_{\theta_s}(x^\star, z_k). \tag{51}$$

and the predictive uncertainty is obtained as the standard deviation of the corresponding Gaussian mixture model obtained by the weight samples and latent variable samples,

$$\sigma_p(x^\star) = \sqrt{\frac{1}{SK}\sum_{i=1}^{S}\sum_{k=1}^{K}\left(\mu_{\theta_s}(x^\star, z_k) - f_m(x^\star)\right)^2 + \sum_{s=1}^{S}\sigma^2_{\theta_s}(x^\star)}. \tag{52}$$

Hyperparameter Overview for BNN LV:

Summary of hyperparameters for the BNN with LV model

Hyperparameter	value range	hints
Number burn-in-epochs	$\approx [0, n]$	after burn-in-epochs train variance and mean outputs.
Loss scale factor β	$\beta \approx [100, 500]$	should depend on parameter and train set size.
Samples during training S_{tr}	$S_{tr} \approx [5, 20]$	depending on network size and computing resources.
Samples during tests and prediction S_{te}	$S_{te} \approx [5, 50]$	depending on network size and computing resources.
Output noise scale σ	$\sigma \approx [1.0, 5.0]$	depending on label noise.
Prior mean μ_p for stochastic parameters	$\mu_p \approx [0, 1.0]$	start with 0.
Prior variance σ_p for stochastic parameters	$\sigma_p \approx [0, 3.0]$	start with 1.0.
Mean initialization for posterior μ_{pr}	$\mu_{pr} \approx [0, 1.0]$	approximate posterior over parameters
Variance initialization for posterior ρ_{pr}	$\rho_{pr} \approx [-6.0, 0.0]$	variance through $\sigma = \log(1 + \exp(\rho))$, approximate posterior over parameters
Prior mean for LV network		usually 0.
Prior variance for LV network	$\gamma \approx \sqrt{d}$	d is dimension of inputs.
LV dimension	$d_z = 1$	usually chosen as 1.
Bayesian layer type	"flipout" or "reparameterization"	
Stochastic module names	list of module names or a list of module numbers	Transform module to be stochastic.

Laplace Approximation: Originally introduced by [36], the Laplace Approximation has been adapted to modern neural networks by [49] and [8] and is an approximate Bayesian method. The goal of the Laplace Approximation is to use a second-order Taylor expansion around the fitted MAP estimate and yield a posterior approximation over the model parameters via a full-rank, diagonal or Kronecker-factorized approach. In order for the Laplace Approximation to be computationally feasible for larger network architectures, we use the Laplace library to include approaches, such as subnetwork selection that have been for example proposed by [9].

The general idea of the Laplace Approximation to obtain a distribution over the network parameters with a Gaussian distribution centered at the MAP estimate of the parameters [9]. In this setting, a prior distribution $p(\theta)$ is defined over our network parameters. Because modern neural networks consists of millions of parameters, obtaining a posterior distribution over the weights θ is intractable. The LA takes MAP estimate of the parameters θ_{MAP} from a trained network $f_{\theta_{MAP}}(x) = \mu_{\theta_{MAP}}(x)$ and constructs a Gaussian distribution around it. The parameters θ_{MAP} are obtained by

$$\theta_{MAP} = \text{argmin}\mathcal{L}(\theta; D), \tag{53}$$

where $\mathcal{L}$ is the mean squared error or also referred to as the ℓ^2 loss, $\mathcal{L}(\theta; \mathcal{D}) := -\sum_{i=1}^{n} log(p(y_i|f_\theta(x_i)))$ and the posterior predictive distribution

$$p(y_i|f_\theta(x_i)). \tag{54}$$

Then with Bayes Theorem, as in [9], one can relate the posterior to the loss,

$$p(\theta|D) = p(D|\theta)p(\theta)/p(D) = \frac{1}{Z} exp(-\mathcal{L}(\theta; D)), \tag{55}$$

with $Z = \int p(D|\theta)p(\theta)d\theta$. Now a second-order expansion of $\mathcal{L}$ around θ_{MAP} is used to construct a Gaussian approximation to the posterior $p(\theta|D)$:

$$-\mathcal{L}(\theta; D) \approx -\mathcal{L}(\theta_{MAP}; D) - \frac{1}{2}(\theta - \theta_{MAP})(\nabla^2_\theta \mathcal{L}(\theta; D)|\theta_{MAP})(\theta - \theta_{MAP}). \tag{56}$$

The term with the first order derivative is zero as the loss is evaluated at a minimum θ_{MAP} [42], and, further, one assumes that the first term is neglible as the loss is evaluated at $\theta = \theta_{MAP}$. Then taking the expontential of both sides allows to identify, after normalization, the Laplace approximation,

$$p(\theta|D) \approx \mathcal{N}(\theta_{MAP}, \Sigma) \qquad \text{with} \qquad \Sigma = (\nabla^2_\theta \mathcal{L}(\theta; D)|\theta_{MAP})^{-1}. \tag{57}$$

As the covariance is just the inverse Hessian of the loss, with $\theta_{MAP} \in \mathcal{R}^W$ and $H^{-1} \in \mathcal{R}^{W \times W}$, with W being the number of weights, the posterior distribution is given by

$$p(\theta|D) \approx \mathcal{N}(\theta_{MAP}, H^{-1}). \tag{58}$$

The computation of the Hessian term is still expensive. Therefore, further approximations are introduced in practice, most commonly the Generalized Gauss-Newton matrix [38]. This takes the following form:

$$H \approx \widetilde{H} = \sum_{n=1}^{N} J_n^T H_n J_n, \tag{59}$$

where $J_n \in \mathcal{R}^{O \times W}$ is the Jacobian of the model outputs with respect to the parameters θ and $H_n \in \mathcal{R}^{O \times O}$ is the Hessian of the negative log-likelihood with respect to the model outputs, where O denotes the model output size and W the number of parameters.

Given (58) during inference on unseen data, one cannot compute the full posterior predictive distribution but instead resort to sampling $\theta_s \sim p(\theta|D)$ for $s \in \{1, ..., S\}$ to approximate the predictions.

Laplace Approximation Regression: for regression the posterior predictive distribution (54) is chosen to be a Gaussian with constant variance σ^2, such that the loss is the mean squared error and a homoskedastic noise model is assumed. Then, the predictive mean is given by

$$f_m(x^\star) = \frac{1}{S} \sum_{s=1}^{S} f_{\theta_s}(x^\star), \tag{60}$$

and obtain the predictive uncertainty by

$$\sigma^2(x^\star) = \sqrt{\frac{1}{S} \sum_{s=1}^{S} \left(f_{\theta_s}(x^\star) - \hat{y}(x^\star)\right)^2 + \sigma^2}. \tag{61}$$

For the subnet strategy, we include the options from the Laplace library for selecting the stochastic parameters.

Laplace Approximation Classification: for classification the posterior predictive distribution (54) is chosen to be a categorical distribution. The prediction of the Laplace Approximation classification model is obtained as the softmax of the mean of logit outputs l_θ with S weight samples from the approximation to the posterior distribution over the weights

$$f_m(x^\star) = \text{softmax}\left(\frac{1}{S} \sum_{i=1}^{S} l_{\theta_s}(x^\star)\right) \in \mathcal{CAT}(c). \tag{62}$$

The predictive uncertainty is given by the standard cross entropy:

$$\sigma_p(x^\star) = -\sum_{q=1}^{c} \mathbb{1}_{y_q^\star} \log\left(f_m(x^\star)_q\right). \tag{63}$$

Hyper Parameter Overview for Laplace Approximation models:

Summary of hyperparameters for the BNN with VI model

Hyperparameter	value range	hints
Number burn-in-epochs	$\approx [0, n]$	after burn-in-epochs train variance and mean outputs.
Loss scale factor β	$\beta \approx [100, 500]$	should depend on parameter and train set size.
Samples during training S_{tr}	$S_{tr} \approx [5, 20]$	depending on network size and computing resources.
Samples during tests and prediction S_{te}	$S_{te} \approx [5, 50]$	depending on network size and computing resources.

SWAG: Is an approximate Bayesian method and uses a low-rank Gaussian distribution as an approximation to the posterior over model parameters. The quality of approximation to the posterior over model parameters is based on using a high SGD learning rate that periodically stores weight parameters in the last few epochs of training [37]. SWAG is based on Stochastic Weight Averaging (SWA), as proposed in [25]. For SWA the weights are obtained by minimising the MSE loss with a variant of stochastic gradient descent. After, a number of burn-in epochs, $\tilde{t} = T - m$, the last m weights are stored and averaged to obtain an approximation to the posterior, by

$$\theta_{SWA} = \frac{1}{m}\sum_{t=\tilde{t}}^{T} \theta_t. \tag{64}$$

For SWAG we use the implementation as proposed by [37]. Here the posterior is approximated by a Gaussian distribution with the SWA mean, (64) and a covariance matrix over the stochastic parameters that consists of a low rank matrix plus a diagonal,

$$p(\theta|D) \approxeq \mathcal{N}\left(\theta_{SWA}, \frac{1}{2}(\Sigma_{diag} + \Sigma_{low-rank})\right). \tag{65}$$

The diagonal part of the covariance is given by

$$\Sigma_{diag} = \text{diag}(\bar{\theta^2} - \theta_{SWA}^2) \tag{66}$$

where,

$$\bar{\theta^2} = \frac{1}{m}\sum_{t=\tilde{t}}^{T} \theta_t^2. \tag{67}$$

The low rank part of the covariance is given by

$$\Sigma_{low-rank} = \frac{1}{m}\sum_{t=\tilde{t}}^{T} (\theta_t - \bar{\theta}_t)(\theta_t - \bar{\theta}_t)^T, \tag{68}$$

where $\bar{\theta}_t$ is the running estimate of the mean of the parameters from the first t epochs or also samples. In order to approximate the mean prediction, we again resort to sampling from the posterior (65).

SWAG Regression: with $\theta_s \sim p(\theta|D)$ for $s \in \{1, ..., S\}$, the mean prediction is given by

$$\hat{y}(x^\star) = \frac{1}{S}\sum_{s=1}^{S} f_{\theta_s}(x^\star), \tag{69}$$

and obtain the predictive uncertainty by

$$\sigma(x^\star) = \sqrt{\frac{1}{S}\sum_{s=1}^{S}\left(f_{\theta_s}(x^\star) - \hat{y}(x^\star)\right)^2}. \tag{70}$$

SWAG Classification: with $\theta_s \sim p(\theta|D)$ for $s \in \{1, ..., S\}$, the prediction is given by

$$f_m(x^\star) = \text{softmax}\left(\frac{1}{S}\sum_{i=1}^{S} l^c_{\theta_s}(x^\star)\right) \in \mathcal{CAT}(c). \tag{71}$$

The predictive uncertainty is given by the standard cross entropy:

$$\sigma_p(x^\star) = -\sum_{q=1}^{c} \mathbb{1}_{y_q^\star} \log\left(f_m(x^\star)_q\right). \tag{72}$$

For the subnet strategy, we include selecting the parameters to be stochastic by module names.

VBLL: variational Bayesian last layer is a Bayesian UQ method using the last layer neural network component introduced in [21]. The method uses a feature extractor $g_\theta : X \to \mathbb{R}^{d_f}$ with weight θ.

VBLL Regression: for regression VBLL models the output layer as a linear Bayesian layer,

$$y^\star = \omega^T g_\theta(x^\star) + \epsilon, \tag{73}$$

where $\epsilon \in \mathcal{N}(0, \Sigma)$. Fixing an independent Gaussian prior, $\omega \sim \mathcal{N}(\bar{\omega}, S)$, yields a predictive posterior distribution for VBLL

$$p(y^\star|x^\star, \theta, \bar{\omega}, S, \Sigma) = \mathcal{N}(\bar{\omega}^T g_\theta(x^\star), g_\theta(x^\star)^T S g_\theta(x^\star) + \Sigma). \tag{74}$$

The loss objective is given by

$$\begin{aligned}&\mathcal{L}(\theta, \omega, S, \Sigma, (x^\star, y^\star)) \\ &= 2\ln\left(2\pi\sigma(x^\star)^2\right) + \frac{1}{2\sigma(x^\star)^2}\left(\omega^T g_\theta(x^\star) - y^\star\right)^2 - \frac{1}{2} g_\theta(x^\star)^T S g_\theta(x^\star)\Sigma^{-1}.\end{aligned} \tag{75}$$

The mean prediction, is then given by

$$f_\theta(x^\star) = \omega^T g_\theta(x^\star). \tag{76}$$

The predictive uncertainty is given by

$$\sigma(x^\star) = g_\theta(x^\star)^T S g_\theta(x^\star) + \Sigma. \tag{77}$$

VBLL Classification: for classification VBLL models the logit output layer as a linear Bayesian layer and the output as a categorical distribution. The predictive posterior distribution is given by

$$p(y^\star|x^\star, \theta, \bar{\omega}) = \text{softmax}(z), \tag{78}$$

with logits $z = \omega g_\theta(x^\star) + \epsilon$, where a Gaussian prior is defined for $\omega \sim \mathcal{N}(\bar{\omega}, S)$. The loss objective is given as

$$\mathcal{L}(\theta, \omega, S, \Sigma, (x^\star, y^\star)) = y^{\star T} \omega g_\theta(x^\star) - \text{LSE}_k \left(\omega_k^T g_\theta(x^\star) + \frac{1}{2} (g_\theta(x^\star)^T S_k g_\theta(x^\star) + \sigma_k^2) \right), \quad (79)$$

where $\text{LSE}_k(\cdot)$ denotes the log-sum-exp function, with the sum over k. The prediction is given by (78) and the uncertainty by the standard cross entropy.

SGLD: Stochastic gradient Langevin dynamics is an approximate sampling method, introduced in [59]. The posterior distribution over the weights is sampled by sampling from the parameter updates obtained by a variant of stochastic gradient descent. In SGLD Gaussian noise is injected into the parameter updates, such that the parameters θ do not collapse to just the MAP solution. The proposed update in [59] is

$$\begin{aligned} \Delta\theta_t &= \frac{\epsilon_t}{2} \left(\nabla \log p(\theta_t) + \frac{N}{n} \sum_{i=1}^{n} \nabla \log p(x_{ti}|\theta_t) \right) + \eta_t \\ \eta_t &\sim \mathcal{N}(0, \epsilon_t). \end{aligned} \quad (80)$$

After, a number of burn-in epochs, $\tilde{t} = T - m$, the last m weights are stored and averaged to obtain an approximation to the posterior. The mean prediction is then obtained as for a weighted ensemble,

$$\hat{y}(x^\star) \simeq \frac{\sum_{t=\tilde{t}}^{T} \epsilon_t f_{\theta_t}(x^\star)}{\sum_{t=1}^{T} \epsilon_t}. \quad (81)$$

Another possibility is to resort to a simpler average as is usually done for MC sampling methods to obtain the mean prediction,

$$\bar{y}(x^\star) \simeq \frac{1}{m} \sum_{t=\tilde{t}}^{T} f_{\theta_t}(x^\star). \quad (82)$$

The predictive uncertainty is then obtained as,

$$\sigma(x^\star) = \sqrt{\frac{1}{m} \sum_{t=\tilde{t}}^{T} \left(f_{\theta_t}(x^\star) - \bar{y}(x^\star) \right)^2}. \quad (83)$$

2.4 Gaussian Process Based UQ Methods

Recap of Gaussian Processes (GPs): The goal of previously introduced methods was to find a distribution over the weights of a parameterized function i.e. a neural network. In contrast, the basic idea of a Gaussian Process (GP) is to instead consider a distribution over possible functions, that fit the data in some way. Formally,

"A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution." [52]

Precisely, a GP can be described by a possibly infinite amount of function values

$$f(x) \sim \mathcal{GP}(m(x), k_\gamma(x)), \quad (84)$$

such that any finite collection of function values f has a joint Gaussian distribution,

$$f = f(X) = [f(x_1), \ldots, f(x_K)]^\top \sim \mathcal{N}(m_X, \mathcal{K}_{X,X}), \quad (85)$$

with a mean vector, $(m_X)i = m(x_i)$, and covariance matrix, $(\mathcal{K}_{X,X})_{ij} = k_\gamma(x_i, x_j)$, stemming from the mean function m and covariance kernel of the GP, k_γ, that is parametrized by γ. A commonly used covariance function is the squared exponential, also referred to as Radial Basis Function (RBF) kernel, exponentiated quadratic or Gaussian kernel:

$$k_\gamma(x, x') = \text{cov}(f(x), f(x')) = \eta^2 \exp\left(-\frac{1}{2l^2}|x - x'|^2\right). \tag{86}$$

Where $\gamma = (\eta^2, l)$ and η^2 can be set to 1 or tuned as a hyperparameter. By default the lengthscale $l = 1$ but can also be optimized over. Now the GP, $f(x) \sim GP(m(x), k(x, x'))$, as a distribution over functions can be used to solve a regression problem. Following [52], consider the simple case where the observations are noise free and you have training data $\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^N$ with $X = (x_i)_{i=1}^N$ and $Y = (y_i)_{i=1}^N$. The joint prior distribution of the training outputs, Y, and the test outputs $f_* = f_*(X_*) = (f(i_k))_{i=1}^m$ where $X_* = (x_i)_{i=1}^m$ are the test points, according to the prior is

$$p(Y, f_*) = \mathcal{N}\left(0, \begin{bmatrix} \mathcal{K}_{X,X} & \mathcal{K}_{X,X_*} \\ \mathcal{K}_{X_*,X} & \mathcal{K}_{X_*,X_*} \end{bmatrix}\right). \tag{87}$$

Here the mean function is assumed to be $m_X = 0$ and $\mathcal{K}_{X,X_*}$ denotes the $N \times m$ matrix of the covariances evaluated at all pairs of training and test points, and similarly for the other entries $\mathcal{K}_{X,X}$, $\mathcal{K}_{X_*,X_*}$ and $\mathcal{K}_{X_*,X}$. To make predictions based on the knowledge of the training points, conditioning on the prior observations is used and yields,

$$\begin{aligned} p(f_*|X_*, X, Y) &= \mathcal{N}(\mathcal{K}_{X_*,X}\mathcal{K}_{X,X}^{-1}Y, \mathcal{K}_{X_*,X_*} - \mathcal{K}_{X_*,X_*}\mathcal{K}_{X,X}^{-1}\mathcal{K}_{X,X_*}) \\ &= \mathcal{N}(m(X, X_*, Y), \tilde{\mathcal{K}}_{X,X_*}). \end{aligned}$$

Now to generate function values on test points, one uses samples from the posterior distribution $f_*(X_*) \sim \mathcal{N}(m(X, X_*, Y), \tilde{K}(X, X_*))$. To illustrate how we can obtain these samples from the posterior distribution, consider a Gaussian with arbitrary mean m and covariance K, i.e. $f_* \sim \mathcal{N}(m, K)$. For this one can use a scalar Gaussian generator, which is available in many packages:

1. Compute the Cholesky decomposition of $K = LL^T$, where L is a lower triangular matrix. This works because K is symmetric by definition.
2. Then, draw multiple $u \sim \mathcal{N}(0, I)$.
3. Now, compute the samples with $f_* = m + Lu$. This has the desired mean, m and covariance $L\mathbb{E}(uu^T)L^T = LL^T = K$.

The above can be extended to incorporate noisy measurements $y \to y + e$, see [52], or noise on the inputs as in [27]. Both of these extensions require tuning of further hyperparameters, yet beneficially allow to incorporate a prediction of aleatoric uncertainty in a GP.

For example, assume additive Gaussian noise on the distribution of the function values,

$$p(y(x)|f(x)) = \mathcal{N}(y(x); f(x), \sigma^2). \tag{88}$$

Then the predictive distribution of the GP evaluated at the K_* test points, X_*, is given by

$$\begin{aligned} p(f_*|X_*, X, Y, \gamma, \sigma^2) &= \mathcal{N}(\mathbb{E}[f_*], \text{cov}(f_*)), \\ \mathbb{E}[f_*] &= m_{X_*} + \mathcal{K}_{X_*,X}[\mathcal{K}_{X,X} + \sigma^2 I]^{-1}Y, \\ \text{cov}(f_*) &= \mathcal{K}_{X_*,X_*} - \mathcal{K}_{X_*,X}[\mathcal{K}_{X,X} + \sigma^2 I]^{-1}\mathcal{K}_{X,X_*}. \end{aligned} \tag{89}$$

Here m_{X_*} is the $K_* \times 1$ mean vector, which is assumed to be zero in the previous case.

In both cases, with and without additive noise on the function values, the GP is trained by learning interpretable kernel hyperparameters. The log marginal likelihood of the targets y - the probability of

the data conditioned only on kernel hyperparameters γ - provides a principled probabilistic framework for kernel learning:

$$\log p(y|\gamma, X) \propto -\left(y^\top (\mathcal{K}_\gamma + \sigma^2 I)^{-1} y + \log |\mathcal{K}_\gamma + \sigma^2 I|\right) , \tag{90}$$

where $\mathcal{K}_\gamma$ is used for $\mathcal{K}_{X,X}$ given γ. Kernel learning can be achieved by optimizing Eq. (90) with respect to γ.

The computational bottleneck for inference is solving the linear system $(\mathcal{K}_{X,X} + \sigma^2 I)^{-1} y$, and for kernel learning it is computing the log determinant $\log |\mathcal{K}_{X,X} + \sigma^2 I|$ in the marginal likelihood. The standard approach is to compute the Cholesky decomposition of the $K \times K$ matrix $\mathcal{K}_{X,X}$, which requires $\mathcal{O}(K^3)$ operations and $\mathcal{O}(K^2)$ storage. After inference is complete, the predictive mean costs $\mathcal{O}(K)$, and the predictive variance costs $\mathcal{O}(K^2)$, per test point x_*.

Deep Kernel Learning Regression: Conceptually DKL consists of a NN architecture that extracts a feature representation of the input x and fits an approximate GP on top of these features to produce a probabilistic output [61]. DKL combines GPs and DNNs in a scalable way. In practice, all parameters, the weights of the feature extractor and the GP parameters are optimized jointly by maximizing the log marginal likelihood of the GP. We utilize GPytorch for our implementation [14] and use a grid approximation where we optimized over the number of inducing points. For DKL the GP is transformed by replacing the inputs x by the outputs of a NN in the following way. The kernel $k_\gamma(x, x')$ with hyperparameters θ is replaced by,

$$k_\gamma(x, x') \to k_\gamma(g(x, \theta), g(x', \theta)) , \tag{91}$$

where $g(x, \theta)$ is a non-linear mapping given by a deep architecture, such as a deep convolutional network mapping into a feature space of dimension J, parametrized by weights θ,

$$\begin{aligned} g(\cdot, \theta) : X &\to \mathbb{R}^J \\ x &\mapsto g(x, \theta). \end{aligned} \tag{92}$$

This so called deep kernel in (91) is now used as the covariance function of a GP to model data $\mathcal{D} = \{x_i, y_i\}_{i=1}^N$. The deep kernel hyperparameters, $\rho = \{\gamma, \theta, \sigma^2\}$, can be *jointly* learned by maximizing the log *marginal likelihood* of the GP (93).

$$\mathcal{L} = \log p(Y|\gamma, X, \theta) \propto -\left(y^\top (K_{\gamma,\theta} + \sigma^2 I)^{-1} y + \log |K_{\gamma,\theta} + \sigma^2 I|\right) , \tag{93}$$

Except for the replacement of input data, one can almost follow the same procedures for learning and inference as for GPs as outlined previously. For optimizing (93) the chain rule is used to compute derivatives of the log marginal likelihood with respect to the deep kernel hyperparameters as in [61].

Exact inference is possible for the regression case, yet the computational complexity scales cubically with the number of data points and makes it not suitable for large datasets. Thus, following [57] in the implementation the sparse GP of [56] and the variational approximation of [22] is used, in order to allow for DKL to scale to large training datasets. The sparse GP approximation of [56] augments the DKL model with M inducing inputs, $Z \in \mathbb{R}^{M \times J}$, where J is the dimensionality of the feature space, as in (92). Moreover, to perform computationally efficient inference we use the the variational approximation introduced by [22], where inducing points Z are treated as variational parameters. U are random variables with prior

$$p(U) = \mathcal{N}(U|m_Z, \mathcal{K}_{Z,Z}), \tag{94}$$

and variational posterior

$$q(U) = \mathcal{N}(U|\widetilde{m}, S), \tag{95}$$

where $\widetilde{m} \in \mathbb{R}^M$ and $S \in \mathbb{R}^{M \times M}$ are variational parameters and initialized at the zero vector and the identity matrix respectively. The approximate predictive posterior distribution at training points X is then

$$p(f|Y) \approx q(f) = \int p(f|U) q(U) dU \tag{96}$$

Here $p(f|U)$ is a Gaussian distribution for which we can find an analytic expression, see [22] for details. Note that we deviate from [22] in that our input points X are mapped into feature space just before computing the base kernel, while inducing points are used as is (they are defined in feature space). The variational parameters Z, $\widetilde{m}$, and S and the feature extractor parameters θ and GP model hyparparameters γ, given by l and η^2, and σ^2 are all learned at once by maximizing a lower bound on the log marginal likelihood of the predictive distribution $p(Y|X)$, the ELBO, denoted by $\mathcal{L}$. For the variational approximation above, this is defined as

$$\log(p(Y|X)) \geq \mathcal{L}(Z, m, S, \gamma, \theta, \sigma^2) = \sum_{i=1}^{N} \mathbb{E}_{q(f)} \left[\log p(y_i|f(x_i))\right] - \beta \mathrm{D}_{\mathrm{KL}}(q(U)||p(U)). \quad (97)$$

Both terms can be computed analytically when the likelihood is Gaussian and all parameters can be learned using stochastic gradient descent. To accelerate optimization gpytorch additionally utilizes the whitening procedure of [39] in their Variational Strategy. The approximate predictive posterior distribution at test points X^* is then

$$p(f_*|Y) \cong q(f_*) = \int p(f_*|U)q(U)dU \quad (98)$$

For regression tasks we directly use the function values f_* above as the predictions. We use the mean of $p(f_*|Y)$ as the prediction, and the variance as the uncertainty.

DKL Classification: For DKL Classificiation the likelihood is replaced wth the softmax (multiclass) likelihood, which is also used used for GP classification. The model head is formed of an independent approximate GP for each output dimension, see [57]. The predictions are obtained by approximating the posterior over the class probabilities. See the appendix in [57] for an overview.

Deterministic Uncertainty Estimation (DUE) - extension of DKL

Algorithm 1 Algorithm for training DUE [57]

1: **Definitions:**
- Residual NN $g_\theta : x \to \mathbb{R}^J$ with feature space dimensionality J and parameters θ.
- Approximate GP with parameters $\rho = \{\gamma, \sigma^2, \omega\}$, where $\gamma = \{l, \eta\}$ and l length scale and η output scale of k_γ, ω GP variational parameters (including m inducing point locations Z)
- Learning rate ζ, loss function $\mathcal{L}$

2: Using a random subset of p points of our training data, $X^{\text{init}} \subset X$, compute:
Initial inducing points: K-means on $g_\theta(X^{\text{init}})$ with $K = m$. Use found centroids as initial inducing point locations Z in GP.
Initial length scale:
$l = \frac{1}{\binom{p}{2}} \sum_{i=0}^{p} \sum_{j=i+1}^{p} |g_\theta(X_i^{\text{init}}) - g_\theta(X_j^{\text{init}})|_2$.

3: **for** minibatch $x_b, y_b \subset X, Y$ **do**
4: $\quad \theta' \leftarrow$ spectral_normalization(θ)
5: $\quad p(y_b'|x_b) \leftarrow$ evaluate_GP$_\theta(g_{\theta'}(x_b))$
6: $\quad \mathcal{L} \leftarrow \text{ELBO}_\theta(p(y_b'|x_b), y_b)$
7: $\quad (\rho, \theta) \leftarrow (\rho, \theta) + \zeta * \nabla_{\rho,\theta}\mathcal{L}$
8: **end for**

DUE builds on DKL by using the same model except for exchanging the feature extractor of the DKL model. With this replacement DUE addresses limitations of DKL and provides potentially robust uncertainty estimates. According to [57] with DKL, data points dissimilar to the training data (also called OOD data) can potentially be mapped close to feature representations of in-distribution points. These feature representations, which are close in some norm, input into the approximate GP yield similar or nearly the same predictions. This is called "feature collapse", and suggests that a constraint must be placed on the deep feature extractor. Based on deterministic uncertainty quantification (DUQ) [58] and spectrally normaplized GPs (SNGP) [34], the authors of [57] propose to use a bi-Lipschitz constraint on a feature extractor. This bi-Lipschitz constraint is enforced by spectral normalization on the weights, [41, 17]. This constraint mitigates so-called "feature collapse",

by forcing the feature representation to be sensitive to changes in the input (lower Lipschitz, avoids feature collapse) but also generalize due to smoothness (upper Lipschitz).

For convolutional and linear layers following [57], we use spectral normalization of the weight matrices to promote approximate bi-Lipschitz continuity. To promote spectral normalization for fully connected layers and 1×1 convolutions online power iteration are used and for larger convolutions an approximate method, as proposed in [17] and was first implemented by [5], is used. Spectral normalization is also extended to batch normalization by rescaling the weights, see [57] for details. Adding spectral normalization to batch normalization layers makes it more likely that the entire network's upper Lipschitz constant is bounded. The mean prediction and predictive uncertainty are obtained as for DKL for both the classification and regression tasks.

Summary of learnable parameters:

- weights of DNN feature extractor θ
- for the GP, parameters γ: noise hyperparamter σ^2, the GP function mean m, the length scale of the GP kernel l and the scale of the kernel η^2. In the above case the GP hyperparameters are learned by optimizing ELBO.

Summary of hyperparameters:

- number of power iterations for spectral normalization, usually set to $r = 1$
- number of initial inducing points M

2.5 Quantile Based UQ Methods

Quantile Regression (QR): The goal of Quantile Regression is to extend a standard regression model to also predict conditional quantiles that approximate the true quantiles of the data at hand. It does not make assumptions about the distribution of errors as is usually common. It is a more commonly used method in Econometrics and Time-series forecasting [30].

In the following we will describe univariate quantile regression. Any chosen conditional quantile $\alpha \in [0, 1]$ can be defined as

$$q_\alpha(x) := \inf\{y \in \mathbb{R} : F(y|X = x) \geq \alpha\}, \tag{99}$$

where $F(y|X = x) = P(Y \leq y|X = x)$ is a strictly monotonic increasing cumulative density function.

For Quantile Regression, the NN f_θ parameterized by θ, is configured to output the number of quantiles that we want to predict. This means that, if we want to predict p quantiles $[\alpha_1, ...\alpha_n]$,

$$f_\theta(x_\star) = (\hat{y}_1(x^\star), ..., \hat{y}_n(x^\star)). \tag{100}$$

The model is trained by minimizing the pinball loss function [30], given by the following loss objective,

$$\mathcal{L}_i(\theta, (x^\star, y^\star)) = \max\{(1 - \alpha_i)(y^\star - \hat{y}_i(x^\star)), \alpha(y^\star - \hat{y}_i(x^\star))\}. \tag{101}$$

Here $i \in \{1, ..., n\}$ denotes the number of the quantile and $100\alpha_i$ is the percentage of the quantile for $\alpha_i \in [0, 1)$. Note that for $\alpha = 1/2$ one recovers the ℓ^1 loss. During inference, the model will output an estimate for the chosen quantiles and these can be used as an indication of aleatoric uncertainty.

Conformalized Quantile Regression (CQR): Conformal Prediction is a post-hoc uncertainty quantification method to yield calibrated predictive uncertainty bands with proven coverage guarantees [4]. Based on a held out calibration set, CQR uses a score function to find a desired coverage quantile

$\hat{q}$ and conformalizes the QR output by adjusting the quantile bands wirh $\hat{q}$ for an unseen test point as follows $x_\star$:

$$T(x_\star) = [\hat{y}_{\alpha/2}(x_\star) - \hat{q}, \hat{y}_{1-\alpha/2}(x_\star) + \hat{q}] \tag{102}$$

where $\hat{y}_{\alpha/2}(x_\star)$ is the lower quantile output and $\hat{y}_{1-\alpha/2}(x_\star)$ is the higher quantile output and α is the desired miscoverage rate [50].

2.6 Diffusion Based UQ Methods

CARD: The classification and regression diffusion (CARD) models, as introduced in [20], combine a denoising diffusion-based conditional generative model and a pre-trained conditional mean estimator in order to obtain a predictive distribution given an input. Given a target $y^\star$ and input $x^\star$ CARD utilizes a series of intermediate predictions $y_{1:T}$ for a number of steps $T \in \mathbb{N}$. The parameters of the diffusion-based conditional generative model are obtained by optimising the following objective

$$\mathcal{L}_{\text{ELBO}}(y^\star, x^\star) = \mathcal{L}_0(y^\star, x^\star) + \sum_{t=2}^{T} \mathcal{L}_{t-1}(y^\star, x^\star) + \mathcal{L}_T(y^\star, x^\star), \tag{103}$$

where the individual terms are given by

$$\mathcal{L}_0(y^\star, x^\star) = \mathbb{E}_q \left[-\log(p_\theta(y^\star | y_1, x) \right] \tag{104}$$

$$\mathcal{L}_{t-1}(y^\star, x^\star) = \mathbb{E}_q \left[D_{\text{KL}}(q(y_{t-1}|y_t, y_0, x)||p_\theta(y_{t-1}|y_t, x)) \right] \tag{105}$$

$$\mathcal{L}_T(y^\star, x^\star) = \mathbb{E}_q \left[D_{\text{KL}}(q(y_T|, y_0, x)||p(y_T|x)) \right] \tag{106}$$

and the predictive distribution $p(y_T|x)$ is obtained by a MAP estimate, in our case the deterministic base model,

$$p(y_T|x) = \mathcal{N}(f_{\theta_{MAP}}(x), \mathbb{I}). \tag{107}$$

Following [47] the forward process of conditional distributions with a diffision schedule $(\beta_t)_{t=1}^T \in (0,1)^T$ is defined such that a closed-form solution exists,

$$p(y_t|y_{t-1}, f_{\theta_{MAP}}(x)) = \mathcal{N}(y_t; \sqrt{1-\beta_t} y_t + (1 - \sqrt{1-\beta_t}) f_{\theta_{MAP}}(x), \beta_t \mathbb{I}), \tag{108}$$

this admits a closed form and non-iterative solution at each time step $t \in \{1, ..., T\}$,

$$p(y_t|y_0, f_{\theta_{MAP}}(x)) = \mathcal{N}(y_t; \sqrt{\alpha_t} y_t + (1 - \sqrt{\alpha_t}) f_{\theta_{MAP}}(x), \beta_t \mathbb{I}), \tag{109}$$

with $\alpha_t = \Pi_{l=1}^t (1 - \beta_l)$. For regression the goal is to reverse the above diffusion process to recover the distribution of the noise term and, hence, obtaining the aleatoric uncertainty of the second moment predictive distribution $p(y|x)$. For this a neural network ϵ_θ is trained that given a sample y_t predicts the corresponding noise $\epsilon \approx \epsilon_\theta(x, y_t, f_{\theta_{MAP}}(x), t)$. The predictive mean and uncertainty, in terms of standard deviation, is obtained by moment matching with the predictive samples y_0 approximating the labels $y^\star$.

2.7 Post-hoc methods

RAPS: As introduced by [3] is a a posthoc conformal method referred to as Regularized Adaptive Prediction Sets (RAPS). RAPS is based on conformal prediction and can be used to adapt classifiers to output a predictive set containing the true label with a user-specified probability, such as 90% which then is true on average given some assumptions. Three desiderata, firstly, coverage desideratum says the sets must provide $1-\alpha$ coverage, secondly, the size desideratum says we want sets of small size, since these convey more detailed information and may be more useful in practice and, thirdly, adaptiveness desideratum says we want the sets to communicate instance-wise uncertainty: they should be smaller for easy test-time examples than for hard one.

Algorithm 2 RAPS Conformal Calibration

Input: α; $s \in [0,1]^{n \times K}$, $I \in \{1,...,K\}^{n \times K}$, and $y \in \{0,1,...,K\}^n$ corresponding respectively to the sorted scores, the associated permutation of indexes, and ground-truth labels for each of n examples in the calibration set; k_{reg}; λ; boolean $rand$

procedure RAPSC(α,s,I,y,λ)
  for $i \in \{1, \cdots, n\}$ **do**
    $L_i \leftarrow j$ such that $I_{i,j} = y_i$
    $E_i \leftarrow \Sigma_{j=1}^{L_i} s_{i,j} + \lambda(L_i - k_{reg})^+$
    if $rand$ **then**
      $U \sim \text{Unif}(0,1)$
      $E_i \leftarrow E_i - U * s_{i,L_i}$
  $\hat{\tau}_{ccal} \leftarrow$ the $\lceil (1-\alpha)(1+n) \rceil$ largest value in $\{E_i\}_{i=1}^n$
  return $\hat{\tau}_{ccal}$

Output: The generalized quantile, $\hat{\tau}_{ccal}$ ▷ The value in Eq. (3)

Figure 1: RAPS Conformal Calibration Algorithm. Figure from [3].

RAPS has three main steps. First, for a feature vector x, the base model computes class probabilities, and we order the classes from most probable to least probable. Then, we add a regularization term to promote small predictive sets. This algorithimic procedure is shown in Figure 1. Finally, we conformally calibrate the penalized prediction sets according to the algorithmic procedure in Figure 2 and, thus obtaining the marginal covergae guarantee on future test sets [3].

Algorithm 3 RAPS Prediction Sets

Input: α, sorted scores s and the associated permutation of classes I for a test-time example, $\hat{\tau}_{ccal}$ from Algorithm 2, k_{reg}, λ, boolean $rand$

procedure RAPS($\alpha, s, I, \hat{\tau}_{ccal}, k_{reg}, \lambda, rand$)
  $L \leftarrow |\{j \in \mathcal{Y} : \Sigma_{i=1}^{j} s_i + \lambda(j - k_{reg})^+ \leq \hat{\tau}_{ccal}\}| + 1$
  if $rand$ **then**
    $U \leftarrow \text{Unif}(0,1)$
    $L \leftarrow L - \mathbb{I}\{(\Sigma_{i=1}^{L} s_i + \lambda(L - k_{reg})^+ - \hat{\tau}_{ccal})/(s_L + \lambda \mathbb{I}(L > k_{reg})) \leq U\}$
  return $\mathcal{C} = \{I_1, ...I_L\}$ ▷ The L most likely classes

Output: The $1-\alpha$ confidence set, $\mathcal{C}$ ▷ The set in Eq. (4)

Figure 2: RAPS Prediction Sets Algorithm. Figure from [3].

Temperature Scaling Temperature scaling optimizes the logits temperature value $\tau > 0$ to calibrate the confidence of a classification model predictions. The temperature describes the scaling of the logits and is considered as $\tau = 1$ during the training procedure. After the training, the scaling of the logits can reduce over- or underconfidence, by adjusting the absolute difference between the logit values before the softmax operation is applied.

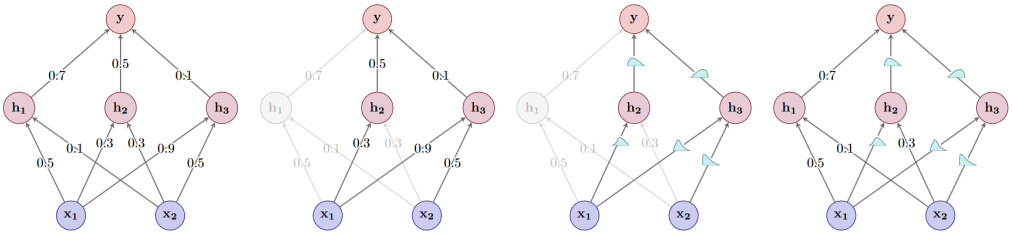
For this we consider a calibration set $D_{\text{cal}} = \{x_m, y_m\}_{m=1}^{M}$ and the cross-entropy loss $\mathcal{L}_{\text{CE}}$, which was also used in the training,

$$\tau^* = \arg\min_{\tau>0} \frac{1}{M} \sum_{(x,y)\in D_{\text{cal}}} \mathcal{L}_{\text{CE}} \left(\text{softmax}\left(\frac{l_\theta(x)}{\tau} \right), y \right) .$$

Depending on the data set size, the temperature scaling is also performed on mini-batches.

2.8 Partially Stochastic Network Strategies

In order to adapt the Bayesian UQ methods to large EO data sets, we support partially stochastic NNs following the approach presented in [54]. In [54] the authors demonstrate experimentally and theoretically that partially stochastic networks can also approximate predictive distributions. There are multiple ways to obtain partially stochastic networks. For the Laplace Approximation and SWAG methods, we use a two-stage training. First, all parameters are obtained by a MAP estimate. Then, in the second training stage the stochastic parameters are obtained. For BNN with VI ELBO and BNN+LV we use joint training, where the stochastic and deterministic parameters are learnt jointly by maximising the evidence lower bound or the so called α-divergence, [11].



(a) Point Estimation (b) Subnetwork Selection (c) Bayesian Inference (d) Prediction

Figure 3: Visualization of partially stochastic networks. Figure from [54].

3 Additional Information on Metrics

Regression tasks are commonly evaluated by accuracy metrics such as Root Mean Squared Error (RMSE) or coefficient of determination, R^2. A better quality of prediction is indicated by a lower RMSE and MAE and a R^2 score close to 1.0. However, these measures only characterize the error between point predictions and available targets. When considering UQ methods, we therefore need additional metrics in the form of proper scoring rules [16] which do not ignore predictive uncertainty. In particular, we consider the negative log-likelihood (NLL) of a Gaussian as a proper scoring rule, [16]. Moreover, we consider calibration as introduced in [31]. As neither the NLL or calibration are sufficient to verify a useful forecast since a model with large predictive uncertainties can be well calibrated and obtain a sufficient NLL, we additionally consider sharpness, which measures the mean of the predictive uncertainties. We use [7] for metric computation and some plots.

The RMSE is computed between the targets $\boldsymbol{y} = (y_i)_{i=1}^N$ and the mean model predictions $\boldsymbol{f}(x) = (f(x_i))_{i=1}^N$ for N samples as

$$\text{RMSE}(\boldsymbol{f}(x), \boldsymbol{y}) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (f(x_i) - y_i)^2}. \tag{110}$$

The MAE is computed as

$$\text{MAE}(\boldsymbol{f}(x), \boldsymbol{y}) = \frac{1}{N} \sum_{i=1}^{N} |f(x_i) - y_i|. \tag{111}$$

The R^2 is computed as

$$R^2 = R^2(\boldsymbol{f}(x), \boldsymbol{y}) = 1 - \frac{\sum_{i=1}^{N}(f(x_i) - y_i)^2}{\sum_{i=1}^{N}\left(f(x_i) - \frac{1}{N}\sum_{j=1}^{N} f(x_j)\right)^2}. \tag{112}$$

However, these measures only characterize the error between point predictions and available targets. In order to compare the predictive uncertainties to the target distribution, we need additional metrics, such as proper scoring rules [16]. We consider the NLL of a Gaussian as a proper scoring rule [16]. We also report the miscalibration area, where a lower miscalibration area indicates a better fit of the predictive uncertainties to the true target distribution. To quantify the overall confidence of a model in a single metric, we consider sharpness which computes the mean of the predictive uncertainties. We use [7] for computing these metrics.

The NLL is computed between the targets $\boldsymbol{y} = (y_i)_{i=1}^N$ and the mean model predictions $\boldsymbol{f}(x) = (f(x_i))_{i=1}^N$ and predictive uncertainties $\boldsymbol{\sigma}(x) = (\sigma(x_i))_{i=1}^N$ for N samples as NLL is computed as

$$\text{NLL}((\boldsymbol{f}(x), \boldsymbol{\sigma}(x)), \boldsymbol{y}) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{2}\ln\left(2\pi\sigma(x_i)^2\right) + \frac{1}{2\sigma(x_i)^2}\left(f(x_i) - y_i\right)^2\right), \tag{113}$$

Additional we consider the scoring rule of the Continuous Ranked Probability Score (CRPS), which for single sample and a predictive distribution that is Gaussian is given by

$$crps(\mathcal{N}(\mu, \sigma), y) = -\sigma\left(\frac{y-\mu}{\sigma}(2\Phi\left(\frac{y-\mu}{\sigma}\right) - 1) + 2\phi\left(\frac{y-\mu}{\sigma}\right) - \frac{1}{\sqrt{\pi}}\right), \tag{114}$$

where Φ is the cumulative density function and ϕ probability distribution of $\mathcal{N}(0, 1)$. Then, we compute the average sum over all predictions and labels, where $f_\theta(x_i^\star) = (\mu(x_i^\star), \sigma(x_i^\star))$, which gives the reported CRPS,

$$CRPS = \frac{1}{N^\star}\sum_{i=1}^{N^\star} crps(f_\theta(x_i^\star), y_i^\star). \tag{115}$$

The miscalibration area is computed based on [7] and is identical to mean absolute calibration error, however the integration here is taken by tracing the area between curves.

The sharpness is computed as

$$\text{sharpness}(\boldsymbol{\sigma}(x)) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\sigma(x_i)^2}. \tag{116}$$

Another key aspect for assessing the reliability of uncertainty estimates is calibration. Calibration refers to the degree to which a predicted distribution matches the true underlying distribution of the data. The mean absolute calibration error, (MACE), gives the mean absolute error of the expected and observed proportions for a given range of quantiles.

References

[1] Moloud Abdar, Farhad Pourpanah, Sadiq Hussain, Dana Rezazadegan, Li Liu, Mohammad Ghavamzadeh, Paul Fieguth, Xiaochun Cao, Abbas Khosravi, U Rajendra Acharya, et al. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information fusion*, 76:243–297, 2021.

[2] Alexander Amini, Wilko Schwarting, Ava Soleimany, and Daniela Rus. Deep evidential regression. *Advances in Neural Information Processing Systems*, 33:14927–14937, 2020.

[3] Anastasios Angelopoulos, Stephen Bates, Jitendra Malik, and Michael I Jordan. Uncertainty sets for image classifiers using conformal prediction. *arXiv preprint arXiv:2009.14193*, 2020.

[4] Anastasios N Angelopoulos and Stephen Bates. A gentle introduction to conformal prediction and distribution-free uncertainty quantification. *arXiv preprint arXiv:2107.07511*, 2021.

[5] Jens Behrmann, Will Grathwohl, Ricky TQ Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. In *International Conference on Machine Learning*, pages 573–582. PMLR, 2019.

[6] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International conference on machine learning*, pages 1613–1622. PMLR, 2015.

[7] Youngseog Chung, Ian Char, Han Guo, Jeff Schneider, and Willie Neiswanger. Uncertainty toolbox: an open-source library for assessing, visualizing, and improving uncertainty quantification. *arXiv preprint arXiv:2109.10254*, 2021.

[8] Erik Daxberger, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig. Laplace redux-effortless bayesian deep learning. *Advances in Neural Information Processing Systems*, 34:20089–20103, 2021.

[9] Erik Daxberger, Eric Nalisnick, James U Allingham, Javier Antorán, and José Miguel Hernández-Lobato. Bayesian deep learning via subnetwork inference. In *International Conference on Machine Learning*, pages 2510–2521. PMLR, 2021.

[10] Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Learning and policy search in stochastic dynamical systems with bayesian neural networks. *arXiv preprint arXiv:1605.07127*, 2016.

[11] Stefan Depeweg, Jose-Miguel Hernandez-Lobato, Finale Doshi-Velez, and Steffen Udluft. Decomposition of uncertainty in bayesian deep learning for efficient and risk-sensitive learning. In *International Conference on Machine Learning*, pages 1184–1193. PMLR, 2018.

[12] Nicolas Dewolf, Bernard De Baets, and Willem Waegeman. Valid prediction intervals for regression problems. *Artificial Intelligence Review*, pages 1–37, 2022.

[13] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[14] Jacob R Gardner, Geoff Pleiss, David Bindel, Kilian Q Weinberger, and Andrew Gordon Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*, 2018.

[15] Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseok Lee, Matthias Humt, Jianxiang Feng, Anna Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, et al. A survey of uncertainty in deep neural networks. *Artificial Intelligence Review*, 56(Suppl 1):1513–1589, 2023.

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

[17] Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael J Cree. Regularisation of neural networks by enforcing lipschitz continuity. *Machine Learning*, 110:393–416, 2021.

[18] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR, 2017.

[19] Fredrik K. Gustafsson, Martin Danelljan, and Thomas B. Schön. How reliable is your regression model's uncertainty under real-world distribution shifts?, 2023.

[20] Xizewen Han, Huangjie Zheng, and Mingyuan Zhou. Card: Classification and regression diffusion models. *Advances in Neural Information Processing Systems*, 35:18100–18115, 2022.

[21] James Harrison, John Willes, and Jasper Snoek. Variational bayesian last layers. In *International Conference on Learning Representations (ICLR)*, 2024.

[22] James Hensman, Alex Matthews, and Zoubin Ghahramani. Scalable variational gaussian process classification. *arXiv preprint arXiv:1411.2005*, 2014.

[23] Jose Hernandez-Lobato, Yingzhen Li, Mark Rowland, Thang Bui, Daniel Hernández-Lobato, and Richard Turner. Black-box alpha divergence minimization. In *International conference on machine learning*, pages 1511–1520. PMLR, 2016.

[24] Jiri Hron, Alexander G de G Matthews, and Zoubin Ghahramani. Variational gaussian dropout is not bayesian. *arXiv preprint arXiv:1711.02989*, 2017.

[25] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

[26] Pavel Izmailov, Sharad Vikram, Matthew D Hoffman, and Andrew Gordon Gordon Wilson. What are bayesian neural network posteriors really like? In *International conference on machine learning*, pages 4629–4640. PMLR, 2021.

[27] Juan Emmanuel Johnson, Valero Laparra, and Gustau Camps-Valls. Accounting for input noise in gaussian process parameter retrieval. *IEEE Geoscience and Remote Sensing Letters*, 17(3):391–395, 2019.

[28] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? *Advances in neural information processing systems*, 30, 2017.

[29] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[30] Roger Koenker and Gilbert Bassett Jr. Regression quantiles. *Econometrica: Journal of the Econometric Society*, pages 33–50, 1978.

[31] Volodymyr Kuleshov, Nathan Fenner, and Stefano Ermon. Accurate uncertainties for deep learning using calibrated regression. In *International conference on machine learning*, pages 2796–2804. PMLR, 2018.

[32] Adrian Lafage and Olivier Laurent. Torch Uncertainty. `https://github.com/ENSTA-U2IS-AI/torch-uncertainty`, 2024.

[33] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

[34] Jeremiah Liu, Zi Lin, Shreyas Padhy, Dustin Tran, Tania Bedrax Weiss, and Balaji Lakshminarayanan. Simple and principled uncertainty estimation with deterministic deep learning via distance awareness. *Advances in Neural Information Processing Systems*, 33:7498–7512, 2020.

[35] Alexander Lyzhov, Yuliya Molchanova, Arsenii Ashukha, Dmitry Molchanov, and Dmitry Vetrov. Greedy policy search: A simple baseline for learnable test-time augmentation. In *Conference on Uncertainty in Artificial Intelligence*, pages 1308–1317. PMLR, 2020.

[36] David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

[37] Wesley J Maddox, Pavel Izmailov, Timur Garipov, Dmitry P Vetrov, and Andrew Gordon Wilson. A simple baseline for bayesian uncertainty in deep learning. *Advances in neural information processing systems*, 32, 2019.

[38] James Martens. New insights and perspectives on the natural gradient method. *The Journal of Machine Learning Research*, 21(1):5776–5851, 2020.

[39] Alexander Graeme de Garis Matthews. *Scalable Gaussian process inference using variational methods*. PhD thesis, University of Cambridge, 2017.

[40] Nis Meinert, Jakob Gawlikowski, and Alexander Lavin. The unreasonable effectiveness of deep evidential regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 9134–9142, 2023.

[41] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

[42] Kevin P Murphy. *Probabilistic machine learning: an introduction*. MIT press, 2022.

[43] Zachary Nado, Neil Band, Mark Collier, Josip Djolonga, Michael W Dusenberry, Sebastian Farquhar, Qixuan Feng, Angelos Filos, Marton Havasi, Rodolphe Jenatton, et al. Uncertainty baselines: Benchmarks for uncertainty & robustness in deep learning. *arXiv preprint arXiv:2106.04015*, 2021.

[44] David A Nix and Andreas S Weigend. Estimating the mean and variance of the target probability distribution. In *Proceedings of 1994 ieee international conference on neural networks (ICNN'94)*, volume 1, pages 55–60. IEEE, 1994.

[45] Ian Osband. Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout. In *NIPS workshop on bayesian deep learning*, volume 192, 2016.

[46] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, David Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. *Advances in neural information processing systems*, 32, 2019.

[47] Kushagra Pandey, Avideep Mukherjee, Piyush Rai, and Abhishek Kumar. Diffusevae: Efficient, controllable and high-fidelity generation from low-dimensional latents. *arXiv preprint arXiv:2201.00308*, 2022.

[48] Rajesh Ranganath, Sean Gerrish, and David Blei. Black box variational inference. In *Artificial intelligence and statistics*, pages 814–822. PMLR, 2014.

[49] Hippolyt Ritter, Aleksandar Botev, and David Barber. A scalable laplace approximation for neural networks. In *6th International Conference on Learning Representations, ICLR 2018-Conference Track Proceedings*, volume 6. International Conference on Representation Learning, 2018.

[50] Yaniv Romano, Evan Patterson, and Emmanuel Candes. Conformalized quantile regression. *Advances in neural information processing systems*, 32, 2019.

[51] Franko Schmähling, Jörg Martin, and Clemens Elster. A framework for benchmarking uncertainty in deep regression. *Applied Intelligence*, pages 1–14, 2022.

[52] Matthias Seeger. Gaussian processes for machine learning. *International journal of neural systems*, 14(02):69–106, 2004.

[53] Florian Seligmann, Philipp Becker, Michael Volpp, and Gerhard Neumann. Beyond deep ensembles: A large-scale evaluation of bayesian deep learning under distribution shift. *Advances in Neural Information Processing Systems*, 36, 2024.

[54] Mrinank Sharma, Sebastian Farquhar, Eric Nalisnick, and Tom Rainforth. Do bayesian neural networks need to be fully stochastic? In *International Conference on Artificial Intelligence and Statistics*, pages 7694–7722. PMLR, 2023.

[55] Laurens Sluijterman, Eric Cator, and Tom Heskes. Optimal training of mean variance estimation neural networks. *arXiv preprint arXiv:2302.08875*, 2023.

[56] Michalis Titsias. Variational learning of inducing variables in sparse gaussian processes. In *Artificial intelligence and statistics*, pages 567–574. PMLR, 2009.

[57] Joost van Amersfoort, Lewis Smith, Andrew Jesson, Oscar Key, and Yarin Gal. On feature collapse and deep kernel learning for single forward pass uncertainty. *arXiv preprint arXiv:2102.11409*, 2021.

[58] Joost Van Amersfoort, Lewis Smith, Yee Whye Teh, and Yarin Gal. Uncertainty estimation using a single deep deterministic neural network. In *International Conference on Machine Learning*, pages 9690–9700. PMLR, 2020.

[59] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688, 2011.

[60] Andrew G Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. *Advances in neural information processing systems*, 33:4697–4708, 2020.

[61] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial intelligence and statistics*, pages 370–378. PMLR, 2016.