Seminar Paper

Uncertainty Quantification in Neural Networks

Department of Statistics
Ludwig-Maximilians-Universität München

Baisu Zhou

Munich, Feburary 28th, 2025

Submitted in partial fulfillment of the requirements for the degree of B. Sc.
Supervised by Dr. Ludwig Bothmann

Abstract

Quantifying uncertainty in predictions is a way to make neural networks as highly complex black-box models more trustworthy. Following a Bayesian approach, one can turn neural networks into probabilistic models capable of estimating uncertainty along with making predictions. However, tracking uncertainty in neural networks imposes great computational challenges. In this paper, we present the theoretical framework and discuss approaches that make uncertainty-aware deep learning feasible.

Contents

1 Introduction

Deep neural netwrks (DNNs) have demonstrated astonishing capabilites of tackling complex predictive tasks in computer vision, natural language processing, and various other fields of science and engineering. While they are often able to achieve high predictive accuracy given large volumes of training data, DNNs are often untrustworthy in real-world applications.

Recent research has highlighted that noisy, corrupted input data can easily mislead neural networks to make wrong predictions. For instance, covering a part of the input image with noise causes a DNN-based vehicle control system to fail at steering angle prediction, which could put human lives in danger (Loquercio et al., 2020). Even a barely perceptible noise, if designed in a certain way, can completely irritate DNN image classifiers (Moosavi-Dezfooli et al., 2017). In the other extreme, certain noise patterns unrecognizable to humans get classified by neural networks as wild animals or everyday objects, and the predictive probabilities assigned to these false classes are very high (Nguyen et al., 2015). Predictions with unreasonably high confidence are not limited to the case where the input data are fine-tuned noise. Guo et al. (2017) showed that over-confidence is a common phenomenon in DNNs. Apart from the objective shortcomings, how human users perceive neural networks also plays a role. In the clinical context, for example, people tend not to trust neural networks due to their nature of being black-box models (Roy et al., 2019).

One approach to making neural networks more trustworthy is to equip them with uncertainty estimation. Besides the predicted label, the model should report the degree of uncertainty in its prediction. If the potentialy unreliable model is uncertain, a reliable human expert can be called to intervene.

In Section 2, we present the Bayesian framework for uncertainty quantification in neural networks. After setting up this theoretical foundation, we introduce three popular uncertainty quantification methods. In Section 3, we derive Laplace approximation and present some of its extensions. In Section 4, we motivate the use of ensemble methods for uncertainty quantification from a Bayesian perspective. In Section 5, we introduce Monte Carlo Dropout, which can be motivated as an ensemble method but enjoys a Bayesian justification. Finally, we apply these methods on toy experiments in Section 6.

2 Bayesian Deep Learning

Adopting a Bayesian view, we use probability distributions to capture uncertainty. This section delves into setting up the framework of Bayesian inference in the context of deep learning.

2.1 Neural Networks as Probabilistic Models

A neural network is a parametric function $f(\,\cdot\,;\boldsymbol{\theta})$ mapping feature objects $x \in \mathcal{X}$ to target values or labels $f(x;\boldsymbol{\theta}) \in \mathcal{Y}$, where $\boldsymbol{\theta}$ is a typically high-dimensional parameter vector. In preparation for a Bayesian interpretation, we view neural network from a probabilistic modeling perspective and establish the equivalence between empirical risk minimization and maximum likelihood estimation.

Let $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ be a training set. We assume the training observations to be i.i.d. For a regression task, in which the network is intended to predict scalar labels, we make the distributional assumption

$$y_i \,|\, \boldsymbol{x}_i \overset{iid}{\sim} \mathcal{N}(f(\boldsymbol{x}_i;\boldsymbol{\theta}); \sigma^2).$$

The negative log-likelihood on $\mathcal{D}$ is given by

$$-\log p(\mathcal{D}\,|\,\boldsymbol{\theta}) = \sum_{i=1}^{n} -\log p(y_i\,|\,\boldsymbol{x}_i,\boldsymbol{\theta}) \propto \sum_{i=1}^{n} (y_i - f(\boldsymbol{x}_i;\boldsymbol{\theta}))^2,$$

which can be interpreted an empirical risk w.r.t. the square loss. In this approach, the variance parameter σ^2 is assumed to be a constant and ignored in the optimization procedure, although the conditional variance can be seen as a natural measure of uncertainty for the target variable. To use this measure, we can predict σ^2 from training data. Accounting for potential heteroskedasticity, we use a network with two outputs: one for the predicted mean $\mu_i \coloneqq \mu(\boldsymbol{x}_i;\boldsymbol{\theta})$, the other for the predicted variance $\sigma_i^2 \coloneqq \sigma^2(\boldsymbol{x}_i;\boldsymbol{\theta})$. The distributional assumption becomes

$$y_i \,|\, \boldsymbol{x}_i \overset{iid}{\sim} \mathcal{N}(\mu(\boldsymbol{x}_i;\boldsymbol{\theta}); \sigma^2(\boldsymbol{x}_i;\boldsymbol{\theta})).$$

The corresponding negative log-likelihood is a sum of logarithms of Gaussian densities.

For a binary classification task, the network outputs a predictive probability for the positive class. The only appropriate distributional assumption is a Bernoulli distribution

$$y_i \,|\, \boldsymbol{x}_i \overset{iid}{\sim} \mathrm{Ber}(f(\boldsymbol{x}_i;\boldsymbol{\theta})).$$

Then the negative log-likelihood on $\mathcal{D}$ amounts to an empirical risk w.r.t. the Bernoulli loss

$$-\log p(\mathcal{D}\,|\,\boldsymbol{\theta}) = \sum_{i=1}^{n} -y_i \log f(\boldsymbol{x}_i;\boldsymbol{\theta}) - (1-y_i)\log(1 - f(\boldsymbol{x}_i;\boldsymbol{\theta})).$$

For a multiclass classification task, the network outputs a probability vector of K entries. Each entry represents the predictive probability for one of the K classes. We consider each label y_i to be a class index, i.e., $y_i \in \{1, \dots, K\}$, and assume the conditional distribution of y_i given $\boldsymbol{x}_i$ to be a categorical distribution

$$y_i \mid \boldsymbol{x}_i \overset{iid}{\sim} \mathrm{Cat}(f(\boldsymbol{x}_i; \boldsymbol{\theta})),$$

which yields

$$-\log p(\mathcal{D} \mid \boldsymbol{\theta}) = \sum_{i=1}^{n} \sum_{k=1}^{K} -\mathbb{I}(y_i = k) \log f(\boldsymbol{x}_i; \boldsymbol{\theta}),$$

where $\mathbb{I}(\,\cdot\,)$ denotes the indicator function. This negative log-likelihood corresponds to an empirical risk w.r.t. the cross entropy loss.

Defining the empirical risk as

$$R_{\mathrm{emp}}(\boldsymbol{\theta}) \coloneqq -\log p(\mathcal{D} \mid \boldsymbol{\theta}) = \sum_{i=1}^{n} \mathcal{L}(y_i, f(\boldsymbol{x}_i; \boldsymbol{\theta})),$$

where $\mathcal{L}$ is a loss function derived from the negative log-likelihood, we see that the empirical risk minimizer

$$\hat{\boldsymbol{\theta}} \coloneqq \arg\min_{\boldsymbol{\theta}} R_{\mathrm{emp}}(\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \log p(\mathcal{D} \mid \boldsymbol{\theta})$$

is identical to the maximum likelihood estimator of $\boldsymbol{\theta}$. In practice, *gradient descent* is used to solve the minimization problem. The algorithm starts at a random initial point $\boldsymbol{\theta}^{(0)}$ in the parameter space and iteratively computes

$$\boldsymbol{\theta}^{(t+1)} \coloneqq \boldsymbol{\theta}^{(t)} - \eta \nabla R_{\mathrm{emp}}\left(\boldsymbol{\theta}^{(t)}\right) = \boldsymbol{\theta}^{(t)} - \eta \nabla \sum_{i=1}^{n} \mathcal{L}(y_i, f(\boldsymbol{x}_i; \boldsymbol{\theta}))$$

for $t = 1, 2, \dots$ until convergence. The hyperparameter $\eta > 0$ is called the learning rate, which may also depend on t. If the training set is large, it is too expensive to re-evaluate the gradient $\nabla R_{\mathrm{emp}}(\boldsymbol{\theta}^{(t)})$ in every iteration t. For computational efficiency, we randomly sample a *mini-batch* $\mathcal{B} \subseteq \mathcal{D}$ of training data and estimate the gradient by

$$\frac{n}{|\mathcal{B}|} \sum_{(\boldsymbol{x}, y) \in \mathcal{B}} \ell(\boldsymbol{x}, y).$$

In every iteration, we generate a new random mini-batch, estimate the loss gradient, and use this estimate to perform gradient descent. This training procedure is called *stochastic gradient descent (SGD).*

2.2 Regularization and MAP Estimation

As highly complex functions, neural networks are prone to overfitting. For higher generalizability, regularization is necessary. L2 regularization is commonly used in deep learning. Instead of the empirical risk, we now minimize the *regularized risk*

$$R_{\mathrm{reg}}(\boldsymbol{\theta}) \coloneqq R_{\mathrm{emp}}(\boldsymbol{\theta}) + \frac{\lambda}{2} \|\boldsymbol{\theta}\|^2,$$

where $\|\cdot\|$ denotes the Euclidean norm and $\lambda > 0$ is a hyperparameter parameter controling the strength of regularization[1]. In practice, L2 regularization is implemented efficiently as *weight decay*. Note that

$$\nabla R_{\text{reg}}(\boldsymbol{\theta}) = \nabla R_{\text{emp}}(\boldsymbol{\theta}) + \lambda\boldsymbol{\theta}.$$

When performing gradient descent under L2 regularization, the update rule becomes

$$\begin{aligned}\boldsymbol{\theta}^{(t+1)} &= \boldsymbol{\theta}^{(t)} - \eta\nabla R_{\text{reg}}\left(\boldsymbol{\theta}^{(t)}\right)\\ &= \boldsymbol{\theta}^{(t)} - \eta\left(\nabla R_{\text{emp}}\left(\boldsymbol{\theta}^{(t)}\right) + \lambda\boldsymbol{\theta}^{(t)}\right)\\ &= \boldsymbol{\theta}^{(t)} - \eta\nabla R_{\text{emp}}(\boldsymbol{\theta}) - \eta\lambda\boldsymbol{\theta}^{(t)}\\ &= (1-\eta\lambda)\boldsymbol{\theta}^{(t)} - \eta\nabla R_{\text{emp}}\left(\boldsymbol{\theta}^{(t)}\right).\end{aligned}$$

Given that $\eta\lambda < 1$, the parameters first decay by a factor $(1-\eta\lambda)$ before the usual gradient step, hence the name "weight decay."

Now we show the equivalence between L2 regularization or weight decay and Bayesian *maximum a posteriori (MAP)* estimation with a Gaussian prior $\boldsymbol{\theta} \sim \mathcal{N}(0, \tau^2 I)$. Suppose $R_{\text{emp}}(\boldsymbol{\theta}) = -\log p(\mathcal{D} \mid \boldsymbol{\theta})$. The parameter estimate under L2 regularization is given by

$$\hat{\boldsymbol{\theta}}_{\text{L2}} := \arg\min_{\boldsymbol{\theta}}\left(R_{\text{emp}}(\boldsymbol{\theta}) + \frac{\lambda}{2}\|\boldsymbol{\theta}\|^2\right) = \arg\min_{\boldsymbol{\theta}}\left(-\log p(\mathcal{D} \mid \boldsymbol{\theta}) + \frac{\lambda}{2}\|\boldsymbol{\theta}\|^2\right). \tag{1}$$

Under the prior assumption $\boldsymbol{\theta} \sim \mathcal{N}(0, \tau^2 I)$, the MAP estimate or *posterior mode* is given by

$$\begin{aligned}\hat{\boldsymbol{\theta}}_{\text{MAP}} &:= \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta} \mid \mathcal{D})\\ &= \arg\max_{\boldsymbol{\theta}} \left(\log p(\mathcal{D} \mid \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})\right)\\ &= \arg\max_{\boldsymbol{\theta}} \left(\log p(\mathcal{D} \mid \boldsymbol{\theta}) - \frac{1}{2\tau^2}\|\boldsymbol{\theta}\|^2\right)\\ &= \arg\min_{\boldsymbol{\theta}} \left(-\log p(\mathcal{D} \mid \boldsymbol{\theta}) + \frac{1}{2\tau^2}\|\boldsymbol{\theta}\|^2\right).\end{aligned}$$

Setting $\tau^2 = \lambda^{-1}$ gives exactly (1). Therefore, if a network is trained using weight decay, we can interpret the parameter estimate as an MAP estimate.

2.3 Predictive Distribution

Given a single MAP estimate $\hat{\boldsymbol{\theta}}$ and a new feature vector $\boldsymbol{x}_*$, one could directly use the likelihood $p(y_* \mid \boldsymbol{x}_*, \hat{\boldsymbol{\theta}})$ to express uncertainty about the predicted target y_*. From a Bayesian perspective, however, we know that the MAP estimate is merely a single point from the posterior. A more reliable representation of the predictive uncertainty should take all information from the posterior into account. This motivates the *predictive distribution*

$$p(y_* \mid \boldsymbol{x}_*, \mathcal{D}) := \int p(y_* \mid \boldsymbol{x}_*, \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid \mathcal{D})\, \mathrm{d}\boldsymbol{\theta} = \mathbb{E}_{\boldsymbol{\theta}\sim p(\boldsymbol{\theta} \mid \mathcal{D})}[p(y_* \mid \boldsymbol{x}_*, \boldsymbol{\theta})]. \tag{2}$$

[1]Scaling λ by $1/2$ simplifies the gradient.

The integral is also called *Bayesian model averaging* (Wilson and Izmailov, 2020). Instead of focusing on a single point estimate, the predictive distribution averages over the entire posterior. Since the posterior distribution is tractable, the true predictive distribution is generally unknown. Even if an approximation of the posterior exists, the integral in (2) is typically intractable (Wilson and Izmailov, 2020, Gawlikowski et al., 2023). A generic approach to approximating this integral is *Monte Carlo integration.* The idea is to approximate the expectation by a sample mean

$$p(y_* \,|\, \boldsymbol{x}_*, \mathcal{D}) = \mathbb{E}_{\boldsymbol{\theta} \sim p(\boldsymbol{\theta} \,|\, \mathcal{D})}[p(y_* \,|\, \boldsymbol{x}_*, \boldsymbol{\theta})] \approx \frac{1}{m} \sum_{i=1}^{m} p(y_* \,|\, \boldsymbol{x}_*, \hat{\boldsymbol{\theta}}_i),$$

where $\hat{\boldsymbol{\theta}}_1, \ldots, \hat{\boldsymbol{\theta}}_m$ are an i.i.d. sample drawn from the posterior or, in practice, an approximation thereof.

In classification, Monte Carlo integration amounts to averaging the predicted probabilities over a sample of parameters. In regression, if the quadratic loss is used and error variance is ignored in training, we do not have the likelihood $p(y_* \,|\, \boldsymbol{x}_*, \hat{\boldsymbol{\theta}})$ fully specified but only know the mean. Then we can estimate the expectation and variance of the predictive distribution using sample mean and sample variance of predicted labels over the Monte Carlo sample of parameters. If error variance is included as a second output of the network, the Monte Carlo sum yields a mixture of Gaussian distributions with mean and variance given by

$$\begin{aligned}
\hat{\mathbb{E}}[y_* \,|\, \boldsymbol{x}_*, \mathcal{D}] &\coloneqq \frac{1}{m} \sum_{i=1}^{m} \mu(\boldsymbol{x}_*; \hat{\boldsymbol{\theta}}_i), \\
\hat{\mathbb{V}}[y_* \,|\, \boldsymbol{x}_*, \mathcal{D}] &\coloneqq \frac{1}{m} \sum_{i=1}^{m} \Big(\sigma^2(\boldsymbol{x}_*; \hat{\boldsymbol{\theta}}_i) + \mu(\boldsymbol{x}; \hat{\boldsymbol{\theta}}_i)^2 \Big) - \hat{\mathbb{E}}[y_* \,|\, \boldsymbol{x}_*, \mathcal{D}]^2,
\end{aligned}$$

where $\mu(\boldsymbol{x}; \boldsymbol{\theta})$ and $\sigma^2(\boldsymbol{x}; \boldsymbol{\theta})$ denote the predicted mean and variance, respectively (Lakshminarayanan et al., 2017).

The mean and variance of the predictive distribution can be used to construct error bands for regression models to visualize uncertainty. For classification, the predictive probabilities already serve as uncertainty measures, while specific summary statistics can be used to capture specific types of uncertainty (Gawlikowski et al., 2023).

2.4 Aleatoric and Epistemic Uncertainty

The form of the predictive distribution (2) motivates a dichotomy of sources of predictive uncertainty into two categories: *data uncertainty*, captured by the likelihood term $p(y_* \,|\, \boldsymbol{x}_*, \boldsymbol{\theta})$, and *model uncertainty*, captured by the posterior $p(\boldsymbol{\theta} \,|\, \mathcal{D})$ (Malinin and Gales, 2018).

Data uncertainty, also called *aleatoric uncertainty*, arises from the inherent stochasticity of the data-generating process. In other words, the relationship between features and target is non-deterministic (Hüllermeier and Waegeman, 2021). This is, e.g., due to random noise in measurements or human-labeling errors (Gawlikowski et al., 2023). Model uncertainty, also called *epistemic uncertainty*, arises from lack of knowledge (Hüllermeier and Waegeman, 2021). On the one hand, multiple parameter settings lead to comparably good performance on the training set, but which one to choose is unclear; On the other hand, we do not know which model architecture is the most appropriate for the task at hand, giving rise to structure uncertainty (Gal, 2016). The prior $p(\boldsymbol{\theta})$ reflects structure uncertainty, as its dispersion is linked to the strength of regularization used in training. A weaker regularization corresponds to a wider prior and higher structure uncertainty.

Sometimes, aleatoric uncertainty is characterized as irreducible, while epistemic uncertainty is considered reducible (Hüllermeier and Waegeman, 2021). This terminology is questionable. For example, we can reduce aleatoric uncertainty due to measurement noise by using more precise measurement devices (Gal, 2016). For example, if we are asked to predict a person's body height based on their gender, the aleatoric uncertainty is high, since people of the same gender vary a lot in height. However, as we gain more information about the person, such as body weight, age, etc., there are fewer and fewer possible individuals with the same traits. By gaining knowledge, we reduce aleatoric uncertainty. This example also illustrates that aleatoric and epistemic uncertainty are context-specific notions. As pointed out by Hüllermeier and Waegeman (2021), categorizing uncertainty components into these two types is only possible with respect to a specific data-generating process and model class.

3 Laplace Approximation

Laplace approximation (LA) is a post-hoc method for analytically approximating the posterior of parameters (Daxberger, Kristiadi, Immer, Eschenhagen, Bauer and Hennig, 2021). Interpreting learned parameters as MAP estimates, we can apply Laplace approximation to neural networks of arbitrary architectures.

3.1 Standard Laplace Approximation

Let $\hat{\boldsymbol{\theta}}_{\text{MAP}}$ be the MAP estimate of the network parameters $\boldsymbol{\theta}$. Suppose that $\boldsymbol{\theta} \in \mathbb{R}^d$. A second-order Taylor approximation of the log-posterior about $\hat{\boldsymbol{\theta}}_{\text{MAP}}$ yields

$$\begin{aligned}\log p(\boldsymbol{\theta} \mid \mathcal{D}) \approx \log p(\hat{\boldsymbol{\theta}}_{\text{MAP}} \mid \mathcal{D}) \\ &+ (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_{\text{MAP}})^\top \nabla_{\boldsymbol{\theta}} \log p(\hat{\boldsymbol{\theta}}_{\text{MAP}} \mid \mathcal{D}) \\ &+ \frac{1}{2}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_{\text{MAP}})^\top \mathbf{H} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_{\text{MAP}}),\end{aligned}$$

where

$$\mathbf{H} = \left(\frac{\partial^2 \log p(\hat{\boldsymbol{\theta}}_{\text{MAP}} \mid \mathcal{D})}{\partial \boldsymbol{\theta}_i \, \partial \boldsymbol{\theta}_j} \right)_{i,j=1}^{d} \tag{3}$$

denotes the Hessian matrix of the log-posterior evaluated at the MAP estimate. Since the gradient at the maximum is zero, the first-order term vanishes and we obtain

$$\log p(\boldsymbol{\theta} \mid \mathcal{D}) \approx \log p(\hat{\boldsymbol{\theta}}_{\text{MAP}} \mid \mathcal{D}) + \frac{1}{2}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_{\text{MAP}})^\top \mathbf{H} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_{\text{MAP}}).$$

Exponentiating both sides yields

$$\begin{aligned} p(\boldsymbol{\theta} \mid \mathcal{D}) &\approx p(\hat{\boldsymbol{\theta}}_{\text{MAP}} \mid \mathcal{D}) \exp\left(\frac{1}{2}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_{\text{MAP}})^\top H (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_{\text{MAP}}) \right) \\ &\propto \exp\left(-\frac{1}{2}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_{\text{MAP}})^\top (-\mathbf{H}) (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_{\text{MAP}}) \right). \end{aligned}$$

The posterior density as a function of $\boldsymbol{\theta}$ is approximately proportional to an exponential of a quadratic form in $\boldsymbol{\theta}$. Thus,

$$p(\boldsymbol{\theta} \mid \mathcal{D}) \approx \mathcal{N}\left(\boldsymbol{\theta}; \hat{\boldsymbol{\theta}}_{\text{MAP}}, (-\mathbf{H})^{-1} \right). \tag{4}$$

Using this posterior approximation, we can approximate the predictive distribution via Monte Carlo integration. The idea to approximate an expectation by a sample mean:

$$p(y_* \mid \boldsymbol{x}_*, \mathcal{D}) = \mathbb{E}_{\boldsymbol{\theta} \sim p(\boldsymbol{\theta} \mid \mathcal{D})}[p(y_* \mid \boldsymbol{x}_*, \boldsymbol{\theta})] \approx \frac{1}{m} \sum_{k=1}^{m} p(y_* \mid \boldsymbol{x}_*, \boldsymbol{\theta}_k), \quad \boldsymbol{\theta}_i \overset{iid}{\sim} \mathcal{N}(\hat{\boldsymbol{\theta}}_{\text{MAP}}, (-\mathbf{H})^{-1}).$$

Note that LA depends on the prior distribution, but only through the Hessian matrix. Assuming the network is trained via standard SGD with weight decay, we obtain the log-posterior

$$\log p(\boldsymbol{\theta} \mid \mathcal{D}) = \log p(\mathcal{D} \mid \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) + \text{const.} = \log p(\mathcal{D} \mid \boldsymbol{\theta}) - \frac{1}{2\tau^2} \|\boldsymbol{\theta}\|^2 + \text{const.}$$

Then the Hessian at the MAP estimate is given by

$$\mathbf{H} = \frac{\partial^2 \log p(\mathcal{D} \mid \hat{\boldsymbol{\theta}}_{\mathrm{MAP}})}{\partial \boldsymbol{\theta} \, \partial \boldsymbol{\theta}^\top} - \tau^{-2}\mathbf{I}, \tag{5}$$

where $\mathbf{I}$ denotes the identity matrix. In theory, the prior precision τ^{-2} should be chosen according to the weight decay constant. In practice, however, τ^{-2} is considered a separate hyperparameter (Ritter et al., 2018, Kristiadi et al., 2020, Immer, Korzepa and Bauer, 2021) and tuned on observed data, e.g., by maximizing the predictive log-likelihood on a validation set (Ritter et al., 2018) or applying the empirical Bayes method (Immer, Bauer, Fortuin, Rätsch and Emtiyaz, 2021).

Note that the Hessian matrix contains $d \times d$ entries, where d is the number of parameters. For a large neural network, Hessian computation is very expensive. We must resort to approximation methods (Daxberger, Kristiadi, Immer, Eschenhagen, Bauer and Hennig, 2021), and we emphasize that such approximation methods are not trivial to implement. The datails are beyond the scope of this paper.

3.2 Linearized Laplace Approximation

Since the approximate posterior is Gaussian, we can exploit properties of Gaussian random variables to develop approximation methods for the predictive distribution other than Monte Carlo integration.

Consider the neural network as a composition $f(x; \boldsymbol{\theta}) = g(h(x; \boldsymbol{\theta}))$ of two functions h and g, where $h(\,\cdot\,; \boldsymbol{\theta}) : \mathcal{X} \longrightarrow \mathbb{R}^K$ maps the input space to the last-layer output space $\mathbb{R}^K$ and g transforms the last-layer output into predicted labels or probabilities. For example, for multiclass classification g is the softmax function. Given a fixed input object $\boldsymbol{x}_*$, the last-layer output $\boldsymbol{h}_* := h(\boldsymbol{x}_*; \boldsymbol{\theta})$ can be seen as a function of the parameters $\boldsymbol{\theta}$. We linearize this function about $\hat{\boldsymbol{\theta}}_{\mathrm{MAP}}$ as

$$\boldsymbol{h}_* = h(\boldsymbol{x}_*; \boldsymbol{\theta}) \approx h(\boldsymbol{x}_*; \hat{\boldsymbol{\theta}}_{\mathrm{MAP}}) + J(\hat{\boldsymbol{\theta}}_{\mathrm{MAP}}; \boldsymbol{x}_*)(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_{\mathrm{MAP}}), \tag{6}$$

where $J(\hat{\boldsymbol{\theta}}_{\mathrm{MAP}}; \boldsymbol{x}_*)$ denotes the Jacobian matrix of the function $\boldsymbol{\theta} \mapsto h(\boldsymbol{x}_*; \boldsymbol{\theta})$ evaluated at the MAP estimate. Since $p(\boldsymbol{\theta} \mid \mathcal{D}) \approx \mathcal{N}(\boldsymbol{\theta}; \hat{\boldsymbol{\theta}}_{\mathrm{MAP}}, (-\mathbf{H})^{-1})$,

$$p(\boldsymbol{h}_* \mid \boldsymbol{x}_*, \mathcal{D}) \approx \mathcal{N}\Big(\boldsymbol{h}_*; h(\boldsymbol{x}_*; \hat{\boldsymbol{\theta}}_{\mathrm{MAP}}), \mathbf{J}(\hat{\boldsymbol{\theta}}_{\mathrm{MAP}}; \boldsymbol{x}_*)(-\mathbf{H})^{-1}\mathbf{J}(\hat{\boldsymbol{\theta}}_{\mathrm{MAP}}; \boldsymbol{x}_*)^\top\Big). \tag{7}$$

This is the *linearized Laplace approximation* (LLA) (Immer, Korzepa and Bauer, 2021). In a regression setting where the network outputs a single scalar, the last-layer output $\boldsymbol{h}_*$ directly is the predicted label, i.e., $g = \mathrm{id}$ and $f(\boldsymbol{x}_*; \boldsymbol{\theta}) = h(\boldsymbol{x}_*; \boldsymbol{\theta}) = \boldsymbol{h}_*$. That means the predictive distribution can be approximated by

$$p(y_* \mid \boldsymbol{x}_*, \mathcal{D}) \approx \mathcal{N}\Big(y_*; f(\boldsymbol{x}_*; \hat{\boldsymbol{\theta}}_{\mathrm{MAP}}), \mathbf{J}(\hat{\boldsymbol{\theta}}_{\mathrm{MAP}}; \boldsymbol{x}_*)(-\mathbf{H})^{-1}\mathbf{J}(\hat{\boldsymbol{\theta}}_{\mathrm{MAP}}; \boldsymbol{x}_*)^\top\Big).$$

For classification, $\boldsymbol{h}_*$ is transformed by a sigmoid or softmax function g into class probabilities. To obtain the predictive distribution, we need to solve the integral

$$\begin{aligned} p(y_* \mid \boldsymbol{x}_*, \mathcal{D}) &= \int p(y_* \mid \boldsymbol{h}_*) p(\boldsymbol{h}_* \mid \boldsymbol{x}_*, \mathcal{D}) \, \mathrm{d}\boldsymbol{h}_* \\ &\approx \int p(y_* \mid \boldsymbol{h}_*) \mathcal{N}\Big(\boldsymbol{h}_*; h(\boldsymbol{x}_*; \hat{\boldsymbol{\theta}}_{\mathrm{MAP}}), \mathbf{J}(\hat{\boldsymbol{\theta}}_{\mathrm{MAP}}; \boldsymbol{x}_*)(-\mathbf{H})^{-1}\mathbf{J}(\hat{\boldsymbol{\theta}}_{\mathrm{MAP}}; \boldsymbol{x}_*)^\top\Big) \, \mathrm{d}\boldsymbol{h}_*. \end{aligned} \tag{8}$$

For binary classification with $y_* \in \{0, 1\}$, $p(y_* \mid \boldsymbol{h}_*) = \mathrm{Ber}(y_*; \mathrm{sigmoid}(\boldsymbol{h}_*))$. For multiclass classification with $y_* \in \{1, \dots, K\}$, $p(y_* \mid \boldsymbol{h}_*) = \mathrm{Cat}(y_*; \mathrm{softmax}(\boldsymbol{h}_*))$. In these two cases, the integral (8) is intractable. One approximation method is again Monte Carlo integration. For the Bernoulli likelihood, probit approximation (Spiegelhalter and Lauritzen, 1990, MacKay, 1992) is an alternative. For the categorical likelihood, extended probit approximation (Gibbs, 1997) or Laplace bridge (Hobbhahn et al., 2022) can be used.

In comparison to standard LA with Monte Carlo integration, LLA with the aforementioned approximation methods is free of sampling, but requires evaluating the Jacobian matrix once for each prediction. Although linearization seems to incur additional error, LLA is often preferrable to standard LA with Monte Carlo integration. The latter approach tends to overestimate predictive uncertainty, while the former yields better uncertainty estimates Immer, Korzepa and Bauer (2021). Moreover, Kristiadi et al. (2020) justifies LLA theoretically by showing its capability of mitigating the overconfidence problem in ReLU classification networks.

3.3 Last-layer Laplace Approximation

LLA faces the same challenge of Hessian computation as standard Laplace approximation. Daxberger, Nalisnick, Allingham, Antoran and Hernandez-Lobato (2021) propose performing Laplace approximation on a sub-network and show that more refined Hessian approximation on a small sub-network outperforms oversimplified Hessian approximation on the whole network. As a special case of sub-network Laplace approximation, applying Laplace approximation only to the last-layer weights and biases not only shows competitive performance to all-layer Laplace approximation (Daxberger, Kristiadi, Immer, Eschenhagen, Bauer and Hennig, 2021), but also shares LLA's nice theoretical properties (Kristiadi et al., 2020).

Last-layer Laplace approximation is particularly efficient. Let $\mathbf{W}$ and $\boldsymbol{b}$ be the weight matrix and bias vector of the last layer, and ϕ be the network up to the second-last layer. Then the last-layer output can be written as

$$\boldsymbol{h}_* = \mathbf{W}\phi(\boldsymbol{x}_*) + \boldsymbol{b}.$$

Using Laplace approximation, the posterior of $\mathbf{W}$ and $\boldsymbol{b}$ are approximated by Gaussians. Since $\boldsymbol{h}_*$ is linear in $\mathbf{W}$ and $\boldsymbol{b}$, it also approximately follows a Gaussian distribution (Eschenhagen et al., 2021). To approximate the predictive distribution, we can therefore employ the same direct approximation methods as for LLA instead of sampling-based Monte Carlo integration. Furthermore, Hessian computation is drastically simplified due to the much smaller number of parameters involved.

4 Ensembles Methods

The loss surface of deep neural networks are non-convex. In particular, there are multiple local minima Li et al. (2018). In light of the equivalence between regularized risk minimization and MAP estimation, this means the posterior of parameters possesses multiple modes. LA approximation does not account for this, but only locally approximate the posterior around a single mode. In constrast, *Deep Ensembles* (Lakshminarayanan et al., 2017) provide a multimodal view of the posterior.

4.1 Deep Ensembles

The multiple modes are obtained by training an ensemble of networks of the same architecture on the same dataset. To enable multimodal posterior approximation, we want different ensemble members to discover different modes. For a neural network ensemble, Lakshminarayanan et al. (2017) argue that random weight initialization and batch generation suffice as sources of variability. By constrast, the traditional ensembling method of bootstrap can even harm performance (Livieris et al., 2021).

Once an ensemble has been trained, we obtain a collection $\hat{\boldsymbol{\theta}}_1, \ldots, \hat{\boldsymbol{\theta}}_m$ of parameter estimates, each of which can be interpreted as a posterior mode. We can interpret the empirical distribution of these parameter estimates as a posterior approximation. That is, the posterior is approximated by a sum of Dirac measures placed at the discovered modes. This approximation does not represent the shape of the posterior around any mode. However, it can be more meaningful than a single-mode alternative such as Laplace approximation, if the true posterior is highly multimodal.

Using all the discovered modes as a sample from the posterior, we approximate the predictive distribution by Monte Carlo integration

$$p(y_* \mid \boldsymbol{x}_*, \mathcal{D}) \approx \frac{1}{m} \sum_{i=1}^{m} p(y_* \mid \boldsymbol{x}_*, \hat{\boldsymbol{\theta}}_i). \tag{9}$$

To implement a Deep Ensemble, we simply train m copies of the same network using different random seeds, and use them jointly for predictions and uncertainty quantification. Since the ensemble members are independently trained, parallelization is possible (Lakshminarayanan et al., 2017). Nevertheless, training m networks is computationally expensive. Moreover, the parameters of m networks must be stored at test time, creating a high storage space demand. These concerns render large ensembles impractical. Lakshminarayanan et al. (2017) suggest using ensembles of 5 or 10 models. Surprisingly, even such small ensembles achieve high performance in predictions and uncertainty quantification.

It is worth mentioning that the Deep Ensemble method was originally proposed as a non-Bayesian approach (Lakshminarayanan et al., 2017). The Bayesian interpretation is due to Wilson and Izmailov (2020).

4.2 Symmetry of Modes

A multimodal approximation of the posterior per se does not guarantee a good quantification of predictive uncertainty. We argue in the following that in the worst case, an ensemble makes no improvement over a single network.

Posterior modes of neural networks exhibits symmetry (Sommer et al., 2024). That is, two different modes $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2$ may correspond to the same function $f(x; \boldsymbol{\theta}_1) = f(x; \boldsymbol{\theta}_2)$ that maps the feature space to the target space. For example, permuation of neurons within each hidden layer or scaling the weights by appropriate constants does not change the output of a fully-connected ReLU network (Grigsby et al., 2023).

Training an ensemble of neural networks with random initialization allows to discover multiple modes, which, however, might parameterize the same function. In the likelihood term $p(y_* \,|\, \boldsymbol{x}_*, \boldsymbol{\theta})$, y_* depends on $\boldsymbol{x}_*$ and $\boldsymbol{\theta}$ through the network output $f(\boldsymbol{x}_*; \boldsymbol{\theta})$. If two parameters correspond to the same function, they yield the same likelihood. In the worst case, the parameters $\hat{\boldsymbol{\theta}}_1, \ldots, \hat{\boldsymbol{\theta}}_m$ of all ensemble members correspond to the same function, so that the ensemble prediction

$$\frac{1}{m} \sum_{i=1}^{m} p(y_* \,|\, \boldsymbol{x}_*, \hat{\boldsymbol{\theta}}_i) = p(y_* \,|\, \boldsymbol{x}_*, \hat{\boldsymbol{\theta}}_1)$$

is no different than that of a single model.

Luckily, the worst case is unlikely. Fort et al. (2020) showed that training ensemble members with random initialization not only leads to different parameter estimates, but also a high functional diversity. Moreover, the authors observed that functions produced by parameters visited in a single optimization process remain similar, but the diversity is high across different training runs.

4.3 Less Expensive Alternatives

Extending standard SGD optimization, Lakshminarayanan et al. (2017) proposed employing adversarial training to improve the robustness of each ensemble member. This happens within the training procedure of each model and does not harm parallelizability.

Furthermore, efforts have been made to lower the computational cost of neural network ensembles. Valdenegro-Toro (2019) proposed *Deep Sub-Ensembles*, in which the ensemble members share the same weights in most layers except the last few. To construct a deep sub-ensemble, we first train a single network as usual, then keep the first layers fixed, and only re-train the last few layers multiple times to obtain the ensemble members. Sharing a larger subnetwork leads to worse prediction performance relative to the standard Deep Ensemble, but the trade-off can be controlled by the user (Valdenegro-Toro, 2019).

BatchEnsemble (Wen et al., 2019) is another approach to simplifying Deep Ensembles. Instead of sharing a subnetwork, this method introduces a layer-wise weight sharing mechanism. Suppose that the weights in a layer can be arranged into an $k \times l$ matrix. Then for each ensemble member $i = 1, \ldots, m$, the weights are generated by an elementwise product of a shared weight matrix $\mathbf{W} \in \mathbb{R}^{k \times l}$ and a member-specific rank one matrix $\mathbf{F}_i = \boldsymbol{s}_i \boldsymbol{r}_i^\top$ with $\boldsymbol{s}_i \in \mathbb{R}^k$ and $\boldsymbol{r}_i \in \mathbb{R}^l$. Without weight sharing, we need to store mkl weights for this layer across the ensemble. With weight sharing, we only need to store $kl + m(k + l)$ weights. The number of stored weights reduces from a cubic function of m, k, l to a quadratic one. The weight sharing mechanism makes parallel training of ensemble members necessary (Gawlikowski et al., 2023). However, this can be carried out per mini-batch without much memory overhead (Wen et al., 2019).

5 Monte Carlo Dropout

Dropout is a regularization technique for multi-layer perceptrons intended to break co-adaptations of neurons and improve prediction performance (Hinton et al., 2012). It can be interpreted as a simplified ensemble method and applied as an efficient uncertainty quantification method.

5.1 Dropout Mechanism

Given the same data and a complex enough network, different weight configurations lead to approximately the same, nearly perfect performance on the training set. For some weight configurations, each neuron extracts information from the input object that is only useful in combination with information provided by several other neurons (Hinton et al., 2012). Srivastava et al. (2014) argued that complex co-adaptations might work well on the training set, but do not generalize well. Therefore, each neuron should be encouraged to learn to extract information that is useful on its own or in a smaller context.

We can break complex co-adaptations by temporarily dropping subsets of neurons from each fully connected layer during training. Suppose the l-th and $(l+1)$-th layers of a network are fully connected layers with D_l and D_{l+1} neurons, respectively. Let $\boldsymbol{z}^{(l-1)} \in \mathbb{R}^{D_{l-1}}$ be the input to the l-th layer. In a forward pass, the two layers compute

$$\begin{aligned}
\boldsymbol{z}^{(l)} &:= g\left(\mathbf{W}^{(l)}\boldsymbol{z}^{(l-1)} + \boldsymbol{b}^{(l)}\right), \\
\boldsymbol{z}^{(l+1)} &:= g\left(\mathbf{W}^{(l+1)}\boldsymbol{z}^{(l)} + \boldsymbol{b}^{(l+1)}\right),
\end{aligned}$$

where $\mathbf{W}^{(l)} \in \mathbb{R}^{D_l \times D_{l-1}}, \boldsymbol{b}^{(l)} \in \mathbb{R}^{D_l}$ and $\mathbf{W}^{(l+1)} \in \mathbb{R}^{D_{l+1} \times D_l}, \boldsymbol{b}^{(l+1)} \in \mathbb{R}^{D_{l+1}}$ denote the weight matrices and bias vectors of the two layers, respectively, and g denotes the activation function. To apply Dropout to the l-th layer, we switch off each neuron in that layer with a probability $\pi \in (0,1)$. This is implemented by multiplying the output vector of l-th layer elementwise with a vector of zeros and ones sampled i.i.d. from the Bernoulli distribution $\mathrm{Ber}(\pi)$. A forward pass with Dropout is as follows:

$$\begin{aligned}
\boldsymbol{z}^{(l)} &:= g\left(\mathbf{W}^{(l)}\boldsymbol{z}^{(l-1)} + \boldsymbol{b}^{(l)}\right), \\
\boldsymbol{r}^{(l)} &:= \left(r_1^{(l)}, \dots, r_{D_l}^{(l)}\right), \quad r_j^{(l)} \overset{iid}{\sim} \mathrm{Ber}(\pi), \\
\tilde{\boldsymbol{z}}^{(l)} &:= \boldsymbol{z}^{(l)} \odot \boldsymbol{r}^{(l)}, \\
\boldsymbol{z}^{(l+1)} &:= g\left(\mathbf{W}^{(l+1)}\tilde{\boldsymbol{z}}^{(l)} + \boldsymbol{b}^{(l+1)}\right),
\end{aligned}$$

where $\odot$ denotes elementwise multiplication. Neurons with output set to zero make no contribution to the following layers, as if they were removed from the network. Therefore, Dropout can be viewed as selecting subnetworks from a full network.

During training, the Dropout vector $\boldsymbol{r}^{(l)}$ is re-sampled for each observation (Srivastava et al., 2014). In other words, a random subnetwork is generated and trained on each observation, while the weights are shared between the subnetworks and continuously updated until convergence. We can interpret Dropout as a way of ensembling (Hinton et al., 2012), which is almost as efficient as training a single network. If n neurons participate in Dropout, a total of 2^n subnetworks can be generated, each of which we can view as an ensemble member. They differ in which neurons from the full network they possess, but the same neuron has the same weights. In practice, not all of the exponentially many ensemble members are actually trained, as the ensemble size is much larger than the typical number of iterations required for SGD to converge.

5.2 Dropout for Uncertainty Quantification

Interpreting Dropout as an ensemble method, we may approximate the predictive distribution following (9). However, averaging over an ensemble of 2^n is infeasible, if the number n of neurons participating in Dropout is large. Thus, we resort to sample-based Monte Carlo integration, where we use Dropout to generate a sufficiently large number of subnetworks, evaluate them on the test instance, and aggregate the predictions. This method is called *Monte Carlo Dropout (MC Dropout).*

Although we motivate MC Dropout as an ensembling method, the MC Dropout method was proposed and justified by Gal and Ghahramani (2016) in a variational inference context. Here we do not present the details of derivation, but only an excerpt of the author's argument which lends some Bayesian flavor to Dropout. The presentation of Dropout in Section 5.1 follows Srivastava et al. (2014), in which the operation of dropping a neuron is implemented by setting output of the neuron to zero. The same can also be achieved by zeroing out weights.

Using the same notation as in Section 5.1, if Dropout is applied to the l-th layer, then the $(l+1)$-layer computes

$$\boldsymbol{z}^{(l+1)} = g\left(\mathbf{W}^{(l+1)}\left(\boldsymbol{z}^{(l)} \odot \boldsymbol{r}^{(l)}\right) + \boldsymbol{b}^{(l+1)}\right) = g\left(\mathbf{W}^{(l+1)} \operatorname{diag}\left(\boldsymbol{r}^{(l)}\right) \boldsymbol{z}^{(l)} + \boldsymbol{b}^{(l+1)}\right),$$

where $\operatorname{diag}(\boldsymbol{r}^{(l)})$ is a diagonal matrix with $r_1^{(l)}, \ldots, r_{D_l}^{(l)}$ on its diagonal. We define

$$\tilde{\mathbf{W}}^{(l+1)} := \mathbf{W}^{(l+1)} \operatorname{diag}\left(\boldsymbol{r}^{(l)}\right).$$

If $r_j^{(l)} = 0$, then all entries in the j-th column of $\tilde{\mathbf{W}}^{(l+1)}$ are zero. Therefore, applying Dropout can be seen as sampling weight matrices from the distribution over matrices with one or more columns set to zero. Using this distribution to construct an approximation to the true posterior, Gal and Ghahramani (2016) showed that minimizing the L_2-regularized risk w.r.t. the square loss is equivalent to minimizing the KL divergence between the approximation and the true posterior.

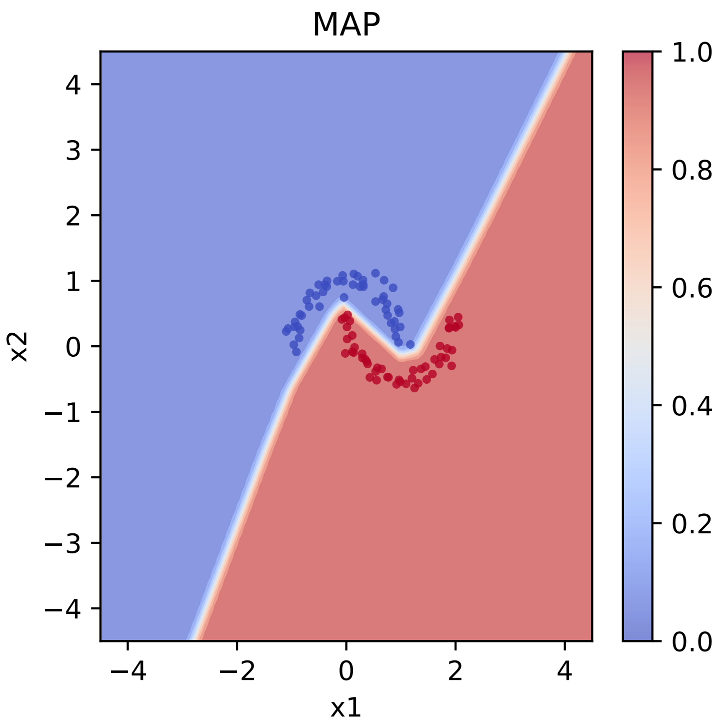

Figure 1: Predictions of a standard MLP. The blue and red points are training data. The color in the background represents the predicted probability $p(y \mid \boldsymbol{x}, \hat{\boldsymbol{\theta}}_{\text{MAP}})$ at each point $\boldsymbol{x} = (x_1, x_2)^\top$ in the feature space.

6 Illustrative Examples

In this section, we apply the three methods for uncertainty quantification on two toy examples to illustrate their properties and make some qualitative comparisons.

6.1 Binary Classification

We train an MLP classifier on the synthetic two-moons dataset from scikit-learn (Pedregosa et al., 2011). The dataset contains two numerical features and a binary target. The network has two-hidden layers with 16 neurons each. The output layer consists of a single neuron. We use ReLU activation and optimize the Bernoulli negative log-likelihood, also known as the binary cross entropy. We train the model for 500 epochs on 128 data points using the SGD optimizer with batch size 2, learning rate 0.5 and weight decay 0.001. The result is shown in Fig. 1. We see that the model is flexible enough to perfectly separate the two classes. However, the predicted probabilities are almost zero or one in most parts of the feature space. Even in regions of no training data, the model makes highly confident predictions. Clearly, the output probabilities of the model themselves do not offer reliable unceratainty quantification.

Since the network is small, we can afford computing the full Hessian matrix for Laplace approximation. In Fig. 2, we demonstrate the sensitivity of this method with respect to

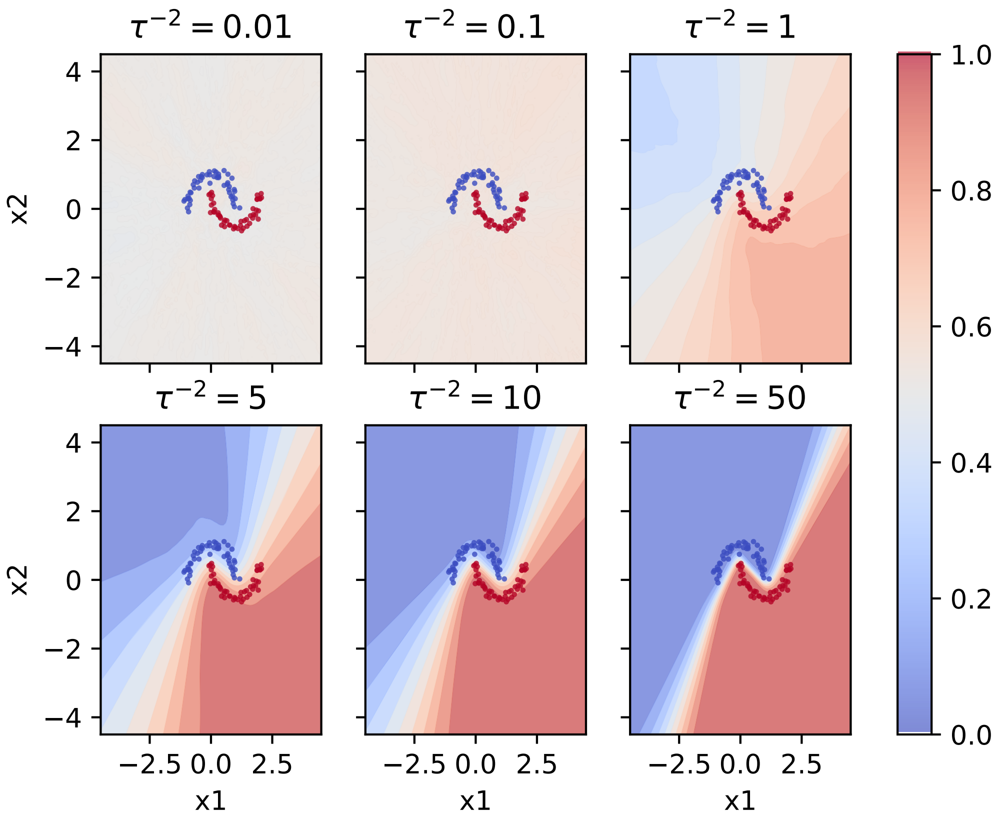

Figure 2: Laplace approximation for different prior precisions τ^{-2}. Monte Carlo integration is used to approximate the predictive distribution. For each test point $(x_1, x_2)^\top$, 1000 parameters are sampled from the Gaussian approximate posterior.

the choice of prior precision τ^{-2}. For a small prior precision or large prior variance, the predictive probabilities are almost 0.5 everywhere. The uncertainty is overestimated. As we increase the prior precision, we can recognize more meaningful structures, and the predictive distribution becomes closer to the MAP result in Fig. 2. A consistent pattern is that the region with predictive probabilities close to 0.5 gets wider as we move away from the training data.

Now we construct an ensemble of 10 models with the same architecture of the MLP. The ensemble members are trained using the same hyperparameter settings, but weight initialization and batch generation are randomized. We plot five ensemble members as well as the ensemsble average in Fig. 3. While each model is highly overconfident, some variation exists between the ensemble members. The predictive distribution given by the ensemble has a similar structure to that given by Laplace approximation with high prior precision. We observe regions of lower confidence expand towards bottom left and top right. However, the issue of overconfidence seems to persist in the other directions.

Finally, we demonstrate Monte Carlo Dropout on the same dataset. We use the same setting to train the same MLP, but now with Dropout applied to the two hidden layers. We set the Dropout rate to $\pi = 0.2$. For such a small network, a high Dropout rate

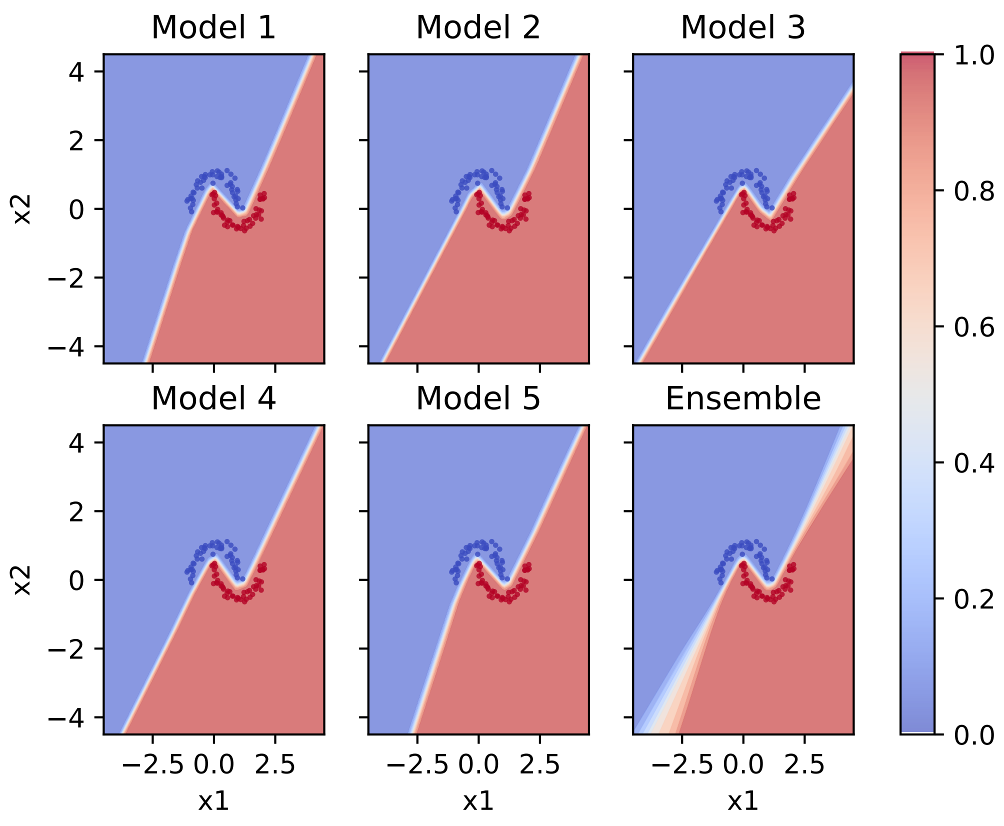

Figure 3: Deep ensemble of 10 models. The first five plots show five individual models. The last plot shows the ensemble average.

harms the prediction performance. In Fig. 4, we observe the same pattern once again: the band of low confidence extends as we move away from data, but larger regions top left and bottom right are still assigned with high confidence. We hypothesize that the unresolvable overconfidence in certain directions is a property of the network we use. Kristiadi et al. (2021) introduced additional hidden units into the network to enhance Laplace approximation. Their approach mitigates this issue on the same dataset.

6.2 Regression

As a toy experiment for regression, we train an MLP with layer dimensions $[1, 16, 16, 1]$ to learn the function $x \mapsto \sin(x)$. We generate $n = 20$ training instances as follows:

$$\begin{aligned} x_i &\overset{iid}{\sim} \text{Uniform}([-3, 3]), \\ \epsilon_i &\overset{iid}{\sim} \mathcal{N}(0, 0.1^2), \\ y_i &:= \sin(x) + \epsilon_i. \end{aligned}$$

We perform no batch generation but use standard gradient descent. The network is optimized with a learning rate of 0.1 for 1000 epochs. The weight decay parameter is set

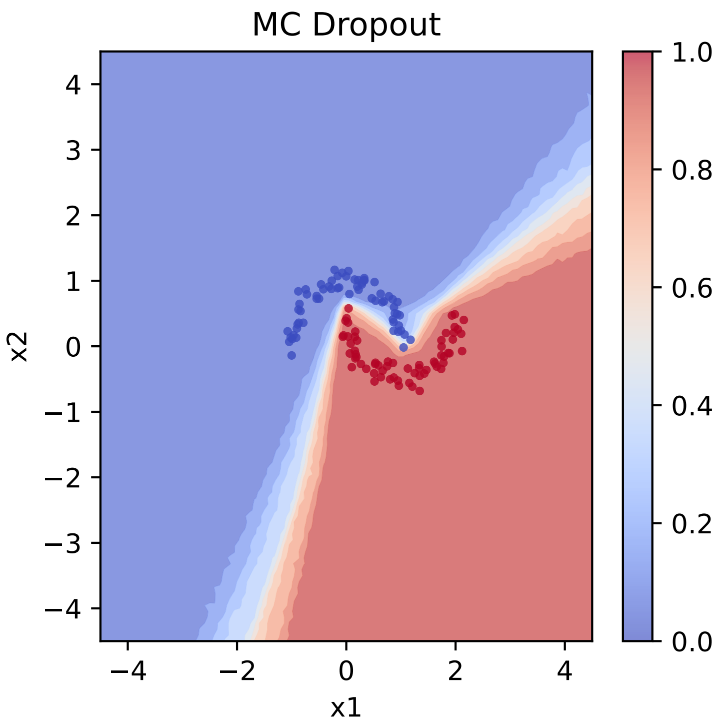

Figure 4: MC Dropout. For each test point $(x_1, x_2)^\top$, a sample of 1000 parameters is used (same as for Laplace approximation).

to 10^{-6}. Fig. 5 presents the trained network (MAP estimate) and the three uncertainty quantification methods. The MAP estimate does not provide any uncertainty quantification, while we can construct error bands using the introduced methods. For Laplace Approximation, we use the last-layer variant with KFAC approximation to the Hessian and optimize the prior precision with respect to the marginal likelihood, as recommended by Daxberger, Kristiadi, Immer, Eschenhagen, Bauer and Hennig (2021). The standard deviation is obtained from the Gaussian approximation to the predictive distribution. The Deep Ensemble consists of 10 models. For MC Dropout, we apply Dropout with rate $\pi = 0.2$ to both hidden layers and use the sample size 1000 for Monte Carlo integration. In the latter two approaches, we estimate the standard deviation using the empirical standard deviation over the ensemble or Monte Carlo sample.

We expect higher uncertainty when moving away from data. This is indeed observed for all three methods. Laplace Approximation is applied in a post-hoc manner. The predictive mean shown by the red curve agrees with the MAP estimate. It successfully captures the increasing uncertainty in extrapolation. Deep Ensemble exhibits higher confidence overall in comparison to Laplace approximation. Lakshminarayanan et al. (2017) suggests letting the model predict the error variance along with the mean, in which case overconfidence is better mitigated. The error band prodced by MC Dropout also expands when extrapolating, although not as quickly.

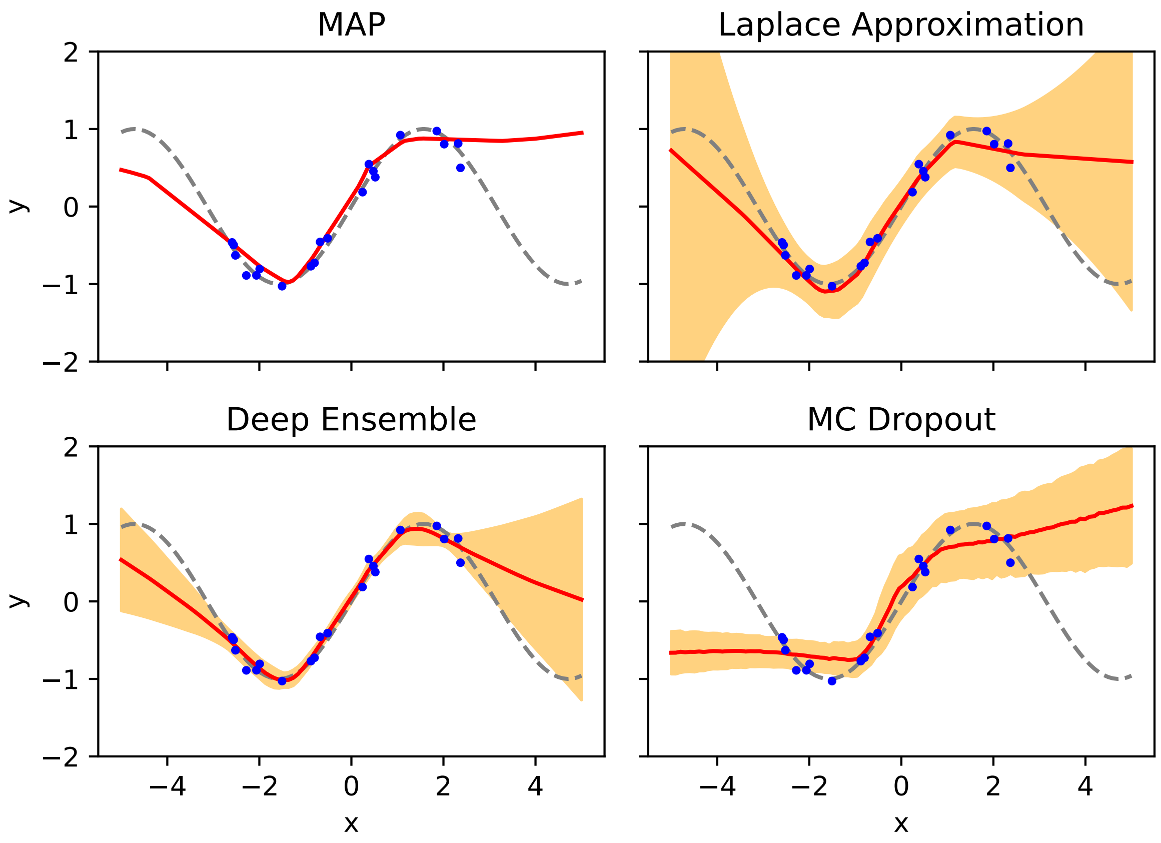

Figure 5: Comparison of MAP and uncertainty quantification methods on the regression toy dataset. The blue points as training data are sampled with Gaussian noise from the grey line. For the MAP plot, the red curve represents the prediction function. For the other plots, the predictive mean (red curve) ± two standard deviations (orange band) are shown.

7 Discussion

Since our illustrative examples are more of sanity checks than benchmark experiments, we compare the methods in terms of their implementation and computational cost, rather than their performance. While Lapalce approximation is applicable in a post-hoc manner, constructing the approximation is non-trivial due to challenges in Hessian computation. Furthermore, the choice of prior precision as a hyperparameter turns out to be crucial for the quality of uncertainty representation. By constrast, Deep Ensembles and MC Dropout require less implementation effort and less fine-tuning. However, they are less efficient in comparison to some variants of Laplace approximation, since it is necessary to perform multiple forward passes to obtain a single uncertainty estimate. While Deep Ensemble captures multimodality, its computational cost is much higher than the other two methods. MC Dropout seems a good alternative to Laplace approximation and Deep Ensembles if one pursues both low implementation effort and low computational cost. However, its use cases are limited to networks trained with Dropout.

8 Conclusion and outlook

In light of the equivalence between regulized risk minimization and MAP estimation, we presented a Bayesian view of deep learning. Our discussion was centered around the predictive distribution which accounts for both aleatroic and espistemic uncertainty as two major types of uncertainty in neural networks. We introduced three methods for approximating this predictive distribution and discussed their strengths and weaknesses.

Throughout this paper, we did not mention how to quantitatively assess these approximation methods. Benchmarking uncertainty quantification is, in fact, a difficult task, because there is no ground-truth uncertainty as reference (Lakshminarayanan et al., 2017). One might be able to control the aleatoric uncertainty by designing a toy experiment, but the epistemic uncertainty inside the model remains unknown. This observation also raises the question whether we can separate these two types of uncertainty. Current research shows that uncertainty disentaglement is hard (Mucsányi et al., 2024, Wimmer et al., 2023). Despite the open challenges, uncertainty quantification methods have been successfully applied to active learning, medical image analysis, robotics, and earth observation (Gawlikowski et al., 2023).

A Electronic appendix

Data, code and figures are provided in electronic form. Code is available on `https://github.com/b-zhou/Seminar-TrustworthyML-UQ`.

References

Daxberger, E., Kristiadi, A., Immer, A., Eschenhagen, R., Bauer, M. and Hennig, P. (2021). Laplace Redux - Effortless Bayesian Deep Learning, *Advances in Neural Information Processing Systems*, Vol. 34, Curran Associates, Inc., pp. 20089–20103.

Daxberger, E., Nalisnick, E., Allingham, J. U., Antoran, J. and Hernandez-Lobato, J. M. (2021). Bayesian Deep Learning via Subnetwork Inference, *Proceedings of the 38th International Conference on Machine Learning*, PMLR, pp. 2510–2521.

Eschenhagen, R., Daxberger, E., Hennig, P. and Kristiadi, A. (2021). Mixtures of Laplace Approximations for Improved Post-Hoc Uncertainty in Deep Learning, *Bayesian Deep Learning Workshop, NeurIPS 2021*.

Fort, S., Hu, H. and Lakshminarayanan, B. (2020). Deep Ensembles: A Loss Landscape Perspective.

Gal, Y. (2016). *Uncertainty in Deep Learning*, PhD Thesis, University of Cambridge.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning, *Proceedings of The 33rd International Conference on Machine Learning*, PMLR, pp. 1050–1059.

Gawlikowski, J., Tassi, C. R. N., Ali, M., Lee, J., Humt, M., Feng, J., Kruspe, A., Triebel, R., Jung, P., Roscher, R., Shahzad, M., Yang, W., Bamler, R. and Zhu, X. X. (2023). A survey of uncertainty in deep neural networks, *Artificial Intelligence Review* **56**(1): 1513–1589.

Gibbs, M. N. (1997). *Bayesian Gaussian Processes for Regression and Classi Cation*, PhD thesis, University of Cambridge.

Grigsby, E., Lindsey, K. and Rolnick, D. (2023). Hidden Symmetries of ReLU Networks, *Proceedings of the 40th International Conference on Machine Learning*, PMLR, pp. 11734–11760.

Guo, C., Pleiss, G., Sun, Y. and Weinberger, K. Q. (2017). On Calibration of Modern Neural Networks, *Proceedings of the 34th International Conference on Machine Learning*, PMLR, pp. 1321–1330.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors.

Hobbhahn, M., Kristiadi, A. and Hennig, P. (2022). Fast predictive uncertainty for classification with Bayesian deep networks, *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence*, PMLR, pp. 822–832.

Hüllermeier, E. and Waegeman, W. (2021). Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods, *Machine Learning* **110**(3): 457–506.

Immer, A., Bauer, M., Fortuin, V., Rätsch, G. and Emtiyaz, K. M. (2021). Scalable Marginal Likelihood Estimation for Model Selection in Deep Learning, *Proceedings of the 38th International Conference on Machine Learning*, PMLR, pp. 4563–4573.

Immer, A., Korzepa, M. and Bauer, M. (2021). Improving predictions of Bayesian neural nets via local linearization, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, PMLR, pp. 703–711.

Kristiadi, A., Hein, M. and Hennig, P. (2020). Being Bayesian, Even Just a Bit, Fixes Overconfidence in ReLU Networks, *Proceedings of the 37th International Conference on Machine Learning*, PMLR, pp. 5436–5446.

Kristiadi, A., Hein, M. and Hennig, P. (2021). Learnable uncertainty under Laplace approximations, *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, PMLR, pp. 344–353.

Lakshminarayanan, B., Pritzel, A. and Blundell, C. (2017). Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles, *Advances in Neural Information Processing Systems*, Vol. 30, Curran Associates, Inc.

Li, H., Xu, Z., Taylor, G., Studer, C. and Goldstein, T. (2018). Visualizing the Loss Landscape of Neural Nets, *Advances in Neural Information Processing Systems*, Vol. 31, Curran Associates, Inc.

Livieris, I. E., Iliadis, L. and Pintelas, P. (2021). On ensemble techniques of weight-constrained neural networks, *Evolving Systems* **12**(1): 155–167.

Loquercio, A., Segu, M. and Scaramuzza, D. (2020). A general framework for uncertainty estimation in deep learning, *IEEE Robotics and Automation Letters* **5**(2): 3153–3160.

MacKay, D. J. C. (1992). The Evidence Framework Applied to Classification Networks, *Neural Computation* **4**(5): 720–736.

Malinin, A. and Gales, M. (2018). Predictive Uncertainty Estimation via Prior Networks, *Advances in Neural Information Processing Systems*, Vol. 31.

Moosavi-Dezfooli, S.-M., Fawzi, A., Fawzi, O. and Frossard, P. (2017). Universal adversarial perturbations, *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 86–94.

Mucsányi, B., Kirchhof, M. and Oh, S. J. (2024). Benchmarking Uncertainty Disentanglement: Specialized Uncertainties for Specialized Tasks.

Nguyen, A., Yosinski, J. and Clune, J. (2015). Deep neural networks are easily fooled: High confidence predictions for unrecognizable images, *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Boston, MA, USA, pp. 427–436.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python, *Journal of Machine Learning Research* **12**: 2825–2830.

Ritter, H., Botev, A. and Barber, D. (2018). A Scalable Laplace Approximation for Neural Networks, *International Conference on Learning Representations*.

Roy, A. G., Conjeti, S., Navab, N. and Wachinger, C. (2019). Bayesian QuickNAT: Model uncertainty in deep whole-brain segmentation for structure-wise quality control, *NeuroImage* **195**: 11–22.

Sommer, E., Wimmer, L., Papamarkou, T., Bothmann, L., Bischl, B. and Rügamer, D. (2024). Connecting the Dots: Is Mode-Connectedness the Key to Feasible Sample-Based Inference in Bayesian Neural Networks?, *Proceedings of the 41st International Conference on Machine Learning*, PMLR, pp. 45988–46018.

Spiegelhalter, D. J. and Lauritzen, S. L. (1990). Sequential updating of conditional probabilities on directed graphical structures, *Networks* **20**(5): 579–605.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting, *Journal of Machine Learning Research* **15**(56): 1929–1958.

Valdenegro-Toro, M. (2019). Deep Sub-Ensembles for Fast Uncertainty Estimation in Image Classification.

Wen, Y., Tran, D. and Ba, J. (2019). BatchEnsemble: An Alternative Approach to Efficient Ensemble and Lifelong Learning, *International Conference on Learning Representations*.

Wilson, A. G. and Izmailov, P. (2020). Bayesian Deep Learning and a Probabilistic Perspective of Generalization, *Advances in Neural Information Processing Systems*, Vol. 33, Curran Associates, Inc., pp. 4697–4708.

Wimmer, L., Sale, Y., Hofman, P., Bischl, B. and Hüllermeier, E. (2023). Quantifying aleatoric and epistemic uncertainty in machine learning: Are conditional entropy and mutual information appropriate measures?, *Proceedings of the Thirty-Ninth Conference on Uncertainty in Artificial Intelligence*, PMLR, pp. 2282–2292.

Declaration of authorship