Low-Cost Bayesian Methods for Fixing Neural Networks' Overconfidence

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Agustinus Kristiadi
aus Bandung, Indonesien

Tübingen
2022

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation: 13.01.2023

Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Philipp Hennig
2. Berichterstatter:	Prof. Dr. Robert Bamler

ABSTRACT

Well-calibrated predictive uncertainty of neural networks—essentially making them know when they do not know—is paramount in safety-critical applications. However, deep neural networks are overconfident in the region both far away and near the training data. In this thesis, we study Bayesian neural networks and their extensions to mitigate this issue. First, we show that being Bayesian, even just at the last layer and in a *post-hoc* manner via Laplace approximations, helps mitigate overconfidence in deep ReLU classifiers. Then, we provide a cost-effective Gaussian-process extension to ReLU Bayesian neural networks that provides a guarantee that ReLU nets will never be overconfident in the region far from the data. Furthermore, we propose three ways of improving the calibration of general Bayesian neural networks in the regions near the data by (i) refining parametric approximations to the Bayesian neural networks' posteriors with normalizing flows, (ii) training the uncertainty of Laplace approximations, and (iii) leveraging out-of-distribution data during training. We provide an easy-to-use library, `laplace-torch`, to facilitate the modern arts of Laplace approximations in deep learning. It gives users a way to turn a standard pre-trained deep net into a Bayesian neural network in a cost-efficient manner.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Philipp Hennig—my main supervisor—for his belief in me. Without his help in guiding me in the world of academia, I will not be able to accomplish much. I would also like to thank Matthias Hein—my second supervisor—for his counsel from a fresh, different perspective. I also acknowledge the support from the third member of my Thesis Advisory Committee, Jörg Stückler, along with the International Max Planck School of Intelligent Systems (IMPRS-IS) community in general.

Special thanks to Runa Eschenhagen, Alexander Immer, and Erik Daxberger for stimulating discussions; and to all excellent scientists who I am fortunate to do research with during my doctoral study. My thank also goes to the Methods of Machine Learning running group, for promoting healthy habit in the midst of busy academic life. Last but not least, I am grateful to Jonathan Wenger, Nathanael Bosch, Marius Hobbhahn, and Runa Eschenhagen for proofreading this thesis.

Contents

Chapter 1
Introduction

Machine learning models, especially neural networks, are increasingly being used in practical applications due to their groundbreaking predictive power. They have thus become an integral part of modern human life. For this reason, neural networks must be safe and robust.

One way to make neural networks safe is by ensuring that their predictions are well-calibrated. Essentially, the goal is to make neural networks know when they do not know. Well-calibrated predictive uncertainty is important since it allows the network to defer the decision-making to a human expert—high-confident wrong predictions could potentially be disastrous, e.g. in self-driving vehicles or medical diagnoses.

In this thesis, we approach the uncertainty calibration of neural networks from the point of view of Bayesian inference. The key idea is that the uncertainty in the parameter space of a neural network, as quantified by an approximate posterior distribution of the network, can give useful information about the lack of knowledge about the correct parameter due to the lack of training data. When this uncertainty is incorporated into the outputs of the network, the networks' predictions should become more calibrated and robust. In this introductory chapter, we shall review the necessary background knowledge for the later chapters.

1.1 Probabilistic Inference

Probability theory provides a principled way of reasoning under uncertainty. Suppose that z_1 and z_2 are random variables, taking values in sets Z_1 and Z_2, respectively, with associated joint probability distribution $p(z_1, z_2)$. The rules of probability can be summarized into the following equations:

(a) PRODUCT RULE: $p(z_1, z_2) = p(z_1 \mid z_2)p(z_2) = p(z_2 \mid z_1)p(z_1)$. Moreover, if z_1 is independent of z_2, then $p(z_1, z_2) = p(z_1)p(z_2)$.

(b) SUM RULE: $p(z_1) = \int p(z_1, z_2)\, dz_2$.

(c) CONDITIONAL PROBABILITY: $p(z_1 \mid z_2) = \frac{p(z_1, z_2)}{p(z_2)}$.

(d) BAYES' RULE: $p(z_1 \mid z_2) = \frac{p(z_2|z_1)p(z_1)}{p(z_2)}$.

For the sum rule, if z_1, z_2 are discrete random variables—i.e., when Z_1, Z_2 are countable—the integral is replaced by a summation. Note that, if z_1, z_2 are continuous random variables—i.e., when Z_1, Z_2 are uncountable—the distributions considered above can also be interchangeably replaced by their probability density function. We henceforth interchangeably refer to $p(z)$ as the distribution and the density function of z, whenever z is a continuous random variable.

Let z be a random variable with a chosen distribution $p(z)$, and let $\mathcal{D} := \{x_i\}_{i=1}^m$ be a set of m independent and identically distributed (i.i.d.) observations. Assume that each observation

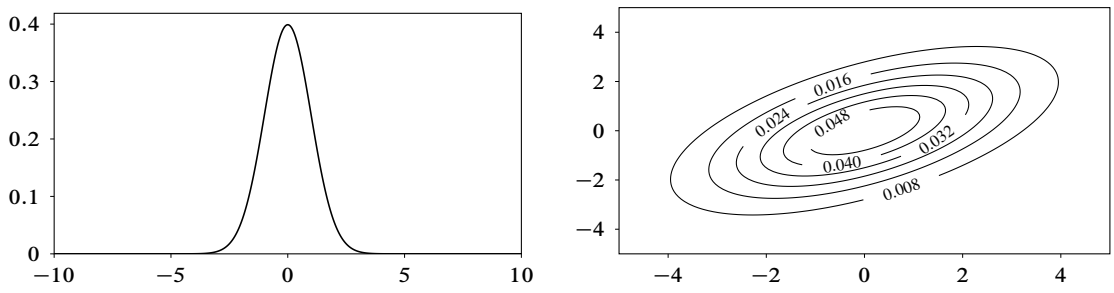


Figure 1.1: Gaussian probability density functions on $\mathbb{R}$ (left) and $\mathbb{R}^2$ (right)—the latter is visualized in terms of its contour.

is modeled via a chosen $p(x_i \mid z)$. Then, the task of ***probabilistic inference*** is to obtain the probability distribution of the random variable z given the data $\mathcal{D}$ using Bayes' rule, that is:

$$p(z \mid \mathcal{D}) = \frac{p(z)p(\mathcal{D} \mid z)}{p(\mathcal{D})} = \frac{p(z)\prod_{i=1}^{m} p(x_i \mid z)}{\int p(z)\prod_{i=1}^{m} p(x_i \mid z)\, dz}. \tag{1.1}$$

The distribution $p(z)$ and $p(x \mid z)$ are called the ***prior*** and ***likelihood***, respectively. Meanwhile, the resulting distribution $p(z \mid \mathcal{D})$ is called the ***posterior***.

Since both $p(x \mid z)$ and $p(z)$ are subjective, i.e. they are modeling choices, they are often chosen to be simple, parametric distributions. The term "parametric" here refers to the property that the distribution is fully characterized by its parameters. In the following, we review some common parametric distributions.

1.1.1 Gaussian Distribution

One of the most commonly-used continuous probability distributions is the ***Gaussian distribution***, defined by its density function on the real line

$$\mathcal{N}\big(x \mid \mu, \sigma^2\big) := \frac{1}{\sqrt{\sigma\, 2\pi}} \exp\!\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right), \tag{1.2}$$

where $\mu \in \mathbb{R}$ and $\sigma^2 \in \mathbb{R}_{>0}$ are its parameters. Its generalization on $\mathbb{R}^d$ with $d > 1$ is given by

$$\mathcal{N}(x \mid \mu, \Sigma) := \left(\det \Sigma^{-1}\right)^{\frac{1}{2}} (2\pi)^{-\frac{d}{2}} \exp\!\left(-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right). \tag{1.3}$$

In this case, $\mu \in \mathbb{R}^d$ and $\Sigma \in S_+(d)$, where $S_+(d)$ denotes the set of $d \times d$ positive-definite matrices. The parameters μ and σ^2 (or Σ) encode the location of the mode and the width of the density function, respectively—see Fig. 1.1. These parameters are also the ***mean*** and ***(co)variance*** of the random variable x, respectively.

Gaussian distributions are important in probabilistic inference since they have many convenient properties. First, all three rules of probability are closed form for Gaussian random variables, see e.g. Bishop (2006, Section 2.3). Second, they have the following properties—useful for later when we discuss approximate Bayesian neural networks:

(a) CONVOLUTION: Let $\mathcal{N}(x \mid \mu_1, \sigma_1^2)$ and $\mathcal{N}(z \mid \mu_2, \sigma_2^2)$ be two Gaussians in $\mathbb{R}$. Their convolution is given by

$$\begin{aligned}\mathcal{N}(x \mid \mu_1, \sigma_1^2) * \mathcal{N}(z \mid \mu_2, \sigma_2^2) &:= \int_{\mathbb{R}} \mathcal{N}(x - z \mid \mu_1, \sigma_1^2)\, \mathcal{N}(z \mid \mu_2, \sigma_2^2)\, dz \\ &= \int_{\mathbb{R}} \mathcal{N}(x \mid \mu_1 + z, \sigma_1^2)\, \mathcal{N}(z \mid \mu_2, \sigma_2^2)\, dz \\ &= \mathcal{N}(x \mid \mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2).\end{aligned} \tag{1.4}$$

As a corollary, we thus have

$$\int_{\mathbb{R}} \mathcal{N}(x \mid z, \sigma_1^2)\, \mathcal{N}(z \mid \mu_2, \sigma_2^2)\, dz = \mathcal{N}(x \mid \mu_2, \sigma_1^2 + \sigma_2^2). \tag{1.5}$$

(b) AFFINE TRANSFORMATION: If $x \sim \mathcal{N}(x \mid \mu, \Sigma)$ is a Gaussian random variable in $\mathbb{R}^n$ and $A \in \mathbb{R}^{k \times n}, b \in \mathbb{R}^k$, then

$$Ax + b \sim \mathcal{N}(Ax + b \mid A\mu + b, A\Sigma A^\top). \tag{1.6}$$

(c) PROBIT INTEGRAL: The ***probit function*** Φ is the cumulative distribution function of the univariate Gaussian $\mathcal{N}(z \mid 0, 1)$ on $\mathbb{R}$, i.e., $\Phi(r) := \int_{-\infty}^{r} \mathcal{N}(z \mid 0, 1)\, dz$. We have

$$\int_{\mathbb{R}} \Phi(ax)\, \mathcal{N}(x \mid \mu, \sigma^2)\, dx = \Phi\left(\frac{\mu}{\sqrt{a^{-2} + \sigma^2}}\right), \tag{1.7}$$

for any $a \in \mathbb{R}$.

1.1.2 Categorical and Dirichlet Distribution

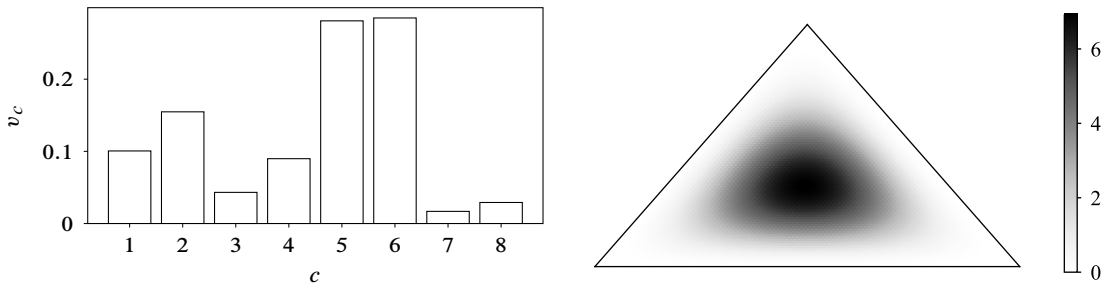
Suppose we have a set $C = \{1, \ldots, k\}$ for a positive integer $k > 1$. A probability distribution on C can be described by an element of the $(k-1)$-probability simplex

$$\Delta^{(k-1)} := \left\{ v \in \mathbb{R}^k_{\geq 0} : \textstyle\sum_{i=1}^k v_i = 1 \right\}. \tag{1.8}$$

That is, for each $c \in C$, the probability of having the value c is v_c—the c-th component of v. In other words, if y is a vector of size k with one component equals one and zero otherwise—the so-called ***one-hot encoding*** of an element $c \in C$; a "corner" of the simplex $\Delta^{(k-1)}$—then we can write the probability mass function as

$$\mathrm{Cat}(y \mid v) := \prod_{i=1}^{k} v_i^{y_i}. \tag{1.9}$$

This distribution is the so-called ***Categorical distribution***. When $k = 2$, it is called the ***Bernoulli distribution***.



(a) Categorical **(b)** Dirichlet

Figure 1.2: A Categorical distribution on $\{1, \ldots, 8\}$ and a Dirichlet distribution on the simplex $\Delta^{(2)}$. The probability vector $(v_1, \ldots, v_k)$ of a Categorical distribution is an element of the probability $(k-1)$-simplex. So, the Dirichlet distribution can also be thought of as a distribution over (Categorical) distributions.

One can also define a probability distribution on the probability simplex $\Delta^{(k-1)}$ itself, called the ***Dirichlet distribution***, via the following density function on $\Delta^{(k-1)}$:

$$\mathrm{Dir}(v \mid \alpha) := \frac{\Gamma\left(\sum_{i=1}^k \alpha_i\right)}{\sum_{i=1}^k \Gamma(\alpha_i)} \prod_{i=1}^k v_i^{\alpha_i - 1}, \tag{1.10}$$

where $\alpha \in \mathbb{R}^k_{>0}$ is the parameter and $\Gamma(z) := \int_0^\infty x^{z-1} \exp(-x)\, dx$ is the Gamma function. The Dirichlet distribution can thus be thought of as a distribution over Categorical distributions.

1.2 Neural Networks

Neural networks are the workhorses of modern machine learning. The main principle of ***neural networks*** is to stack simple parametric nonlinear functions to end up with a much more sophisticated one.[1] That is, if $f : \mathbb{R}^n \to \mathbb{R}^k$ is a neural network function from the input space $\mathbb{R}^n$ to the output space $\mathbb{R}^k$, then it can be written as

$$f := f^{(L)}_{\theta^{(L)}} \circ \cdots \circ f^{(1)}_{\theta^{(1)}}, \tag{1.11}$$

for some functions $f^{(1)}_{\theta^{(1)}}, \ldots, f^{(L)}_{\theta^{(L)}}$ parametrized by $\theta^{(1)}, \ldots, \theta^{(L)}$, respectively. Note that, the dimensions of each $f^{(\ell)}_{\theta^{(\ell)}}$ must agree with its "neighbors": if $f^{(\ell)}_{\theta^{(\ell)}}$ is a function from $\mathbb{R}^{n_{\ell-1}} \to \mathbb{R}^{n_\ell}$, then $f^{(\ell-1)}_{\theta^{(\ell-1)}}$ must be a function with codomain $\mathbb{R}^{n_{\ell-1}}$ and $f^{(\ell+1)}_{\theta^{(\ell+1)}}$ must be a function with domain $\mathbb{R}^{n_\ell}$. Note also that defining

$$\theta := \left\{\mathrm{vec}\,\theta^{(1)}, \ldots, \mathrm{vec}\,\theta^{(L)}\right\} \in \mathbb{R}^d,$$

to be the concatenation of the vectorization of all its weights, the neural network f is often also written as f_θ or $f(\cdot\,; \theta)$. Finally, the number L is referred to as the ***depth*** of the neural network.

[1] There are many variants of neural networks, but here we focus on classic neural networks.

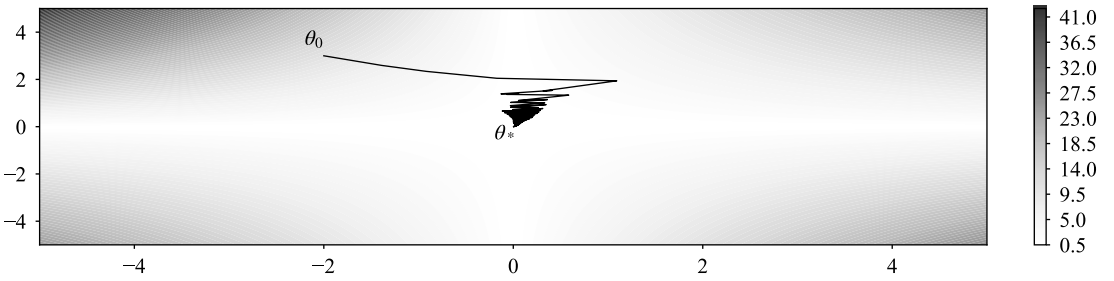


Figure 1.3: Gradient descent's dynamics. Background shade is the contour of the loss function.

What specifically is each $f^{(\ell)}_{\theta^{(\ell)}} : \mathbb{R}^{n_{\ell-1}} \to \mathbb{R}^{n_\ell}$? Commonly, it is defined as a composition of an affine function and a nonlinear function φ. That is, $f^{(\ell)}_{\theta^{(\ell)}}(x) = \varphi(Wx + b)$, where in this case $\theta^{(\ell)} = \{W \in \mathbb{R}^{n_\ell \times n_{\ell-1}}, b \in \mathbb{R}^{n_\ell}\}$. Note that, this definition includes common neural network building blocks such as the dense layer and the convolution layer. As for the nonlinearity φ, it is often chosen to be a simple component-wise nonlinear function. For example, the ***ReLU activation function*** acts on each component z_i of a vector z via

$$\mathrm{ReLU}(z_i) := \max(0, z_i), \tag{1.12}$$

Meanwhile, the ***hyperbolic-tangent activation function*** acts component-wise on a vector z via

$$\tanh(z_i) := \frac{\exp(z_i) - \exp(-z_i)}{\exp(z_i) + \exp(-z_i)}. \tag{1.13}$$

The ReLU activation function is virtually the *de facto* nonlinearity for deep neural networks.

1.2.1 Training

Let $\mathcal{D} := \{(x_i, y_i) \in \mathbb{R}^n \times \mathbb{R}^k\}_{i=1}^m$ be an i.i.d. dataset. The goal of neural network training is to "learn" the unknown function $\mathbb{R}^n \to \mathbb{R}^k$ by finding a suitable parameter value θ of a neural network $f_\theta : \mathbb{R}^n \to \mathbb{R}^k$ given the dataset $\mathcal{D}$. This setup is known as ***supervised learning***, as opposed to ***unsupervised learning*** where the dataset is *not* constructed by pairs of input-output observations, i.e. $\mathcal{D} = \{x_i \in \mathbb{R}^n\}_{i=1}^m$.

To that end, one must define a ***loss function*** $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ which measures how well the network parameter θ, and thus the network function f_θ, fit the dataset $\mathcal{D}$. For instance, the ***sum squared error loss*** is defined as

$$\mathcal{L}(\theta) := \frac{1}{2\beta} \sum_{i=1}^{m} \sum_{j=1}^{k} (f_\theta(x_i)_j - y_{ij})^2, \tag{1.14}$$

where $\beta > 0$ is freely chosen. On other hand, when the output variable y takes values in $\{1, \dots, k\}$, which is isomorphic to the k corners of the simplex $\Delta^{(k-1)} \subset \mathbb{R}^k$, one often uses the ***cross-entropy loss***, written under the one-hot encoded y by

$$\mathcal{L}(\theta) := -\sum_{i=1}^{m} \sum_{j=1}^{k} y_{ij} \log \operatorname{softmax}(f_\theta(x_i))_j, \tag{1.15}$$

where the ***softmax function*** $\operatorname{softmax} : \mathbb{R}^k \to \Delta^{(k-1)}$ is defined by

$$\operatorname{softmax}(z) := \left(\frac{\exp(z_1)}{\sum_{i=1}^k \exp(z_i)}, \dots, \frac{\exp(z_k)}{\sum_{i=1}^k \exp(z_i)} \right). \tag{1.16}$$

It is worth noting that when $k = 2$, one can alternatively define a real-valued neural network $f_\theta : \mathbb{R}^n \to \mathbb{R}$ and use the ***logistic function*** $\sigma : \mathbb{R} \to \Delta^{(1)} \simeq [0, 1]$ instead, defined by

$$\sigma(z) := \frac{1}{1 + \exp(-z)} = \frac{\exp(z)}{1 + \exp(z)}. \tag{1.17}$$

The softmax and logistic functions are instances of nontrivial (inverse) ***link functions***, because it "links" between the output space of the network f_θ and the space where y takes values in.[2]

Training the neural network f_θ can thus be reduced to minimizing the loss function:

$$\theta_* = \operatorname*{argmin}_{\theta \in \mathbb{R}^d} \mathcal{L}(\theta). \tag{1.18}$$

The most common way to carry out this optimization is via ***backpropagation***: Assuming that $\mathcal{L}$ is almost-everywhere differentiable, one computes the gradient $\nabla_\theta \mathcal{L}$ via the chain rule, exploiting the layered structure of f_θ, and then uses the gradient to iteratively update the value of θ. The update part of backpropagation can be done in various ways, the simplest of which is via ***gradient descent***:

$$\theta_{i+1} = \theta_i - \alpha \nabla_\theta \mathcal{L}|_{\theta_i}, \tag{1.19}$$

where $\alpha \in \mathbb{R}_{>0}$ is a ***step size*** which governs how much one moves away from θ_i in the direction of the negative-gradient. This update rule can be generalized to ***preconditioned gradient descent*** which employs a positive-definite matrix field $P : \mathbb{R}^d \to S_+(d)$ to take into account the geometry of the parameter space $\mathbb{R}^d$:

$$\theta_{i+1} = \theta_i - \alpha P(\theta)^{-1} \nabla_\theta \mathcal{L}|_{\theta_i}, \tag{1.20}$$

For efficiency reason, $P(\theta)$ is often restricted to be diagonal matrix (Duchi et al., 2011; Hinton et al., 2012; Kingma & Ba, 2015, etc.).

One can further reduce the computational burden of performing gradient descent by computing the gradient $\nabla_\theta \mathcal{L}$ using only a subset of the dataset. The resulting methods are subsumed in ***stochastic gradient descent*** methods—the word "stochastic" indicates that the update rule is subject to a noise that arises from the subsampling process of $\mathcal{D}$.

Optimizing the loss function $\mathcal{L}$ as-is risks ***overfitting***: a phenomenon where the network f_θ only memorizes the dataset $\mathcal{D}$ and does not approximate the underlying function $x \mapsto y$ well.

[2] In the case where y takes values in $\mathbb{R}^k$, then the link function is simply the identity function $\operatorname{id}(x) = x$.

To prevent this, one can augment the loss function with a ***regularization*** term which limits the network's complexity. This can be done, for example, by encouraging θ to have small norm:

$$\widetilde{\mathcal{L}}(\theta) := \mathcal{L}(\theta) + \frac{1}{2\gamma}\theta^\top\theta. \tag{1.21}$$

This regularization is often referred to as the ***weight decay*** regularization and is almost always employed when training a neural network.

1.2.2 Probabilistic Interpretation

Many loss functions can be interpreted probabilistically. Let us use the sum squared error as a first example. Denoting the identity matrix in $\mathbb{R}^{d\times d}$ by I_d, notice how (1.14) can be written as

$$\begin{aligned}\mathcal{L}(\theta) &= -\sum_{i=1}^{m}\left(-\frac{1}{2\beta}\sum_{j=1}^{k}(y_{ij} - f_\theta(x_i)_j)^2\right)\\ &= -\sum_{i=1}^{m}\left(-\frac{1}{2}(y_i - f_\theta(x_i))^\top(\beta^{-1}I_d)(y_i - f_\theta(x_i))\right)\\ &= -\log\prod_{i=1}^{m}\exp\left(-\frac{1}{2}(y_i - f_\theta(x_i))^\top(\beta^{-1}I_d)(y_i - f_\theta(x_i))\right).\end{aligned} \tag{1.22}$$

We can thus identify the term inside the product to be proportional to a Gaussian on $\mathbb{R}^k$ with mean $f_\theta(x_i)$ and covariance $\beta^{-1}I_d$. Note that adding a constant term to take into account the normalization constant of this Gaussian does not change the optimization result. So, by the definition of likelihood in (1.1), we can think of the sum squared loss as the negative-log Gaussian likelihood.

Similarly, we can write the cross-entropy loss as

$$\mathcal{L}(\theta) = -\log\prod_{i=1}^{m}\prod_{j=1}^{k}\operatorname{softmax}(f_\theta(x_i))_j^{y_{ij}} = -\log\prod_{i=1}^{m}\operatorname{Cat}(y_i \mid \operatorname{softmax}(f_\theta(x_i))). \tag{1.23}$$

Thus, comparing it against (1.15), we can immediately see that the cross-entropy loss is equivalent to the negative-log Categorical likelihood.

Furthermore, one can also interpret the weight decay probabilistically. We can identify the second term in (1.21) as the negative log-probability of a Gaussian on $\mathbb{R}^d$ with mean zero and covariance γI_d. Notice that this term does not depend on the data—comparing it to (1.1), we can identify this term as a Gaussian prior over the parameter θ. The regularized loss function (1.21) can thus be seen as the negative of the unnormalized posterior probability over the parameter θ of f_θ under the dataset $\mathcal{D}$. The solution $\theta_* = \operatorname{argmin}_\theta \widetilde{\mathcal{L}}(\theta)$ is thus the ***mode*** of the posterior density $p(\theta \mid \mathcal{D})$. For this reason, the problem of minimizing $\widetilde{\mathcal{L}}$ is also called ***maximum a posteriori (MAP) estimation***. The solution θ_* of MAP estimation is also written as θ_{MAP}.

While MAP estimation does indeed work with the posterior density $p(\theta \mid \mathcal{D})$, it only concerns in finding a *single*, most-likely parameter θ. That is, one can see MAP estimation as finding the best Dirac-delta approximation $\delta(\theta - \theta_{\text{MAP}})$ of the posterior. Note that, the Dirac-delta function can be seen as the limit of a Gaussian with variance that goes to zero. So, MAP estimation lacks

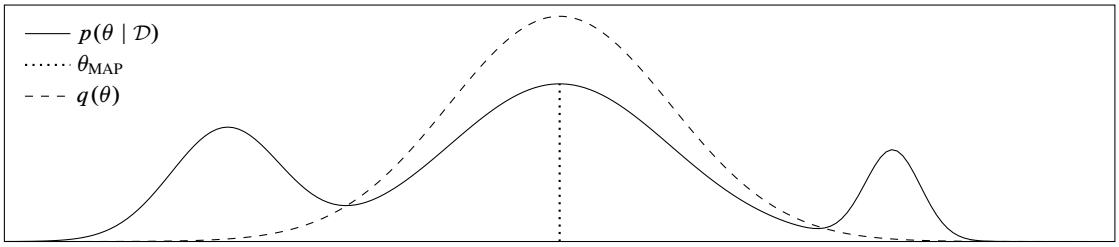


Figure 1.4: A MAP estimate θ_{MAP} and its (Gaussian-approximated) Bayesian counterpart $q(\theta)$. While MAP estimation is an approximation to the posterior $p(\theta \mid \mathcal{D})$, the resulting network is not a Bayesian neural network since it does not capture a nontrivial probability mass of the posterior, either via an approximate parametric density or via samples.

an uncertainty estimate. To mitigate this issue, one can capture the uncertainty by moving away from point-estimation, via the so-called Bayesian neural network formulation of f_θ.

1.3 Bayesian Neural Networks

Let $\theta \in \mathbb{R}^d$ be a random parameter of a neural network $f_\theta : \mathbb{R}^n \to \mathbb{R}^k$ and $p(\theta \mid \mathcal{D})$ be its posterior under an independent and identically distributed dataset $\mathcal{D} := \{(x_i, y_i)\}_{i=1}^m$, written via Bayes' rule as

$$p(\theta \mid \mathcal{D}) = \frac{1}{\int p(\mathcal{D} \mid \theta) p(\theta)\, d\theta}\, p(\mathcal{D} \mid \theta) p(\theta) =: \frac{1}{Z} h(\theta), \tag{1.24}$$

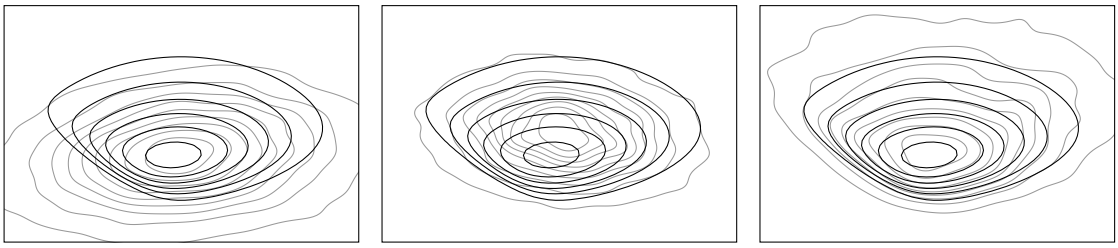
where $p(\mathcal{D} \mid \theta) := \prod_{i=1}^m p(y_i \mid f_\theta(x_i))$ and $p(\theta)$ are a likelihood and a prior, respectively. The quantity Z is a function of the data and often also called the ***marginal likelihood*** or the ***evidence*** $p(\mathcal{D})$. A neural network equipped with its non-point-mass posterior distribution is called a ***Bayesian neural network***. Note in particular that this definition excludes MAP-estimated neural networks. However, the posterior of a Bayesian neural network is generally intractable since the integral Z is, because (i) $p(\mathcal{D} \mid \theta)$ is nonlinear in θ and (ii) θ is high-dimensional.

Approximating $p(\theta \mid \mathcal{D})$ is thus necessary for Bayesian neural networks. Two paradigms exist: (i) approximating $p(\theta \mid \mathcal{D})$ with a simpler, parametric distribution such as the Gaussian and (ii) obtaining samples from $p(\theta \mid \mathcal{D})$ based solely on the readily available $h(\theta)$. We shall review the Laplace approximation and variational Bayes for the former, and Markov Chain Monte Carlo methods for the latter.

1.3.1 The Laplace Approximation

Let $\theta_{\text{MAP}} := \operatorname{argmax}_\theta \log p(\theta \mid \mathcal{D}) = \operatorname{argmax}_\theta \log h(\theta)$ be a (local) maximum of the posterior—the so-called ***maximum a posteriori (MAP)*** estimate. Taylor-expanding $\log h$ around θ_{MAP} up to second order yields

$$\log h(\theta) \approx \log h(\theta_{\text{MAP}}) - \frac{1}{2}(\theta - \theta_{\text{MAP}})^\top \Lambda\, (\theta - \theta_{\text{MAP}}), \tag{1.25}$$



(a) Laplace approximation **(b)** Variational Bayes **(c)** MCMC

Figure 1.5: Comparison between the approximations obtained via the Laplace approximation, Gaussian-based variational Bayes, and MCMC. Black contours are the true posterior density. Grey contours are the density estimates of the aforementioned approximations, computed using their samples. The Laplace approximation provides a good approximation around the mode of the true distribution but can yield inaccurate results far from it. Variational Bayes fixes this issue but MCMC methods yield good approximations in general since they are not constrained to be parametric and they are guaranteed to yield samples from the true posterior if the Markov chain is long enough.

where $\Lambda := -\nabla^2_\theta \log h|_{\theta_{\text{MAP}}}$ is the negative Hessian matrix of the log-unnormalized posterior at θ_{MAP}. Integrating (1.25), we thus obtain a (multivariate) Gaussian integral, the analytic solution of which is readily available:

$$\begin{aligned} Z &\approx \exp(\log h(\theta_{\text{MAP}})) \int \exp\left(-\frac{1}{2}(\theta - \theta_{\text{MAP}})^\top \Lambda\, (\theta - \theta_{\text{MAP}})\right) d\theta \\ &= h(\theta_{\text{MAP}})(2\pi)^{\frac{d}{2}} (\det \Lambda)^{-\frac{1}{2}}. \end{aligned} \tag{1.26}$$

Plugging the approximations (1.25) and (1.26) back into the expression of $p(\theta \mid \mathcal{D})$, we obtain

$$p(\theta \mid \mathcal{D}) = \frac{1}{Z} h(\theta) \approx (\det \Lambda)^{\frac{1}{2}} (2\pi)^{-\frac{d}{2}} \exp\left(-\frac{1}{2}(\theta - \theta_{\text{MAP}})^\top \Lambda\, (\theta - \theta_{\text{MAP}})\right), \tag{1.27}$$

which we can immediately identify as the Gaussian density $\mathcal{N}(\theta \mid \theta_{\text{MAP}}, \Sigma)$ with mean θ_{MAP} and covariance matrix $\Sigma := \Lambda^{-1}$. This Gaussian is called the ***Laplace approximation*** of $p(\theta \mid \mathcal{D})$.

Since both the MAP estimate θ_{MAP} and the Hessian Λ are local quantities, the Laplace approximation is thus a local approximation around a mode of $p(\theta \mid \mathcal{D})$. Intuitively, it can be seen as "surrounding" a point estimate with a Gaussian probability mass, cf. Fig. 1.5a. Nevertheless, the Laplace approximation is among the cheapest approximate inference method for Bayesian neural networks due to the fact that it can be done in a *post-hoc* manner, i.e. it can be applied to any MAP pre-trained neural networks—the Hessian matrix Λ only need to be computed once and can be done easily thanks to recent advances in the field of second-order optimization (Dangel et al., 2020). Given the ubiquity of MAP training in deep learning, the Laplace approximation is thus invaluable to essentially transform a standard neural network into a Bayesian neural network without much hassle.

1.3.2 Variational Bayes

Another way of obtaining a parametric approximation of the posterior $p(\theta \mid \mathcal{D})$ is via variational inference. Let $M := \{q_\varphi(\theta) : \varphi \in \Phi\}$ be a set of parametric densities, usually chosen to be the set of Gaussian densities, i.e. $\varphi := (\mu, \Sigma)$ and $\Phi := \mathbb{R}^d \times S_+(d)$, where $S_+(d)$ is the space of $d \times d$ positive definite matrices. The goal of variational Bayesian inference is to minimize the reverse ***Kullback-Leiber (KL) divergence*** between q_φ and $p(\theta \mid \mathcal{D})$ w.r.t. φ—intuitively, minimizing the non-symmetric distance between q_φ and $p(\theta \mid \mathcal{D})$:

$$\min_{\varphi \in \Phi} D_{\mathrm{KL}}\big(q_\varphi(\theta), p(\theta \mid \mathcal{D})\big) = \min_{\varphi \in \Phi} \int_{\mathbb{R}^d} \log \frac{q_\varphi(\theta)}{p(\theta \mid \mathcal{D})}\, q_\varphi(\theta)\, d\theta. \tag{1.28}$$

However, the integral above is intractable in general since it implies that one can evaluate the normalizing constant Z of $p(\theta \mid \mathcal{D})$.

To mitigate this issue, one can consider the upper bound of the KL-divergence above. To obtain it, we add and substract the log-marginal likelihood $\log p(\mathcal{D})$ to the KL-divergence and note that $p(\theta, \mathcal{D}) = p(\theta \mid \mathcal{D})p(\mathcal{D}) = p(\mathcal{D} \mid \theta)p(\theta)$:

$$\begin{aligned} D_{\mathrm{KL}}\big(q_\varphi(\theta), p(\theta \mid \mathcal{D})\big) &= \mathbb{E}_{q_\varphi}\left(\log \frac{q_\varphi(\theta)}{p(\theta \mid \mathcal{D})}\right) - \log p(\mathcal{D}) + \log p(\mathcal{D}) \\ &= \mathbb{E}_{q_\varphi}\left(\log \frac{q_\varphi(\theta)}{p(\theta \mid \mathcal{D})p(\mathcal{D})}\right) + \log p(\mathcal{D}) \\ &= -\mathbb{E}_{q_\varphi}\left(\log \frac{p(\mathcal{D} \mid \theta)p(\theta)}{q_\varphi(\theta)}\right) + \log p(\mathcal{D}) \\ &=: -\mathrm{ELBO}(\varphi) + \log p(\mathcal{D}). \end{aligned} \tag{1.29}$$

The function $\mathrm{ELBO} : \Phi \to \mathbb{R}$ is thus an approximation to the KL-divergence—the nomenclature ELBO (evidence lower bound) is coming from the fact that it is a lower bound of $\log p(\mathcal{D})$, due to the nonnegativity of the KL-divergence. Note that, unlike $D_{\mathrm{KL}}\big(q_\varphi(\theta), p(\theta \mid \mathcal{D})\big)$, ELBO is easy to work with since it only depends on $h(\theta) = p(\mathcal{D} \mid \theta)p(\theta)$ and $q_\varphi(\theta)$. In particular, it does not depend on Z.

In practice, generally, the expectation presents in ELBO does not have an analytic solution, especially for Bayesian neural networks. Thus, it is often further approximated via ***Monte Carlo integration*** by:

$$\mathrm{ELBO}(\varphi) \approx \frac{1}{s} \sum_{i=1}^{s} \log p(\mathcal{D} \mid \theta_i) + \log p(\theta_i) + \mathbb{H}(q_\varphi); \quad \theta_i \sim q_\varphi(\theta), \tag{1.30}$$

where $\mathbb{H}(q_\varphi) := -\mathbb{E}_{q_\varphi} \log q_\varphi$ is the ***entropy*** of q_φ, which has an analytic solution for common parametric densities such as Gaussian.

While it is more flexible than the Laplace approximation—see Fig. 1.5—variational Bayes is more expensive since it cannot be done *post-hoc* and the dimensionality of the optimization problem is larger than the standard MAP optimization.

1.3.3 Markov Chain Monte Carlo

Let $\theta_1, \ldots, \theta_t$ be a sequence of random variables in $\mathbb{R}^d$. Suppose we assume that θ_i is independent of $(\theta_j)_{j=1}^{i-1}$ given θ_{i-1} for each $i = 1, \ldots, t$, the so-called ***Markov assumption***, then we have the following joint distribution:

$$p(\theta_1, \ldots, \theta_t) = p(\theta_1) \prod_{i=2}^{t} p(\theta_i \mid \theta_{i-1}). \tag{1.31}$$

The sequence $(\theta_i)_{i=1}^t$ is then called a ***Markov chain*** and the probability $p(\theta_i \mid \theta_{i-1})$ is called the ***transition probability***. Once $p(\theta_i \mid \theta_{i-1})$ is defined, a Markov chain provides us with an easy way to obtain sequence of random variables: simply sample θ_i from $p(\theta_i \mid \theta_{i-1})$ given the previous sample θ_{i-1}.

Let θ_i, θ_{i+1} be two random variables in a Markov chain with a transition probability $p(\theta_{i+1} \mid \theta_i)$. Suppose that the distribution of θ is given by $q(\theta)$. Then we say that q is the ***stationary distribution*** of the Markov chain if

$$q(\theta_{i+1}) = \int_\theta p(\theta_{i+1} \mid \theta_i)\, q(\theta_i)\, d\theta_i. \tag{1.32}$$

That is, the distribution of θ is unchanged under the Markov chain transition $p(\theta' \mid \theta)$. Intuitively, if a Markov chain has reached its stationary distribution at "time" i, then every θ_j with $j \geq i$ comes from the same distribution q.

The main idea of ***Markov Chan Monte Carlo (MCMC)*** methods for Bayesian inference is to construct a Markov chain on $\mathbb{R}^d$—i.e. defining the transition function $p(\theta_{i+1} \mid \theta_i)$—which stationary distribution q is the posterior $p(\theta \mid \mathcal{D})$. This way, once the Markov chain enters the stationary distribution, running the Markov chain equals sampling from the true posterior $p(\theta \mid \mathcal{D})$. How soon the Markov chain arrives at its stationary distribution, often referred to as the ***mixing speed*** of the Markov chain, is important in practice since it determines the costs of sampling from the posterior.

One commonly-used MCMC algorithm is the ***Hamiltonian Monte Carlo*** method (Neal, 2012): Let $v \sim \mathcal{N}(0, M)$ be an auxiliary variable in $\mathbb{R}^d$ with $M \in S_+(d)$, and let

$$p(\theta, r \mid \mathcal{D}) \propto \exp\left(\log h(\theta) - \frac{1}{2} v^\top M^{-1} v\right) \tag{1.33}$$

be the augmented posterior. To generate samples from this distribution, one then follow the Hamiltonian dynamics under the Hamiltonian function $H(\theta, v) := -\log h(\theta) + v^\top M^{-1} v$. That is, one simulates the following discretized ordinary differential equation to obtain a sample θ_{i+1} given θ_i for m steps, under the inital conditions $v^{(0)} \sim \mathcal{N}(0, M)$ and $\theta^{(0)} = \theta_i$:

$$\begin{aligned} \theta^{(t)} &= \theta^{(t-1)} + \alpha M^{-1} v^{(t-1)}, \\ v^{(t)} &= v^{(t-1)} - \alpha \nabla_\theta \log h|_{\theta^{(t-1)}}, \end{aligned} \tag{1.34}$$

where α is a step size, and set $\theta_i = \theta^{(m)}$. The cost-effectiveness of this method has been improved further by enabling noisy, minibatch computation of the gradient (Chen et al., 2014). The resulting method is subsumed under the name ***stochastic-gradient MCMC*** methods.

1.3.4 Predictive Distributions

Given a parametric approximation $q(\theta)$ of the posterior $p(\theta \mid \mathcal{D})$, how does one make a prediction y_* on a new input $x_* \in \mathbb{R}^n$? Given the likelihood $p(y_* \mid f_\theta(x_*))$, one must take into account the fact that one has infinitely-many possible values for θ, as weighted by the approximate density $q(\theta)$. The answer thus comes from the rules of probability, discussed in Section 1.1:

$$p(y_* \mid x_*) = \int_{\mathbb{R}^d} p(y \mid f_\theta(x_*))\, q(\theta)\, d\theta. \tag{1.35}$$

However, in the case of neural networks, no analytic solution to this integral exists even when both the likelihood $p(y_* \mid f_\theta(x_*))$ and the approximate posterior $q(\theta)$ are Gaussian, due to the nonlinearity of f_θ. One must then rely on further approximation to obtain $p(y \mid x_*)$.

1.3.4.1 Monte Carlo Integration

The most straightforward approximation is via Monte Carlo integration:

$$p(y_* \mid x_*) \approx \frac{1}{s} \sum_{i=1}^{s} p(y \mid f_{\theta_i}(x_*)); \qquad \theta_i \sim q(\theta). \tag{1.36}$$

This approximation is advantageous since it is *unbiased*, i.e. the average above tends to the true integral as $s \to \infty$, due to the law of large numbers. However, a small value of s is often used in practice due to the cost of computing each $f_{\theta_i}(x_*)$. The associated error, which scales like $1/\sqrt{s}$, can thus be high for Bayesian neural networks. Note that Monte Carlo integration is the only choice of computing $p(y_* \mid x_*)$ under a Markov Chain Monte Carlo approximation.

1.3.4.2 Linearization

Recall that Gaussians are closed under linear/affine transformations. Thus, one way to obtain an analytic approximation to the distribution of the network output $f_* := f(x_*) \in \mathbb{R}^k$, where the parameter θ has been marginalized out, under a Gaussian approximation $q(\theta) = \mathcal{N}(\theta \mid \mu, \Sigma)$ is via a linearization of f_θ around μ:[3]

$$f_\theta(x_*) \approx f_\mu(x_*) + J(x_*)\,(\theta - \mu), \tag{1.37}$$

where $J(x_*) := \left(\nabla_\theta f_\theta(x_*) \mid_\mu\right)$ is the $d \times k$ Jacobian of $f_\theta(x_*)$ w.r.t. θ at μ. Then, using the fact that the Dirac delta function δ can be written as a Gaussian with variance tending to zero, we can use the convolution and affine-transformation properties of Gaussians to obtain

$$p(f_* \mid x_*) \approx \int_{\mathbb{R}^d} \delta(f_* - f_\theta(x_*))\, q(\theta)\, d\theta = \mathcal{N}(f_* \mid f_\mu(x_*), J(x_*)^\top \Sigma J(x_*)). \tag{1.38}$$

This approximation is often useful for theoretical analysis due to its analytic nature. It is also useful in practice since one can use this k-variate Gaussian, instead of the d-variate Gaussian $q(\theta)$ in Monte Carlo integration, to do the computation in downstream tasks.

[3] Note that the network function $x \mapsto f_\theta(x)$ is still nonlinear.

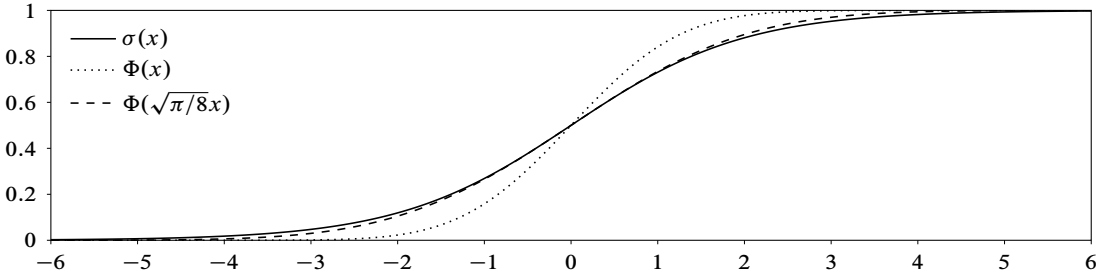


Figure 1.6: Approximating the logistic function with the probit function. The scaling factor $\sqrt{\pi/8}$ for the probit function is chosen so that they both have the same derivative at zero.

Given the Gaussian $p(f_* \mid x_*)$, one more integration is needed to obtain the final prediction:

$$p(y_* \mid x_*) = \int_{\mathbb{R}^k} p(y_* \mid f_*)\, p(f_* \mid x_*)\, df_*. \tag{1.39}$$

As before, this can be approximated via Monte Carlo integration. Unlike the Monte Carlo integration in (1.36), however, (1.39) is more cost-efficient due to the smaller domain of integration and due to the fact that no costly evaluation of the network $f_\theta(x_*)$ for each sample $\theta \sim q(\theta)$ is needed. Thus, one can use a larger number of samples in (1.39), leading to more accurate approximations.

Nevertheless, having a Gaussian distribution over the network output enables us to either analytically solve or approximate the integral (1.39). For regression tasks with a Gaussian likelihood $p(y_* \mid f_*) = \mathcal{N}(y_* \mid f_*, \beta)$, this can be done exactly:

$$p(y_* \mid x_*) = \mathcal{N}(y_* \mid f_\mu(x_*), \beta + J(x_*)^\top \Sigma J(x_*)), \tag{1.40}$$

since the integral above is a convolution of two Gaussians.

For classifications, both with the logistic and softmax link functions, a further approximation is needed. Let us start with binary classification with the logistic function $p(y_* \mid f_*) = \sigma(f_*)$, Using the approximation $\sigma(z) \approx \Phi(\sqrt{\pi/8}\, z)$ and the probit integral (1.7), we obtain the approximation

$$p(y_* \mid x_*) = \int_{\mathbb{R}} \sigma(f_*)\, p(f_* \mid x_*)\, df_* \approx \sigma\left(\frac{f_\mu(x_*)}{\sqrt{1 + \pi/8\, J(x_*)^\top \Sigma J(x_*)}}\right). \tag{1.41}$$

This approximation is called the ***probit approximation***, due to Spiegelhalter & Lauritzen (1990) and MacKay (1992b).

Meanwhile, for multiclass classification with the likelihood $p(y_* \mid f_*) = \text{softmax}(f_*)$, due to the similarity between each component in (1.16) with the logistic function (1.17), we can

obtain an analytic approximation to the softmax-Gaussian integral via the probit approximation, due to Gibbs (1998):

$$\begin{aligned} p(y_* \mid x_*) &= \int_{\mathbb{R}^k} \operatorname{softmax}(f_*)\, p(f_* \mid x_*)\, df_* \\ &\approx \operatorname{softmax}\left(\frac{f_\mu(x_*)}{\sqrt{1 + \pi/8 \operatorname{diag}(J(x_*)^\top \Sigma J(x_*))}} \right), \end{aligned} \tag{1.42}$$

where the division above is taken componentwise. We call this approximation the ***multiclass probit approximation***. See Appendix A for detailed derivations of the (multiclass) probit approximation.

Laplace Bridge The Laplace bridge is a more expressive alternative to the multiclass probit approximation. The main idea is to perform a Laplace approximation to the Dirichlet distribution, which has support on the simplex $\Delta^{(k-1)}$, by first writing it as a distribution over $\mathbb{R}^k$ with the help of the softmax function (MacKay, 1998). This way, the Laplace approximation can be reasonably applied to approximate the Dirichlet, which can be thought of as mapping the Dirichlet $\operatorname{Dir}(\alpha)$ to a Gaussian $\mathcal{N}(\mu, \Sigma)$. The pseudo-inverse of this map, mapping (μ, Σ) to α where for each $i = 1, \ldots, k$, the i-th component α is given by the simple closed-form expression

$$\alpha_i = \frac{1}{\Sigma_{ii}} \left(1 - \frac{2}{k} + \frac{\exp(\mu_i)}{k^2} \sum_{j=1}^{k} \exp(-\mu_j) \right),$$

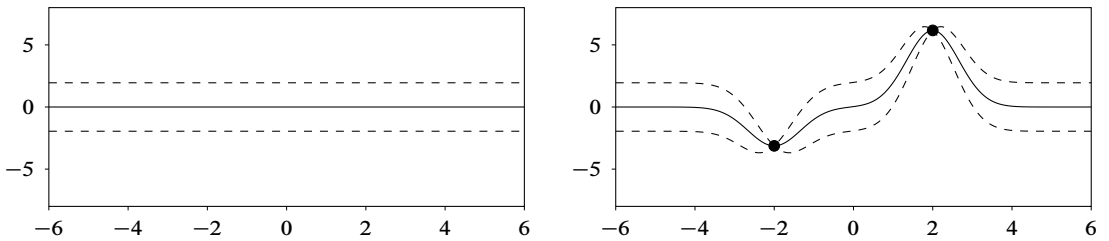
is the ***Laplace bridge***. Unlike the multiclass probit approximation, it yields a *full distribution* over the solutions of the softmax-Gaussian integral (1.42). So, the Laplace bridge is a richer yet comparably simple approximation to the integral (Hobbhahn et al., 2022).

Specifically for Bayesian neural networks, Hobbhahn et al. (2022) further proposed to add corrections to the standard Laplace bridge, which (i) project the Gaussian $\mathcal{N}(\mu, \Sigma)$ to one that adhere the constraint that the random variable sums to zero, and (ii) scale both parameters of the Gaussia by a factor of $(\operatorname{tr} \Sigma)/\sqrt{K/2}$. These correction terms are useful to (i) fullfill the original assumption by MacKay (1998), and (ii) to alleviate the observation that the Laplace bridge does not work well when the variance in Σ is high.

Recall that the Laplace bridge provides a distribution over the softmax-Gaussian integral. In some practical applications of Bayesian neural networks, however, one often only needs a single estimate of the integral. This can be done analytically in the Laplace bridge by summarizing the resulting Dirichlet by its mean. That is, we approximate

$$p(y_* = i \mid x_*) \approx \frac{\alpha_i}{\sum_{j=1}^{k} \alpha_j} \qquad \text{for each } i = 1, \ldots, k, \tag{1.43}$$

to obtain an approximation to the predictive distribution $p(y_* \mid x_*)$, where α is obtained via the Laplace bridge from the Gaussian over the network output f_* with mean $f_\mu(x_*)$ and covariance $J(x_*)^\top \Sigma J(x_*)$.



(a) Prior **(b)** Posterior

Figure 1.7: A Gaussian process prior of a function $\mathbb{R} \to \mathbb{R}$ with a kernel $K(x, x') := \exp(-(x-x')^2/2)$, along with its corresponding posterior. Solid and dashed curves are mean and 95% credible intervals, respectively. Dots represent data points.

1.4 Gaussian Processes

Suppose we have a linear model $f : \mathbb{R}^n \to \mathbb{R}$ defined by

$$f(x) := \phi(x)^\top w,$$

where $w \in \mathbb{R}^d$ is the parameter and $\phi : \mathbb{R}^n \to \mathbb{R}^d$ is an arbitrary feature map. Given a Gaussian prior $p(w) := \mathcal{N}(w \mid 0, \sigma^2 I_d)$ on $\mathbb{R}^d$, we can show by the property of the Gaussian distribution (1.6) that the distribution over the output $p(f(x))$ is also a Gaussian with mean 0 and variance $\sigma^2\phi(x)^\top\phi(x)$. Moreover, for a pair of inputs $x, x' \in \mathbb{R}^n$, the covariance between $f(x)$ and $f(x')$ is given by $\sigma^2\phi(x)^\top\phi(x')$. Thus, the joint distribution $p(f(x), f(x'))$ is also given by a Gaussian

$$p(f(x), f(x')) = \mathcal{N}\left(\begin{pmatrix} f(x) \\ f(x') \end{pmatrix} \middle| \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \sigma^2 \begin{pmatrix} \phi(x)^\top\phi(x) & \phi(x)^\top\phi(x') \\ \phi(x)^\top\phi(x') & \phi(x')^\top\phi(x') \end{pmatrix}\right). \tag{1.44}$$

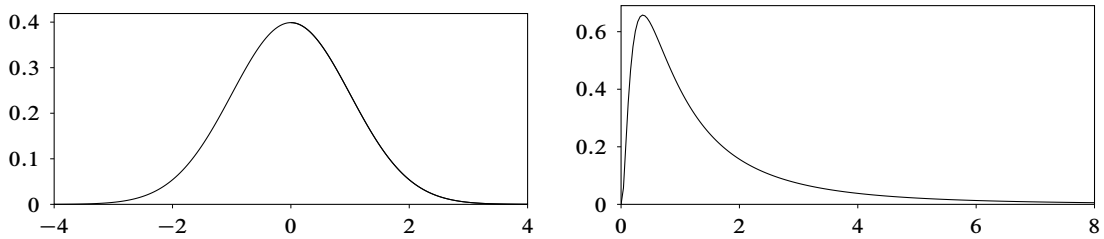
It is easy to see that this property also holds for any finite collection $(x_1, \ldots, x_m)$ of inputs, or, in other words, a finite collection $(f(x_1), \ldots, f(x_m))$ of function outputs. That is, writing $X := (x_i)_{i=1}^m$ and $f(X) := (f(x_i))_{i=1}^m$, we have a Gaussian on $\mathbb{R}^m$:

$$p(f(X)) = \mathcal{N}(f(X) \mid 0, K(X, X)), \tag{1.45}$$

where $K(X, X)$ is the covariance matrix in (1.44). This leads to the definition of ***Gaussian process***: It is a collection of random variables where any finite number of which is distributed as a joint Gaussian distribution.

In applications, often the focus is on the function outputs $f(x)$ itself, regardless of the underlying linear model. Thus, it is useful to "skip" the construction (1.4) and instead directly model the joint Gaussian. Notice that this can be done by generalizing $K(X, X)$ so that it does not depend on the feature map ϕ used in (1.4). A useful object for this is a *kernel*. A ***(positive-definite) kernel*** is a function $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ s.t. for any finite collection $X = (x_1, \ldots, x_m)$ of points in $\mathbb{R}^n$, its matrix

$$K(X, X) := \begin{pmatrix} K(x_1, x_1) & \ldots & K(x_1, x_m) \\ \vdots & \ddots & \vdots \\ K(x_m, x_1) & \ldots & K(x_m, x_m) \end{pmatrix} \tag{1.46}$$



(a) $p(x)$ **(b)** $p(\exp(x))$

Figure 1.8: A change of density under the diffeomorphism $\exp : \mathbb{R} \to (0, \infty)$.

is positive-definite. The matrix $K(X, X)$ is called the ***kernel matrix*** of K. Given a kernel K and noticing that the graph of a function f can be represented as an infinite collection $(f(x_1), f(x_2), \dots)$, we can think of a Gaussian process as a prior over function, denoted by

$$p(f) = \mathcal{GP}(f \mid 0, K). \tag{1.47}$$

where $f : \mathbb{R}^n \to \mathbb{R}$.

Gaussian processes are useful for approximating an unknown function $\mathbb{R}^n \to \mathbb{R}$ given a set of observations $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^m =: (X, Y)$ where each x_i and y_i is in $\mathbb{R}^n$ and $\mathbb{R}$, respectively—this is essentially the task of (Bayesian) regression. Suppose a Gaussian process prior $p(f) = \mathcal{GP}(0, K)$ is used and assume an observation noise $\varepsilon \sim \mathcal{N}(\varepsilon \mid 0, \sigma_y^2)$ on y. Then, the posterior predictive distribution over $f(X_*)$ under a finite collection $X_* = (x_{*1}, \dots, x_{*m_*})$ of m_* test points in $\mathbb{R}^n$, is also given by a Gaussian process with mean and covariance (Rasmussen & Williams, 2005, Sec. 2.2)

$$\mathbb{E}(f(X_*)) = K(X_*, X)\big(K(X, X) + \sigma_y^2\big)^{-1} Y \tag{1.48}$$

and

$$\mathrm{Cov}(f(X_*), f(X_*)) = K(X_*, X_*) - K(X_*, X)\big(K(X, X) + \sigma_y^2\big)^{-1} K(X, X_*). \tag{1.49}$$

See Fig. 1.7 for an illustration.

Notice that we have so far only considered a Gaussian process with a mean zero. However, (1.47) can be generalize further to take into account non-zero mean functions. For instance, Qiu et al. (2020) uses a fixed pre-trained neural network $f_{\theta_{\mathrm{MAP}}}$ as the mean function of a Gaussian process prior. In this case, the mean of Gaussian process posterior in (1.48) becomes (Rasmussen & Williams, 2005, Sec. 2.7):

$$\mathbb{E}(f(X_*)) = f_{\theta_{\mathrm{MAP}}}(X_*) + K(X_*, X)\big(K(X, X) + \sigma_y^2\big)^{-1}(Y - f_{\theta_{\mathrm{MAP}}}(X)),$$

while the covariance in (1.49) stays unchanged.

1.5 Normalizing Flows

Let $F : \mathbb{R}^n \to \mathbb{R}^n$ be a diffeomorphism—a smooth function with a smooth inverse—in the sense of $C^k(\mathbb{R}^n)$ for a fixed $k \geq 1 \in \mathbb{N}$. If $p(x)$ is a density of a random variable x in $\mathbb{R}^n$,[4] then the density q of the random variable $y = F(x)$ is given by

$$q(y) = p(F^{-1}(y)) \, |\det J_{F^{-1}}(y)|, \tag{1.50}$$

where $J_{F^{-1}} = J_F^{-1}$ is the $d \times d$ Jacobian matrix of F^{-1}. This is the so-called ***change of variable formula*** of probability density functions. See Fig. 1.8 for an example.

A ***normalizing flow*** is a method exploiting the change of variable formula (Rezende & Mohamed, 2015; Rippel & Adams, 2013; Dinh et al., 2015). Specifically, its main idea is to assume a parametrized diffeomorphism $F_\theta : \mathbb{R}^n \to \mathbb{R}^n$, constructed as a composition of simpler ones—similar to the idea of neural networks:

$$F_\theta := F^{(1)}_{\theta^{(1)}} \circ \cdots \circ F^{(L)}_{\theta^{(L)}}, \tag{1.51}$$

for some $L \in \mathbb{N}$ and $\theta \in \mathbb{R}^d$. Note that, different from standard neural networks, F_θ—the so-called ***flow***—is required to be invertible, which can be satisfied by requiring each $F^{(\ell)}_{\theta^\ell}$'s to be invertible. Given an "initial density" $p_0(\theta)$—often chosen to be a simple density such as $\mathcal{N}(\theta \mid 0, I_d)$—and using the change of variable formula, we can thus define

$$q_\theta(y) := p_0(F_\theta^{-1}(y)) \left| \prod_{\ell=1}^{L} \det J^{-1}_{F^{(\ell)}_{\theta^\ell}}(y) \right| \tag{1.52}$$

to be the resulting density under the flow F_θ. Then, this parametric density can be used for approximating an unknown density $p_*(\theta)$ by optimizing some objective function w.r.t. the parameter θ. When the objective function is the ELBO (1.29), one can think of this task as a variational approximation scheme where the feasible set of variational approximations $\Phi := \{q_\theta(y) : \theta \in \mathbb{R}^d\}$ is induced by the parameter of the flow.

An example of the flow $F^{(\ell)}_{\theta^{(\ell)}}$ is the ***radial flow*** (Rezende & Mohamed, 2015) defined by

$$F^{(\ell)}_{\theta^{(\ell)}}(x) := x + \beta h(\alpha, r)(x - x_0), \tag{1.53}$$

where $r := \|x - x_0\|$ for some norm $\|\cdot\|$, $h(\alpha, r) := 1/(\alpha + r)$, and the parameter is given by

$$\theta^{(\ell)} := \{x_0 \in \mathbb{R}^n, \alpha \in \mathbb{R}_{>0}, \beta \in \mathbb{R}\}.$$

Meanwhile, the ***planar flow***—also proposed by Rezende & Mohamed (2015)—is defined by

$$F^{(\ell)}_{\theta^{(\ell)}}(x) := x + uh(w^\top x + b), \tag{1.54}$$

where

$$\theta^{(\ell)} := \{w \in \mathbb{R}^n, u \in \mathbb{R}^n, \beta \in \mathbb{R}\}$$

is the parameter.

[4] All densities considered here are assumed to be w.r.t. the Lebesgue measure.

1.6 Predictive Uncertainty Quantification

The key advantage of Bayesian neural networks (and Bayesian predictive systems in general) compared to standard point-estimated neural networks is the fact that they produce uncertainty estimates associated with the networks' outputs. This additional information can thus be leveraged to make neural networks more reliable on tasks such as predictive uncertainty calibration and out-of-distribution data detection.

1.6.1 Calibration

Let $f : \mathbb{R}^n \to \mathbb{R}^k$ be a neural network classifier and let $\mathcal{D} := \{(x_i \in \mathbb{R}^n, y_i \in \{1, \dots, k\})\}_{i=1}^m$ dataset. For any input $x_* \in \mathbb{R}^n$, we can define the ***confidence*** associated with the network prediction on x_* to be the predictive probability associated with the predicted label, i.e.

$$\text{conf}(x_*) := \max_{i \in \{1,\dots,k\}} p(y = i \mid f(x_*), \mathcal{D}), \tag{1.55}$$

where $p(y \mid f(x_*), \mathcal{D})$ is the predictive distribution of f under $\mathcal{D}$—e.g. $p(y \mid f(x_*), \mathcal{D}) = \text{softmax}(f_{\theta_{\text{MAP}}}(x_*))$ under a MAP estimation θ_{MAP} of f or $p(y \mid f(x_*), \mathcal{D})$ equals (1.42) if we perform an approximate Bayesian inference on f.

We call the confidence estimates of the network f to be ***calibrated*** if its confidence matches the probability of its prediction being correct. For example, given 100 predictions under f, each with 0.6 confidence, then f is well-calibrated if f correctly classifies 60 of them (Guo et al., 2017). More formally, let $\widehat{y}(x) := \text{argmax}_i \, p(y = i \mid f(x), \mathcal{D})$ be the predicted label. Then, f is ***well-calibrated*** if for all $r \in [0, 1]$ (Wenger et al., 2019):

$$\mathbb{E}_{x,y \sim p(x,y)}([\widehat{y}(x) = y] \mid \text{conf}(x) = r) = r, \tag{1.56}$$

where $p(x, y)$ is the data distribution—the distribution $\mathcal{D}$ is assumed to be sampled from—and $[\cdot]$ is the Iverson bracket, i.e. it equals one if its argument is true and zero otherwise.

It is useful to measure the miscalibration of f by measuring the discrepancy between the confidence estimate and accuracy at each confidence level r. This can be done in practice by partitioning the interval $[0, 1]$ into b bins $B := \{B_i\}_{i=1}^b$ where $\bigsqcup_{i=1}^b B_i = [0, 1]$. Given a test set $\mathcal{D}_{\text{test}}$ of size m_{test}, we denote by $\text{acc}(B_i)$ and $\text{conf}(B_i)$ the accuracy and confidence of a subset of $S_i \subseteq \mathcal{D}_{\text{test}}$ whose confidences all fall into the bin B_i, respectively. Then, we define the ***expected calibration error (ECE)*** metric (Naeini et al., 2015) by

$$\text{ECE} := \sum_{i=1}^{b} \frac{|S_i|}{m_{\text{test}}} |\text{acc}(B_i) - \text{conf}(B_i)|, \tag{1.57}$$

where $|S_i|$ is the size of the subset S_i. See Fig. 1.9 for an illustration.

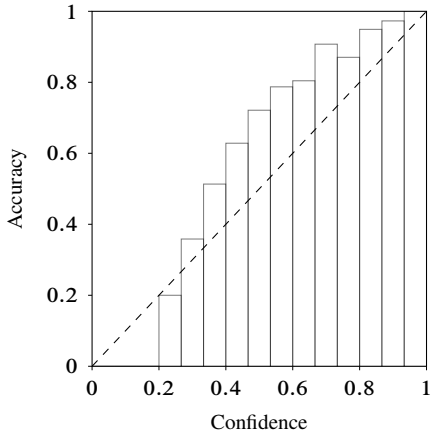
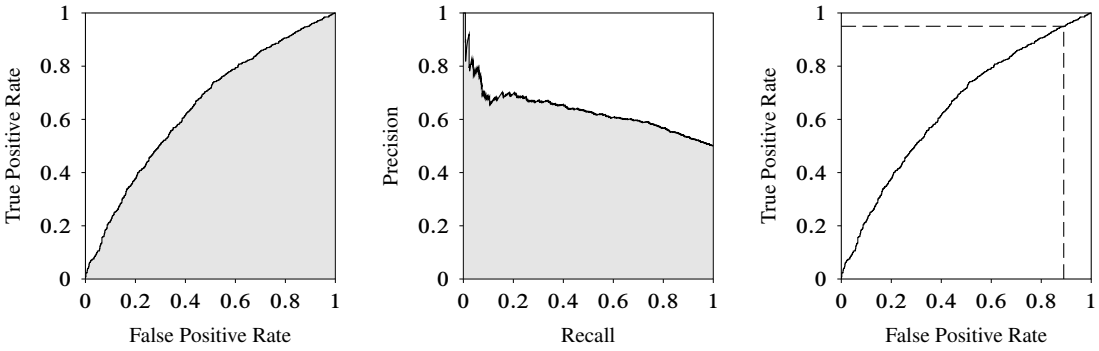


Figure 1.9: The ECE metric is the sum of the absolute discrepancy between the confidence and accuracy in each bin.



(a) AUROC **(b)** AUPRC **(c)** FPR@95

Figure 1.10: The AUROC, AUPRC, and FPR@95 metrics. The area-under-curves are the shaded regions.

Alternatively, one can use the ***Brier score*** to evaluate the calibration of a model (Brier et al., 1950), defined by the mean-squared error between the predicted probability vectors $p(y \mid f(x_*), \mathcal{D})$ and the one-hot encoded true label y_* under a test set $D_{\text{test}} = \{(x_*, y_*)\}$:

$$\text{Brier} := \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{x_*, y_* \in \mathcal{D}_{\text{test}}} (p(y \mid f(x_*), \mathcal{D}) - y_*)^2. \tag{1.58}$$

Finally, one can also use the ***(Categorical) negative log-likelihood (NLL)*** score under $\mathcal{D}_{\text{test}}$:

$$\text{NLL} := -\frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{x_*, y_* \in \mathcal{D}_{\text{test}}} \sum_{i=1}^{k} y_{*i} \log p(y = i \mid f(x_*), \mathcal{D}), \tag{1.59}$$

where again we have assumed the label y_* is one-hot encoded.

1.6.2 Out-of-Distribution Data Detection

The task of ***out-of-distribution data detection***, or OOD detection for short, is concerned with the question of how well a (neural network) classifier distinguishes between data sampled from the same distribution as the training data $\mathcal{D}$ and data sampled from another distribution. The former are called the ***in-distribution (ID) data*** while the latter are called the ***OOD data***.

Notice that we can formulate OOD detection as a binary classification problem. Assume without loss of generality that we assign the label 1 if an input point x is an ID point and the label 0 otherwise. Then, we can assign a threshold $\tau \in [0, 1]$ to obtain a binary classifier where the prediction $\widehat{y}$ of a point $x \in \mathbb{R}^n$ is given by

$$\widehat{y}(x) := \begin{cases} 0 & \text{if } \text{conf}(x) < \tau, \\ 1 & \text{if } \text{conf}(x) \geq \tau, \end{cases} \tag{1.60}$$

where the confidence function is defined as in (1.55)—in particular, it depends on the formulation of the predictive distribution of the network f. Note that, $\widehat{y}(x)$ predicts whether x is ID or OOD.

To measure how useful the confidence estimates of f are for distinguishing ID against OOD data, one can use standard metrics used in classic binary classification tasks. Let $\mathcal{D}_{\text{in}}$ and $\mathcal{D}_{\text{out}}$ be ID and OOD test sets, respectively, and let $\mathcal{D}_{\text{test}} := \mathcal{D}_{\text{in}} \amalg \mathcal{D}_{\text{out}}$. A popular metric for OOD detection is the ***area under the receiver operating characteristic curve (AUROC)***. The receiver operating characteristic (ROC) curve itself is defined by taking into account the true-positive and false-positive rates of the binary classifier $\widehat{y}$ under the test set $\mathcal{D}_{\text{test}}$. A commonly-used alternative to the AUROC metric is the ***area under the precision-recall curve (AUPRC)***, which is defined by simply replacing the false-positive rates above with the precision of $\widehat{y}$. For both the area-under-curve metrics above, higher values are better. Finally, one can also fix a particular threshold value $\widehat{\tau}$ and evaluate the ROC curve at that particular true-positive rate. The resulting metric is called the ***FPR@$\widehat{\tau}$TPR*** or simply ***FPR$\widehat{\tau}$***. Commonly, $\widehat{\tau}$ is chosen to be 95 percent (Hein et al., 2019; Meinke & Hein, 2020). Note that, lower FPR$\widehat{\tau}$ values are better for a fixed $\widehat{\tau}$. See Fig. 1.10 for illustrations.

Chapter 2
Laplace Approximations for Deep Learning

The contents of this chapter are primarily based on:

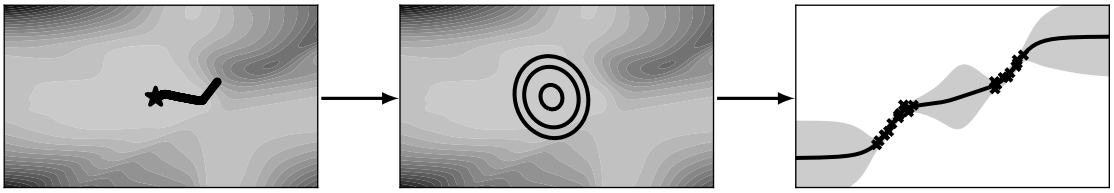
> Erik Daxberger*, Agustinus Kristiadi*, Alexander Immer*, Runa Eschenhagen*, Matthias Bauer, and Philipp Hennig. Laplace Redux—Effortless Bayesian Deep Learning. Advances in Neural Information Processing Systems (NeurIPS), 2021.[1]

	Idea	**Analysis**	**Experiment**	**Code**	**Writing**
Erik Daxberger*	16.7%	20%	25%	25%	20%
Agustinus Kristiadi*	16.7%	20%	25%	25%	20%
Alexander Immer*	16.7%	20%	25%	25%	20%
Runa Eschenhagen*	16.7%	20%	25%	25%	20%
Matthias Bauer	16.7%	10%	0%	0%	15%
Philipp Hennig	16.7%	10%	0%	0%	5%

Despite their successes, modern neural networks (NNs) still suffer from several shortcomings that limit their applicability in some settings. These include (i) poor calibration and overconfidence, especially when the data distribution shifts between training and testing (Guo et al., 2017), (ii) catastrophic forgetting of previously learned tasks when continuously trained on new tasks (Kirkpatrick et al., 2017), and (iii) the difficulty of selecting suitable NN architectures and hyperparameters (Hutter et al., 2019). Bayesian modeling (Barber, 2012; Ghahramani, 2015) provides a principled and unified approach to tackle these issues by (i) equipping models with robust uncertainty estimates (Gal & Ghahramani, 2016), (ii) enabling models to learn continually by capturing past information (Nguyen et al., 2018), and (iii) allowing for automated model selection by optimally trading off data fit and model complexity (MacKay, 1995).

Even though this provides compelling motivation for using Bayesian neural networks (BNNs), they have not gained much traction in practice. Common criticisms include that BNNs are difficult to implement, finicky to tune, expensive to train, and hard to scale to modern models and datasets. For instance, popular variational Bayesian methods (Hinton & Van Camp, 1993; Graves, 2011; Blundell et al., 2015, etc.) require considerable changes to the training procedure

[1] Asterisk indicates randomly-ordered equal-contribution authors.



(a) MAP Estimation (b) Laplace Approximation (c) Prediction

Figure 2.1: Probabilistic predictions with the Laplace approximation. **(a)** We find a MAP estimate (star) via standard training (background contours is the log-posterior landscape on the parameter space). **(b)** We locally approximate the posterior landscape by fitting a Gaussian centered at the MAP estimate, with covariance matrix equal to the negative inverse Hessian of the loss at the MAP—this is the Laplace approximation. **(c)** We use the this Gaussian to make predictions with error bars—here, the black curve is the predictive mean, and the shading covers the 95% confidence interval.

and model architecture. Also, their optimization process is slower and typically more unstable unless carefully tuned (Osawa et al., 2019). Other methods, such as deep ensembles (Lakshminarayanan et al., 2017), Monte Carlo dropout (Gal & Ghahramani, 2016), and Gaussian stochastic weight averaging (Maddox et al., 2019a) promise to bring uncertainty quantification to standard NNs in simple manners. But these methods either require a significant cost increase compared to a single network, have limited empirical performance, or an unsatisfying Bayesian interpretation.

In this chapter we argue that the Laplace approximation (LA) is a simple and cost-efficient, yet competitive approximation method for inference in Bayesian deep learning. First proposed in this context by MacKay (1992a), the LA dates back to the 18th century (Laplace, 1774). Recall from Section 1.3.1 that it locally approximates the posterior with a Gaussian distribution centered at a local maximum, with covariance matrix corresponding to the local curvature. Two key advantages of the LA are that the local maximum is readily available from standard *maximum a posteriori* (MAP) training of NNs (Section 1.2.1), and that curvature estimates can be easily and efficiently obtained thanks to recent advances in second-order optimization, both in terms of more efficient approximations to the Hessian (Heskes, 2000; Martens & Grosse, 2015; Botev et al., 2017) and easy-to-use software libraries (Dangel et al., 2020; Osawa, 2021b). Together, they make the LA practical and readily applicable to many already-trained NNs—the LA essentially enables practitioners to turn their high performing point-estimate NNs into BNNs easily and quickly, without loss of predictive performance. Furthermore, the Laplace approximation to the marginal likelihood may even be used for Bayesian model selection or NN training (MacKay, 1995; Immer et al., 2021a). Figure 2.1 provides an intuition of the LA: we first fit a point estimate of the model, and then estimate a Gaussian distribution around that.

Yet, despite recent progress in scaling and improving the LA for deep learning (Ritter et al., 2018a;b; Khan et al., 2019; Immer et al., 2021b; Daxberger et al., 2021b; Kristiadi et al., 2020; Lee et al., 2020), it is far less widespread than other methods. This is likely due to misconceptions, like that the LA is hard to implement due to the Hessian computation, that it must necessarily perform worse than the competitors due to its local nature, or quite simply that it is old and too simple. Here, we show that these are indeed misconceptions. Moreover, we argue that the LA deserves a wider adoption in both practical and research-oriented deep learning.

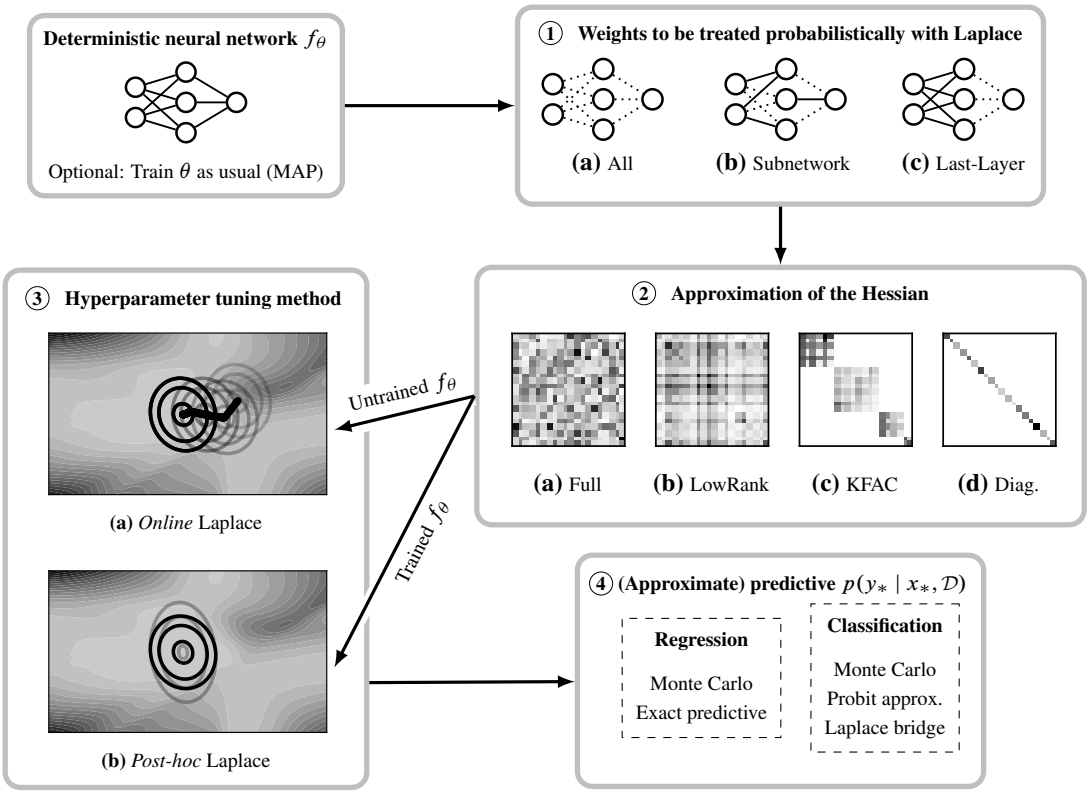


Figure 2.2: Four key components to scale and apply the LA to a neural network f_θ (with randomly-initialized or pre-trained weights θ). ① First, choose which part of the network we want to perform inference over with the LA. ② Then, select how to to approximate the Hessian. ③ We can then perform model selection using the evidence: **(a)** If we started with an untrained model f_θ, we can jointly train the model and use the evidence to tune hyperparameters *online*. **(b)** If we started with a pre-trained model, we can use the evidence to tune the hyperparameters *post-hoc*. Here, shades represent the loss landscape, while contours represent LA log-posteriors—faded contours represent intermediate iterates during hyperparameter tuning to obtain the final log-posterior (thick contours). ④ Finally, to make predictions for a new input, we can use the options described in Section 1.3.4.

2.1 Modern Laplace Approximations in Deep Learning

The LA can be used in two different ways to benefit deep learning: Firstly, we can use the LA to approximate the model's *posterior distribution* to enable *probabilistic predictions* (as also illustrated in Fig. 2.1) or other downstream applications such as continual learning (Ritter et al., 2018b). Secondly, we can use the LA to approximate the *model evidence* to enable *model selection* (e.g. hyperparameter tuning).

The canonical form of (supervised) deep learning is that of empirical risk minimization. Given, e.g., an i.i.d. classification dataset $\mathcal{D} := \{(x_i \in \mathbb{R}^n, y_i \in \mathbb{R}^k)\}_{i=1}^m$, the weights $\theta \in \mathbb{R}^d$ of an L-layer NN $f_\theta : \mathbb{R}^n \to \mathbb{R}^k$ are trained to minimize the (regularized) empirical risk, which typically decomposes into a sum over empirical loss terms $\ell(x_i, y_i; \theta)$ and a regularizer $r(\theta)$,

$$\theta_{\text{MAP}} = \operatorname{argmin}_{\theta \in \mathbb{R}^d} \mathcal{L}(\mathcal{D}; \theta) = \operatorname{argmin}_{\theta \in \mathbb{R}^d} \big(r(\theta) + \textstyle\sum_{i=1}^m \ell(x_i, y_i; \theta) \big). \tag{2.1}$$

From the Bayesian viewpoint, these terms can be identified with i.i.d. log- ***likelihoods*** and a log-***prior***, respectively and, thus, θ_{MAP} is indeed a ***maximum a-posteriori (MAP)*** estimate:

$$\ell(x_i, y_i; \theta) = -\log p(y_i \mid f_\theta(x_i)) \qquad \text{and} \qquad r(\theta) = -\log p(\theta) \tag{2.2}$$

For example, the widely used weight regularizer $r(\theta) = \frac{1}{2}\gamma^{-2}\|\theta\|^2$ (a.k.a. weight decay) corresponds to a centered Gaussian prior $p(\theta) = \mathcal{N}(\theta \mid 0, \gamma^2 I)$, and the cross-entropy loss amounts to a categorical likelihood. Hence, the exponential of the negative training loss $\exp(-\mathcal{L}(\mathcal{D}; \theta))$ amounts to an ***unnormalized posterior***. By normalizing it, we obtain

$$p(\theta \mid \mathcal{D}) = \tfrac{1}{Z}\, p(\mathcal{D} \mid \theta)\, p(\theta) = \tfrac{1}{Z} \exp(-\mathcal{L}(\mathcal{D}; \theta)), \qquad Z := \int p(\mathcal{D} \mid \theta)\, p(\theta)\, d\theta \tag{2.3}$$

with an intractable ***normalizing constant*** Z. ***Laplace approximations*** (Laplace, 1774) use a second-order expansion of $\mathcal{L}$ around θ_{MAP} to construct a Gaussian approximation to $p(\theta \mid \mathcal{D})$. I.e., we consider:

$$\mathcal{L}(\mathcal{D}; \theta) \approx \mathcal{L}(\mathcal{D}; \theta_{\text{MAP}}) + \tfrac{1}{2}(\theta - \theta_{\text{MAP}})^\top \big(\nabla^2_\theta \mathcal{L}(\mathcal{D}; \theta)|_{\theta_{\text{MAP}}}\big)(\theta - \theta_{\text{MAP}}), \tag{2.4}$$

where the first-order term vanishes at θ_{MAP}. Then we can identify the Laplace approximation as

$$p(\theta \mid \mathcal{D}) \approx \mathcal{N}(\theta \mid \theta_{\text{MAP}}, \Sigma) \qquad \text{with} \qquad \Sigma := \big(\nabla^2_\theta \mathcal{L}(\mathcal{D}; \theta)|_{\theta_{\text{MAP}}}\big)^{-1}. \tag{2.5}$$

The normalizing constant Z (which is typically referred to as the ***marginal likelihood*** or ***evidence***) is useful for model selection and can also be approximated as

$$Z \approx \exp(-\mathcal{L}(\mathcal{D}; \theta_{\text{MAP}}))\, (2\pi)^{D/2}\, (\det \Sigma)^{1/2}. \tag{2.6}$$

Thus, to obtain the approximate posterior, we first need to find the argmax θ_{MAP} of the log-posterior function, i.e. do "standard" deep learning with regularized empirical risk minimization. The only *additional* step is to compute the inverse of the Hessian matrix at θ_{MAP} (see Fig. 2.1(b)). The LA can therefore be constructed *post-hoc* to a pre-trained network, even one downloaded off-the-shelf. As we discuss below, the Hessian computation can be offloaded to recently advanced automatic differentiation libraries (Dangel et al., 2020). LAs are widely used to approximate the posterior distribution in logistic regression (Spiegelhalter & Lauritzen, 1990), Gaussian process classification (Williams & Barber, 1998; Rasmussen & Williams, 2005), and also for Bayesian neural networks (BNNs), both shallow (MacKay, 1992b) and deep (Ritter et al., 2018a). The latter is the focus of this work.

Generally, any prior with twice differentiable log-density can be used. Due to the popularity of the weight decay regularizer, we assume that the prior is a zero-mean Gaussian $p(\theta) = \mathcal{N}(\theta \mid 0, \gamma^2 I)$ unless stated otherwise.[2] The Hessian $\nabla^2_\theta \mathcal{L}(\mathcal{D}; \theta)|_{\theta_{\text{MAP}}}$ then depends both on the (simple) log-prior / regularizer and the (complicated) log-likelihood / empirical risk:

$$\nabla^2_\theta \mathcal{L}(\mathcal{D}; \theta)|_{\theta_{\text{MAP}}} = -\gamma^{-2} I - \textstyle\sum_{i=1}^{m} \nabla^2_\theta \log p(y_i \mid f_\theta(x_i))|_{\theta_{\text{MAP}}}. \tag{2.7}$$

A naive implementation of the Hessian is infeasible because the second term in (2.7) scales quadratically with the number of network parameters, which can be in the millions or even

[2] One can also consider a per-layer or even per-parameter weight decay, which corresponds to a more general, but still comparably simple Gaussian prior. In particular, the Hessian of this prior is still diagonal and constant.

billions (He et al., 2016; Shoeybi et al., 2019). In recent years, several works have addressed scalability, as well as other factors that affect approximation quality and predictive performance of the LA. In the following, we identify, review, and discuss four key components that allow LAs to scale and perform well on modern deep architectures. See Fig. 2.2 for an overview and Section 2.1 for a more detailed version of the review and discussion. Further discussion regarding (approximate) predictive distributions is in Section 1.3.4.

Components of Scalable Laplace Approximations for Deep Neural Networks

① Inference over all Weights or Subsets of Weights

In most cases, it is possible to treat *all* weights probabilistically when using appropriate approximations of the Hessian, as we discuss below in ②. Another simple way to scale the LA to large NNs (without Hessian approximations) is the ***subnetwork LA*** (Daxberger et al., 2021b), which only treats a *subset* of the model parameters probabilistically with the LA and leaves the remaining parameters at their MAP-estimated values. An important special case of this applies the LA to only the *last linear layer* of an L-layer NN, while fixing the feature extractor defined by the first $L-1$ layers at its MAP estimate (Snoek et al., 2015; Kristiadi et al., 2020). This ***last-layer LA*** is cost-effective yet compelling both theoretically and in practice (Kristiadi et al., 2020).

② Hessian Approximations and Their Factorizations

One advance in second-order optimization that the LA can benefit from are positive semi-definite approximations to the (potentially indefinite) Hessian of the log-likelihoods of NNs in the second term of (2.7) (Martens, 2020). The ***Fisher information matrix*** (Amari, 1998), abbreviated as *the Fisher* and defined by

$$F := \sum_{i=1}^{m} \mathbb{E}_{\widehat{y}\sim p(y|f_\theta(x_i))}\big[(\nabla_\theta \log p(\widehat{y} \mid f_\theta(x_i))|_{\theta_{\text{MAP}}})(\nabla_\theta \log p(\widehat{y} \mid f_\theta(x_i))|_{\theta_{\text{MAP}}})^\top\big], \quad (2.8)$$

is one such choice.[3] One can also use the ***generalized Gauss-Newton matrix (GGN)*** matrix (Schraudolph, 2002)

$$G := \sum_{i=1}^{m} J(x_i)\Big(\nabla_f^2 \log p(y_i \mid f)|_{f=f_{\theta_{\text{MAP}}}(x_i)}\Big)J(x_i)^\top, \quad (2.9)$$

where $J(x_i) := \nabla_\theta f_\theta(x_i)|_{\theta_{\text{MAP}}}$ is the NN's Jacobian matrix. As the Fisher and GGN are equivalent for common log-likelihoods (Martens, 2020), we will henceforth refer to them interchangeably. In deep LAs, they have emerged as the default choice (Ritter et al., 2018a;b; Kristiadi et al., 2020; Lee et al., 2020; Daxberger et al., 2021b; Immer et al., 2021b, etc.).

As F and G are still quadratically large, we typically need further factorization assumptions. The most lightweight is a ***diagonal factorization*** which ignores off-diagonal elements (LeCun et al., 1990; Denker & LeCun, 1990). More expressive alternatives are block-diagonal factorizations such as ***Kronecker-factored approximate curvature (KFAC)*** (Heskes, 2000; Martens & Grosse, 2015; Botev et al., 2017), which factorizes each within-layer Fisher[4] as a Kronecker

[3] If, instead of taking expectation in (2.8), we use the training label y_i, we call the matrix the ***empirical Fisher***, which is distinct from the Fisher (Martens, 2020; Kunstner et al., 2019).

[4] The elements F or G corresponding to the weight $W_l \subseteq \theta$ of the ℓ-th layer of the network.

product of two smaller matrices. KFAC has been successfully applied to the LA (Ritter et al., 2018a;b) and can be improved by low-rank approximations of the KFAC factors (Lee et al., 2020) by leveraging their eigendecompositions (George et al., 2018). Finally, recent work has studied/enabled ***low-rank approximations*** of the Hessian/Fisher (Madras et al., 2020; Maddox et al., 2020; Sharma et al., 2021).

③ Hyperparameter Tuning

As with all approximate inference methods, the performance of the LA depends on the (hyper)parameters of the prior and likelihood. For instance, it is typically beneficial to tune the prior variance γ^2 used for inference (Ritter et al., 2018a; Kristiadi et al., 2020; Daxberger et al., 2021b; Immer et al., 2021b;a). Commonly, this is done through ***cross-validation***, e.g. by maximizing the validation log-likelihood (Ritter et al., 2018a; Foong et al., 2019) or, additionally, using out-of-distribution data (Kristiadi et al., 2020; 2021). When using the LA, however, ***marginal likelihood maximization*** (a.k.a. ***empirical Bayes*** or ***the evidence framework*** (MacKay, 1992b; Bernardo & Smith, 2009)) constitutes a more principled alternative to tune these hyperparameters, and requires no validation data. Immer et al. (2021a) showed that marginal likelihood maximization with LA can work in deep learning and even be performed in an online manner jointly with the MAP estimation. Note that such approach is not necessarily feasible for other approximate inference methods because most do not provide an estimate of the marginal likelihood. Other recent approaches for hyperparameter tuning for the LA include Bayesian optimization (Humt et al., 2020) or the addition of dedicated, trainable hidden units for the sole purpose of uncertainty tuning (Kristiadi et al., 2021).

④ Approximate Predictive Distribution

To predict using a posterior (approximation) $p(\theta \mid \mathcal{D})$, we need to compute $p(y \mid f(x_*), \mathcal{D}) = \int p(y \mid f_\theta(x_*))\, p(\theta \mid \mathcal{D})\, d\theta$ for any test point $x_* \in \mathbb{R}^n$, which is intractable in general. The simplest but most general approximation to $p(y \mid x_*, \mathcal{D})$ is Monte Carlo integration using s samples $(\theta_i)_{i=1}^s$ from $p(\theta \mid \mathcal{D})$: $p(y \mid f(x_*), \mathcal{D}) \approx s^{-1} \sum_{i=1}^s p(y \mid f_{\theta_i}(x_*))$. However, for LAs with GGN and Fisher Hessian approximations Monte Carlo integration can perform poorly (Foong et al., 2019; Immer et al., 2021b). Immer et al. (2021b) attribute this to the inconsistency between Hessian approximation and the predictive and suggest to use a linearized predictive instead, which can also be useful for theoretic analyses (Kristiadi et al., 2020). For the last-layer LA, the Hessian coincides with the GGN and the linearized predictive is exact.

The predictive of a ***linearized neural network*** with a LA approximation to the posterior $p(\theta \mid \mathcal{D}) \approx \mathcal{N}(\theta \mid \theta_{\text{MAP}}, \Sigma)$ results in a Gaussian distribution on neural network outputs $f_* := f(x_*)$ and therefore enables simple approximations or even a closed-form solution. The distribution on the outputs is given by $p(f_* \mid x_*, \mathcal{D}) \approx \mathcal{N}(f_* \mid f_{\theta_{\text{MAP}}}(x_*), J(x_*)^\top \Sigma J(x_*))$ and is typically significantly lower-dimensional (number of outputs C instead of parameters D). It can also be inferred entirely in function space as a Gaussian process (Khan et al., 2019; Immer et al., 2021b). Given the distribution on outputs f_*, the predictive distribution can be obtained by integration against the likelihood: $p(y \mid x_*, \mathcal{D}) = \int p(y \mid f_*) p(f_* \mid x_*, \mathcal{D})\, d\theta$. In the case of regression with a Gaussian likelihood with variance σ^2, the solution can even be obtained analytically: $p(y \mid x_*, \mathcal{D}) \approx \mathcal{N}(y \mid f_{\theta_{\text{MAP}}}(x_*), J(x_*)^\top \Sigma J(x_*) + \sigma^2 I)$. For non-Gaussian likelihoods, e.g. in classification, a further approximation is needed. Again, the simplest approximation to this is ***Monte Carlo integration***. In the binary case, we can employ the ***probit***

```python
from laplace import Laplace

# Load pre-trained model
model = load_map_model()

# Define and fit LA variant with custom settings
la = Laplace(
    model, 'classification',
    subset_of_weights='all', hessian_structure='diag'
)
la.fit(train_loader)

# Hyperparameter tuning via cross-validation
la.optimize_prior_precision(method='CV', val_loader=val_loader)

# Make prediction with custom predictive approx.
pred = la(x, pred_type='glm', link_approx='probit')
```

Listing 2.1: Fit diagonal LA over all weights of a pre-trained classification model, do *post-hoc* tuning of the prior precision hyperparameter using cross-validation, and make a prediction for input x with the probit approximation.

approximation (Spiegelhalter & Lauritzen, 1990; MacKay, 1992a) which approximates the logistic function with the probit function. In the multi-class case, we can use its generalization, the ***extended probit approximation*** (Gibbs, 1998). Finally, first proposed for non-BNN applications (MacKay, 1998; Hennig et al., 2012), the ***Laplace bridge*** approximates the softmax-Gaussian integral via a Dirichlet distribution (Hobbhahn et al., 2022). The key advantage is that it yields a *distribution* of the integral solutions.

2.2 The `laplace-torch` toolkit

Implementing the LA is non-trivial, as it requires efficient computation and storage of the Hessian. While this is not fundamentally difficult, there exists no complete, easy-to-use, and standardized implementation of various LA flavors—instead, it is common for deep learning researchers to repeatedly re-implement the LA and Hessian computation with varying efficiency (Maddox et al., 2019b; Kristiadi, 2020; Lee & Humt, 2020, etc.). An efficient implementation typically requires hundreds of lines of code, making it hard to quickly prototype with the LA. To address this, we introduce `laplace-torch`: a simple, easy-to-use, extensible library for scalable LAs of deep NNs in PyTorch (Paszke et al., 2019). The `laplace-torch` library enables *all* sensible combinations of the four components discussed in Fig. 2.2. Listings 2.1 to 2.4 show code examples.

The core of `laplace-torch` consists of efficient implementations of the LA's key quantities: (i) posterior (i.e. Hessian computation and storage), (ii) marginal likelihood, and (iii) posterior predictive. For (i), to take advantage of advances in automatic differentiation, we outsource the Hessian computation to state-of-the-art, optimized second-order optimization libraries: BackPACK (Dangel et al., 2020) and ASDL (Osawa, 2021a). Moreover, we design `laplace-torch` in a modular manner that makes it easy to add new backends and approximations in the future. For (ii), we follow Immer et al. (2021a) in our implementation of the LA's marginal likelihood—it is thus both efficient and differentiable and allows the user to implement both *online* and *post-hoc* marginal likelihood tuning, cf. Listing 2.3. Note that `laplace-torch`

```python
from laplace import Laplace

# Load un- or pre-trained model
model = load_map_model()

# Fit default, recommended LA variant, last-layer KFAC LA
la = Laplace(model, 'regression')
la.fit(train_loader)

# Post-hoc marginal likelihood
la.optimize_prior_precision(method='marglik')
```

Listing 2.2: Fitting a KFAC LA over the last layer of a pre- or un-trained regression model and performing a *post-hoc* hyperparameter tuning via marginal likelihood maximization.

also supports standard cross-validation for hyperparameter tuning (Ritter et al., 2018a; Kristiadi et al., 2020), as shown in Listing 2.1. Finally, for (iii), `laplace-torch` supports all approximations to the posterior predictive distribution discussed shown in Fig. 2.2—it thus provides the user with flexibility in making predictions, depending on the computational budget.

Default behavior To abstract away from a large number of options available, we provide the following default choices based on our extensive experiments (Section 2.3); they should be applicable and perform decently in the majority of use cases: we assume a pre-trained network and treat only the last-layer weights probabilistically (last-layer LA), use the KFAC factorization of the GGN and tune the hyperparameters *post-hoc* using empirical Bayes. To make predictions, we use the closed-form Gaussian predictive distribution for regression and the (extended) probit approximation for classification. Of course, the user can pick custom choices (Listings 2.1 and 2.3).

Limitations Because `laplace-torch` employs external libraries (BackPACK (Dangel et al., 2020) and ASDL (Osawa, 2021a)) as backends, it inherits the available choices of Hessian factorizations from these libraries. For instance, the LA variant proposed by Lee et al. (2020) can currently not be implemented via `laplace-torch`, because neither backend supports eigenvalue-corrected KFAC (George et al., 2018) (yet).

2.3 Applications

We benchmark various LAs implemented via `laplace-torch`. Section 2.3.1 addresses the question of "which are the best design choices for the LA", in light of Fig. 2.2. Section 2.3.2 shows that the LA is competitive to strong Bayesian baselines in in-distribution, dataset-shift, and out-of-distribution (OOD) settings. We then showcase some applications of the LA in downstream tasks. Section 2.3.3 demonstrates the applicability of the (last-layer) LA on various data modalities and NN architectures (including transformers (Vaswani et al., 2017))—settings where other Bayesian methods are challenging to implement. Section 2.3.4 shows how the LA can be used as an easy-to-use yet strong baseline in continual learning. In all results, arrows behind metric names denote if lower (↓) or higher (↑) values are better.

```python
from laplace import Laplace

prec0 = nn.Parameter(...)  # Or its log
hyper_optimizer = ...

for train_epoch in range(n_train_epochs):
    for x, y in train_loader:
        out = model(x)
        loss = -loglik(out, y) - logprior(prec0)
        loss.backward()
        optimizer.step()

    # Tune prior precision with LA marginal likelihood
    la = Laplace(model, 'classification', prior_precision=prec0)
    la.fit(train_loader)

    for tune_epoch in range(n_tune_epochs):
        lml = -la.log_marginal_likelihood(prec0)
        lml.backward()
        hyper_optimizer.step()
```

Listing 2.3: Online empirical Bayes (Immer et al., 2021a) with `laplace-torch`.

2.3.1 Choosing the Right Laplace Approximation

In Section 2.1 we presented multiple options for each component of the design space of the LA, resulting in a large number of possible combinations, all of which are supported by `laplace-torch`. Here, we try to reduce this complexity and make suggestions for sensible default choices that cover common application scenarios. To this end, we performed a comprehensive comparison between most variants; we measured in- and out-of-distribution performance on standard image classification benchmarks (MNIST, FashionMNIST, CIFAR-10) but also considered the computational complexity of each variant.

Hyperparameter tuning and parameter inference. We can apply the LA purely *post-hoc* (only tune hyperparameters of a pre-trained network) or online (tune hyperparameters and train the network jointly, as e.g. suggested by Immer et al. (2021a)). We find that the online LA only works reliably when it is applied to all weights of the network. In contrast, applying the LA *post-hoc* only on the last layer instead of all weights typically yields better performance due to less underfitting, and is significantly cheaper. For problems where a pre-trained network or optimal hyperparameters are available, e.g. for well-studied data sets, we, therefore, suggest using the *post-hoc* variant on the last layer. This LA has the benefit that it has minimal overhead over a standard neural network forward pass (cf. Fig. 2.6) while performing on par or better than state-of-the-art approaches (cf. Fig. 2.5). When hyperparameters are unknown or no validation data is available, we suggest training the neural network online by optimizing the marginal likelihood, following Immer et al. (2021a) (cf Listing 2.3). Figure 2.3 illustrates this on CIFAR-10: for CIFAR-10 with data augmentation, strong pre-trained networks and hyperparameters are available and the *post-hoc* methods directly profit from that while the online methods merely reach the same performance. On the less studied CIFAR-10 without data augmentation, the online method can improve the performance over the *post-hoc* methods.

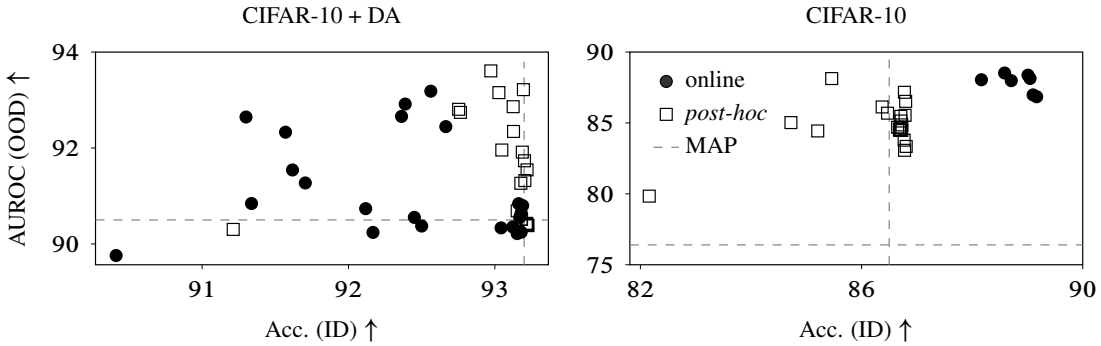


Figure 2.3: In- vs. out-of-distribution (ID and OOD, resp.) performance on CIFAR-10 of different LA configurations (dots), each being a combination of settings for 1) subset-of-weights, 2) covariance structure, 3) hyperparameter tuning, and 4) predictive approximation. "DA" stands for "data augmentation". Post-hoc performs better with DA and a strong pre-trained network, while online performs better without DA where optimal hyperparameters are unknown.

Covariance approximation and structure. Generally, we find that a more expressive covariance approximation improves performance, as would be expected. However, a full covariance is in most cases intractable for full networks or networks with large last layers. The KFAC structured covariance provides a good trade-off between expressiveness and speed. Diagonal approximations perform significantly worse than KFAC and are therefore not suggested. Independent of the structure, we find that the empirical Fisher (EF) approximations perform better on out-of-distribution detection tasks while GGN approximations tend to perform better on in-distribution metrics.

Predictive distribution. Considering in- and out-of-distribution (OOD) performance as well as cost, the probit provides the best approximation to the predictive for the last-layer LA. MC integration can sometimes be superior for OOD detection but at an increased computational cost. The Laplace bridge has the same cost as the probit approximation but typically provides inferior results in our experiments. When using the LA online to optimize hyperparameters, we find that the resulting MAP predictive provides good performance in-distribution, but a probit or MC predictive improves OOD performance.

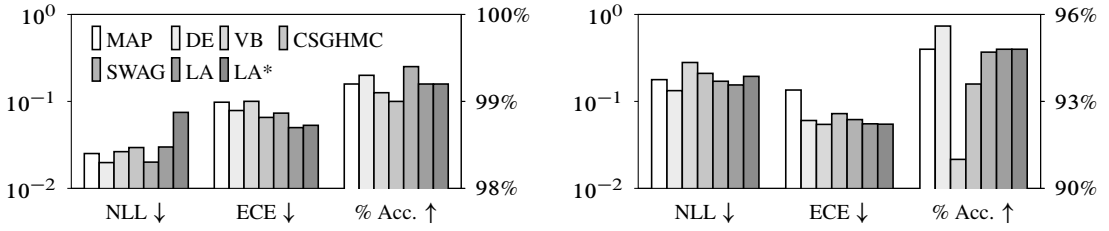
Overall recommendation. Following the experimental evidence, the default Laplace approximation in `laplace-torch` is a *post-hoc* KFAC last-layer LA with a GGN approximation to the Hessian. This default is applicable to all architectures that have a fully-connected last layer and can be easily applied to pre-trained networks. For problems where trained networks are unavailable or hyperparameters are unknown, the online KFAC LA with a GGN or empirical Fisher provides a good baseline with minimal effort.

2.3.2 Predictive Uncertainty Quantification

We consider two flavors of LAs: the default flavor of `laplace-torch` (**LA**) and the most robust one in terms of distribution shift found in Section 2.3.1 (**LA***—last-layer, with a full empirical Fisher Hessian approximation, and the probit approximation). We compare them with the MAP network (**MAP**) and various popular and strong Bayesian baselines: Deep Ensemble

Table 2.1: OOD detection performance averaged over all test sets (see Appendix B.2 for details). Confidence is defined as the maximum of the predictive probability vector (Hendrycks & Gimpel, 2017). LA and especially LA* reduce the overconfidence of MAP and achieve better results than the VB, CSGHMC, and SWAG baselines.

	Confidence ↓		AUROC ↑	
Methods	**MNIST**	**CIFAR-10**	**MNIST**	**CIFAR-10**
MAP	75.0±0.4	76.1±1.2	96.5±0.1	92.1±0.5
DE	65.7±0.3	65.4±0.4	97.5±0.0	94.0±0.1
VB	73.2±0.8	58.8±0.7	95.8±0.2	88.7±0.3
CSGHMC	69.2±1.7	69.4±0.6	96.1±0.2	90.6±0.2
SWAG	75.8±0.3	68.1±2.3	96.5±0.1	91.3±0.8
LA	67.5±0.4	69.0±1.3	96.2±0.2	92.2±0.5
LA*	56.1±0.5	55.7±1.2	96.4±0.2	92.4±0.5



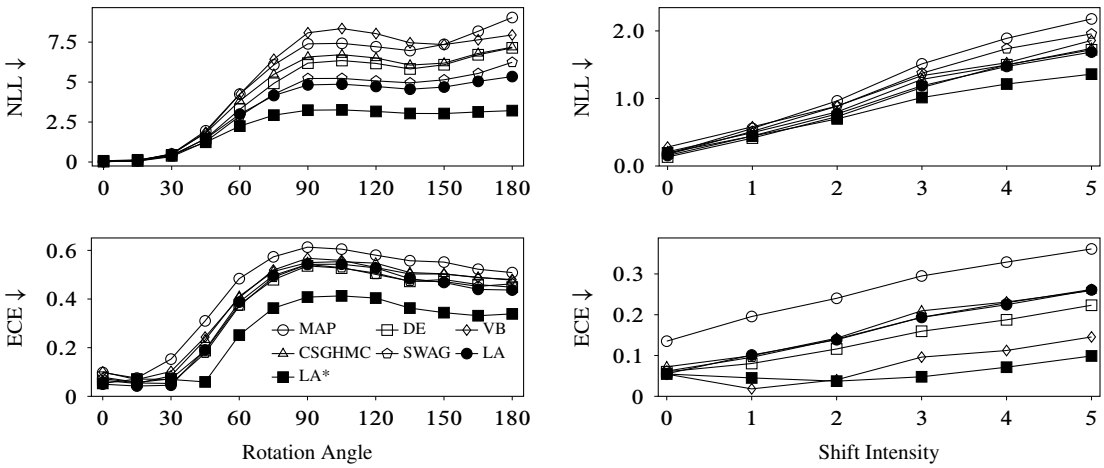
(a) MNIST **(b)** CIFAR-10

Figure 2.4: Assessing model calibration on in-distribution data. We report accuracy and, to measure calibration, negative log-likelihood (NLL) and expected calibration error (ECE)—all evaluated on the standard test sets. LA is the best-calibrated method in terms of ECE, while also retaining the accuracy of MAP (unlike VB and CSGHMC).

(**DE**, Lakshminarayanan et al., 2017), mean-field variational Bayes (**VB**, Graves, 2011; Blundell et al., 2015) with the flipout estimator (Wen et al., 2018), cyclical stochastic-gradient Hamiltonian Monte Carlo (**CSGHMC**, Zhang et al., 2020), and Gaussian stochastic weight averaging (**SWAG**, Maddox et al., 2019a). For each baseline, we use the hyperparameters recommended in the original paper—see Appendix B.2 for details.

First, Figs. 2.4 and 2.5 shows that LA and LA* are, respectively, competitive with and superior to the baselines in trading-off between in-distribution calibration and dataset-shift robustness. Second, Table 2.1 shows that LA and LA* achieve better results on out-of-distribution (OOD) detection than even VB, CSGHMC, and SWAG. The LA shines even more when we consider its (time *and* memory) cost relative to the other, more complex baselines.

In Fig. 2.6 we show the wall-clock times of each method relative to MAP's for training and prediction. As expected, DE, VB, and CSGHMC are slow to train and in making predictions: they are between two to five times more expensive than MAP. Meanwhile, despite being *post-hoc*, SWAG is almost twice as expensive as MAP during training due to the need for sampling and updating its batch normalization statistics. Moreover, with 30 samples, as recommended by its authors (Maddox et al., 2019a), it is very expensive at prediction time—more than ten times more expensive than MAP. Meanwhile, LA (and LA*) is the cheapest of all methods



(a) Rotated-MNIST **(b)** CIFAR-10-C

Figure 2.5: Assessing model calibration under distribution shift, for the MNIST (top row) and CIFAR-10 (bottom row) datasets. We use the Rotated-MNIST **(a)** and Corrupted-CIFAR-10 **(b)** benchmarks (Hendrycks & Dietterich, 2019; Ovadia et al., 2019). Even though *post-hoc*, all LAs achieve competitive results, even to DE. In particular, LA* achieves the best results.

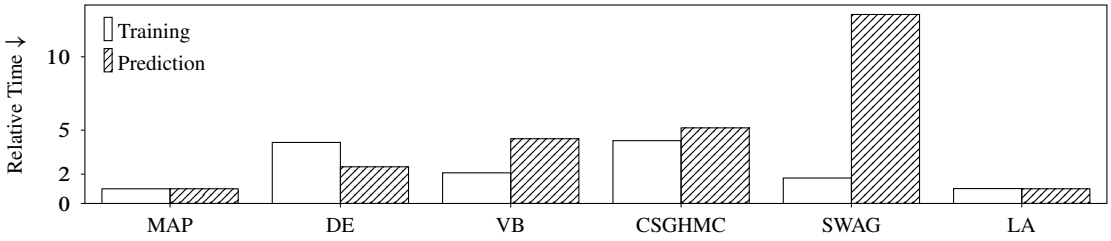


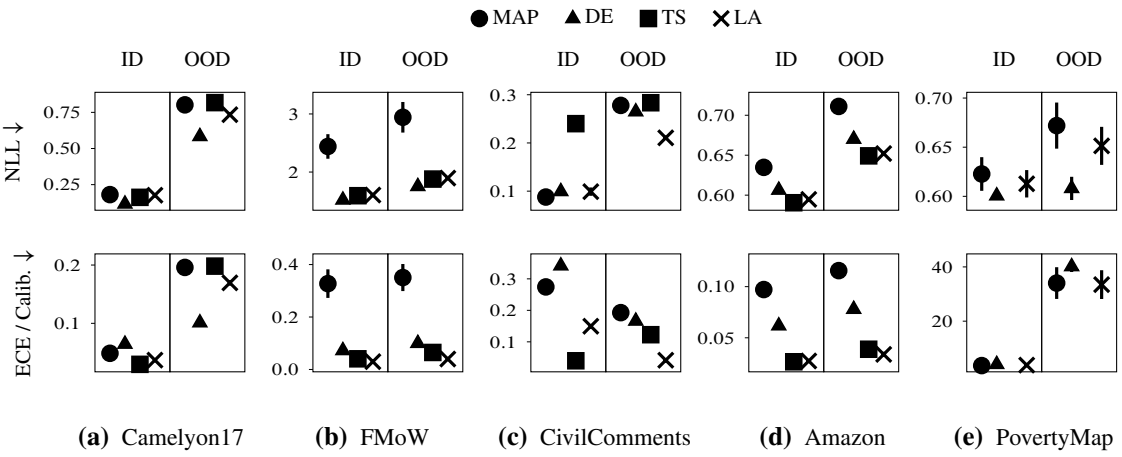
Figure 2.6: Wall-clock time costs relative to MAP. LA introduces negligible overhead over MAP, while all other baselines are significantly more expensive.

considered: it only incurs a negligible overhead on top of the costs of MAP. This shows that the LA is significantly more compute-efficient than all the other methods considered, adding minimal overhead over MAP inference and prediction.

Finally, Table 2.2 compares the theoretical memory complexity and actual memory footprint (of a WideResNet 16-4 on CIFAR-10) of the different methods. There, M denotes the number of model parameters, H denotes the number of neurons in the last layer, K denotes the number of model outputs, R denotes the number of SWAG snapshots, S denotes the number of CSGHMC samples, and N denotes the number of deep ensemble's members. VB has a complexity of $2M$ as it needs to store a variance vector of size M in addition to the mean vector of size M. For the actual memory footprints, we assume $R = 40$ SWAG snapshots,

Table 2.2: The memory complexities of all methods in big-O notation. To get a better idea of what these complexities translate to in practice, we also report the actual memory footprints (in megabytes) of a WideResNet 16-4 (WRN) on CIFAR-10.

Method	Mem. Complexity	WRN on CIFAR-10
MAP	M	11 MB
LA	$\boldsymbol{M + H^2 + K^2}$	**12 MB**
VB	$2M$	22 MB
DE	NM	55 MB
CSGHMC	SM	132 MB
SWAG	RM	440 MB



(a) Camelyon17 **(b)** FMoW **(c)** CivilComments **(d)** Amazon **(e)** PovertyMap

Figure 2.7: Assessing real-world distribution shift robustness on five datasets from the `WILDS` benchmark Koh et al. (2020), covering different data modalities, model architectures, and output types. We plot means ± standard errors of the NLL (top) and ECE (for classification) or regression calibration error Kuleshov et al. (2018) (bottom).

$S = 12$ CSGHMC samples, and $N = 5$ ensemble members, which are the hyperparameters recommended in the original papers. It can be seen that the proposed default KFAC-last-layer approximation poses a small memory overhead of $O(H^2 + K^2)$ on top of the MAP estimate.

Altogether, the results in Fig. 2.6 and Table 2.2 make the LA particularly attractive for practitioners, especially in low-resource environments. Together with Fig. 2.5 and Table 2.1, this justifies our default flavor of Laplace approximation in `laplace-torch`, and importantly, shows that Bayesian deep learning does not have to be expensive.

2.3.3 Realistic Distribution Shift

So far, our experiments focused on comparably simple benchmarks, allowing us to comprehensively assess different LA variants and compare to more involved Bayesian methods such as VB, MCMC, and SWAG. In more realistic settings, however, where we want to improve the uncertainty of complex and costly-to-train models, such as transformers (Vaswani et al., 2017), these methods would likely be difficult to get to work well and expensive to run. However, one might often have access to a pre-trained model, allowing for the cheap use of *post-hoc* methods such as the LA. To demonstrate this, we show how `laplace-torch` can improve the distribution shift robustness of complex pre-trained models in large-scale settings. To this end, we use the WILDS testbed (Koh et al., 2020), a recently proposed benchmark of realistic distribution shifts encompassing a variety of real-world datasets across different data modalities and application domains. While the WILDS models employ complex (e.g. convolutional or transformer) architectures as feature extractors, they all feed into a linear output layer, allowing us to conveniently and cheaply apply the last-layer LA. As baselines, we consider: 1) the pre-trained MAP models (Koh et al., 2020), 2) *post-hoc* temperature scaling of the MAP models (for classification tasks) Guo et al. (2017), and 3) deep ensembles (Lakshminarayanan et al., 2017).[5] More details on the experimental setup are provided in Appendix B.3. Fig. 2.7 shows the results on five different

[5]We simply construct deep ensembles from the various pre-trained models provided by Koh et al. (2020).

```python
from laplace import Laplace

# Initialize Gaussian prior N(0, I)
prior_mean = torch.zeros(...)
prior_prec = torch.eye(...)

for task in range(num_of_tasks):
    # MAP training with L2 regularization
    model = train(train_loader[task], prior_mean, prior_prec)

    # Fit LA with the current prior mean and precision
    la = Laplace(model, 'classification',
                 prior_mean=prior_mean,
                 prior_precision=prior_prec)
    la.fit(train_loader[task])

    prediction = la(x_test[task])

    # Set the current posterior as the next prior
    prior_mean = la.mean
    prior_prec = la.posterior_precision
```

Listing 2.4: Bayesian continual learning with `laplace-torch`.

WILDS datasets. Overall, the LA is significantly better calibrated than MAP, and competitive with temperature scaling and ensembles, especially on the OOD splits.

2.3.4 Continual learning

Beyond predictive uncertainty quantification, the LA is useful in wide range of applications such as Bayesian optimization (Snoek et al., 2015), bandits (Chapelle & Li, 2011), active learning (MacKay, 1992b; Park et al., 2011), and continual learning (Ritter et al., 2018b). The `laplace-torch` library conveniently facilitates these applications. As an example, we demonstrate the performance of the LA on the standard continual learning benchmark with the Permuted-MNIST dataset, consisting of ten tasks each containing pixel-permuted MNIST images (Goodfellow et al., 2013). Figure 2.8 shows how the all-layer diagonal and Kronecker-factored LAs can overcome *catastrophic forgetting*. In this experiment, we update the LAs after each task as suggested by Ritter et al. (2018b) and improve upon their result by tuning the prior precision through marginal likelihood optimization during training, following Immer et al. (2021a) (details in Appendix B.4). Using this scheme, the performance after 10 tasks is at around 96% accuracy, outperforming other Bayesian approaches for continual learning (Nguyen et al., 2018; Titsias et al., 2020; Pan et al., 2020). Concretely, we show that the KFAC LA, while much simpler when applied via `laplace-torch`, can achieve better performance to a recent VB baseline (VOGN, Osawa et al., 2019). Our library thus provides an easy and quick way of constructing a strong baseline for this application.

Related Work

The LA is fundamentally a local approximation that covers a single mode of the posterior; similarly, other Gaussian approximations such as mean-field variational inference (Graves, 2011; Blundell et al., 2015; Osawa et al., 2019) or SWAG (Maddox et al., 2019a) also only capture

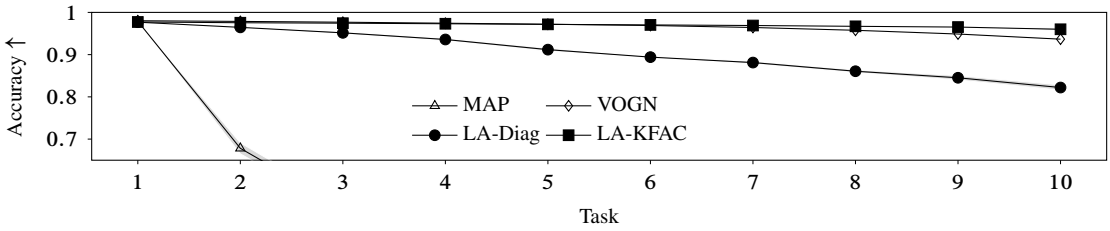


Figure 2.8: Continual learning results on Permuted-MNIST. MAP fails catastrophically as more tasks are added. The Bayesian approaches substantially outperform MAP, with LA-KFAC performing the best, closely followed by VOGN.

local information. SWAG uses the first and second empirical moment of SGD iterates to form a diagonal plus low-rank Gaussian approximation but requires storing many NN copies and applying a (costly) heuristic related to batch normalization at test time. In contrast, the LA directly uses curvature information of the loss around the MAP and can be applied *post-hoc* to pre-trained NNs.

In contrast to local Gaussian approximations, (stochastic-gradient) MCMC methods (Welling & Teh, 2011; Wenzel et al., 2020a; Zhang et al., 2020; Izmailov et al., 2021b; Garriga-Alonso & Fortuin, 2021, etc.) and deep ensembles (Lakshminarayanan et al., 2017) can explore several modes. Nevertheless, prior works—also validated in our experiments in Section 2.3—indicate that using a single mode might not be as limiting in practice as one might think. Wilson & Izmailov (2020) conjecture that this is due to the complex, nonlinear connection between the parameter space and the function (output) space of NNs. Moreover, while unbiased compared to its simpler alternatives, MCMC methods are notoriously expensive in practice and, thus, often require further approximations such as distillation (Korattikara et al., 2015; Wang et al., 2018). Finally, note that both the LA as well as SWAG can be extended to ensembles of modes in a *post-hoc* manner (Eschenhagen et al., 2021; Wilson & Izmailov, 2020).

Chapter 3
Fixing Asymptotic Overconfidence

The contents of this chapter are primarily based on:

> Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. "Being Bayesian, even just a bit, fixes overconfidence in ReLU networks." International Conference on Machine Learning (ICML). 2020.

	Idea	**Analysis**	**Experiment**	**Code**	**Writing**
Agustinus Kristiadi	80%	100%	100%	100%	85%
Matthias Hein	10%	0%	0%	0%	5%
Philipp Hennig	10%	0%	0%	0%	10%

And:

> Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. "An Infinite-Feature Extension for Bayesian ReLU Nets That Fixes Their Asymptotic Overconfidence." Advances in Neural Information Processing Systems (NeurIPS). 2021.

	Idea	**Analysis**	**Experiment**	**Code**	**Writing**
Agustinus Kristiadi	70%	90%	100%	100%	85%
Matthias Hein	5%	10%	0%	0%	5%
Philipp Hennig	25%	0%	0%	0%	10%

ReLU networks are currently among the most widely used neural architectures. This class comprises any network that can be written as a composition of linear layers (including fully-connected, convolutional, and residual layers) and a ReLU activation function. But, while ReLU networks often achieve high accuracy, the *uncertainty* of their predictions has been shown to be miscalibrated (Guo et al., 2017). Indeed, Hein et al. (2019) demonstrated that ReLU networks are *always* overconfident "far away from the data": scaling a training point x (a vector in a Euclidean input space) with a scalar δ yields predictions of arbitrarily high confidence in the limit $\delta \to \infty$. This means ReLU networks are susceptible to out-of-distribution examples. Meanwhile, probabilistic methods (in particular Bayesian methods) have long been known empirically

to improve predictive uncertainty estimates. MacKay (1992b) demonstrated experimentally that the predictive uncertainty of Bayesian neural networks will naturally be high in regions not covered by training data. Although the theoretical analysis is still lacking, results like this raise the hope that the overconfidence problem of ReLU networks, too, might be mitigated by the use of probabilistic and Bayesian methods.

This chapter offers theoretical analyses of ReLU classification networks. In Section 3.2 we show that equipping a two-class ReLU network with a Gaussian approximate distribution over the weights mitigates the aforementioned theoretical problem, in the sense that the predictive confidence far away from the training data approaches a known limit, bounded away from one, whose value is controlled by the covariance. In the case of Laplace approximations (MacKay, 1992c; Ritter et al., 2018a), this treatment in conjunction with the probit approximation (Spiegelhalter & Lauritzen, 1990; MacKay, 1992b) does not change the decision boundary of the trained network, so it has no negative effect on the predictive performance. Furthermore, we show that a sufficient condition for this desirable property to hold is to apply a Gaussian approximation *only* to the last layer of a ReLU network. This motivates the commonly used approximation scheme where an L-layer network is decomposed into a fixed feature map composed of the first $L-1$ layers and a Bayesian linear classifier (Wilson et al., 2016a; Riquelme et al., 2018; Ober & Rasmussen, 2019; Brosse et al., 2020, etc.). This particular result implies that just being "a bit" Bayesian—at low-cost overhead—already gives desirable benefits. Meanwhile, in Section 3.3 we propose a method to incorporate infinitely-many ReLU features in Gaussian-approximated ReLU networks in a cheap and *post-hoc* manner. Using this method, we provide a stronger result: the augmented multiclass ReLU classification network will always attain the ideal *uniform confidence* far away from the training data.

3.1 The Asymptotic Overconfidence Problem

We call a function $f : \mathbb{R}^n \to \mathbb{R}^k$ ***piecewise affine*** if there exists a finite set of polytopes $\{Q_i\}_{i=1}^r$, referred to as the ***linear regions*** of f, such that $\cup_{i=1}^r Q_i = \mathbb{R}^n$ and $f|_{Q_i}$ is an affine function for every Q_r. ***ReLU networks*** are networks that result in piecewise affine classifier functions (Arora et al., 2018), which include networks with fully-connected, convolutional, and residual layers where just ReLU or leaky-ReLU are used as activation functions and max or average pooling are used in convolution layers.

Let $\mathcal{D} := \{x_i \in \mathbb{R}^n, y_i \in \mathbb{R}^k\}_{i=1}^m$ be a dataset, where the targets are $y_i \in \{0, 1\}$ or $y_i \in \{1, \ldots, k\}$ for the binary and multi-class case, respectively. Let $\phi : \mathbb{R}^n \to \mathbb{R}^d$ be an arbitrary fixed feature map. Given a neural network f_θ, we consider the distribution $p(\theta \mid \mathcal{D})$ over its parameters. The *predictive* distribution for the binary case is

$$p(y = 1 \mid x, \mathcal{D}) = \int \sigma(f_\theta(x))\, p(\theta \mid \mathcal{D})\, d\theta, \tag{3.1}$$

and for the multi-class case

$$p(y = i \mid x, \mathcal{D}) = \int \operatorname{softmax}(f_\theta(x), i)\, p(\theta \mid \mathcal{D})\, d\theta. \tag{3.2}$$

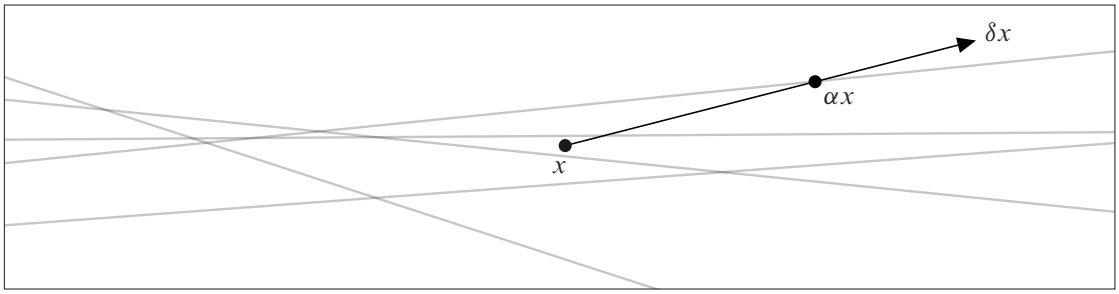


Figure 3.1: A sketch of the situation in Theorem 3.2. The intersections of gray lines are the linear regions induced by a ReLU network f. Far enough from the data region—the region around x—one can always find a linear region that extends to infinity, such that δx will always be contained in that region for δ large enough, i.e. $\delta \geq \alpha$. In this case, analyzing f is simple since $f(\delta x)$ is a simple affine function.

The functions $\lambda_i(\cdot)$, $\lambda_{\max}(\cdot)$, and $\lambda_{\min}(\cdot)$ return the i-th, maximum, and minimum eigenvalue (which are assumed to exist) of their matrix argument, respectively.[1] Similarly for the function $s_i(\cdot)$, $s_{\max}(\cdot)$, and $s_{\min}(\cdot)$ which return singular values instead. Finally, we assume that $\|\cdot\|$ is the ℓ^2 norm.

The following lemma shows the existence of an "outer linear region" induced by a ReLU network which contains the scaled input δx for δ sufficiently large.

Lemma 3.1 (Hein et al., 2019). *Let $\{Q_m\}_{m=1}^r$ be the set of linear regions associated to the ReLU network $f : \mathbb{R}^n \to \mathbb{R}^k$. For any $x \neq 0 \in \mathbb{R}^n$ there exists an $\alpha > 0$ and $t \in \{1, \ldots, r\}$ such that $\delta x \in Q_t$ for all $\delta \geq \alpha$. Furthermore, the restriction of f to Q_t can be written as an affine function $U^\top x + c$ for some suitable $U \in \mathbb{R}^{n \times k}$ and $c \in \mathbb{R}^k$.* □

Based on the previous lemma, the following theorem shows that ReLU networks exhibit arbitrarily high confidence far away from the training data: If a point $x \in \mathbb{R}^n$ is scaled by a sufficiently large scalar $\delta > 0$, the input δx attains arbitrarily high confidence.

Theorem 3.2 (Hein et al., 2019). *Let $\mathbb{R}^n = \cup_{j=1}^r Q_j$ and $f|_{Q_j}(x) = U_j x + c_j$ be the piecewise affine representation of the output of a ReLU network on a linear region Q_j. Suppose that U_j does not contain identical rows for all $j = 1, \ldots, r$, then for almost any $x \neq 0 \in \mathbb{R}^n$ and any $\varepsilon > 0$, there exists a $\delta > 0$ and a class $i \in \{1, \ldots, k\}$ such that it holds* $\operatorname{softmax}(f(\delta x), i) \geq 1 - \varepsilon$. *Moreover,* $\lim_{\delta \to \infty} \operatorname{softmax}(f(\delta x), i) = 1$. □

The proofs of Lemma 3.1 and Theorem 3.2 can be found in the work of Hein et al. (2019). The intuition of Theorem 3.2 is that due to the existence of an outer linear region (Lemma 3.1), far away from the training data, a ReLU network becomes a simple affine function that is almost always increasing or decreasing. Because of this, as $\delta \to \infty$, the softmax output of the network tends to a one-hot vector and thus has confidence of 1. See Fig. 3.1 for an illustration.

[1] We assume they are sorted in a descending order.

3.2 Being A Bit Bayesian Mitigates Asymptotic Overconfidence

In this section we focus on binary classification problems. It is standard to treat neural networks as probabilistic models of the conditional distribution $p(y \mid x, \theta)$ over the prediction y. In this case, we define the ***confidence*** of any input point x as the maximum predictive probability, which, in the case of a binary problem, can be written as

$$\text{conf}(x) := \max_{i \in \{0,1\}} p(y = i \mid x, \theta) = \sigma(|f_\theta(x)|). \tag{3.3}$$

Standard training corresponds to assigning a maximum a posteriori (MAP) estimate θ_{MAP} to the weights, ignoring potential uncertainty on θ. We will show that this lack of uncertainty is the primary cause of the overconfidence discussed by Hein et al. (2019) and argue that it can be mitigated by considering the marginalized prediction—where uncertainty over the weights are taken into account—in (3.1) instead.

Even for a binary linear classifier parametrized by a single weight matrix $\theta = w$, there is generally no analytic solution for (3.1). But, good approximations exist when the distribution over the weights is a Gaussian $p(w \mid \mathcal{D}) \approx \mathcal{N}(w \mid \mu, \Sigma)$ with mean μ and covariance Σ. One such approximation is given by the probit approximation (Spiegelhalter & Lauritzen, 1990; MacKay, 1992b). Using this approximation and the Gaussian assumption, if we let $a := w^\top \phi(x)$, we get

$$p(y = 1 \mid x, \mathcal{D}) \approx \sigma\left(\frac{\mu^\top \phi(x)}{\sqrt{1 + \pi/8\, \phi(x^\top \Sigma \phi(x)}}\right) =: \sigma(z(x)). \tag{3.4}$$

In the case of $\mu = w_{\text{MAP}}$, (3.4) can be seen as the "softened" version of the MAP prediction of the classifier, using the covariance of the Gaussian. The predictive confidence on an input x in this case is defined by

$$\text{conf}(x) = \max_{i \in \{0,1\}} p(y = i \mid x, \mathcal{D}) = \sigma(|z(x)|). \tag{3.5}$$

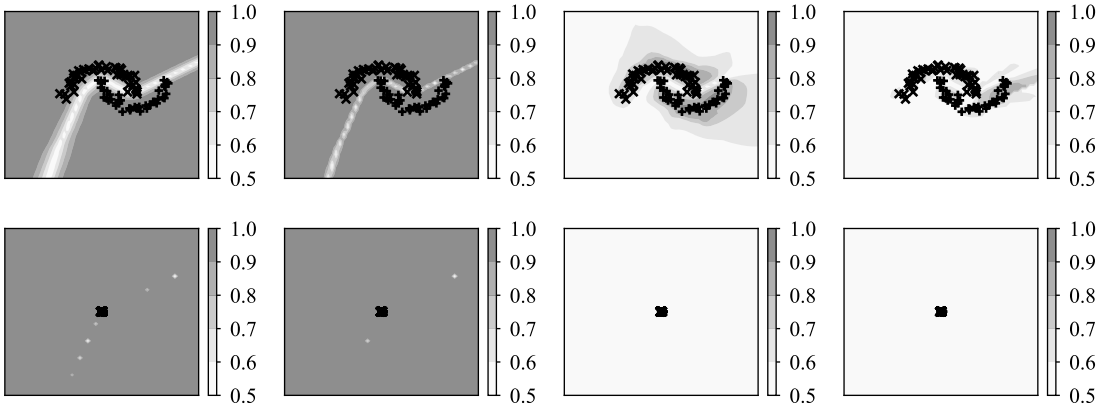
We can generalize the previous insight to the case where the parameters of the feature map ϕ are also approximated by a Gaussian. Let $\theta \in \mathbb{R}^d$ be the parameter vector of a NN $f_\theta : \mathbb{R}^n \to \mathbb{R}$ with a given Gaussian approximation $p(\theta \mid \mathcal{D}) \approx \mathcal{N}(\theta \mid \mu, \Sigma)$. Let $x \in \mathbb{R}^n$ be an arbitrary input point. Letting $g(x) := \nabla f_\theta(x)|_\mu$, we do a first-order Taylor expansion of f_θ at μ (MacKay, 1995): $f_\theta(x) \approx f_\mu(x) + g(x)^\top(\theta - \mu)$. This implies that the distribution over $f_\theta(x)$ is given by $p(f_\theta(x) \mid x, \mathcal{D}) \approx \mathcal{N}(f_\theta(x) \mid f_\mu(x), g(x)^\top \Sigma g(x))$. Therefore, we have

$$z(x) := \frac{f_\mu(x)}{\sqrt{1 + \pi/8\, g(x)^\top \Sigma g(x)}}. \tag{3.6}$$

It is easy to see that z in (3.4) is indeed a special case of (3.6).

As the first notable property of this approximation, we show that, in contrast to some other methods for uncertainty quantification (e.g. Monte Carlo dropout, Gal & Ghahramani, 2016) it preserves the decision boundary induced by the MAP estimate.

Proposition 3.3 (Invariance property). *Let $f_\theta : \mathbb{R}^n \to \mathbb{R}$ be a binary classifier network parametrized by θ and let $\mathcal{N}(\theta \mid \mu, \Sigma)$ be the distribution over θ. Then for any $x \in \mathbb{R}^n$, we have $\sigma(f_\mu(x)) = 0.5$ if and only if $\sigma(z(x)) = 0.5$.*



(a) MAP **(b)** Temp. scaling **(c)** Bayesian (last-layer) **(d)** Bayesian (all-layer)

Figure 3.2: Binary classification on a toy dataset using a MAP estimate, temperature scaling (Guo et al., 2017), and both last-layer and all-layer Gaussian approximations over the weights which are obtained via Laplace approximations. Background shades represent confidence estimates. Bottom row shows a zoomed-out view of the top row. The Bayesian approximations—even in the last-layer case—give desirable uncertainty estimates: confident close to the training data and uncertain otherwise. MAP and temperature scaling yield overconfident predictions.

Proof. Let $x \in \mathbb{R}^n$ be arbitrary and denote $\mu_f := f_{\theta_{\text{MAP}}}(x)$ and $v_f := g(x)^\top \Sigma g(x)$. For the forward direction, suppose that $\sigma(\mu_f) = 0.5$. This implies that $\mu_f = 0$, and we have $\sigma(0/(1 + \pi/8\, v_f)^{1/2}) = \sigma(0) = 0.5$. For the reverse direction, suppose that $\sigma(\mu_f/(1 + \pi/8\, v_f)^{1/2}) = 0.5$. This implies $\mu_f/(1 + \pi/8\, v_f)^{1/2} = 0$. Since the denominator of the l.h.s. is positive, it follows that μ_f must be 0, implying that $\sigma(\mu_f) = 0.5$. □

This property is useful in practice, particularly whenever $\mu = \theta_{\text{MAP}}$, since it guarantees that employing a Gaussian approximation on top of a MAP-trained network will not reduce the original classification accuracy. Virtually all state-of-the-art models in deep learning are trained via MAP estimation and sacrificing the classification performance that makes them attractive in the first place would be a waste.

As our central theoretical contribution, we show that for any $x \in \mathbb{R}^n$, as $\delta \to \infty$, the value of $|z(\delta x)|$ in (3.6) goes to a quantity that only depends on the mean and covariance of the Gaussian over the weights. Moreover, this property also holds in the finite asymptotic regime, far enough from the training data. This result implies that one can drive the confidence closer to the uniform (one-half) far away from the training points by shifting $|z(x)|$ closer to zero by controlling the Gaussian. The following lemma is needed to get the desired result.

Lemma 3.4. *Let $A \in \mathbb{R}^{m\times n}$ and $z \in \mathbb{R}^n$ with $m \geq n$, then $\|Az\|^2 \geq s^2_{\text{min}}(A)\|z\|^2$.*

Proof. By SVD, $A = USV^\top$. Notice that U, V are orthogonal and thus are isometries, and that S is a rectangular diagonal matrix with n non-zero elements. Therefore,

$$\|U(SV^\top z)\|^2 = \|SV^\top z\|^2 = \sum_{i=1}^{n} s_i^2(A)(V^\top z)_i^2$$

$$\geq s_{\min}^2(A) \sum_{i=1}^{n} (V^\top z)_i^2$$
$$= s_{\min}^2(A) \|V^\top z\|^2 = s_{\min}^2(A) \|z\|^2 , \tag{3.7}$$

thus the proof is complete. □

We formalize our main result in the following theorem.

Theorem 3.5 (All-layer approximation). *Let $f_\theta : \mathbb{R}^n \to \mathbb{R}$ be a binary ReLU classification network parametrized by $\theta \in \mathbb{R}^d$ with $d \geq n$, and let $\mathcal{N}(\theta \mid \mu, \Sigma)$ be the Gaussian approximation over the parameters. Then for any input $x \in \mathbb{R}^n$,*

$$\lim_{\delta \to \infty} \sigma(|z(\delta x)|) \leq \sigma\left(\frac{\|u\|}{s_{\min}(J) \sqrt{\pi/8 \, \lambda_{\min}(\Sigma)}} \right), \tag{3.8}$$

where $u \in \mathbb{R}^n$ is a vector depending only on μ and the $n \times p$ matrix $J := \frac{\partial u}{\partial \theta}\Big|_\mu$ is the Jacobian of u w.r.t. θ at μ. Moreover, if f_θ has no bias parameters, then there exists $\alpha > 0$ such that for any $\delta \geq \alpha$, we have that

$$\sigma(|z(\delta x)|) \leq \lim_{\delta \to \infty} \sigma(|z(\delta x)|).$$

Proof. By Lemma 3.1 there exists an $\alpha > 0$ and a linear region R, along with $u \in \mathbb{R}^n$ and $c \in \mathbb{R}$, such that for any $\delta \geq \alpha$, we have that $\delta x \in R$ and the restriction $f_\theta|_R$ can be written as $u^\top x + c$. Note that, for any such δ, the vector u and scalar c are constant w.r.t. δx. Therefore for any such δ, we can write the gradient $g(\delta x)$ as follows:

$$g(\delta x) = \frac{\partial(\delta u^\top x)}{\partial \theta}\bigg|_\mu + \frac{\partial c}{\partial \theta}\bigg|_\mu = \delta \frac{\partial u}{\partial \theta}\bigg|_\mu^\top x + \frac{\partial c}{\partial \theta}\bigg|_\mu$$
$$= \delta \left(J^\top x + \frac{1}{\delta} \nabla_\theta \, c|_\mu \right). \tag{3.9}$$

Hence, by (3.6),

$$|z(\delta x)| = \frac{|\delta u^\top x + c|}{\sqrt{1 + \pi/8 \, g(\delta x)^\top \Sigma g(\delta x)}}$$
$$= \frac{|\delta (u^\top x + \frac{1}{\delta} c)|}{\sqrt{1 + \pi/8 \, \delta^2 (J^\top x + \frac{1}{\delta} \nabla_\theta \, c|_\mu)^\top \Sigma (J^\top x + \frac{1}{\delta} \nabla_\theta \, c|_\mu)}}$$
$$= \frac{\delta |(u^\top x + \frac{1}{\delta} c)|}{\delta \sqrt{\frac{1}{\delta^2} + \pi/8 \, (J^\top x + \frac{1}{\delta} \nabla_\theta \, c|_\mu)^\top \Sigma (J^\top x + \frac{1}{\delta} \nabla_\theta \, c|_\mu)}}$$

Now, notice that as $\delta \to \infty$, $1/\delta^2$ and $1/\delta$ goes to zero. So, in the limit, we have that

$$\lim_{\delta \to \infty} |z(\delta x)| = \frac{|u^\top x|}{\sqrt{\pi/8 \, (J^\top x)^\top \Sigma (J^\top x)}}.$$

Using the Cauchy-Schwarz inequality and Lemma 3.6, we can upper-bound this limit with

$$\lim_{\delta \to \infty} |z(\delta x)| \leq \frac{\|u\| \|x\|}{\sqrt{\pi/8 \, \lambda_{\min}(\Sigma) \|J^\top x\|^2}}.$$

Notice that $J^\top \in \mathbb{R}^{d \times n}$ with $d \geq n$ by our hypothesis. Therefore, using Lemma 3.4 on $\|J^\top x\|^2$ in conjunction with $s_{\min}(J) = s_{\min}(J^\top)$, we conclude that

$$\begin{aligned} \lim_{\delta \to \infty} |z(\delta x)| &\leq \frac{\|u\| \|x\|}{\sqrt{\pi/8 \, \lambda_{\min}(\Sigma) \, s_{\min}^2(J) \|x\|^2}} \\ &= \frac{\|u\|}{s_{\min}(J) \sqrt{\pi/8 \, \lambda_{\min}(\Sigma)}}, \end{aligned} \tag{3.10}$$

thus the first result is proved.

To prove the second statement, let $L := \lim_{\delta \to \infty} |z(\delta x)|$. Since L is the limit of $|z|_Q(\delta x)|$ in the linear region Q given by Lemma 3.1, it is sufficient to show that the function $(0, \infty] \to \mathbb{R}$ defined by $\delta \mapsto |z|_Q(\delta x)|$ is increasing.

For some suitable choices of $u \in \mathbb{R}^n$ that depends on μ, we can write the restriction of the "point-estimated" ReLU network $f_\mu|_Q(x)$ as $u^\top x$ by definition of ReLU network and since we assume that f has no bias parameters. Furthermore, we let the matrix $J := \frac{\partial u}{\partial \theta}|_\mu$ to be the Jacobian of u w.r.t. θ at μ. Therefore for any $\delta \geq \alpha$, we can writ:

$$\begin{aligned} |z|_Q(\delta x)| &= \frac{|\delta u^\top x|}{\sqrt{1 + \pi/8 \, \delta^2 \, (J^\top x)^\top \Sigma (J^\top x)}} \\ &=: \frac{|\delta a|}{\sqrt{1 + \pi/8 \, \delta^2 \, b}}, \end{aligned}$$

where for simplicity we have let $a := u^\top x$ and $b := (J^\top x)^\top \Sigma (J^\top x)$. The derivative is therefore given by

$$\frac{d}{d\delta} |z|_Q(\delta x)| = \frac{\delta |a|}{(1 + \delta^2 b)^{\frac{3}{2}} |\delta|}$$

and since Σ is positive-definite, it is non-negative for $\delta \in (0, \infty]$. Thus we conclude that $|z|_Q(\delta x)|$ is an increasing function. □

The following question is practically interesting: Do we have to construct a probabilistic (Gaussian) uncertainty to the whole ReLU network for the previous property (Theorem 3.5) to hold? Surprisingly, the answer is no. The following theorem establishes that a guarantee similar to Theorem 3.5, is feasible even if *only the last layer's weights* are assigned a Gaussian distribution. This amounts to a form of Bayesian logistic regression where the features are provided by the ReLU network. First, we state the following lemma.

Lemma 3.6. *Let $x \in \mathbb{R}^n$ be a vector and $A \in \mathbb{R}^{n \times n}$ be an SPD matrix. If $\lambda_{\min}(A)$ is the minimum eigenvalue of A, then $x^\top A x \geq \lambda_{\min} \|x\|^2$.*

Proof. Since A is SPD, it admits an eigendecomposition $A = Q\Lambda Q^\top$ and $\Lambda = \Lambda^{\frac{1}{2}}\Lambda^{\frac{1}{2}}$ makes sense. Therefore, by keeping in mind that $Q^\top x$ is a vector in $\mathbb{R}^n$, we have

$$\begin{aligned} x^\top Ax &= x^\top Q\Lambda^{\frac{1}{2}}\Lambda^{\frac{1}{2}}Q^\top x = \|\Lambda^{\frac{1}{2}}Q^\top x\|^2 \\ &= \sum_{i=1}^{n} \lambda_i(A)(Q^\top x)_i^2 \geq \lambda_{\min}(A)\sum_{i=1}^{n}(Q^\top x)_i^2 \\ &= \lambda_{\min}(A)\|Q^\top x\|^2 = \lambda_{\min}(A)\|x\|^2, \end{aligned}$$

where the last equality is obtained since $\|Q^\top x\|^2 = x^\top Q^\top Qx$ and by noting that Q is an orthogonal matrix. □

Using this lemma, we can show the result.

Theorem 3.7 (Last-layer approximation). *Let $g : \mathbb{R}^d \to \mathbb{R}$ be a binary linear classifier defined by $g(\phi(x)) := w^\top\phi(x)$ where $\phi : \mathbb{R}^n \to \mathbb{R}^f$ is a fixed ReLU network and let $\mathcal{N}(w \mid \mu, \Sigma)$ be the Gaussian approximation over the last-layer's weights. Then for any input $x \in \mathbb{R}^n$,*

$$\lim_{\delta\to\infty} \sigma(|z(\delta x)|) \leq \sigma\left(\frac{\|\mu\|}{\sqrt{\pi/8\,\lambda_{\min}(\Sigma)}}\right). \tag{3.11}$$

Moreover, if ϕ has no bias parameters, then there exists $\alpha > 0$ such that for any $\delta \geq \alpha$, we have that

$$\sigma(|z(\delta x)|) \leq \lim_{\delta\to\infty} \sigma(|z(\delta x)|).$$

Proof. By Lemma 3.1 of Hein et al. (2019) there exists $\alpha > 0$ and a linear region R, along with $U \in \mathbb{R}^{d\times n}$ and $c \in \mathbb{R}^d$, such that for any $\delta \geq \alpha$, we have that $\delta x \in R$ and the restriction $\phi|_R$ can be written as $Ux + c$. Therefore, for any such δ,

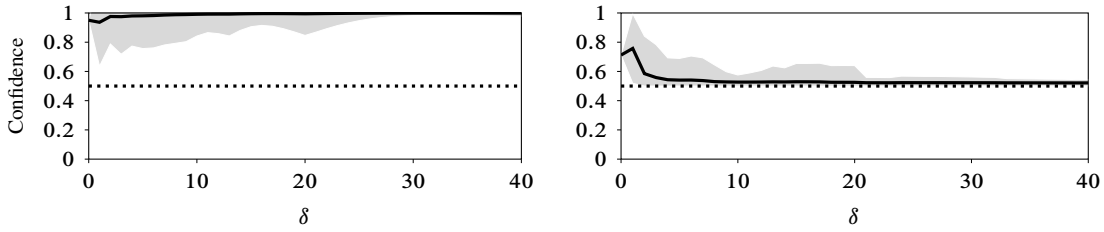
$$\begin{aligned} |z \circ (\phi|_R)(\delta x)| &= \frac{|\mu^\top(\delta Ux + c)|}{\sqrt{1 + \pi/8\,(\delta Ux + b)^\top\Sigma(\delta Ux + c)}} \\ &= \frac{|\mu^\top(Ux + \frac{1}{\delta}c)|}{\sqrt{\frac{1}{\delta^2} + \pi/8\,(Ux + \frac{1}{\delta}c)^\top\Sigma(Ux + \frac{1}{\delta}c)}} \end{aligned}$$

Now, notice that as $\delta \to \infty$, $1/\delta^2$ and $1/\delta$ goes to zero. So, in the limit, we have that

$$\lim_{\delta\to\infty} |z \circ (\phi|_R)(\delta x)| = \frac{|\mu^\top(Ux)|}{\sqrt{\pi/8\,(Ux)^\top\Sigma(Ux)}}.$$

Using the Cauchy-Schwarz inequality and the Lemma 3.6, we can upper-bound the limit with

$$\begin{aligned} \lim_{\delta\to\infty} |z \circ (\phi|_R)(\delta x)| &\leq \frac{\|\mu\|\|Ux\|}{\sqrt{\pi/8\,\lambda_{\min}(\Sigma)\|Ux\|^2}} \\ &= \frac{\|\mu\|}{\sqrt{\pi/8\,\lambda_{\min}(\Sigma)}}, \end{aligned}$$



(a) MAP **(b)** Bayesian

Figure 3.3: Confidence (mean and ±3 standard deviation) of the dataset in Fig. 3.2 as a function of δ.

which concludes the proof for the first statement.

For the second statement, since the previous limit is the limit of $|(z|_R)(\delta x)|$ in the linear region R, it is sufficient to show that the function $(0, \infty] \to \mathbb{R}$ defined by $\delta \mapsto |z|_R(\delta x)|$ is increasing. For some $U \in \mathbb{R}^{d \times n}$ that depends on the fixed parameter of ϕ, we write the restriction $\phi|_R(x)$ as Ux by definition of ReLU network and since ϕ is assumed to have no bias parameters. Therefore for any $\delta \geq \alpha$, we can write as a function of δ:

$$\begin{aligned} |(z|_Q)(\delta x)| &= \frac{|\delta \mu^\top U x|}{\sqrt{1 + \pi/8\, \delta^2\, (Ux)^\top \Sigma (Ux)}} \\ &=: \frac{|\delta a|}{\sqrt{1 + \pi/8\, \delta^2\, b}}, \end{aligned}$$

where for simplicity we have let $a := \mu^\top U x$ and $b := (Ux)^\top \Sigma (Ux)$. The derivative is therefore given by

$$\frac{d}{d\delta} |(z|_Q)(\delta x)| = \frac{\delta |a|}{(1 + \delta^2 b)^{\frac{3}{2}} |\delta|}$$

and since Σ is positive-definite, it is non-negative for $\delta \in (0, \infty]$. Thus we conclude that $|(z|_Q)(\delta x)|$ is an increasing function. □

We show, using the same toy dataset and Gaussian-based last-layer Bayesian method as in Fig. 3.2, an illustration of the previous results in Fig. 3.3. Confirming the findings, for each input x, the Gaussian approximation drives $|z(\delta x)|$ to a constant for sufficiently large δ. Note that on true data points ($\delta = 1$), the confidences remain high and the convergence occurs at some finite δ.

Taken together, the results above formally validate the usage of the common Gaussian approximations of the weights distribution, both in Bayesian (MacKay, 1992c; Graves, 2011; Blundell et al., 2015, etc.) or non-Bayesian (Franchi et al., 2019; Lu et al., 2020, etc.) fashions, on ReLU networks for mitigating overconfidence problems. Furthermore, Theorem 3.7 shows that an all-layer Gaussian approximation or Bayesian treatment (i.e. on all layers of a NN) is not required to achieve control over the confidence far away from the training data. Put simply, even being "just a bit Bayesian" is enough to overcome at least asymptotic overconfidence.

We will show in the experiments (Section 3.2.6) that the same Bayesian treatment also mitigates asymptotic confidence in the multi-class case. However, extending the theoretical analysis

to this case is not straightforward, even with analytic approximations such as those by Gibbs (1998) and Wu et al. (2019).

3.2.1 Laplace Approximations

The results in the previous section imply that the asymptotic confidence of a binary ReLU classifier with a Gaussian approximation—either via a full or last-layer approximation—can be driven closer to uniform by controlling the covariance. In this section, we analyze the case when a Bayesian method in the form of a Laplace approximation is employed for obtaining the Gaussian. Although Laplace approximations are currently less popular than variational Bayes (VB), they have useful practical benefits: (i) they can be applied to any pre-trained network, (ii) whenever the approximation (3.6) can be employed, Proposition 3.3 holds, and (iii) no re-training is needed. Indeed, Laplace approximations can be attractive to practitioners who already have a working MAP-trained network, but want to enhance its uncertainty estimates further without decreasing performance.

The principle of Laplace approximations is as follows. Let $p(\theta \mid \mathcal{D}) \propto p(\theta) \prod_{x,y \in \mathcal{D}} p(y \mid x, \theta)$ be the posterior of a network f_θ. Then we can obtain a Gaussian approximation $p(\theta \mid \mathcal{D}) \approx \mathcal{N}(\theta \mid \mu, \Sigma)$ of the posterior by setting $\mu = \theta_{\text{MAP}}$ and $\Sigma := (-\nabla^2_\theta \log p(\theta \mid \mathcal{D})|_{\theta_{\text{MAP}}})^{-1}$, the inverse Hessian of the negative log-posterior at the mode. In the binary classification case, the likelihood $p(y \mid x, \theta)$ is assumed to be a Bernoulli distribution with parameter $\sigma(f_\theta(x))$. The prior $p(\theta)$ is assumed to be an isotropic Gaussian $\mathcal{N}(\theta \mid 0, \sigma_0^2 I)$.

While the prior variance σ_0^2 is tied to the MAP estimation (it can be derived from the weight decay), it is often treated as a separate hyperparameter and tuned after training (Ritter et al., 2018a). This treatment is useful in the case when one has only a pre-trained network and not the original training hyperparameters. Under this situation, in the following proposition, we analyze the effect of σ_0^2 on the asymptotic confidence presented by Theorem 3.5.

Proposition 3.8 (All-layer Laplace). *Let f_θ be a binary ReLU classification network modeling a Bernoulli distribution $p(y \mid x, \theta) = \text{Ber}(\sigma(f_\theta(x)))$ with parameter $\theta \in \mathbb{R}^d$. Let $\mathcal{N}(\theta \mid \mu, \Sigma)$ be the posterior obtained via a Laplace approximation with prior $\mathcal{N}(\theta \mid 0, \sigma_0^2 I)$, H be the Hessian of the negative log-likelihood at μ, and J be the Jacobian as in Theorem 3.5. Then for any input $x \in \mathbb{R}^n$, the confidence $\sigma(|z(x)|)$ is a decreasing function of σ_0^2 with limits*

$$\lim_{\sigma_0^2 \to \infty} \sigma(|z(x)|) \leq \sigma\left(\frac{|f_\mu(x)|}{1 + \sqrt{\pi/8\, \lambda_{\max}(H) \|Jx\|^2}}\right)$$
$$\lim_{\sigma_0^2 \to 0} \sigma(|z(x)|) = \sigma(|f_\mu(x)|).$$

Proof. The assumption on the prior implies that $-\log p(\theta) = 1/2\, \theta^\top (1/\sigma_0^2 I)\theta + \text{const}$, which has Hessian $1/\sigma_0^2 I$. Thus, the Hessian of the negative log posterior $-\log p(\theta \mid \mathcal{D}) = -\log p(\theta) - \log \prod_{x,y \in \mathcal{D}} p(y \mid x, \theta)$ is $1/\sigma_0^2 I + H$. This implies that the posterior covariance Σ of the Laplace approximation is given by

$$\Sigma = \left(\frac{1}{\sigma_0^2} I + H\right)^{-1}. \tag{3.12}$$

Therefore, the ith eigenvalue of Σ for any $i = 1, \ldots, n$ is

$$\lambda_i(\Sigma) = \frac{1}{1/\sigma_0^2 + \lambda_i(H)} = \frac{\sigma_0^2}{1 + \sigma_0^2 \lambda_i(H)}. \tag{3.13}$$

For all $i = 1, \ldots, n$, the derivative of $\lambda_i(\Sigma)$ w.r.t. σ_0^2 is $1/(1 + \sigma_0^2 \lambda_i(H))^2$ which is non-negative. This tells us that $\lambda_i(\Sigma)$ is a non-decreasing function of σ_0^2. Furthermore, it is also clear that $\sigma_0^2/(1 + \sigma_0^2 \lambda_i(H))$ goes to $1/\lambda_i(H)$ as σ_0^2 goes to infinity, while it goes to 0 as σ_0^2 goes to zero.

Now, we can write

$$|z(x)| = \frac{|f_\mu(x)|}{\sqrt{1 + \pi/8 \sum_{i=1}^d \lambda_i(\Sigma)(Q^\top d)_i^2}}, \tag{3.14}$$

where $\Sigma = Q \operatorname{diag}(\lambda_i(\Sigma), \ldots, \lambda_d(\Sigma))\, Q^\top$ is the eigendecomposition of Σ. It is therefore clear that the denominator of the r.h.s. is a non-decreasing function of σ_0^2. This implies $|z(x)|$ is a non-increasing function of σ_0^2.

For the limits, it is clear that $\lambda_{\min}(\Sigma)$ has limits $1/\lambda_{\max}(H)$ and 0 whenever $\sigma_0^2 \to \infty$ and $\sigma^2 \to 0$, respectively. From these facts, the right limit is immediate from Lemma 3.6 while the left limit is directly obtained by noticing that the denominator goes to 1 as $\sigma_0^2 \to 0$. □

The following is the last-layer version of the previous proposition.

Proposition 3.9 (Last-layer Laplace). *Let $g : \mathbb{R}^d \to \mathbb{R}$ be a binary linear classifier defined by $g \circ \phi(x) := w^\top \phi(x)$ where $\phi : \mathbb{R}^n \to \mathbb{R}^d$ is a ReLU network, modeling a Bernoulli distribution $p(y \mid x, w) = \mathcal{B}(\sigma(g \circ \phi(x)))$ with parameter $w \in \mathbb{R}^d$. Let $\mathcal{N}(w \mid \mu, \Sigma)$ be the posterior obtained via a Laplace approximation with prior $\mathcal{N}(\theta \mid 0, \sigma_0^2 I)$ and H be the Hessian of the negative log-likelihood at μ. Then for any input $x \in \mathbb{R}^n$, the confidence $\sigma(|z(x)|)$ is a non-increasing function of σ_0^2 with limits*

$$\lim_{\sigma_0^2 \to \infty} \sigma(|z(x)|) \leq \sigma\left(\frac{|\mu^\top \phi(x)|}{1 + \sqrt{\pi/8\, \lambda_{\max}(H) \|\phi(x)\|^2}}\right)$$
$$\lim_{\sigma_0^2 \to 0} \sigma(|z(x)|) = \sigma(|\mu^\top \phi(x)|).$$

Proof. The assumption on the prior implies that $-\log p(w) = 1/2\, w^\top (1/\sigma_0^2 I) w + \text{const}$, which has Hessian $1/\sigma_0^2 I$. Thus, the Hessian of the negative log posterior $-\log p(w \mid \mathcal{D}) = -\log p(w) - \log \prod_{x,y \in \mathcal{D}} p(y \mid x, w)$ is $1/\sigma_0^2 I + H$. This implies that the posterior covariance Σ of the Laplace approximation is given by

$$\Sigma = \left(\frac{1}{\sigma_0^2} I + H\right)^{-1}. \tag{3.15}$$

Therefore, the ith eigenvalue of Σ for any $i = 1, \ldots, n$ is

$$\lambda_i(\Sigma) = \frac{1}{1/\sigma_0^2 + \lambda_i(H)} = \frac{\sigma_0^2}{1 + \sigma_0^2 \lambda_i(H)}. \tag{3.16}$$

For all $i = 1, \dots, n$, the derivative of $\lambda_i(\Sigma)$ w.r.t. σ_0^2 is $1/(1 + \sigma_0^2 \lambda_i(H))^2$ which is non-negative. This tells us that $\lambda_i(\Sigma)$ is a non-decreasing function of σ_0^2. Furthermore, it is also clear that $\sigma_0^2/(1 + \sigma_0^2 \lambda_i(H))$ goes to $1/\lambda_i(H)$ as σ_0^2 goes to infinity, while it goes to 0 as σ_0^2 goes to zero.

Now, we can write.

$$|z(x)| = \frac{|\mu^\top \phi(x)|}{\sqrt{1 + \pi/8 \sum_{i=1}^{d} \lambda_i(\Sigma)(Q^\top \phi(x))_i^2}}, \tag{3.17}$$

where $\Sigma = Q \operatorname{diag}(\lambda_i(\Sigma), \dots, \lambda_d(\Sigma))\, Q^\top$ is the eigendecomposition of Σ. It is therefore clear that the denominator of the r.h.s. is a non-decreasing function of σ_0^2. This implies $|z(x)|$ is a non-increasing function of σ_0^2.

For the limits, it is clear that $\lambda_{\min}(\Sigma)$ has limits $1/\lambda_{\max}(H)$ and 0 whenever $\sigma_0^2 \to \infty$ and $\sigma^2 \to 0$, respectively. From these facts, the right limit is immediate from Lemma 3.6 while the left limit is directly obtained by noticing that the denominator goes to 1 as $\sigma_0^2 \to 0$. □

The result above shows that the "far-away" confidence decreases (up to some limit) as the prior variance increases. Meanwhile, we recover the far-away confidence induced by the MAP estimate as the prior variance goes to zero. One could therefore pick a value of σ_0^2 as high as possible for mitigating overconfidence. However, this is undesirable since it also lowers the confidence of the training data and test data around them (i.e. the so-called *in-distribution data*), thus, causing underconfident predictions. Another common way to set this hyperparameter is by maximizing the validation log-likelihood (Ritter et al., 2018a). This is also inadequate for our purpose since it only considers points close to the training data.

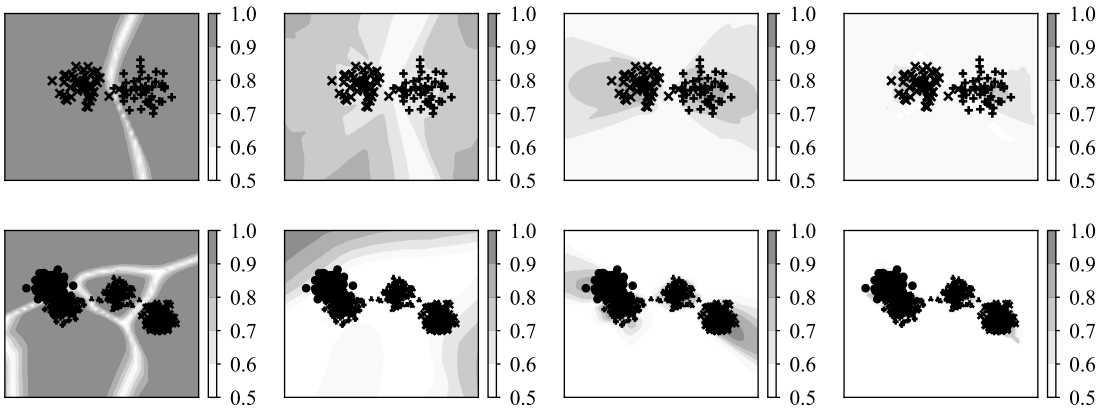
Inspired by Hendrycks et al. (2019) and Hein et al. (2019), we simultaneously prefer high confidence on the in-distribution validation set and low confidence (high entropy) on the out-of-distribution validation set. Let $\widehat{\mathcal{D}} := \{(\widehat{x}_i, \widehat{y}_i)\}_{i=1}^m$ be a validation set and $\widetilde{\mathcal{D}} := \{\widetilde{x}_i\}_{i=1}^m$ be an out-of-distribution dataset. We then pick the optimal σ_0^2 by solving the following one-parameter optimization problem:

$$\underset{\sigma_0^2}{\operatorname{argmax}} \frac{1}{m} \sum_{i=1}^{m} \log p(\widehat{y}_i \mid \widehat{x}_i, \mathcal{D}) + \lambda \mathbb{H}[p(y \mid \widetilde{x}_i, \mathcal{D})], \tag{3.18}$$

where $\lambda \in [0, 1]$ is controlling the trade-off between both terms. The first term in (3.18) is the standard cross-entropy loss over $\widehat{\mathcal{D}}$ while the second term is the negative predictive entropy over $\widetilde{\mathcal{D}}$. Alternatively, the second term can be replaced by the cross-entropy loss where the target is the uniform probability vector. In all our experiments, we simply assume that $\widetilde{\mathcal{D}}$ is a collection of uniform noise in the input space.

3.2.2 Related Work

The overconfidence problem of deep neural networks, and thus ReLU networks, has long been known in the deep learning community (Nguyen et al., 2015), although a formal description was only delivered recently. Many methods have been proposed to combat or at least detect this issue. *Post-hoc* heuristics based on temperature or Platt scaling (Platt et al., 1999; Guo et al.,



(a) MAP **(b)** Temp. scaling **(c)** LLLA **(d)** Full Laplace

Figure 3.4: Binary (top) and multi-class (bottom) toy classification problem. Background shades represent confidence estimates. Background shades in **(d)** are obstructed by the data—the full Laplace yields confidence estimates that are high only in the data regions.

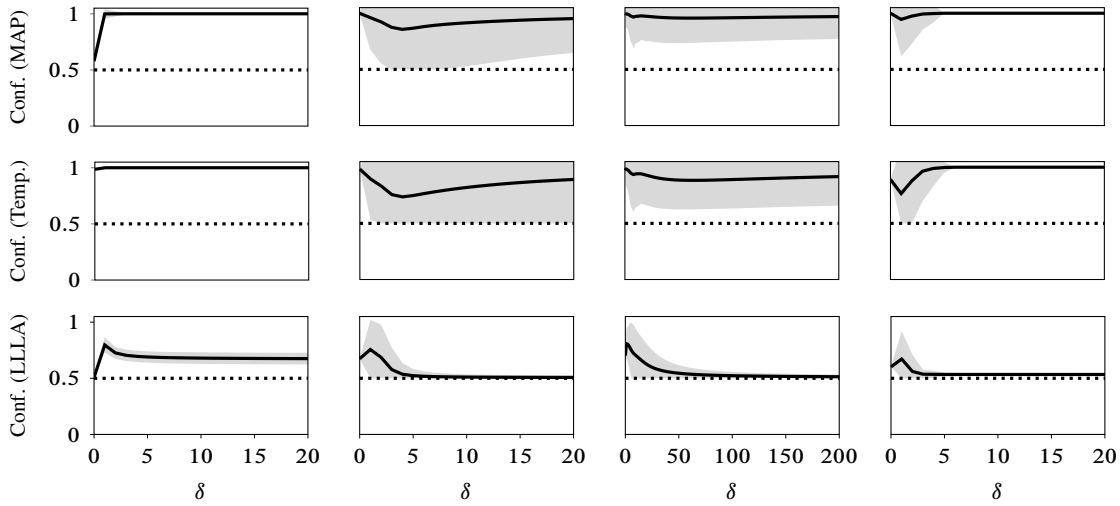
2017; Liang et al., 2018) are unable to detect inputs with arbitrarily high confidence far away from the training data (Hein et al., 2019).

Many works on uncertainty quantification in deep learning have recently been proposed. Gast & Roth (2018) proposed lightweight probabilistic networks via assumed density filtering. Malinin & Gales (2018; 2019); Sensoy et al. (2018) employ a Dirichlet distribution to model the distribution of a network's output. Lakshminarayanan et al. (2017) quantify predictive uncertainty based on the idea of model ensembling and frequentist calibration. Hein et al. (2019) proposed enhanced training objectives based on robust optimization to mitigate this issue. Meinke & Hein (2020) proposed a similar approach with provable guarantees. However, they either lack in their theoretical analysis or do not employ probabilistic or Bayesian approximations. Our results, meanwhile, provide a theoretical justification to the commonly-used Gaussian approximations of NNs' weights, both Bayesian (Graves, 2011; Blundell et al., 2015; Louizos & Welling, 2016; Maddox et al., 2019a, etc.) and non-Bayesian (Franchi et al., 2019; Lu et al., 2020, etc.).

Bayesian methods have long been thought to mitigate the overconfidence problem on any neural network (MacKay, 1992b). Empirical evidence supporting this intuition has also been presented (Liu et al., 2019; Wu et al., 2019, etc.). Our results complement these with a theoretical justification for the ReLU-logistic case. Furthermore, our theoretical results show that, in some cases, an expensive Bayesian treatment over all layers of a network is not necessary (Theorem 3.7). Our results are thus theoretically validating the usage of Bayesian generalized linear models (Gelman et al., 2008) (especially in conjunction with ReLU features) and last-layer Bayesian methods (Snoek et al., 2015; Wilson et al., 2016a;b; Riquelme et al., 2018); and complementing the empirical analyses of Ober & Rasmussen (2019) and Brosse et al. (2020).

3.2.3 Experiments

We corroborate our theoretical results via four experiments using various Bayesian methods with Gaussian approximations. In Section 3.2.4 we visualize the confidence of 2D binary and multi-class toy datasets. In Section 3.2.5 we empirically validate our main result that the confidence of



(a) Bin.-MNIST **(b)** Bin.-CIFAR-10 **(c)** Bin.-SVHN **(d)** Bin.-CIFAR-100

Figure 3.5: Confidence of MAP (top row), temperature scaling (middle row), and LLLA (bottom row) as functions of δ over the test sets of binary classification datasets. Thick blue lines and shades correspond to means and ± 3 standard deviations, respectively. Dotted lines signify the ideal uniform confidence.

binary classification datasets approaches finite constants as δ increases. Furthermore, we show empirically that this property also holds in the multi-class case, along with the usefulness of Bayesian methods in standard OOD detection tasks in Section 3.2.6. Finally, in Section 3.2.7, we show that our results also hold in the case of last-layer Gaussian processes.

Unless stated otherwise, we use LeNet (for MNIST) or ResNet-18 (for CIFAR-10, SVHN, CIFAR-100) architectures. We train these networks by following the procedure described by Meinke & Hein (2020): We train all networks we use in Table 3.2 for 100 epochs with batch size of 128. We use ADAM and SGD with 0.9 momentum with the initial learning rates of 0.001 and 0.1 for MNIST and CIFAR-10/SVHN/CIFAR-100 experiments, respectively, and we divide them by 10 at epoch 50, 75, and 95. Standard data augmentations, i.e. random crop and standardization are also used for training the network on CIFAR-10. To obtain the optimal hyperparameter σ_0^2, we follow (3.18) with λ set to 0.25.

We mainly use a ***last-layer Laplace approximation (LLLA)*** where a Laplace approximation with an exact Hessian or its Kronecker factors is applied only to the last layer of a network. Whenever the (binary) probit approximation cannot be used, we compute the predictive distribution via Monte Carlo integration with 100 posterior samples. Other Laplace approximations that we use will be introduced in the subsection where they are first employed. Besides the vanilla MAP method, we use the temperature scaling method (Guo et al., 2017) as a baseline since it claims to give calibrated predictions in the frequentist sense. In particular, the optimal temperature is found via a validation log-likelihood maximization using PyCalib (Wenger et al., 2019). For each dataset that we use, we obtain a validation set via a random split from the respective test set.[2] Lastly, all numbers reported in this section are averages along with their standard deviations over 10 trials.

[2]We use 50, 1000, and 2000 points for the toy, binary, and multi-class classification cases, respectively.

Table 3.1: OOD detection for far-away points in binary classification settings. The in-distribution datasets are Binary-MNIST, Binary-CIFAR-10, Binary-SVHN, and Binary-CIFAR-100. Each OOD dataset is obtained by scaling uniform noise images in the corresponding input space of the in-distribution dataset with $\delta = 100$. All values are means and standard deviations over 10 trials.

	MAP		**+Temp.**		**+LLLA**	
	MMC ↓	AUROC ↑	MMC ↓	AUROC ↑	MMC ↓	AUROC ↑
Binary-MNIST	99.9±0.0	-	100.0±0.0	-	79.4±0.9	-
Noise ($\delta = 100$)	100.0±0.0	0.2±0.1	100.0±0.0	45.1±5.8	**67.5±0.8**	**99.6±0.1**
Binary-CIFAR-10	96.3±0.3	-	90.5±0.6	-	76.4±0.3	-
Noise ($\delta = 100$)	98.9±1.0	11.3±10.3	97.6±2.2	11.3±10.3	**50.6±0.1**	**99.5±0.1**
Binary-SVHN	99.4±0.0	-	98.2±0.1	-	80.7±0.1	-
Noise ($\delta = 100$)	98.8±0.6	50.5±42.3	95.9±3.0	50.5±42.3	**51.2±0.6**	**99.8±0.1**
Binary-CIFAR-100	94.5±0.5	-	74.5±2.9	-	66.7±0.5	-
Noise ($\delta = 100$)	100.0±0.0	1.5±0.7	100.0±0.0	0.0±0.0	**53.5±0.1**	**93.6±1.8**

3.2.4 Toy Dataset

Here, the dataset is constructed by sampling the input points from k independent Gaussians. The corresponding targets indicate from which Gaussian the point was sampled. We use a 3-layer ReLU network with 20 hidden units at each layer. We use the exact Hessian and the full generalized-Gauss-Newton (GGN) approximation of the Hessian for the case of LLLA and all-layer Laplace approximations, respectively.

We show the results for the binary and multi-class cases in Fig. 3.4. The MAP predictions have high confidence everywhere except at the region close to the decision boundary. Temperature scaling assigns low confidence to the training data, while assigning high confidence far away from them. LLLA, albeit simple, yields high confidence close to the training points and high uncertainty otherwise, while maintaining the MAP's decision boundary. Furthermore, we found that the all-layer Laplace approximation makes the aforementioned finding stronger: the boundaries of the high-confidence regions are now closer to the training data.

3.2.5 Binary Classification

We validate our theoretical finding by plotting the test confidence of various binary classification datasets as functions of δ. Each dataset is constructed by picking two classes which are most difficult to distinguish, based on the confusion matrix of the corresponding multi-class problem.

As shown in Fig. 3.5, both MAP (top row) and temperature scaling (middle row) methods are overconfident for sufficiently large δ. Meanwhile, LLLA which represents Bayesian methods, mitigates this issue: As δ increases, the confidence converges to some constant close to the uniform confidence (one-half). Moreover, when $\delta = 1$ (the case of in-distribution data), LLLA retains higher confidence.

Table 3.1 further quantifies the results where we treat collections of 2000 uniform noise images scaled by $\delta = 100$ as the OOD datasets. Note that, while the resulting data points are not in the image space anymore, this construction is useful to assess the effectiveness of the Bayesian methods in unbounded problems. We report the standard metrics proposed by Hendrycks & Gimpel (2017): mean-maximum-confidence (MMC) and area-under-ROC-curve

Table 3.2: Multi-class OOD detection results for MAP, last-layer Laplace (LLLA), (all-layers) diagonal Laplace (DLA), and (all-layers) Kronecker-Factored Laplace (KFLA). Each "far-away" Noise dataset is constructed as in Table 3.1 with $\delta = 2000$. All values are averages and standard deviations over 10 trials.

	MAP		**+Temp.**		**+LLLA**		**+DLA**		**+KFLA**	
	MMC ↓	AUROC ↑	MMC ↓	AUROC ↑	MMC ↓	AUROC ↑	MMC ↓	AUROC ↑	MMC ↓	AUROC ↑
MNIST - MNIST	99.2±0.0	-	99.5±0.1	-	98.4±0.2	-	84.5±0.2	-	92.9±0.3	-
MNIST - EMNIST	82.3±0.0	89.2±0.1	87.6±1.4	88.9±0.2	70.2±1.9	**92.0±0.4**	**54.5±0.3**	87.7±0.4	58.7±0.4	89.6±0.3
MNIST - FMNIST	66.3±0.0	97.4±0.0	75.2±2.5	97.1±0.1	56.0±1.8	98.2±0.2	42.5±0.1	96.3±0.1	**39.9±0.5**	**98.6±0.1**
MNIST - Noise ($\delta = 2000$)	100.0±0.0	0.1±0.0	100.0±0.0	6.8±4.1	99.9±0.0	9.6±0.7	84.9±1.3	53.7±3.1	**55.6±2.0**	**97.3±0.4**
CIFAR-10 - CIFAR-10	97.1±0.1	-	95.4±0.2	-	92.8±1.1	-	88.4±0.1	-	86.5±0.1	-
CIFAR-10 - SVHN	62.5±0.0	95.8±0.1	54.6±0.6	96.1±0.0	45.9±1.6	**96.4±0.1**	43.3±0.1	95.5±0.1	**43.0±0.1**	94.8±0.1
CIFAR-10 - LSUN	74.5±0.0	91.9±0.1	66.9±0.6	92.2±0.1	57.4±1.9	92.7±0.4	49.0±0.5	**92.8±0.3**	**47.6±0.4**	92.2±0.2
CIFAR-10 - Noise ($\delta = 2000$)	98.7±0.2	10.9±0.4	98.4±0.2	10.0±0.5	**17.4±0.0**	**100.0±0.0**	60.7±2.0	89.6±1.1	61.8±1.5	87.6±0.9
SVHN - SVHN	98.5±0.0	-	97.4±0.2	-	93.2±1.0	-	88.8±0.0	-	90.8±0.0	-
SVHN - CIFAR-10	70.4±0.0	95.4±0.0	64.1±0.9	95.4±0.0	43.4±2.1	97.2±0.1	**38.0±0.1**	**97.6±0.0**	41.2±0.1	97.5±0.0
SVHN - LSUN	71.7±0.0	95.5±0.0	65.4±1.0	95.6±0.0	44.3±2.3	97.3±0.1	**39.5±0.7**	**97.5±0.2**	42.0±0.6	**97.5±0.1**
SVHN - Noise ($\delta = 2000$)	98.7±0.1	11.9±0.6	98.4±0.1	11.0±0.6	**27.5±0.1**	**99.6±0.0**	60.8±1.6	92.8±0.6	62.4±2.0	94.0±0.5
CIFAR-100 - CIFAR-100	81.2±0.1	-	78.9±0.8	-	74.6±0.2	-	76.4±0.2	-	73.4±0.2	-
CIFAR-100 - SVHN	53.5±0.0	78.8±0.1	49.2±1.2	79.2±0.1	42.7±0.3	**80.4±0.2**	46.0±0.1	79.6±0.2	**41.4±0.1**	80.1±0.2
CIFAR-100 - LSUN	50.7±0.0	81.0±0.1	46.8±1.1	81.1±0.1	39.8±0.2	**82.6±0.2**	43.5±0.3	81.5±0.2	**39.7±0.4**	81.6±0.3
CIFAR-100 - Noise ($\delta = 2000$)	99.5±0.1	2.8±0.2	99.4±0.1	2.6±0.2	**5.9±0.0**	**99.9±0.0**	41.5±1.5	84.2±0.9	37.1±1.3	84.2±0.8

(AUROC). Confirming our finding in Fig. 3.5, LLLA is able to detect OOD data with high accuracy: for the chosen values of δ, the MMC and AUROC values are close to the ideal values of 50 and 100, respectively. Both MAP and temperature scaling fail to do so since their confidence estimates saturate to one. These results (i) confirm our theoretical analysis, (ii) show that even a simple Bayesian method yields good uncertainty estimates, and (iii) temperature scaling is not calibrated for outliers far-away from the training data.[3]

3.2.6 Multi-class Classification

We also show empirically that Bayesian methods yield a similar behavior in multi-class settings. On top of LLLA, representing Bayesian methods, we employ various other scalable Laplace approximation techniques: diagonal Laplace approximation (DLA) where a diagonal Gaussian is used to approximate the posterior over all layers of a network, and Kronecker-factored Laplace approximation (KFLA) (Ritter et al., 2018a) where a matrix-variate normal is used to approximate the posterior over all layers. We use 20 posterior samples for both DLA and KFLA.

Table 3.3: Wall-clock time (in second) of posterior inferences and predictions over test sets.

	MNIST	**CIFAR-10**	**SVHN**	**CIFAR-100**
Inference				
MAP	-	-	-	-
+Temp.	0.0	0.0	0.0	0.0
+LLLA	1.8	23.0	33.7	23.1
+DLA	1.3	22.9	33.6	23.0
+KFLA	4.3	78.1	115.1	78.6
Prediction				
MAP	0.4	1.2	2.7	1.2
+Temp.	0.4	1.2	2.7	1.2
+LLLA	0.9	1.7	4.3	1.6
+DLA	7.3	100.0	260.6	100.4
+KFLA	21.5	151.0	392.8	151.7

For each training dataset we evaluate all methods both in the non-asymptotic (the corresponding OOD test datasets, e.g. SVHN and LSUN for CIFAR-10) and asymptotic (Noise datasets) regime. Each "far-away" Noise dataset is constructed by scaling 2000 uniform noise images in the corresponding input space with $\delta = 2000$. As in the previous section, we report the MMC and AUROC metrics.

[3]This confirms the theoretical arguments of Hein et al. (2019).

As presented in Table 3.2, all the Bayesian methods improve the OOD detection performance of the base models both in the non-asymptotic and asymptotic regime. Especially in the asymptotic regime, all the Bayesian methods perform well, empirically confirming our hypothesis that our theoretical analysis carries over to the multi-class setting. Meanwhile, both MAP's and temperature scaling's MMC and AUROC are close to 100 and 0, respectively.[4] Moreover, while LLLA is the simplest Bayesian method in this experiment, it often outperforms DLA and KFLA. Our finding agrees with the prior observation that last-layer Bayesian approximations are often sufficient (Ober & Rasmussen, 2019; Brosse et al., 2020).

In Table 3.3, we present the computational cost analysis in terms of wall-clock time. We measure the time required for each method to do posterior inference (or finding the optimal temperature) and to make predictions. While MAP and temperature scaling are fast, as we have shown in the previous results, they are overconfident. Among the Bayesian methods, since the cost of LLLA is constant w.r.t. the network depth, we found that it is up to two orders of magnitude faster than DLA and KFLA when making predictions. All in all, this finding, combined with the previous results, makes this simple Bayesian method attractive in applications.

3.2.7 Last-layer Gaussian Processes

It has been shown that Gaussian-approximated linear models with infinitely many features, e.g. two-layer networks with infinitely wide hidden layers, are equivalent to Gaussian processes (Neal, 2012). In the language of Theorem 3.7, this is the case when the dimension of the feature space $\mathbb{R}^d$ goes to infinity. It is therefore interesting to see, at least empirically, whether low asymptotic confidence is also attained in this case.

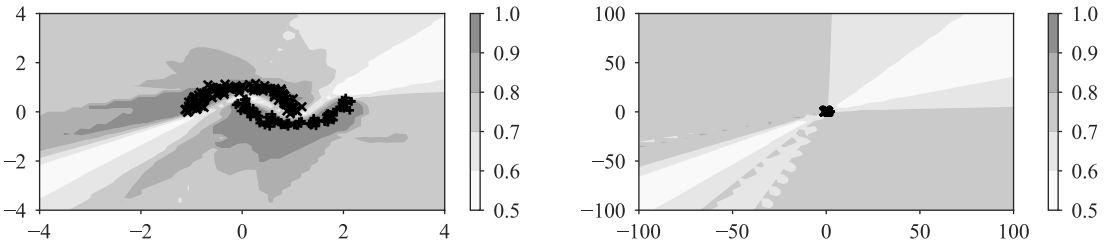
Table 3.4: Multi-class OOD detection results for deep kernel learning (DKL). Each "far-away" Noise dataset is constructed as in Table 3.2 with $\delta = 2000$. Values are averages and std. dev. over 10 trials.

Train - Test	MMC ↓	AUROC ↑
MNIST - MNIST	99.6±0.0	-
MNIST - EMNIST	83.9±0.0	94.4±0.1
MNIST - FMNIST	70.6±0.1	98.8±0.0
MNIST - Noise	58.6±0.5	99.7±0.0
CIFAR-10 - CIFAR-10	97.5±0.0	-
CIFAR-10 - SVHN	50.6±0.1	98.6±0.0
CIFAR-10 - LSUN	77.9±0.3	93.4±0.1
CIFAR-10 - Noise	56.5±0.7	98.5±0.1
SVHN - SVHN	98.6±0.0	-
SVHN - CIFAR-10	72.7±0.0	96.0±0.0
SVHN - LSUN	76.7±0.1	95.1±0.1
SVHN - Noise	48.6±0.7	99.4±0.0
CIFAR-100 - CIFAR-100	80.5±0.0	-
CIFAR-100 - SVHN	72.7±0.1	63.1±0.1
CIFAR-100 - LSUN	66.8±0.4	69.7±0.3
CIFAR-100 - Noise	43.1±1.1	90.3±0.7

While unlike LLLA, deep kernel learning (DKL, Wilson et al., 2016a;b) is not a *post-hoc* method, it is a suitable model for showcasing our theory in the case of last-layer Gaussian processes. We therefore train stochastic variational DKL models (Wilson et al., 2016b) which use the same networks used in the previous experiment (minus the top layer) as their feature extractors, following the implementation provided by GPyTorch (Gardner et al., 2018). The training protocol is identical as before. To compute each prediction, we use 20 samples from the Gaussian process posterior. We are mainly interested in the performance of DKL in term of multi-class OOD detection (both in asymptotic and non-asymptotic regimes), similar to the previous section.

The results are presented in Table 3.4. When compared to the results of the MAP estimation in Table 3.2, we found that DKL is able to mitigate asymptotic overconfidence (see results

[4] I.e. the worst values for those metrics.



(a) Zoomed in **(b)** Zoomed out

Figure 3.6: Confidence estimates of a Laplace-approximated BNN.

against the Noise dataset). These results empirically verify that our analysis also holds in the non-parametric infinite-width regime. Nevertheless, we found that LLLA generally outperforms DKL both in term of MMC and AUROC metrics. This finding, along with the simplicity and efficiency of LLLA make it more attractive than DKL, especially since DKL requires retraining and thus cannot simply be applied to pre-trained ReLU networks.

3.3 ReLU-GP Residual

We have seen that BNNs, even with a simple Gaussian approximate posterior, can help to mitigate this problem in binary classifications. The crux of the proof is the observation that in an outer linear region, the predictive distribution (via the probit approximation) over the prediction y_* of a test input x_* is given by[5]

$$p(y_* = 1 \mid \delta x_*, \mathcal{D}) \approx \sigma\left(\frac{\delta u^\top x_*}{\sqrt{1 + \pi/8\, v(\delta x_*)}}\right), \tag{3.19}$$

where σ is the logistic-sigmoid function, u is the parameter vector corresponding to the linear region and the quadratic function v maps δx_* to the variance of the network output. Unfortunately, both the numerator and denominator above are linear in δ and thus altogether $p(y = 1 \mid \delta x_*, \mathcal{D})$ only converges to a constant strictly less than 1 as $\delta \to \infty$, not necessarily the ideal uniform confidence prediction. BNNs can therefore still be overconfident, albeit less so than the point-estimated counterpart (Fig. 3.6).

From Section 3.1 it becomes clear that the asymptotic miscalibration of ReLU BNNs is due to the finite number of ReLU features used, which results in only quadratic variance growth. An infinite-ReLU GP with the cubic spline kernel has cubic variance growth, which, combined with the probit approximation, yields uniform confidence in the limit. But of course, full GP inference is prohibitively expensive. In this section, we propose a cheap, *post-hoc* way to extend any pre-trained ReLU BNN with the aforementioned GP by extending the cubic spline kernel and exploiting its two important properties. We will see that the resulting model approximates the full GP posterior and combines the predictive power of the BNN with a guarantee for asymptotically uniform confidence. While in our analysis we employ network linearization for analytical tractability, the method can be applied via MC-integration as well (cf. Section 3.3.5).

[5] We omit the bias parameter for simplicity.

3.3.1 The Double-Sided Cubic Spline Kernel

The ReLU activation function $\text{ReLU}(z) := \max(0, z)$ (Nair & Hinton, 2010) has become the *de facto* choice of non-linearity in deep learning. Given an arbitrary real number c, it can be generalized as $\text{ReLU}(z; c) := \max(0, z - c)$, with the "kink" at location c. An alternative formulation, useful below, is in terms of the Heaviside function H as[6]

$$\text{ReLU}(z; c) = H(z - c) \cdot (z - c).$$

We may define a collection of d such ReLU functions evaluated at some point in $\mathbb{R}$ as a function $\phi : \mathbb{R} \to \mathbb{R}^d$ defined by

$$\phi(z) := (\text{ReLU}(z; c_1), \ldots, \text{ReLU}(z; c_d))^\top.$$

We call this function the ***ReLU feature map***, which can be interpreted as "placing" ReLU functions at different locations in $\mathbb{R}$.

Consider a linear model $f : \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$ defined by $g(x; w) := w^\top \phi(x)$. Suppose ϕ regularly places d generalized ReLU functions centered at evenly-spaced points $(c_i)_{i=1}^d$ on $[c_{\min}, c_{\max}] \subset \mathbb{R}$, where $c_{\min} < c_{\max}$. If we consider a Gaussian prior

$$p(w) := \mathcal{N}\big(w \mid 0, \sigma^2 d^{-1}(c_{\max} - c_{\min})I\big),$$

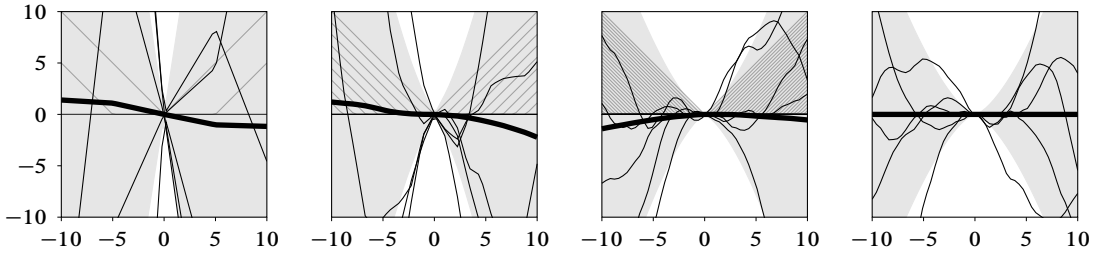
then as $d \to \infty$, the distribution over g is a Gaussian process with mean 0 and covariance:

$$\begin{aligned}&\widehat{K}^1(x, x'; c_{\min}, \sigma^2) \\ &\quad := \sigma^2 H(\bar{x} - c_{\min})\left(\frac{1}{3}(\bar{x}^3 - c_{\min}^3) - \frac{1}{2}(\bar{x}^2 - c_{\min}^2)(x + x') + (\bar{x} - c_{\min})xx'\right),\end{aligned} \tag{3.20}$$

where the superscript 1 denotes the fact that this function is over a 1-dimensional input space, $\bar{x} := \min(x, x')$, and H is the Heaviside step function. To show this, note that the covariance of the output of f is given by

$$\begin{aligned}\text{cov}(f(x), f(x')) &= \sigma^2 \frac{c_{\max} - c_{\min}}{d} \phi(x)^\top \phi(x') \\ &= \sigma^2 \frac{c_{\max} - c_{\min}}{d} \sum_{i=1}^{d} \text{ReLU}(x; c_i)\text{ReLU}(x'; c_i) \\ &= \sigma^2 \frac{c_{\max} - c_{\min}}{d} \sum_{i=1}^{d} H(x - c_i)H(x' - c_i)(x - c_i)(x' - c_i) \\ &= \sigma^2 \frac{c_{\max} - c_{\min}}{d} \sum_{i=1}^{d} H(\min(x, x') - c_i)\big(c_i^2 - c_i(x + x') + xx'\big),\end{aligned} \tag{3.21}$$

[6] $H(x) = 1$ if $x > 0$ and $H(x) = 0$ otherwise.



(a) $d = 6$ **(b)** $d = 20$ **(c)** $d = 60$ **(d)** $d \to \infty$

Figure 3.7: The construction of a GP prior with the proposed "ReLU kernel", as the limiting covariance of the output of a Bayesian linear model with d ReLU features (grey), arranged at regular intervals, oriented away from the origin. Red curves are function samples with the thick one being the mean, and the red shade their std. dev. With finite d **(a-c)**, the variance grows only quadratically, leading to the asymptotic overconfidence in ReLU BNNs. But, with $d \to \infty$ **(d)**, the variance grows *cubically* away from the origin. The fact that this kernel has zero mean and negligible variance near the origin enables us to easily combine this GP with standard finite pre-trained ReLU BNNs.

where the last equality follows from the fact that both x and x' must be greater than or equal to c_i, and by expanding the quadratic form in the second line. Since (3.21) is a Riemann sum, in the limit of $d \to \infty$, it is expressed by the following integral

$$\begin{aligned}\lim_{d\to\infty} \operatorname{cov}(f(x), f(x')) &= \sigma^2 \int_{c_{\min}}^{c_{\max}} H(\bar{x} - c)\big(c^2 - c(x + x') + xx'\big)\, dc \\ &= \sigma^2 H(\bar{x} - c_{\min}) \int_{c_{\min}}^{\min(\bar{x}, c_{\max})} c^2 - c(x + x') + xx'\, dc \\ &= \sigma^2 H(\bar{x} - c_{\min}) \left[\frac{1}{3}(z^3 - c_{\min}^3) - \frac{1}{2}(z^2 - c_{\min}^2)(x + x') + (z - c_{\min})xx'\right]\end{aligned}$$

where we have defined $z := \min(\bar{x}, c_{\max})$. The term $H(\bar{x} - c_{\min})$ has been added in the second equality as the previous expression is zero if $\bar{x} \leq c_{\min}$ (since in this region, all the ReLU functions evaluate to zero). Note that

$$H(\bar{x} - c_{\min}) = H(x - c_{\min}) H(x' - c_{\min})$$

is itself a positive definite kernel. We also note that $c_{\max}$ can be chosen sufficiently large so that $[-c_{\max}, c_{\max}]$ contains the data for sure, e.g. this is anyway true for data from bounded domains like images, and thus we can set $z = \bar{x} = \min(x, x')$.

Since the expression (3.20) does not depend on $c_{\max}$, we can consider the limit $c_{\max} \to \infty$, and thus this kernel is non-zero on $(c_{\min}, \infty)$. This covariance function is the ***cubic spline kernel*** (Wahba, 1990). The name indicates that posterior mean of the associated GP is piecewise-cubic. But it also has variance $\widehat{K}^1(x, x; c_{\min}, \sigma^2)$ which is cubic in x and negligible for x close to $c_{\min}$.

The cubic spline kernel is one-sided in the sense that it has zero variance on $(-\infty, c_{\min})$, and therefore is unsuitable for modeling over the entire domain. This is easy to fix by first setting $c_{\min} = 0$ to obtain a kernel $K_r^1(x, x'; \sigma^2) := \widehat{K}^1(x, x'; 0, \sigma^2)$ which is non-zero only on $(0, \infty)$. Now, by an entirely analogous construction with infinitely many ReLU functions pointing to the opposite direction (i.e. left) via $\mathrm{ReLU}(-z; c)$, we obtain another kernel $K_l^1(x, x'; \sigma^2) :=$

$K_r^1(-x, -x'; \sigma^2)$, which is non-zero only on $(-\infty, 0)$. Combining them together, we obtain the following kernel, which covers the whole real line:

$$K^1(x, x'; \sigma^2) := K_l^1(x, x'; \sigma^2) + K_r^1(x, x'; \sigma^2).$$

See Fig. 3.7 for an intuition. Note in particular that the variance $K^1(0, 0)$ at the origin is zero. This is a key feature of this kernel that enables us to efficiently combine the resulting GP prior with a pre-trained BNN.

For multivariate input domains, we define

$$K(x, x'; \sigma^2) := \frac{1}{n} \sum_{i=1}^{n} K^1(x_i, x_i'; \sigma^2) \tag{3.22}$$

for any $x, x' \in \mathbb{R}^n$ with $n > 1$. We here deliberately use a summation, instead of the alternative of a product, since we want the associated GP to add uncertainty whenever *at least* one input dimension has non-zero value. (By contrast, a product $K(x, x')$ is zero if one of the $K^1(x_i, x_i')$ is zero.) We call this kernel the ***double-sided cubic spline (DSCS) kernel***. Similar to the one-dimensional case, two crucial properties of this kernel are that it has negligible variance around the origin of $\mathbb{R}^n$ and for any $x_* \in \mathbb{R}^n$ and $\delta \in \mathbb{R}$, the value $K(\delta x_*, \delta x_*)$ is *cubic* in δ.

3.3.2 ReLU-GP Residual

For simplicity, we start with real-valued BNNs and discuss the generalization to multi dimensional output later. Let $f : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}$ be an L-layer real-valued ReLU BNN parametrized by $\theta \in \mathbb{R}^d$. Since f by itself can be asymptotically overconfident, it has *residual* in its uncertainty estimates far from the data. Our goal is to extend f with the GP prior that arises from the DSCS kernel, to model this uncertainty residual. We do so by placing infinitely many ReLU features over its input space $\mathbb{R}^n$ by following the DSCS kernel construction in the previous section. Then, we arrive at a zero-mean GP prior $\mathcal{GP}(\widehat{f} \mid 0, K)$ over a real-valued random function $\widehat{f} : \mathbb{R}^n \to \mathbb{R}$. Following previous works (Wahba, 1978; O'Hagan, 1978; Qiu et al., 2020), we use this GP prior to model the residual of f by defining

$$\widetilde{f} := f + \widehat{f}, \qquad \text{where } \widehat{f} \sim \mathcal{GP}(0, K), \tag{3.23}$$

and call this method ***ReLU-GP residual (RGPR)***.

We now analyze RGPR. Besides linearization, we assume that the DSCS kernel has, without loss of generality, a negligibly small value at the data, i.e. $K(x_i, x_*) \approx 0$ for all $(x_i)_{i=1}^m$ and any i.i.d. test point x_*. Note that this can always be satisfied by centering and scaling. The error of this approximation is stated in the following.

Lemma 3.10. *Let $0 < \delta < 1$, and let $\sigma^2 > 0$ be a constant. For any $x, x' \in \mathbb{R}^n$ with $\|x\|^2, \|x'\|^2 \leq \delta$ we have $K(x, x'; \sigma^2) \in O(\delta^3)$.*

Proof. First, note that $\|x\|^2, \|x'\|^2 \leq \delta$ implies $x_i, x_i' \leq \delta$ for all $i = 1, \ldots, n$. By definition of the 1D DSCS kernel $K_r^1(x_i, x_i'; \sigma^2)$, it is upper bounded by $\sigma^2(\frac{1}{3}\delta^3)$ since $\bar{x_i} = \min(x_i, x_i') \leq \delta$; and similarly for $K_l^1(x_i, x_i'; \sigma^2)$ by the symmetry of the DSCS kernel. Thus $K^1(x_i, x_i'; \sigma^2) \in O(\delta^3)$ and hence $K(x, x'; \sigma^2)$ also is, since it is just the average of $\{K^1(x_i, x_i'; \sigma^2)\}_{i=1}^n$. □

Using this approximation and the following lemma by Higham (1994), we show the approximate GP posterior of $\widetilde{f}$.

Lemma 3.11 (Higham, 1994). *Let $Am = b$ and $(A + \Delta A)n = b + \Delta b$, and let E and d be a matrix and vector with non-negative components, respectively. Assume that $\|\Delta A\| \leq \varepsilon \|E\|$ and $\|\Delta b\| \leq \varepsilon \|d\|$, and that $\varepsilon \|A^{-1}\| \|E\| < 1$, where $\varepsilon > 0$. Then,*

$$\|m - n\| \leq \frac{\varepsilon(\|A^{-1}\| \|d\| + \|m\| \|A^{-1}\| \|E\|)}{1 - \varepsilon \|A^{-1}\| \|E\|}. \tag{3.24}$$

□

Proposition 3.12 (RGPR's GP Posterior). *Let $f : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}$ be a ReLU BNN with weight distribution $\mathcal{N}(\theta \mid \mu, \Sigma)$, and let $\mathcal{D} := (x_i, y_i)_{i=1}^m =: (X, Y)$ be a dataset. Assume that $\|x_i\|^2, \|x\|^2 \leq \delta$ for all $i = 1, \ldots, m$ and any i.i.d. test point $x \in \mathbb{R}^n$, with $0 < \delta < 1$. Then given an i.i.d. input point $x_* \in \mathbb{R}^n$, under the linearization of f w.r.t. $theta$ around μ, the GP posterior over $\widetilde{f}_*$ is a Gaussian with mean and variance*

$$\mathbb{E}(\widetilde{f}_* \mid \mathcal{D}) \approx f(x_*; \mu) + h_*^\top C^{-1}(Y - f(X; \mu)), \tag{3.25}$$

$$\mathrm{Var}(\widetilde{f}_* \mid \mathcal{D}) \approx g(x_*)^\top \Sigma g(x_*) + K(x_*, x_*) - h_*^\top C^{-1} h_*, \tag{3.26}$$

respectively, where $h_* := (\mathrm{Cov}(f(x_*), f(x_1)), \ldots, \mathrm{Cov}(f(x_*), f(x_m)))^\top$, while C is the covariance matrix $\big(\mathrm{Cov}(f(x_i), f(x_j))\big)_{ij}^m$, and $f(X; \mu) := (f(x_1; \mu), \ldots, f(x_m; \mu))^\top$. Moreover, the approximation errors are in

$$O\big((\delta^6 \|C^{-1}\| \|C^{-1}(Y - f(X; \mu))\|)/(1 - \delta^3 \|C^{-1}\|)\big)$$

and

$$O\big((\delta^6(\|C^{-1}\| + \|C^{-1}\| \|C^{-1} h_*\|))/(1 - \delta^3 \|C^{-1}\|)\big)$$

for (3.25) *and* (3.26)*, respectively.*

Proof. Under the linearization of f w.r.t. θ around μ, we have

$$f(x; \theta) \approx f(x; \mu) + \underbrace{\nabla_\theta f(x; \theta)|_\mu}_{=: g(x)}{}^\top (\theta - \mu).$$

So, the distribution over the function output $f(x)$, where θ has been marginalized out, is given by $f(x) \sim \mathcal{N}(f(x; \mu), g(x)^\top \Sigma g(x))$—see (1.38). The definition of RGPR in (3.23) thus implies that

$$\widetilde{f}(x) \sim \mathcal{N}(f(x; \mu), g(x)^\top \Sigma g(x) + K(x, x)), \tag{3.27}$$

since $\widetilde{f}(x)$ is a sum of two Normal r.v.s. Note that we can see this distribution as a marginal distribution of a Gaussian process with a mean function $f(\cdot\,; \mu)$ and a kernel $\overline{K}$ defined by

$$(x, x') \mapsto g(x)^\top \Sigma g(x') + K(x, x').$$

Thus, we write the following GP prior

$$\widetilde{f} \sim \mathcal{GP}(f(\cdot;\mu), \overline{K}). \tag{3.28}$$

Our goal is to find the corresponding GP posterior under the dataset $\mathcal{D}$.

Let $x_* \in \mathbb{R}^n$ be an arbitrary test point. The GP posterior at x_*, i.e. the predictive distribution of $\widetilde{f_*} := f(x_*)$, is thus identified by the following mean and variance (see Section 1.4):

$$\mathbb{E}(\widetilde{f_*} \mid \mathcal{D}) = f(x_*;\mu) + \overline{K}(x_*, X)^\top \overline{K}(X,X)^{-1}(Y - f(X;\mu)) \tag{3.29}$$

$$\mathrm{Var}(\widetilde{f_*} \mid \mathcal{D}) = \overline{K}(x_*, x_*) - \overline{K}(x_*, X)^\top \overline{K}(X,X)^{-1}\overline{K}(x_*, X), \tag{3.30}$$

where we have used the shorthand $\overline{K}(x_*, X) := (\overline{K}(x_*, x_1), \ldots, \overline{K}(x_*, x_m))^\top$ and $\overline{K}(X,X)$ is the $m \times m$ kernel matrix of $\overline{K}$ under the training inputs X. For the latter we can also write $\overline{K}(X,X) = C + K(X,X)$, where C is the kernel matrix of $g(x)^\top \Sigma g(x')$ under X.

Since we assume $\|x_m\|^2, \|x\|^2 \leq \delta$ for all $i = 1, \ldots, m$ and any i.i.d. test point $x \in \mathbb{R}^n$, we have $K(x, x_i) \approx 0$. Thus, we have $\overline{K}(X,X) \approx C$ and

$$\begin{aligned}\overline{K}(x_*, X) &\approx (g(x_*)^\top \Sigma g(x_1), \ldots, g(x_*)^\top \Sigma g(x_m))^\top \\ &= (\mathrm{Cov}(f(x_*), f(x_1)), \ldots, \mathrm{Cov}(f(x_*), f(x_1)))^\top = h_*,\end{aligned}$$

where the covariances above are of the network's outputs under the linearization. And so the mean and the variance of the GP posterior simplify to

$$\mathbb{E}(\widetilde{f_*} \mid \mathcal{D}) \approx f(x_*;\mu) + h_*^\top C^{-1}(Y - f(X;\mu))$$

and

$$\mathrm{Var}(\widetilde{f_*} \mid \mathcal{D}) \approx g(x_*)^\top \Sigma g(x_*) + K(x_*, x_*) - h_*^\top C^{-1} h_*.$$

We have thus obtained both (3.25) and (3.26).

The only thing that remains is to obtain the approximation errors of both the mean and variance above. Using Lemma 3.11, we find the error of $(C + K(X,X))^{-1}(Y - f(X;\mu))$ in (3.29) due to RGPR, i.e. we quantify the error caused by δ presents in $K(X,X)$. We set $A = C$, $\Delta A = K(X,X)$, and $b = Y - f(X;\mu)$. Moreover, we set $m = C^{-1}(Y - f(X;\mu))$ and $n = (C + K(X,X))^{-1}(Y - f(X;\mu))$. For simplicity, we let E be a matrix where all its components are 1 and set $\varepsilon = \delta^3 c$ for some constant c s.t. the conditions in Lemma 3.11 are satisfied. Note that the δ^3 term in ε is so that the condition $\|\Delta A\| \leq \varepsilon \|E\|$ is satisfied, since one can write $\|\Delta A\| = c_0 \|E\|$ where $c_0 \in O(\delta^3)$. Moreover, we set $d = 0$ since $\Delta b = 0$. Plugging these into (3.24), we thus have

$$\|m - n\| \in O\left(\frac{\delta^3 \|A^{-1}\| \|m\|}{1 - \delta^3 \|A^{-1}\|}\right).$$

Combining this with the $O(\delta^3)$ error in the approximation $\overline{K}(x_*, X) \approx h_*$, we conclude that using (3.25) as an approximation of (3.29) incurs an error of

$$O\left(\frac{\delta^6 \|A^{-1}\| \|m\|}{1 - \delta^3 \|A^{-1}\|}\right),$$

which is small since $\delta \in (0, 1)$.

For the approximation error of the variance, we use A, ΔA, E, and ε as before. But, here we set $b = h_*$, $\Delta b = K(x_*, X)$, and $d = 1$. Moreover, we set $m = C^{-1}h_*$ and $n = (C + K(X, X))^{-1}(h_* + K(x_*, X))$. Then, plugging them into Lemma 3.11, we obtain

$$\|m - n\| \in O\left(\frac{\delta^3(\|A^{-1}\| + \|A^{-1}\|\|m\|)}{1 - \delta^3\|A^{-1}\|}\right).$$

Combining this with the approximation error in $\overline{K}(x_*, X) \approx h_*$ as before, we obtain the desired result. □

While this result is applicable to any Gaussian weight distribution, an interesting special case is where we assume that the BNN is *well-trained*, i.e. we have a Gaussian (approximate) posterior $p(\theta \mid \mathcal{D})$ which induces (i) *accurate prediction* and (ii) *high output confidence* on each of the training data. In this case, due to (i), the last term of (3.25) is negligible since the residual $Y - f(X; \mu)$ is close zero. Moreover, the last term in (3.26) can be upper-bounded by

$$h_*^\top C^{-1} h_* \leq \lambda_{\max}\|h_*\|^2 = \lambda_{\max} \sum_{i=1}^{m} \mathrm{Cov}(f(x_*), f(x_i))^2,$$

where $\lambda_{\max}$ denotes the largest eigenvalue of C^{-1}. The last summand above can further be upper-bounded via the Cauchy-Schwarz inequality by

$$\mathrm{Cov}(f(x_*), f(x_m))^2 \leq \mathrm{Var}(f(x_*))\mathrm{Var}(f(x_m)).$$

But assumption (ii) implies that $\mathrm{Var}(f(x_i))$ is close to zero for all $i = 1, \ldots, m$. Thus, if f is a pre-trained ReLU BNN, we approximately have

$$\widetilde{f_*} \sim \mathcal{N}(f(x_*; \mu), g_*^\top \Sigma g_* + K_*), \tag{3.31}$$

which can be thought of as arising from the sum of two Gaussian random variables

$$f_* \sim \mathcal{N}(f(x_*; \mu), g_*^\top \Sigma g_*) \qquad \text{and} \qquad \widehat{f_*} \sim \mathcal{N}(0, K_*).$$

Thus, we are back to the definition of RGPR in (3.23). That is, unlike previous works on modeling residuals with GPs (Wahba, 1978; O'Hagan, 1978; Qiu et al., 2020), the *GP posterior* of RGPR can approximately be written as *a posteriori* f plus *a priori* $\widehat{f}$. RGPR can hence be applied *post-hoc*, after the usual training process of the BNN. Furthermore, we see that RGPR does indeed model only the uncertainty residual of the BNN since it only affects the predictive variance. In particular, it does not affect the output mean of the BNN and thus preserves its predictive accuracy—this is often desirable in practice since the main reason for using deep ReLU nets is due to their accurate predictions.

Generalization to BNNs with multiple outputs is straightforward. Let $f : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^k$ be a vector-valued, pre-trained, L-layer ReLU BNN with posterior $\mathcal{N}(\theta \mid \mu, \Sigma)$ on $\mathbb{R}^d$. We assume that the following real-valued random functions $(\widehat{f^{(i)}} : \mathbb{R}^n \to \mathbb{R})_{i=1}^k$ are i.i.d. as the GP prior $\mathcal{GP}(0, k)$ (3.23). Thus, for any $x_* \in \mathbb{R}^n$, defining $\widehat{f_*} := (\widehat{f_*^{(1)}}, \ldots, \widehat{f_*^{(k)}})^\top$, we have

$p(\widehat{f_*}) = \mathcal{N}(0, K_* I)$, and so under the linearization of f, this implies that the marginal GP posterior of RGPR is approximately given by the following k-variate Gaussian

$$p(\widetilde{f_*} \mid x_*, \mathcal{D}) \approx \mathcal{N}(f_\mu(x_*), J_*^\top \Sigma J_* + K_* I). \tag{3.32}$$

We can do so since intuitively (3.32) is simply obtained as a result of "stacking" k independent $\widetilde{f_*^{(i)}}$'s, each of which satisfies Proposition 3.12. The following lemma shows that asymptotically, the marginal variances of $\widetilde{f_*}$ grow cubically as we scale the test point.

Proposition 3.13 (Asymptotic Variance Growth). *Let $f : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^k$ be a pre-trained ReLU network with posterior $\mathcal{N}(\theta \mid \mu, \Sigma)$ and $\widetilde{f}$ be obtained from f via RGPR. Suppose that the linearization of f w.r.t. θ around μ is employed. For any $x_* \in \mathbb{R}^n$ with $x_* \neq 0$ there exists $\alpha > 0$ such that for any $\delta \geq \alpha$ and each $c = 1, \ldots, k$, the variance* $\mathrm{Var}(\widetilde{f^{(c)}}(\delta x_*))$ *under* (3.32) *is in* $\Theta(\delta^3)$.

Proof. Let $x_* \in \mathbb{R}^n$ with $x_* \neq 0$ be arbitrary. By Lemma 3.1 and definition of ReLU network, there exists a linear region R and real number $\alpha > 0$ such that for any $\delta \geq \alpha$, the restriction of f to R can be written as

$$f|_R(\delta x; \theta) = W(\delta x) + b,$$

for some matrix $W \in \mathbb{R}^{k \times N}$ and vector $b \in \mathbb{R}^k$, which are functions of the parameter θ, evaluated at μ. In particular, for each $c = 1, \ldots, k$, the c-th output component of $f|_R$ can be written as

$$f^{(c)}|_R = w_c^\top (\delta x) + b_c,$$

where w_c and b_c are the c-th row of W and b, respectively.

Let $c \in \{1, \ldots, C\}$ and let $j_c(\delta x_*)$ be the c-th column of the Jacobian $J(\delta x_*)$ as defined in (1.37). Then by definition of $p(\widetilde{f_*} \mid x_*, \mathcal{D})$, the variance of $\widetilde{f^{(c)}}|_R(\delta x_*)$—the c-th diagonal entry of the covariance of $p(\widetilde{f_*} \mid x_*, \mathcal{D})$—is given by

$$\mathrm{Var}(\widetilde{f^{(c)}}|_R(\delta x_*)) = j_c(\delta x_*)^\top \Sigma j_c(\delta x_*) + K(\delta x_*, \delta x_*).$$

Now, from the definition of the DSCS kernel in (3.22), we have

$$k(\delta x_*, \delta x_*) = \frac{1}{n} \sum_{i=1}^{n} K^1(\delta x_{*i}, \delta x_{*i}) = \frac{1}{n} \sum_{i=1}^{n} \delta^3 \frac{\sigma^2}{3} x_{*i}^3 = \frac{\delta^3}{n} \sum_{i=1}^{n} K^1(x_{*i}, x_{*i}) \in \Theta(\delta^3).$$

Furthermore, we have

$$j_c(\delta x_*)^\top \Sigma j_c(\delta x_*) = \Big(\delta(\nabla_\theta w_c|_\mu)^\top x + \nabla_\theta b_c|_\mu\Big)^\top \Sigma \Big(\delta(\nabla_\theta w_c|_\mu)^\top x + \nabla_\theta b_c|_\mu\Big).$$

Thus, $j_c(\delta x_*)^\top \Sigma j_c(\delta x_*)$ is quadratic in δ. Therefore, $\mathrm{Var}(\widetilde{f^{(c)}}|_R(\delta x_*))$ is in $\Theta(\delta^3)$. □

Equipped with this result, we are now ready to state our main result. The following theorem shows that RGPR yields the ideal asymptotic uniform confidence of $1/k$ given any pre-trained ReLU classification BNN with an arbitrary number of classes.

Theorem 3.14 (Uniform Asymptotic Confidence). *Let $f : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^k$ be a k-class pre-trained ReLU network equipped with the posterior $\mathcal{N}(\theta \mid \mu, \Sigma)$ and let $\widetilde{f}$ be obtained from f via RGPR. Suppose that the linearization of f and the multiclass probit approximation* (1.42) *is used for approximating the predictive distribution $p(y_* \mid \delta x_*, \widetilde{f}, \mathcal{D})$ under $\widetilde{f}$. For any nonzero input $x_* \in \mathbb{R}^n$ and for every class $c = 1, \ldots, k$,*

$$\lim_{\delta \to \infty} p(y_* = c \mid \delta x_*, \widetilde{f}, \mathcal{D}) = \frac{1}{k}.$$

Proof. Let $x_* \neq 0 \in \mathbb{R}^n$ be arbitrary. By Lemma 3.1 and definition of ReLU network, there exists a linear region R and real number $\alpha > 0$ such that for any $\delta \geq \alpha$, the restriction of f to R can be written as

$$f|_R(\delta x) = W(\delta x) + b,$$

where the matrix $W \in \mathbb{R}^{k \times n}$ and vector $b \in \mathbb{R}^k$ are functions of the parameter θ, evaluated at μ. Furthermore, for $i = 1, \ldots, k$ we denote the i-th row and the i-th component of W and b as w_i and b_i, respectively. Under the linearization of f, the marginal distribution (3.32) over the output $\widetilde{f}(\delta x)$ holds. Hence, for each $c = 1, \ldots, k$, under the multiclass probit approximation, the predictive distribution restricted to R is given by

$$\begin{aligned}\widetilde{p}(y_* = c \mid \delta x_*, \mathcal{D}) &\approx \frac{\exp(m_c(\delta x_*)\, \kappa_c(\delta x_*))}{\sum_{i=1}^k \exp(m_i(\delta x_*)\, \kappa_i(\delta x_*))} \\ &= \frac{1}{1 + \sum_{i \neq c}^k \exp(\underbrace{m_i(\delta x_*)\, \kappa_i(\delta x_*) - m_c(\delta x_*)\, \kappa_c(\delta x_*)}_{=: z_{ic}(\delta x_*)})},\end{aligned}$$

where for all $i = 1, \ldots, k$,

$$m_i(\delta x_*) = f^{(i)}|_R(\delta x; \mu) = w_i^\top(\delta x) + b_i \in \mathbb{R}, \tag{3.33}$$

and

$$\kappa_i(\delta x) = (1 + \pi/8\, (v_{ii}(\delta x_*) + K(\delta x_*, \delta x_*)))^{-\frac{1}{2}} \in \mathbb{R}_{>0}. \tag{3.34}$$

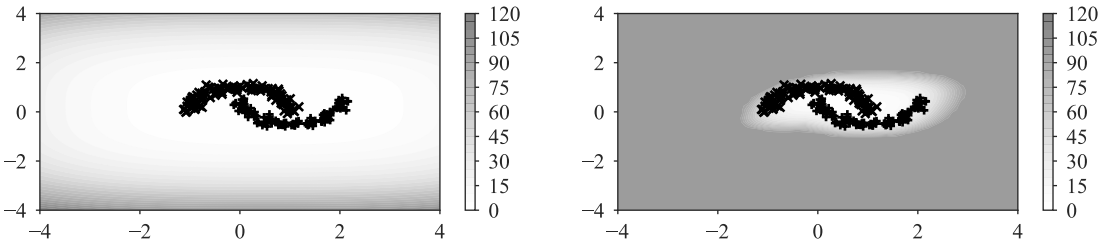
In particular, for all $i = 1, \ldots, k$, note that $m(\delta x_*)_i \in \Theta(\delta)$ and $\kappa(\delta x)_i \in \Theta(1/\delta^{\frac{3}{2}})$ since $v_{ii}(\delta x_*) + k(\delta x_*, \delta x_*)$ is in $\Theta(\delta^3)$ by Proposition 3.13. Now, notice that for any $c = 1, \ldots, k$ and any $i \in \{1, \ldots, k\} \setminus \{c\}$, we have

$$\begin{aligned}z_{ic}(\delta x_*) &= (m_i(\delta x_*)\, \kappa_i(\delta x_*)) - (m_c(\delta x_*)\, \kappa_c(\delta x_*)) \\ &= (\underbrace{\kappa_i(\delta x_*)\, w_i}_{\Theta\left(1/\delta^{\frac{3}{2}}\right)} - \underbrace{\kappa_c(\delta x_*)\, w_c}_{\Theta\left(1/\delta^{\frac{3}{2}}\right)})^\top (\delta x_*) + \underbrace{\kappa_i(\delta x_*)\, b_i}_{\Theta\left(1/\delta^{\frac{3}{2}}\right)} - \underbrace{\kappa_c(\delta x_*)\, b_c}_{\Theta\left(1/\delta^{\frac{3}{2}}\right)}.\end{aligned}$$

Thus, it is easy to see that $\lim_{\delta \to \infty} z_{ic}(\delta x_*) = 0$. Hence we have

$$\lim_{\delta \to \infty} \widetilde{p}(y_* = c \mid \delta x_*, \mathcal{D}) = \lim_{\delta \to \infty} \frac{1}{1 + \sum_{i \neq c}^k \exp(z_{ic}(\delta x_*))} = \frac{1}{1 + \sum_{i \neq c}^k \exp(0)} = \frac{1}{k},$$

as required. □



(a) Input only **(b)** Input and hiddens

Figure 3.8: The variance of $\widehat{f}$, under the assumption that RGPR is applied only on the input space **(a)** or on the input and hidden spaces of a ReLU network f **(b)**.

As a sketch of the proof for this theorem, consider the special case of binary classification. Here, we notice that the variance v in the probit approximation (3.19) is now a cubic function of δ under RGPR, due to Proposition 3.13. Thus, it is easy to see that the term inside of σ decays like $1/\sqrt{\delta}$ far away from the training data. Therefore, in this case, $p(y = 1 \mid \delta x_*, \mathcal{D})$ evaluates to $\sigma(0) = 1/2$ as $\delta \to \infty$, and hence we obtain the asymptotic maximum entropy prediction.

We remark that the pre-trained assumption on f in Proposition 3.13 and Theorem 3.14 can be removed. Intuitively, this is because under the scaling of δ on x_*, the last term of (3.26) is in $\Theta(\delta^2)$. Thus, it is asymptotically dominated by the $\Theta(\delta^3)$ growth induced by the DSCS kernel in the second term. We however present the statements as they are since they support the *post-hoc* spirit of RGPR.

3.3.3 Extending RGPR to Non-Asymptotic Regimes

While the previous construction is sufficient for modeling uncertainty far away from the data, it does not necessarily model the uncertainty *near* the data region well. Figure 3.8(a) shows this behavior: the variance of the GP prior equipped with the DSCS kernel grows slowly around the data and hence, even though Theorem 3.14 will still apply in the limit, RGPR has a minimal effect on the uncertainty of the BNN in non-asymptotic regimes.

A way to address this is to adapt RGPR's notion of proximity between input points. This can be done by using the higher-level data representations already available from the pre-trained NN—a test point close to the data in the input space can be far from them in the representation space, thereby the DSCS kernel might assign a large variance. Based on this intuition, we extend RGPR by additionally placing infinite ReLU features on the representation spaces of the point-estimated network f_μ induced by the BNN f, where μ is the mean of the Gaussian posterior of f, as follows.

For each $\ell = 1, \ldots, L-1$ and any input x_*, let n_ℓ be the size of the ℓ-th hidden layer of f_μ and $h_*^{(\ell)}$ be the ℓ-th hidden representation of x_*. By convention, we assume that $n_0 = n$ and $h_*^{(0)} = x_*$. Now, we place for each $\ell = 0, \ldots, L-1$ an infinite number of ReLU features on the representation space $\mathbb{R}^{n_\ell}$, and thus we obtain a random function $\widehat{f}^{(\ell)} : \mathbb{R}^{n_\ell} \to \mathbb{R}$ distributed as $\mathcal{GP}(0, K)$. Then, given that $\widehat{N} := \sum_{\ell=0}^{L-1} n_\ell$, we define $\widehat{f} : \mathbb{R}^{\widehat{n}} \to \mathbb{R}$ by $\widehat{f} := \widehat{f}^{(0)} + \cdots + \widehat{f}^{(L-1)}$, i.e. we assume that $\{\widehat{f}^{(\ell)}\}_{l=0}^{L-1}$ are independent. This function is therefore a function over *all* representation (including the input) spaces of f_μ, distributed as the additive Gaussian

Algorithm 1 MC-prediction for RGPR. Differences from the standard procedure are in **grey**.

Input:

Pre-trained L-layer, ReLU BNN classifier f with posterior $\mathcal{N}(\theta \mid \mu, \Sigma)$. Test point $x_* \in \mathbb{R}^n$. Centering and scaling function $\texttt{std}$. Hyperparameters $(\sigma_\ell^2)_{\ell=0}^{L-1}$. Number of MC samples s.

$(h_*^{(\ell)})_{l=1}^{L-1} = \texttt{forward}(f_\mu, x_*)$ ▷ Get representations of x_* via a forward pass on f_μ.

$v_s(x_*) = \sum_{l=0}^{L-1} k(\texttt{std}(h_*^{(\ell)}), \texttt{std}(h_*^{(\ell)}); \sigma_\ell^2)$ ▷ Compute the prior variance of $\widehat{f}_*$.

for $i = 1, \ldots, s$ **do**

  $\theta_i \sim \mathcal{N}(\theta \mid \mu, \Sigma)$

  $f_i(x_*) = f(x_*; \theta_i)$

  $\widehat{f}_i(x_*) \sim \mathcal{N}(0, v_i(x_*) I)$ ▷ Sample from the marginal of GP-DSCS.

  $\widetilde{f}_i(x_*) = f_i(x_*) + \widehat{f}_i(x_*)$ ▷ Compute RGPR's output $\widetilde{f}(x_*; \theta_i)$.

end for

return $s^{-1} \sum_{i=1}^{s} \text{softmax}(\widetilde{f}_i(x_*))$ ▷ Monte Carlo averaging under RGPR.

process $\mathcal{GP}(0, \sum_{\ell=0}^{L-1} K)$. In other words, given all representations $h_* := (h_*^{(\ell)})_{\ell=0}^{L-1}$ of x_* under f_μ, the marginal over the function output $\widehat{f}(h_*)$ is given by

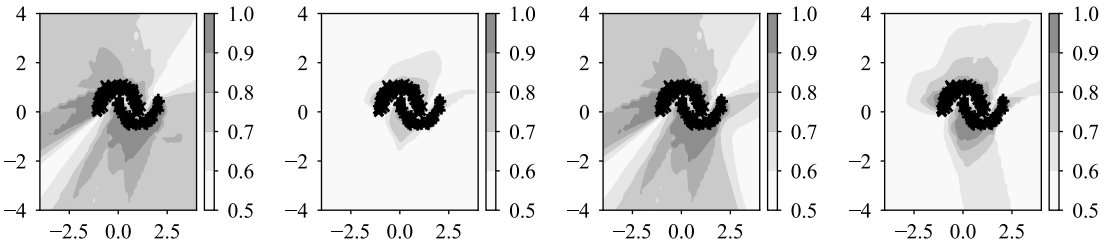
$$p(\widehat{f}_*) = \mathcal{N}\Big(0, \textstyle\sum_{\ell=0}^{L-1} K\Big(h_*^{(\ell)}, h_*^{(\ell)}; \sigma_\ell^2\Big)\Big). \tag{3.35}$$

We can then use this definition of $\widehat{f}$ as a drop-in replacement in (3.23) to define RGPR. Figure 3.8(b) visualizes the effect: the low-variance region modeled by $\widehat{f}$ becomes more compact around the data.

The analysis from the previous section still applies here since it is easy to see that the variance of $\widehat{f}_*$ in (3.35) is still cubic in δ. In practice, however, it is not necessarily true anymore that each $h_*^{(\ell)}$ is close to the origin in $\mathbb{R}^{n_\ell}$. To fix this, one can center and scale each $h_*^{(\ell)}$ via standardization using the mean $\mathbb{E}_{x \in \mathcal{D}}(h^{(\ell)}(x))$ and scaled standard deviation $r\sqrt{\text{Var}_{x \in \mathcal{D}}(h^{(\ell)}(x))}$ with $r > 1$, before evaluating the kernel in (3.35) (these quantities only need to be computed once). Note that by tuning the DSCS kernel's hyperparameter σ^2 such that confidence over the training data is preserved (cf. the next section), RGPR becomes insensitive to the choice of r since intuitively the tuning procedure will make sure that the DSCS kernel does not assign large variance to the training data. Therefore, in practice we set $r = 1$.

Algorithm 1 provides a pseudocode of RGPR for classification predictions via MC integration. The only overhead compared to the usual MC-integrated BNN prediction step are (marked in grey) (i) a single additional forward-pass over f_μ, (ii) L evaluations of the DSCS kernel K, and (iii) sampling from a k-dimensional diagonal Gaussian. The additional costs are negligible compared to the cost of obtaining the standard MC-prediction of f in the first place, which, in particular, requires multiple forward passes.

The kernel hyperparameters $(\sigma_\ell^2)_{\ell=0}^{L-1} =: \sigma^2$ control the variance growth of the DSCS kernel. Since RGPR is a GP model, one way to tune σ^2 is via marginal likelihood maximization. However, this leads to an expensive procedure even if a stochastic approximation (Hensman et al., 2015) is employed since the computation of the RGPR kernel (3.32) requires the network's Jacobian and the explicit kernel matrix need to be formed. Note however that those quantities are not needed for the computation of the predictive distribution via MC-integration (Algorithm 1).



(a) No RGPR **(b)** Marg. Likelihood **(c)** CV with $\mathcal{L}_{\text{LL}}$ **(d)** CV with $\mathcal{L}_{\text{OOD}}$

Figure 3.9: Different objectives for tuning σ^2. Shades are predictive confidence. $\mathcal{D}_{\text{out}}$ consists of uniform noise images.

Hence, a cheaper yet still valid option to tune σ^2 is to use a cross-validation (CV) which depends only on predictions over validation data $\mathcal{D}_{\text{val}}$ (Rasmussen & Williams, 2005, Ch. 5).

A straightforward way to perform CV is by maximizing the validation log-likelihood (LL). That is, we maximize the objective

$$\mathcal{L}_{\text{LL}}(\sigma^2) := \sum_{x_*, y_* \in \mathcal{D}_{\text{val}}} \log p(y_* \mid x_*, \mathcal{D}; \sigma^2).$$

However, this tends to yield overconfident results outside the training data (Fig. 3.9). Thus, we can optionally add an auxiliary term to $\mathcal{L}_{\text{LL}}$ that depends on some OOD dataset $\mathcal{D}_{\text{out}}$, resulting in

$$\mathcal{L}_{\text{OOD}}(\sigma^2) := \mathcal{L}_{\text{LL}}(\sigma^2) + \frac{\lambda}{k} \sum_{x_* \in \mathcal{D}_{\text{out}}} \sum_{i=1}^{k} \log p(y = i \mid x_*, \mathcal{D}; \sigma^2).$$

In particular, the additional term is simply the negative cross-entropy between the predictive distribution and the uniform probability vector of length k, with $\lambda = 0.5$ as proposed by Hendrycks et al. (2019). Note that both objectives can be optimized via gradient descent without the need of backprop through the network. See Fig. 3.9 for comparison between different objectives. In Section 3.3.5, we discuss the choice of $\mathcal{D}_{\text{out}}$.

3.3.4 Related Work

Mitigation of asymptotic overconfidence has been studied recently: Hein et al. (2019) noted, demonstrated, and analyzed this issue, but their proposed method does not work for large δ. Kristiadi et al. (2020) showed that a Bayesian treatment could mitigate this issue even as $\delta \to \infty$. However, the analysis is restricted to binary classification and the asymptotic confidence of standard ReLU BNNs only converges to a constant in $(0, 1)$. In a non-Bayesian framework, Meinke & Hein (2020) used density estimation to achieve the uniform confidence far away from the data. Nevertheless, this property has not been previously achieved in the context of BNNs.

Unlike a line of works that connects NNs and GPs (Cho & Saul, 2009; Lee et al., 2018a; Khan et al., 2019, etc.) which studies properties of NNs as GPs in an infinite-width limit, we focus on combining *finite-width* BNNs with a GP *a posteriori*. Though similar in spirit, our method thus differs from Wilson et al. (2020) which propose a combination of a weight-space prior and a function-space posterior for efficient GP posterior sampling. Our method is also distinct

from other methods that model the residual of a predictive model with a GP (Blight & Ott, 1975; Wahba, 1978; O'Hagan, 1978; Qiu et al., 2020, etc.) since RGPR models the *uncertainty residual* of BNNs, in contrast to the predictive residual of point-estimated networks, and RGPR does not require further posterior inference given a pre-trained BNN.

Cho & Saul (2009) proposed a family of kernels for deep learning, called the arc-cosine kernels. The first-order arc-cosine kernel can be interpreted as a ReLU kernel but it only has a quadratic variance growth and thus is not suitable to guarantee the uniform asymptotic confidence. While higher-order arc-cosine kernels have super-quadratic variance growth, they ultimately cannot be interpreted as ReLU kernels, and hence are not as natural as the cubic-spline kernel in the context of ReLU BNNs.

3.3.5 Empirical evaluations

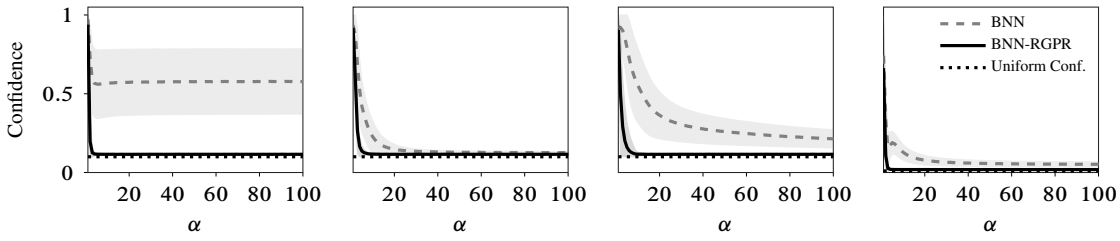
We empirically validate Theorem 3.14 in the asymptotic regime and the effect of RGPR on non-asymptotic confidence estimates in multiclass image classification. The LeNet architecture (LeCun et al., 1998) is used for MNIST, while ResNet-18 (He et al., 2016) is used for CIFAR-10, SVHN, and CIFAR-100—details in Appendix D.2. For each dataset, we tune σ^2 via a validation set of size 2000 obtained by splitting the corresponding test set. Following Hein et al. (2019), $\mathcal{D}_{\text{out}}$ consists of smoothed noise images, which are obtained via random permutation, blurring, and contrast rescaling of the original dataset—they do not preserve the structure of the original images and thus can be considered as synthetic noise images. Particularly for ResNet, we use the outputs of its residual blocks to obtain input representations h_*.

3.3.6 Asymptotic Regime

In this experiment, we use the last-layer Laplace approximation (LLL) as the base BNN. Results with other, more sophisticated BNNs (Ritter et al., 2018a; Maddox et al., 2019a; Wilson et al., 2016b) are in Appendix D.2—we observe similar results there. Figure 3.10 shows confidence estimates of both the BNN and the RGPR-imbued BNN over 1000 samples obtained from each of MNIST, CIFAR-10, SVHN, and CIFAR-100 test sets, as the scaling factor δ increases. As expected, the vanilla BNN does not achieve the ideal uniform confidence prediction, even for large δ. This issue is most pronounced on MNIST, where the confidence estimates are far away from the ideal confidence of 0.1. Overall, this observation validates the hypothesis that BNNs have residual uncertainty, leading to asymptotic overconfidence that can be severe. We confirm that RGPR fixes this issue. Moreover, its convergence appears at a finite, small δ; without a pronounced effect on the original confidence.

3.3.7 Non-Asymptotic Regime

We report results on standard dataset shift and out-of-distribution (OOD) detection tasks. For the former, we use the standard rotated-MNIST and CIFAR-10-C datasets (Ovadia et al., 2019; Hendrycks & Dietterich, 2019) and measure the performance using the following metrics: negative log-likelihood (NLL), the Brier score, expected calibration error (ECE), accuracy, and average confidence. Meanwhile, for OOD detection, we use five OOD sets for each in-distribution dataset. The FPR@95 metric measures the false positive rate of an OOD detector at a 95% true positive rate. We use LLL as the base BNN for RGPR and compare it against the MAP-trained network, temperature scaling (TS, Guo et al., 2017), the method of Qiu et al. (2020) with the



(a) MNIST **(b)** CIFAR-10 **(c)** SVHN **(d)** CIFAR-100

Figure 3.10: Confidence of a vanilla BNN (LLL) and the same BNN with RGPR, as a function of δ. Test data are constructed by scaling the original test set. Curves are means, shades are ± 1 std. devs. Note that in **(b)** and **(d)**, even though close, the BNN does not achieve the uniform confidence.

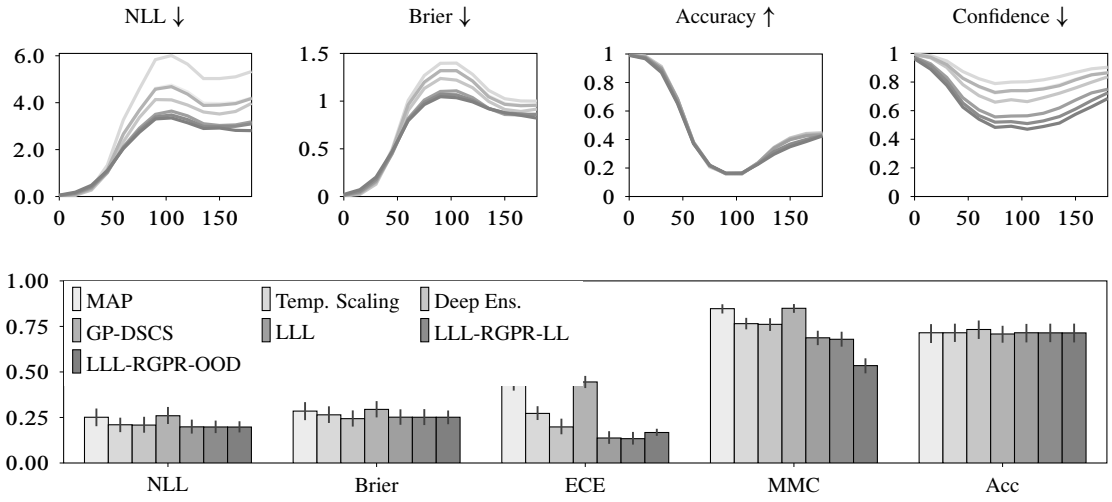


Figure 3.11: (Top) Rotated-MNIST x-axes are rotation angles. **(Bottom)** Corrupted-CIFAR-10—values are normalized to [0, 1] and are averages over all types of corruption and all severity levels.

DSCS kernel (GP-DSCS), and Deep Ensemble (DE, Lakshminarayanan et al., 2017), which is a strong baseline in this regime (Ovadia et al., 2019). We denote the RGPR tuned via $\mathcal{L}_{\text{LL}}$ and $\mathcal{L}_{\text{OOD}}$ with the suffixes "-LL" and "-OOD", respectively. More results are in Appendix D.2.

On the rotated-MNIST benchmark, we observe in Fig. 3.11 that RGPR consistently improves the base LLL, especially when tuned with $\mathcal{L}_{\text{OOD}}$, while still preserving the calibration of LLL on the clean data. LLL-RGPR attains better results than GP-DSCS, which confirms that applying a GP on top of a trained BNN is more effective than on top of MAP-trained nets. Some improvements, albeit less pronounced are also observed in CIFAR-10-C. For OOD detection (Table 3.5) we find that LLL is already competitive with all baselines, but RGPR can still improve it further, making it better than Deep Ensemble. Further results comparing RGPR to recent non-Bayesian baselines (Lee et al., 2018c; Van Amersfoort et al., 2020) are in Appendix D.2.

Finally, we discuss the limitation of $\mathcal{L}_{\text{OOD}}$. While the use of additional OOD data in tuning σ^2 improves both dataset-shift and OOD detection results, it is not without a drawback: $\mathcal{L}_{\text{OOD}}$ induces slightly worse calibration in terms of ECE (Table D.5 in Appendix D.2). This implies that one can somewhat trade the exactness of RGPR (as assumed by Proposition 3.12) off with

Table 3.5: OOD data detection in terms of FPR@95. All values are in percent and averages over five OOD test sets and over 5 prediction runs.

Methods	**MNIST**	**CIFAR-10**	**SVHN**	**CIFAR-100**
MAP	28.2	38.9	17.8	72.2
TS	28.4	34.9	17.6	71.9
DE	23.0	51.0	11.3	74.7
GP-DSCS	27.8	46.7	19.1	69.1
LLL	24.8	29.8	15.7	69.5
LLL-RGPR-LL	3.9	29.6	13.8	65.8
LLL-RGPR-OOD	**3.6**	**24.2**	**9.6**	**63.0**

better OOD detection. This trade-off is expected to a degree since OOD data are often close to the training data. Hence, the single multiplicative hyperparameter σ_l^2 of each the DSCS kernel in (3.35) cannot simultaneously induce high variance on outliers and low variance on the nearby training data. Table D.9 (Appendix D.2) corroborates this: When a $\mathcal{D}_{\text{out}}$ "closer" to the training data (the 32×32 ImageNet dataset (Chrabaszcz et al., 2017)) is used, the ECE values induced by $\mathcal{L}_{\text{OOD}}$ become worse (but the OOD performance improves further). Note that this negative correlation between ECE and OOD detection performance also presents in state-of-the-art OOD detectors (Section D.2.2.5). So, if the in-distribution calibration performance is more crucial in applications of interest, $\mathcal{L}_{\text{LL}}$ is a better choice for tuning σ^2 since it still gives benefits on non-asymptotic outliers, but preserves calibration better than $\mathcal{L}_{\text{OOD}}$.

Chapter 4
Improving Non-Asymptotic Confidence Estimates

The contents of this chapter are primarily based on:

> Agustinus Kristiadi, Runa Eschenhagen, and Philipp Hennig. "Posterior Refinement Improves Sample Efficiency in Bayesian Neural Networks." Advances in Neural Information Processing Systems (NeurIPS). 2022.

	Idea	**Analysis**	**Experiment**	**Code**	**Writing**
Agustinus Kristiadi	60%	80%	50%	50%	90%
Runa Eschenhagen	20%	20%	50%	50%	5%
Philipp Hennig	20%	0%	0%	0%	5%

And:

> Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. "Learnable uncertainty under Laplace approximations." Uncertainty in Artificial Intelligence (UAI). 2021.

	Idea	**Analysis**	**Experiment**	**Code**	**Writing**
Agustinus Kristiadi	30%	80%	100%	100%	85%
Matthias Hein	0%	0%	0%	0%	5%
Philipp Hennig	70%	20%	0%	0%	10%

And:

> Kristiadi, Agustinus, Matthias Hein, and Philipp Hennig. "Being a Bit Frequentist Improves Bayesian Neural Networks." International Conference on Artificial Intelligence and Statistics (AISTATS). 2022.

	Idea	**Analysis**	**Experiment**	**Code**	**Writing**
Agustinus Kristiadi	95%	95%	100%	100%	90%
Matthias Hein	5%	0%	0%	0%	5%
Philipp Hennig	0%	5%	0%	0%	5%

We have seen in Chapter 3 that Bayesian neural networks are useful for calibrating the uncertainty of test points that lie far away from the training data. However, as shown by Guo et al. (2017) and Ovadia et al. (2019), standard Bayesian neural networks are also not generally calibrated *near* the training data. In this chapter, we, therefore, propose several methods that can be used to improve the non-asymptotic uncertainty calibration of Bayesian neural networks.

In Section 4.1 we study the *de facto* method for approximating the predictive distributions of Bayesian neural networks—few-sample Monte Carlo integration. While theoretically, it requires many samples to be accurate, Monte Carlo integration can be used with few samples provided that the weight-space approximate posterior is accurate. However, accurately approximating a neural network's posterior is very challenging and expensive. We thus propose a cheap, *post-hoc* method for refining a parametric approximate posterior, which calibration performance can match that of the gold-standard full-batch Hamiltonian Monte Carlo.

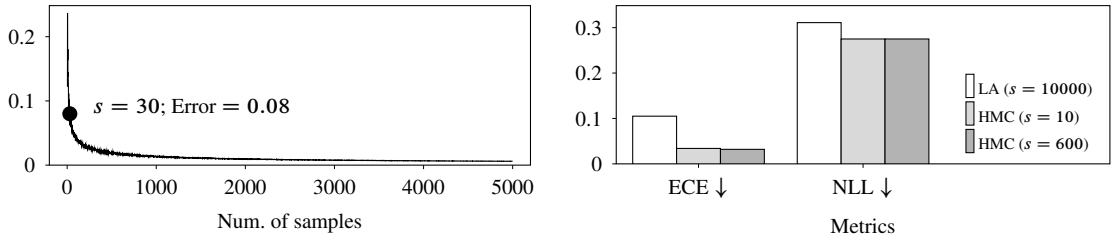
Meanwhile, in Section 4.2 we propose a *post-hoc* way to "train" the predictive uncertainty of a Laplace approximation, without affecting its accuracy. The crux of the method is to essentially embed the parameter space of a neural network into a higher dimensional space in a particular way. Then, due to the added degrees of freedom, in this "augmented space", one can find a point that yields the same predictions as the original parameter but with a different local curvature. Moreover, the Hessian matrix at that point depends on the free parameters introduced by the augmentation. Thus, the Hessian matrix and therefore the uncertainty under a Laplace approximation can be tuned to make the resulting Bayesian neural network better calibrated.

Finally, in Section 4.3, we show that a key to improving the calibration of a Bayesian neural network is to follow the standard method used in non-Bayesian uncertainty quantification literature: the out-of-distribution (OOD) training (Hendrycks et al., 2019; Hein et al., 2019, etc). We show several ways of incorporating OOD training data in Bayesian neural networks and show that the simplest, most philosophically clean method, which simply adds an additional class to the original network, is preferable.

4.1 Refining the Approximate Posteriors of Neural Networks

A prediction in a BNN amounts to an integration of the likelihood w.r.t. the (approximate) posterior measure. Due to the non-linearity of NNs, no analytic solution to the integral exists, even when the likelihood and the approximate posterior are both Gaussian. A low-cost, unbiased, stochastic approximation can be obtained via Monte Carlo (MC) integration: obtain s samples from the approximate posterior and then compute the empirical expectation of the likelihood w.r.t. these samples. While MC integration is accurate for large s, because of the sheer size of modern (B)NNs, virtually all BNNs use small s (typically 10 to 30, see (Blundell et al., 2015; Louizos & Welling, 2016; 2017; Ritter et al., 2018a; Osawa et al., 2019; Maddox et al., 2019a; Dusenberry et al., 2020, etc.)). Due to its well-known error scaling of $\Theta(1/\sqrt{s})$, intuitively, MC integration with a small s is inadequate for an accurate prediction—yet, this has not been studied in depth for BNNs. Furthermore, while linearization of an NN around a point estimate in the parameter space is an alternative to MC integration (Immer et al., 2021b; MacKay, 1992b; Foong et al., 2019), it is generally costly due to the computation of *per-example* Jacobian matrices.

In this section, we study the quality of MC integration for making predictions in BNNs. We show that few-sample MC integration is inaccurate for even an "easy" integral such as when the domain of integration is the output space of a binary-classification BNNs, cf. Fig. 4.1 (left). Further, we show that its alternative, the network linearization, disagrees with large-sample MC in-



(a) The Monte Carlo standard error to the integral $\int \text{softmax}(x)\,\mathcal{N}(x \mid 5, 10^2)\,dx \in [0, 1]$

(b) Calibration on the Fashion-MNIST dataset

Figure 4.1: Left: Few-sample MC integration is inaccurate for computing classification predictive distribution: the standard error of the MC integration of the logistic-Gaussian integral is large for the commonly-used (few) number of samples. **Right:** But one can still obtain good predictive performance with a small s if a high-quality posterior approximation, e.g. HMC, is used.

tegration, making it unsuitable as a general-purpose predictive approximation method, i.e. when the Gauss-Newton matrix is *not* employed.[1] Meanwhile, as indicated by full-batch Markov Chain Monte Carlo methods (Neal et al., 2011; Hoffman et al., 2014) or even (the Bayesian interpretation of) ensemble methods (Lakshminarayanan et al., 2017), few-sample MC integration might still be useful for making predictions, provided that the approximate posterior measure is "close enough" to the true posterior—cf. Fig. 4.1 (right). This implies that one should focus on improving the accuracy of posterior approximations.

Nevertheless, prior methods for obtaining expressive posteriors (e.g., Louizos & Welling, 2017; Zhang et al., 2020) require either significant modification to the NN or storing many copies of the parameters. Moreover, they require training from scratch and thus introduce a significant overhead, which can be undesirable in practical applications. Therefore, we propose a *post-hoc* method for "refining" a Gaussian approximate posterior by leveraging normalizing flows (Rezende & Mohamed, 2015). Contrary to the existing normalizing flow methods with *a priori* base distribution (e.g. $\mathcal{N}(0, I)$), the proposed refinement method converges faster with shorter flows, making it cheaper than the naïve application of normalizing flows in BNNs. When used in conjunction with last-layer BNNs, which have been shown to be competitive to their all-layer counterparts (Daxberger et al., 2021a), the proposed method is simple, cheap, yet competitive to even the gold-standard Hamiltonian Monte Carlo in terms of predictive performance.

4.1.1 Pitfalls of BNNs' Approximate Predictive Distributions

We begin our analysis with some observations in obtaining high-performing approximate predictive distributions in BNNs. First, we observe that while accurate MC integration requires many samples (Fig. 4.1(a)), one can "get away" with much fewer samples when the posterior approximation accurately reflects the true posterior, as in the case of Hamiltonian Monte Carlo (HMC). However, such accurate posterior approximations do not come cheaply: HMC and other MCMC algorithms are generally prohobitively expensive for large neural networks (Izmailov et al., 2021b). Alternatives to MC integration for obtaining predictive distributions are not generally applicable either and come with some biases, making the resulting approximation less accurate. Moreover, linearization is also expensive during test time since per-example Jacobian

[1] The generalized Gauss-Newton matrix is the *exact* Hessian matrix of a linearized network.

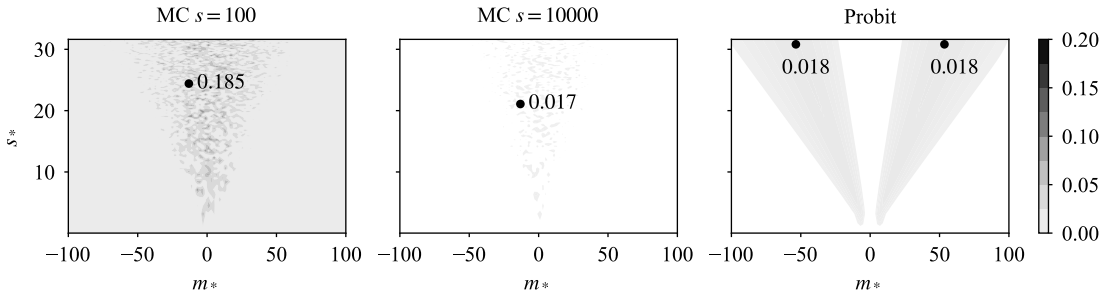


Figure 4.2: The (absolute) errors of the MC integration and the probit approximation for computing $I(m_*, s_*)$ across different values of m_* and s_*. A trapezoid method with 20000 evaluation points is used as a gold-standard baseline. Dots indicate the maximum errors. White indicates (near) zero error.

matrices need to be computed. All those observations motivate us to obtain a cheap—preferrably *post-hoc* like the Laplace approximation—yet accurate like HMC, which we will discuss in the next section.

4.1.1.1 Accurate MC integration requires many samples

We begin our analysis with the simplest yet practically relevant case. Let $x_* \in \mathbb{R}^n$ and $f_* := f(x_*) \in \mathbb{R}$ be the output of a binary classifier f, with $p(f_* \mid x_*, \mathcal{D}) = \mathcal{N}(f_* \mid m_*, s_*^2)$ for some $m_* \in \mathbb{R}$ and $s_*^2 \in \mathbb{R}_{>0}$. We are interested in approximating the integral (cf. Fig. 1.6)

$$I(m_*, s_*) := p(y \mid x_*, \mathcal{D}) = \int_{\mathbb{R}} \sigma(f_*)\, \mathcal{N}(f_* \mid m_*, s_*^2)\, df_*.$$

Note that this integral is prevalent in practice, e.g. in linearized or last-layer classification BNNs (Daxberger et al., 2021b; Immer et al., 2021b; Eschenhagen et al., 2021). We compare the MC integration

$$I(m_*, s_*) \approx \frac{1}{s} \sum_{i=1}^{s} \sigma(f_{*s}) \qquad \text{where} \quad f_{*s} \sim \mathcal{N}(f_* \mid m_*, s_*^2), \tag{4.1}$$

with $s = 100$ against the probit approximation, using a trapezoid quadrature with 20000 evaluation points to represent a gold-standard baseline. That is, we compute the discrepancy $|\widetilde{I}(m_*, s_*) - I_*(m_*, s_*)|$ where $\widetilde{I}(m_*, s_*)$ is $I(m_*, s_*)$ computed either with MC integration or the probit approximation, and $I_*(m_*, s_*)$ obtained via the trapezoid method.

The results are in Fig. 4.2: Even with $s = 100$ samples—larger than the usual $s =$ 10-30—MC integration is inaccurate. Its error can be as high as 0.18, which is substantial considering $I(m_*, s_*) \in [0, 1]$. As s increases beyond 10000, MC integration improves and eventually overtakes the probit approximation, as to be expected given its theoretical guarantees. This highlights the flaw of few-sample MC integration—accurate MC integration generally requires large number of samples.

4.1.1.2 Many-sample MC integration is not sufficient

However, even with a large s, MC integration can still fail to yield good predictive performance in BNNs. This can happen when the approximation $q(\theta)$ used in MC integration is an inaccurate approximation of $p(\theta \mid \mathcal{D})$—virtually the case for every parametric BNN. Thankfully, there is some evidence that the error of MC integration might be relatively small in comparison to the error generated by crude posterior approximations. As an extreme example, Deep Ensemble (Lakshminarayanan et al., 2017) and its variants, even though they perform MC integration with a small number of samples (usually $s = 5$), generally yield better approximations to the predictive distributions. This is perhaps due to their multimodality, i.e. due to their finer-grained posterior approximations.

Table 4.1: The expected calibration error (ECE) and negative log-likelihood (NLL) of a LA ($s = 10000$) and HMC ($s = 10$).

Methods	ECE ↓	NLL ↓
F-MNIST		
LA	10.5±0.4	0.311±0.005
HMC	**3.4**±0.2	**0.275**±0.004
CIFAR-10		
LA	4.9±0.2	0.161±0.001
HMC	**4.2**±0.2	**0.158**±0.001

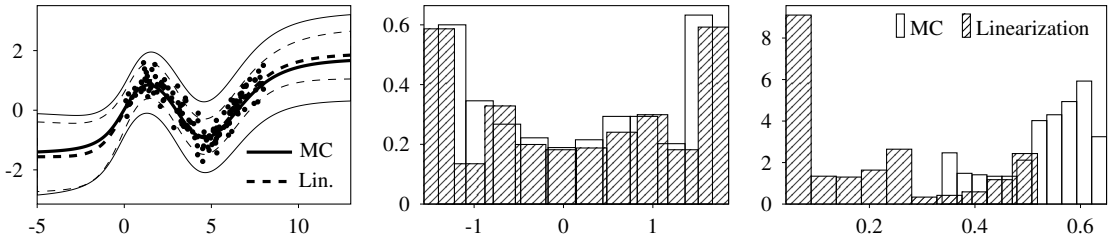
To show this more concretely, consider the following experiment. We take Fashion-MNIST (F-MNIST) pre-trained LeNet and CIFAR-10 pre-trained WideResNet-16-4. For each case, we perform a last-layer Laplace approximation with the exact Hessian and a full-batch NUTS-HMC (Hoffman et al., 2014), under the same prior and likelihood. We find in Table 4.1 that HMC, even with few samples, yields better-calibrated predictive distribution than the LA with three orders of magnitude more samples. This finding validates the widely-believed wisdoms (Louizos & Welling, 2016; 2017; Dusenberry et al., 2020; Eschenhagen et al., 2021, etc.) that highly accurate posterior approximations are most important for BNNs. Furthermore, this also shows that with a fine-grained posterior approximation, the predictive performance of a BNN is less sensitive to the number of MC samples, leading to better test-time efficiency.

4.1.1.3 Analytic alternatives to MC integration are not the definitive answer

Of course, fine-grained posterior approximations are expensive. So, a natural question is whether one can keep a cheap but crude posterior approximation by replacing MC integration. Network linearization seems to be a prime candidate to replace MC integration in the case of Gaussian approximations (Immer et al., 2021b; MacKay, 1992b). But it poses several problems: First, it requires relatively expensive computation of the per-example Jacobian, where *for each* test point x_*, one must store the associated $d \times k$ matrix, where d could easily be tens of millions (e.g. in ResNets) and k could be in the order of thousands (e.g. ImageNet). Second, while Immer et al. (2021b) argued that network linearization is the correct way to make predictions in Gauss-Newton-based Gaussian approximations, it is not generally applicable. We show this in Fig. 4.3: Everything else being equal, MC integration ($s = 10000$) and linearization yield different results especially in terms of predictive uncertainty. Since one should prefer MC integration in this many-sample regime, linearization is thus not accurate for the general cases.

Table 4.2: Calibration of MC integration $s = 10000$ and the multi-class probit approximation.

Methods	ECE ↓	NLL ↓
F-MNIST		
MPA	**3.3**±0.2	**0.281**±0.002
MC	10.5±0.4	0.311±0.005
CIFAR-10		
MPA	**3.8**±0.1	**0.161**±0.001
MC	4.9±0.2	**0.161**±0.001



(a) Predictive distributions. **(b)** Densities of pred. means. **(c)** Densities of pred. std. dev.

Figure 4.3: Predictive distributions (means and 95% confidence intervals), computed via a MC integration ($s = 10000$) and network linearization, of a BNN with a weight-space Gaussian approximate posterior (a LA with the exact Hessian on a two-layer NN under a toy regression dataset).

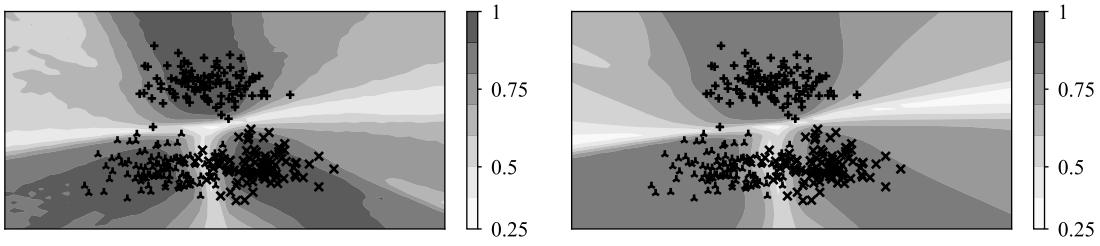
Moreover, we show that the multiclass probit approximation (MPA), which is often used on top of a linearized output distribution $p(f(x_*) \mid x_*, \mathcal{D})$ to obtain the predictive distribution $p(y_* \mid x_*, \mathcal{D})$ (Daxberger et al., 2021a; Eschenhagen et al., 2021), can also obscure the predictive performance, albeit often for the better (Daxberger et al., 2021a; Eschenhagen et al., 2021). To that end, we use the same experiment setting as Section 4.1.1.2 and Table 4.1, and compare the MPA against MC integration. The results are in Table 4.2.

Indeed, we see that the MPA yields better-calibrated predictive distributions. However, this results should be taken with a grain of salt: The MPA ignores the off-diagonal elements of the covariance of $p(f(x_*) \mid x_*, \mathcal{D})$, as shown in (1.42). That is, the MPA "biases" the predictive distribution $p(y \mid x_*, \mathcal{D})$ since it generally assumes that f_* has lower uncertainty than it actually has. And since $p(f_* \mid x_*, \mathcal{D})$ is usually induced by the LA or VB which are often underconfident on large networks (Immer et al., 2021b; Sun et al., 2019; Lotfi et al., 2022), the bias of MPA towards overconfidence thus counterbalances the underconfidence of $p(f_* \mid x_*, \mathcal{D})$. Therefore, in this case, the MPA can yield better-calibrated predictive distributions than MC integration (Eschenhagen et al., 2021; Daxberger et al., 2021a). Nevertheless, it can also fail even in simple cases such as in Fig. 4.4,[2] where the MPA yields underconfident predictions even near the training data. Moreover, the structured nature of the error of the MPA (cf. Fig. 4.2 for the binary case) might also contribute to the cases where the MPA differs from MC integration. Both examples above thus highlight the need of careful consideration when analytic alternatives to MC integration is employed.

4.1.2 Refining Gaussian Approximate Posteriors

The previous analysis indicates that accurate approximate posteriors $q(\theta)$ are most important for BNNs' predictive distributions. While one can obtain them via MCMC methods such as HMC (Neal et al., 2011; Hoffman et al., 2014), in practice, those methods are *very* expensive since they require a full-batch of data in their updates (Izmailov et al., 2021b). While mini-batch versions of MCMC methods exist, they do not seem to yield as good of results as their full-batch counterparts—indeed, Daxberger et al. (2021a) even showed that a well-tuned *last-layer* LA can outperform a state-of-the-art *all-layer* stochastic-gradient MCMC method (Zhang et al., 2020). Furthermore, these sample-based methods—both the full- and mini-batch versions, along with deep ensembles and their variants—are costly in terms of storage since one effectively must

[2] An all-layer full-Hessian LA on a three-layer `tanh` network is used.



(a) MC **(b)** MPA

Figure 4.4: Confidence estimate of MC integration and the MPA. The MPA is less confident compared to MC near the data.

store s copies of the network. We, therefore, propose a simple *post-hoc* technique for "refining" Gaussian approximations using normalizing flows. The resulting method is thus parametric yet can produce high-quality samples.

Let f_θ be a NN equipped with a Gaussian approximate posterior $q(\theta) = \mathcal{N}(\theta \mid \mu, \Sigma)$ (e.g., via a LA, VB, or SWAG (Maddox et al., 2019a)) on the parameter space under a dataset $\mathcal{D}$. Given a NF F_ϕ of length ℓ with parameter ϕ, we obtain the *refined posterior* by

$$\widetilde{q}_\phi(\widetilde{\theta}) = q(F_\phi^{-1}(\widetilde{\theta})) \left|\det J_{F_\phi}(\widetilde{\theta})\right|^{-1}. \tag{4.2}$$

Then, a *refinement* of $q(\theta)$ amounts to minimizing the reverse KL-divergence to the true posterior, using the evidence lower bound (ELBO) as a proxy ($\mathbb{H}$ below is the entropy functional):

$$\phi^* = \operatorname*{argmax}_\phi \; \mathbb{E}_{\widetilde{\theta}\sim\widetilde{q}_\phi}\Big[\log p(\mathcal{D} \mid f_{\widetilde{\theta}}) + \log p(\widetilde{\theta})\Big] + \mathbb{H}\big[\widetilde{q}_\phi\big], \tag{4.3}$$

Given a refined posterior $q_{\phi^*}(\widetilde{\theta})$ and a test point x^*, we can obtain the predictive distribution via MC integration:

$$p(y^* \mid x^*, \mathcal{D}) \approx \frac{1}{s}\sum_{i=1}^{s} p\Big(y^* \mid f_{\widetilde{\theta}_i}(x^*)\Big); \;\; \text{where } \widetilde{\theta}_i = F_{\phi^*}(\theta_i); \;\; \theta_i \sim q(\theta) \quad \forall i = 1, \ldots, s.$$

Due to the expressiveness of NFs (Papamakarios et al., 2021), we can expect based on the previous analysis that a large s is not necessary here to obtain good predictive performance. We shall validate this in Section 4.1.4. Last but not least, this refinement technique is especially useful for last-layer BNNs since their parameter spaces typically have manageable dimensions—e.g., WideResNets' last-layer features' dimensionality typically range from 256 to 2048 (Zagoruyko & Komodakis, 2016).

4.1.3 Related Work

Normalizing flows have previously been used for approximate Bayesian inference in BNNs. An obvious way to do so, based on the flexibility of NFs in approximating any density (Papamakarios et al., 2021), would be to apply a NF on top of the standard normal distribution $\mathcal{N}(\theta \mid 0, I)$ to approximate $p(\theta \mid \mathcal{D})$, see e.g. Izmailov et al. (2019) and the default implementation of

variational approximation with NF in Pyro (Bingham et al., 2019). We show in Section 4.1.4.2 that the subtle difference that we make—using an approximate posterior instead of an *a priori* distribution—is more cost-effective. In a more sophisticated model, Louizos & Welling (2017) combine VB with NF by assuming a compound distribution on each NN's weight matrix and use the NF to obtain an expressive mixing distribution. However, their method requires training both the BNN and the NF jointly from scratch. In an adjacent field, Maroñas et al. (2021) use NFs to transform Gaussian process priors.

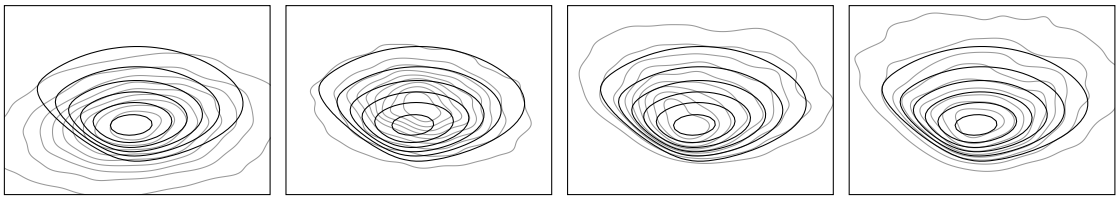
Posterior refinement in approximate Bayesian inference has recently been studied. Immer et al. (2021b) proposes to refine the LA using a Gaussian-based VB and Gaussian processes. However, this implies that they still assume a Gaussian posterior. To obtain a non-Gaussian posterior, Miller et al. (2017) form a mixture-of-Gaussians approximation by iteratively adding component distributions. But, at *every* iteration, their methods require a full ELBO optimization, making it costly for BNNs. A lower-cost LA-based alternative to their work has also been proposed by Eschenhagen et al. (2021). These methods have high storage costs since they must store many high-dimensional Gaussians. By contrast, our method only does an ELBO optimization once and only requires storage of the base Gaussian and the parameters of a NF. In a similar vein, Havasi et al. (2021) perform an iterative ELBO optimization for refining a sample of a variational approximation. But, this inner-loop optimization needs to be conducted each time a sample is drawn, e.g., during MC integration at test time. Our method, on the other hand, only requires sampling from a Gaussian and evaluations of a NF, cf. (4.1.2).

4.1.4 Experiments

Datasets We validate our method using standard classification datasets: Fashion-MNIST (F-MNIST), CIFAR-10, and CIFAR-100. For the out-of-distribution (OOD) detection task, we use three standard OOD test sets for each in-distribution dataset, see Appendix E.0.2 for the full list. Finally, for the toy logistic regression experiment, a dataset of size 50 is generated by sampling from a bivariate, bimodal Gaussian.

Network architectures For the F-MNIST experiments, we use the LeNet-5 architecture (LeCun et al., 1998). Meanwhile, for the CIFAR experiments, we use the WideResNet architecture with a depth of 16 and widen factor of 4 (WRN-16-4, Zagoruyko & Komodakis, 2016). For the NF, we use the radial flow (Rezende & Mohamed, 2015) which is among the simplest and cheapest non-trivial NF architectures.

Baselines We focus on last-layer Bayesian methods to validate the refinement technique. We use the LA to obtain the base distribution for our refinement method—following recent practice (Daxberger et al., 2021a), we tune the prior precision via *post-hoc* marginal likelihood maximization. The No-U-Turn-Sampler Hamiltonian Monte Carlo (HMC, Neal et al., 2011; Hoffman et al., 2014) with 600 samples is used as a gold-standard baseline—all HMC results presented in this paper are well-converged in term of the Gelman-Rubin diagnostic (Gelman & Rubin, 1992), see Appendix E. Furthermore, we compare our method against recent, all-layer BNN baselines: variational Bayes with the Flipout estimator (VB, Wen et al., 2018) and cyclical stochastic-gradient HMC (CSGHMC, Zhang et al., 2020). For all methods, we use MC integration with 20 samples to obtain the predictive distribution, except for HMC and CSGHMC where we use $s = 600$ and $s = 12$, respectively. Finally, we use prior precisions of 510 and 40 for the



(a) LA (0.040) **(b)** VB (0.016) **(c)** LA+Refine (0.002) **(d)** HMC

Figure 4.5: Comparison of approximate posterior densities, visualized via a kernel density estimation, on a 2D logistic regression problem. **Black contour:** The exact posterior contour, up to a normalizing constant. **Grey contour:** The kernel density estimate obtained from the posterior samples of each method. **Number:** The MMD distance to HMC's samples; lower is better.

Table 4.3: In-distribution calibration performance. The proposed method can make the LA closer to HMC, both in terms of the weight-space approximation and the calibration performance.

	F-MNIST		CIFAR-10		CIFAR-100	
Methods	**MMD** ↓	**NLL** ↓	**MMD** ↓	**NLL** ↓	**MMD** ↓	**NLL** ↓
MAP	1.093±0.003	0.3116±0.0049	0.438±0.001	0.1698±0.0009	0.416±0.001	0.9365±0.0063
LA	0.418±0.002	0.3076±0.0046	0.299±0.001	0.1672±0.0009	0.063±0.000	0.9865±0.0057
LA-Refine-1	0.356±0.004	0.2752±0.0031	0.346±0.000	0.1616±0.0007	0.063±0.000	0.9548±0.0062
LA-Refine-5	0.022±0.002	0.2699±0.0028	0.290±0.000	0.1582±0.0007	0.018±0.000	0.9073±0.0062
LA-Refine-10	0.013±0.002	0.2701±0.0028	0.130±0.001	0.1577±0.0008	0.019±0.000	0.9037±0.0058
LA-Refine-30	0.012±0.002	0.2701±0.0028	0.002±0.000	0.1581±0.0008	0.020±0.000	0.9035±0.0055
HMC	-	0.2699±0.0028	-	0.1581±0.0008	-	0.8849±0.0047

last-layer F-MNIST and CIFAR experiments, respectively. These prior precisions are obtained via grid search on the respective HMC baseline, maximizing validation log-likelihood. More implementation details are in Appendix E.

Metrics We use standard metrics to measure both calibration and OOD-detection performance. For the former, we use the negative log-likelihood (NLL) and the expected calibration error (ECE, Naeini et al., 2015). For the latter, we employ the false-positive rate at 95% true-positive rate (FPR95). Finally, to measure the closeness between an approximate posterior to the true posterior, we use the maximum-mean discrepancy (MMD, Gretton et al., 2012) distance between the said approximation's samples to the HMC samples, i.e. we use HMC as a proxy to the true posterior.

4.1.4.1 Toy example

We visualize different approximations to the posterior of the toy logistic regression problem in Fig. 4.5. Note that the true posterior density is non-symmetric and non-Gaussian. The LA matches the weight-space posterior mean, but inaccurate the further away from it. It even assigns probability mass on what are supposed to be low-density regions. While VB yields a more accurate result than the LA, it still assigns some probability mass on low-density regions due to the symmetry of the Gaussian approximation. Furthermore, it is unable to match the posterior's

mode well. The proposed refinement method, on the other hand, is able to make the LA more accurate—it yields a skewed, non-Gaussian approximation, similar to HMC.

We further quantify the previous observation using the MMD distance between each approximation's samples and HMC's samples. The LA, as expected, obtain the worst weight-space MMD. While VB is better than the LA with an MMD, the refined LA achieve even better MMD. This quantifies the previous visual observation.

4.1.4.2 Image classification

We present the calibration results in Table 4.3 using the LA and HMC as baselines, which represents a two "extremes" in the continuum of posterior approximations. As has previously shown by e.g. Guo et al. (2017), the vanilla MAP approximation yields uncalibrated, low-quality predictive distributions in terms of NLL and ECE. While the LA is a cheap way to improve MAP predictive, its predictive performance is still lagging behind HMC. By refining it with a NF, the LA becomes even better and closer to the HMC predictive. We also note that one does not need a complicated nor long (thus expensive) NF to achieve these improvements. Furthermore, we observe a positive correlation between posterior-approximation quality (measured via MMD) and the predictive quality. Considering that $s = 20$ is used, this validates our hypothesis that one can "get away" with fewer number of MC samples when accurate weight-space posterior approximations are employed.

Moreover, we present out-of-distribution (OOD) data detection in Table 4.4. We observe that while the last-layer LA baseline can already be better than all-layer baselines—as also observed by Daxberger et al. (2021a)—refining it can yield better results: Even with a small NF, e.g., $\ell = 5$, the OOD detection performance of the refined LA is close to that of the gold-standard HMC.

Table 4.4: OOD detection in terms of FPR95 (in percent, lower is better), averaged over three test sets and five seeds. All-layer baselines are asterisk-marked.

Methods	CIFAR-10	CIFAR-100
VB*	62.9±2.0	80.8±1.0
CSGHMC*	58.7±1.6	79.3±1.0
LA	49.2±2.4	79.6±1.0
LA-Refine-1	47.7±2.1	77.3±0.7
LA-Refine-5	46.8±2.2	77.8±0.7
LA-Refine-10	46.2±2.3	77.8±0.7
LA-Refine-30	46.1±2.3	77.9±0.8
HMC	46.0±2.3	77.8±0.9

We show that it is indeed desirable to do *refinement*, i.e. using an approximate posterior as the base distribution of the NF, instead of starting from scratch, i.e. starting from a data-independent distribution such as $\mathcal{N}(0, I)$. As shown in Fig. 4.7, starting from an approximate posterior yields better predictive distributions faster than when $\mathcal{N}(0, I)$ is used as the base distribution of the NF. This is particularly important since the computational cost of a NF depends on its length: We see a 53% increase in training time from $\ell = 5$ to $\ell = 10$—the latter is required for the *a priori* NF approximation to yield similar predictive performance to the refined posterior.

Finally, to validate that the refinement technique yields accurate posterior approximations, we plot the empirical (i.e., obtained via samples) marginal densities $q(w_i \mid \mathcal{D})$ of randomly selected weight $w_i \in \theta$ in Fig. 4.6. We validate that the refinement method makes the crude, base LA posteriors closer to HMC in the weight space.

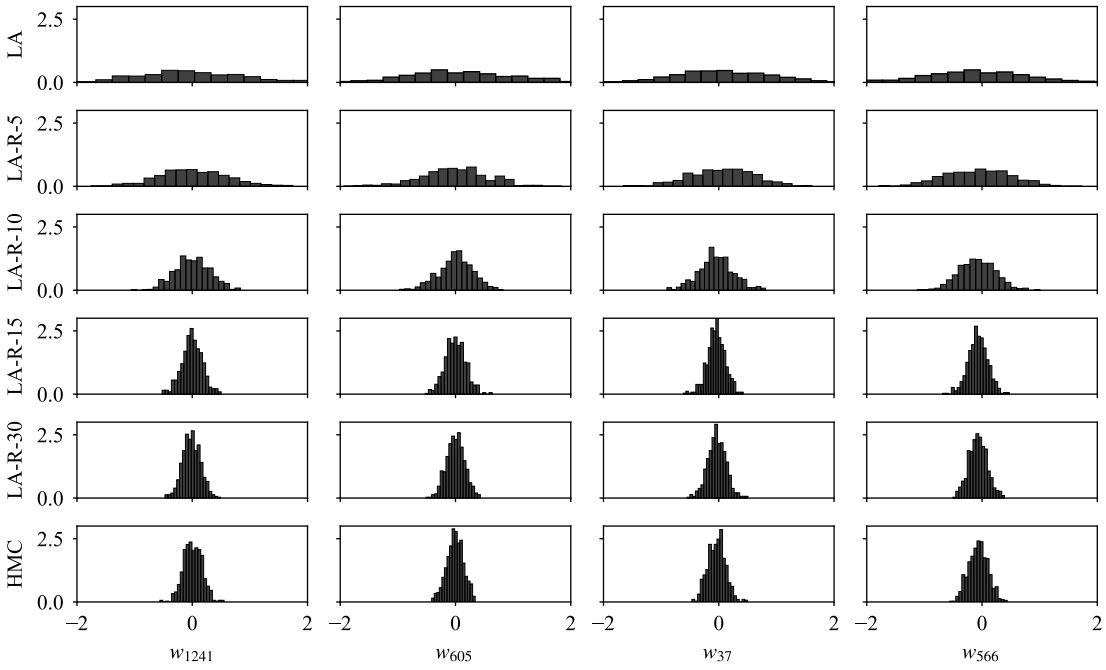


Figure 4.6: Marginal weight distributions on CIFAR-10.

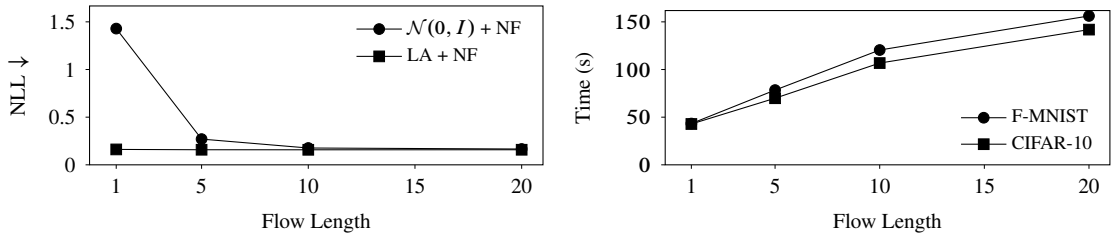


Figure 4.7: Calibration and wall-clock refinement time vs. flow length.

4.1.4.3 Costs

The proposed refinement technique is *post-hoc* and cheap when applied to last-layer BNNs. Suppose one already has a last-layer Gaussian approximate posterior. Using a standard consumer GPU (Nvidia RTX 2080Ti), each epoch of a length-5 NF's optimization takes around 3.4 seconds. From our experiments, we found that a low number of epochs (we use 20) is already sufficient for improving a crude approximate posterior. Thus, the entire refinement process is quick, especially when compared to MAP estimation, ELBO optimization, or HMC sampling.

4.2 Learnable Uncertainty under Laplace Approximations

A standard practice in contemporary LAs is to tune a single hyperparameter—the prior precision—to calibrate their predictive uncertainty (Chapter 2). However, this scalar parametrization allows only for a very limited form of uncertainty calibration. Here, we propose a more flexible framework to tune the uncertainty of Laplace-approximated BNNs without changing their point estimates. The idea is to introduce additional hidden units, associated with partly zero weights,

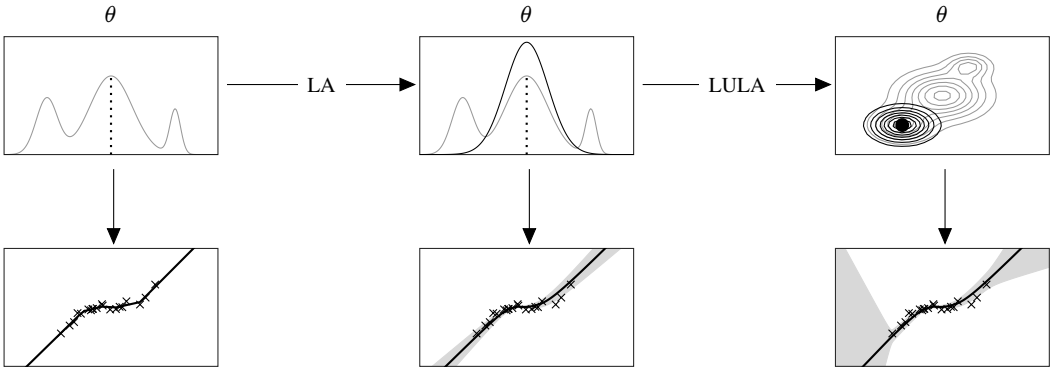


Figure 4.8: A schematic of our LULA. **Top row:** grey and black curves represent the true and the Laplace-approximated posteriors over the parameter space, respectively—the point estimates are dotted. **Bottom row:** predictions induced by the respective Laplace approximation—curves and shades are predictive means and 95% confidence intervals, respectively. Our method adds further degrees of freedom to the parameter space—as induced by additional hidden units with a particular weight structure—and finds a point in the augmented space that induces the same predictions but with better-calibrated uncertainty estimates (esp. w.r.t. outliers), under a Laplace approximation.

to the hidden layers of any MAP-trained network. Because of their weight structure, they are partly inactive and do not affect the prediction of the underlying network. However, they can still contribute to the Hessian of the loss with respect to the parameters, and hence induce additional structure to the posterior covariance under a Laplace approximation—these units are thus *uncertainty units* under Laplace approximations. Furthermore, the non-zero weights associated with these units can then be trained via an uncertainty-aware objective (Lee et al., 2018b; Hendrycks et al., 2019, etc.), such that they improve the predictive uncertainty quantification performance of the Laplace-approximated BNN. Figure 4.8 provides intuition.

To that end, we introduce *learnable uncertainty under Laplace approximations (LULA)* method. The premise is to add *uncertainty units*, which can be added to the layers of any MAP-trained network (Section 4.2.1) and trained via an uncertainty-aware loss (Section 4.2.2) to improve uncertainty calibration under Laplace approximations.

4.2.1 Construction

Let $f : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^k$ be a MAP-trained L-layer neural network with parameters $\theta_{\text{MAP}} = (W^{(\ell)}_{\text{MAP}}, b^{(\ell)}_{\text{MAP}})^L_{\ell=1}$. The premise of the method is simple: At each hidden layer $\ell = 1, \ldots, L-1$, we add $m_\ell \in \mathbb{Z}_{\geq 0}$ additional hidden units (under the original activation function) to $h^{(\ell)}$—as a consequence, the ℓ-th weight matrix and bias vector need to be extended to accommodate them. The method augments these parameters in such a way that for any input $x \in \mathbb{R}^n$, the original network output $f(x; \theta_{\text{MAP}})$ is preserved, as follows.

For each layer $\ell = 1, \dots, L-1$ of the network f, we expand the MAP-estimated weight matrix $W_{\text{MAP}}^{(\ell)} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ and the bias vector $b_{\text{MAP}}^{(\ell)} \in \mathbb{R}^{n_\ell}$ to obtain the following block matrix and vector:

$$\widetilde{W}^{(\ell)} := \begin{pmatrix} W_{\text{MAP}}^{(\ell)} & 0 \\ \widehat{W}_1^{(\ell)} & \widehat{W}_2^{(\ell)} \end{pmatrix} \in \mathbb{R}^{(n_\ell + m_\ell) \times (n_{\ell-1} + m_{\ell-1})},$$
$$\widetilde{b}^{(\ell)} := \begin{pmatrix} b_{\text{MAP}}^{(\ell)} \\ \widehat{b}^{(\ell)} \end{pmatrix} \in \mathbb{R}^{n_\ell + m_\ell}, \tag{4.4}$$

to take into account the additional m_ℓ hidden units. We do not add additional units to the input layer, so $m_0 = 0$. Furthermore, for $\ell = L$, we define

$$\widetilde{W}^{(L)} := (W_{\text{MAP}}^{(L)}, 0) \in \mathbb{R}^{k \times (n_{L-1} + m_{L-1})};$$
$$\widetilde{b}^{(L)} := b_{\text{MAP}}^{(L)} \in \mathbb{R}^k, \tag{4.5}$$

so that the output dimensionality is also unchanged. For brevity, we denote by $\widehat{\theta}^{(\ell)}$ the non-zero additional parameters in (4.4), i.e. we define $\widehat{\theta}^{(\ell)}$ to be the tuple $(\widehat{W}_1^{(\ell)}, \widehat{W}_2^{(\ell)}, \widehat{b}^{(\ell)})$. Altogether, considering all layers $\ell = 1, \dots, L-1$, we denote

$$\widehat{\theta} := (\theta^{(\ell)})_{\ell=1}^{L-1},$$

to be the tuple of all non-zero additional parameters of the network f. Furthermore, we write the resulting augmented network as $\widetilde{f}$ and the resulting overall parameter vector—consisting of $(\widetilde{W}^{(\ell)}, \widetilde{b}^{(\ell)})_{\ell=1}^{L}$—as $\widetilde{\theta}_{\text{MAP}} \in \mathbb{R}^{\widetilde{d}}$, where $\widetilde{d}$ is the resulting number of parameters. Refer to Fig. 4.9 for an illustration. Note that we can easily extend this construction to convolutional networks by expanding the "channel" of hidden convolution layers.[3]

Let us inspect the implication of this construction. Here for each $\ell = 1, \dots, L-1$, the sub-matrices $\widehat{W}_1^{(\ell)}$, $\widehat{W}_2^{(\ell)}$ and the sub-vector $\widehat{b}^{(\ell)}$ contain parameters for the additional m_ℓ hidden units in the ℓ-th layer. We are free to choose the values of these parameters since the upper-right quadrant of $\widetilde{W}^{(\ell)}$, i.e. the zero part of the additional weights, *deactivates* the $m_{\ell-1}$ additional hidden units in the previous layer, hence they do not contribute to the original hidden units in the ℓ-th layer. Part (a) of the following proposition thus guarantees that the additional hidden units will not change the output of the network.

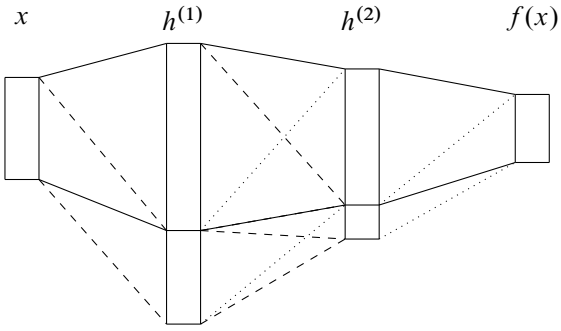


Figure 4.9: An illustration of the proposed construction. **Rectangles** represent layers, **solid lines** represent connection between layers, given by the original weight matrices. The additional units are represented by the additional block at the bottom of each layer. **Dashed lines** correspond to the free parameters $\widehat{\theta}$, while **dotted lines** to the zero weights.

Proposition 4.1 (Properties). *Let $f : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^k$ be a MAP-trained L-layer network*

[3] E.g. if the hidden units are a 3D array of (channel × height × width), then we expand the first dimension.

under dataset $\mathcal{D}$, and let θ_{MAP} be the MAP estimate. Suppose $\widetilde{f} : \mathbb{R}^n \times \mathbb{R}^{\widetilde{d}} \to \mathbb{R}$ and $\widetilde{\theta}_{MAP} \in \mathbb{R}^{\widetilde{d}}$ are obtained via the previous construction, and $\widetilde{\mathcal{L}}$ is the resulting loss function under $\widetilde{f}$.

(a) For an arbitrary input $x \in \mathbb{R}^n$, we have $\widetilde{f}(x; \widetilde{\theta}_{MAP}) = f(x; \theta_{MAP})$.
(b) The gradient of $\widetilde{\mathcal{L}}$ w.r.t. the additional weights in $\widetilde{W}^{(L)}$ is non-linear in $\widehat{\theta}$.

Proof. For each layer $\ell = 1, \dots, L$ we denote the hidden units and pre-activations of $\widetilde{f}$ as $\widetilde{h}^{(\ell)}$ and $\widetilde{a}^{(\ell)}$, respectively.

We begin with (a). Let $x \in \mathbb{R}^n$ be arbitrary. We need to show that the output of $\widetilde{f}$, i.e. the last pre-activations $\widetilde{a}^{(L)}$, is equal to the last pre-activations $a^{(L)}$ of f. For the first layer, we have that

$$\begin{aligned} \widetilde{a}^{(1)} &= \widetilde{W}^{(1)} x + \widetilde{b}^{(1)} \\ &= \begin{pmatrix} W^{(1)} \\ \widehat{W}_1^{(1)} \end{pmatrix} x + \begin{pmatrix} b^{(1)} \\ \widehat{b}^{(1)} \end{pmatrix} \\ &= \begin{pmatrix} W^{(1)} x + b^{(1)} \\ \widehat{W}_1^{(1)} x + \widehat{b}^{(1)} \end{pmatrix} =: \begin{pmatrix} a^{(1)} \\ \widehat{a}^{(1)} \end{pmatrix}. \end{aligned} \tag{4.6}$$

For every layer $\ell = 1, \dots, L-1$, we denote the hidden units as the block vector

$$\widetilde{h}^{(\ell)} = \begin{pmatrix} \varphi(a^{(\ell)}) \\ \varphi(\widehat{a}^{(\ell)}) \end{pmatrix} = \begin{pmatrix} h^{(\ell)} \\ \widehat{h}^{(\ell)} \end{pmatrix}.$$

Now, for the intermediate layer $l = 2, \dots, L-1$, we observe that

$$\begin{aligned} \widetilde{a}^{(\ell)} &= \widetilde{W}^{(\ell)} \widetilde{h}^{(\ell-1)} + \widetilde{b}^{(\ell)} \\ &= \begin{pmatrix} W^{(\ell)} & 0 \\ \widehat{W}_1^{(\ell)} & \widehat{W}_2^{(\ell)} \end{pmatrix} \begin{pmatrix} h^{(\ell-1)} \\ \widehat{h}^{(\ell-1)} \end{pmatrix} + \begin{pmatrix} b^{(\ell)} \\ \widehat{b}^{(\ell)} \end{pmatrix} \\ &= \begin{pmatrix} W^{(\ell)} h^{(\ell-1)} + 0 + b^{(\ell)} \\ \widehat{W}_1^{(\ell)} h^{(\ell-1)} + \widehat{W}_2^{(\ell)} \widehat{h}^{(\ell-1)} + \widehat{b}^{(\ell)} \end{pmatrix} =: \begin{pmatrix} a^{(\ell)} \\ \widehat{a}^{(\ell)} \end{pmatrix}. \end{aligned} \tag{4.7}$$

Finally, for the last layer, we get

$$\begin{aligned} \widetilde{a}^{(L)} &= \widetilde{W}^{(L)} x + \widetilde{b}^{(L)} \\ &= \begin{pmatrix} W^{(L)} & 0 \end{pmatrix} \begin{pmatrix} h^{(L-1)} \\ \widehat{h}^{(L-1)} \end{pmatrix} + b^{(L)} \\ &= W^{(L)} h^{(L-1)} + 0 + b^{(L)} \\ &= a^{(L)}, \end{aligned} \tag{4.8}$$

and thus we have the desired invariance.

For part (b), we denote the additional (zero) weights in $\widetilde{W}^{(L)}$ by $\widehat{W}^{(L)}$. It is clear from (4.8) that the gradient $\nabla_{\widehat{W}^{(L)}} a^{(L)}$ is given by $\widehat{h}^{(L-1)}$. Hence, by chain rule we have

$$\begin{aligned} \nabla_{\widehat{W}^{(L)}} \widetilde{\mathcal{L}} &= (\nabla_{a^{(L)}} \widetilde{\mathcal{L}}) \, (\nabla_{\widehat{W}^{(L)}} a^{(L)}) \\ &= (\nabla_{a^{(L)}} \widetilde{\mathcal{L}}) \, \widehat{h}^{(L-1)}. \end{aligned}$$

By observing (4.6) and (4.7), along the fact that the non-linearity φ is used in the forward pass, it is clear that $\widehat{h}^{(L-1)}$ is non-linear in $\widehat{\theta} = (\widehat{W}_1^{(1)}, \widehat{b}^{(1)}, \ldots, \widehat{W}_1^{(L-1)}, \widehat{W}_2^{(L-1)}, \widehat{b}^{(L-1)})$ and therefore $\nabla_{\widehat{W}^{(L)}} \widetilde{\mathcal{L}}$ also is. □

Part (b) of the last proposition tells us that the additional non-zero weights $\widehat{\theta}$ affect the loss landscape in a non-trivial way, and they, in general, induce non-trivial curvatures along the additional dimensions in the last-layer weight matrix (4.5) of the network. Therefore this construction non-trivially affects the covariance matrix in a LA. The implication of this insight to predictive uncertainty can be seen clearly in real-valued networks with diagonal LA posteriors, as the following proposition shows.

Proposition 4.2 (Predictive Uncertainty). *Suppose $f : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}$ is a real-valued network and $\widetilde{f}$ is as constructed above. Suppose further that diagonal Laplace approximations $\mathcal{N}(\theta_{MAP}, \mathrm{diag}(\sigma))$, $\mathcal{N}(\widetilde{\theta}_{MAP}, \mathrm{diag}(\widetilde{\sigma}))$ are employed for f and $\widetilde{f}$, respectively. Under the network linearization, for any input $x \in \mathbb{R}^n$, the variance over the output $\widetilde{f}(x; \widetilde{\theta})$ is at least that of $f(x; \theta)$.*

Proof. Let us denote the random variable taking values in the augmented parameter space by $\widetilde{\theta}$. W.l.o.g. we re-arrange $\widetilde{\theta}$ as $(\theta^\top, \widehat{\theta}^\top)^\top$ where $\widehat{\theta} \in \mathbb{R}^{\widetilde{d}-d}$ contains the weights corresponding to the the additional LULA units. If $g(x)$ is the gradient of the output $f(x; \theta)$ w.r.t. θ at θ_{MAP}, then the gradient of $\widetilde{f}(x; \widetilde{\theta})$ w.r.t. $\widetilde{\theta}$ at $\widetilde{\theta}_{\mathrm{MAP}}$, say $\widetilde{g}(x)$, can be written as the concatenation $(g(x)^\top, \widehat{g}(x)^\top)^\top$ where $\widehat{g}(x)$ is the corresponding gradient w.r.t. $\widehat{\theta}$. Furthermore, $\mathrm{diag}(\widetilde{\sigma})$ has diagonal elements

$$\left(\sigma_{11}, \ldots, \sigma_{dd}, \widehat{\sigma}_{11}, \ldots, \widehat{\sigma}_{\widetilde{d}-d, \widetilde{d}-d}\right)^\top =: (\sigma^\top, \widehat{\sigma}^\top)^\top.$$

Let $x \in \mathbb{R}^n$ be an arbitrary input. Denoting the output variance of $\widetilde{f}(x; \widetilde{\theta})$ by $\widetilde{v}(x)$, we have

$$\begin{aligned}
\widetilde{v}(x) &= \widetilde{g}(x)^\top \mathrm{diag}(\widetilde{\sigma}) \widetilde{g}(x) \\
&= \underbrace{g(x)^\top \mathrm{diag}(\sigma) g(x)}_{=v(x)} + \widehat{g}(x)^\top \mathrm{diag}(\widehat{\sigma}) \widehat{g}(x) \\
&\geq v(x),
\end{aligned}$$

since $\mathrm{diag}(\widehat{\sigma})$ is positive-definite by definition. □

In summary, the construction along with Propositions 4.1 and 4.2 imply that the additional hidden units we have added to the original network are *uncertainty units* under Laplace approximations, i.e. hidden units that *only* contribute to the Laplace-approximated uncertainty and not the predictions. Furthermore, by part (b) of Proposition 4.1, the values of $\widehat{\theta}$—which can be set freely without affecting the output—influence the loss-landscape Hessian in a non-trivial way. They are thus learnable and so we call these units ***learnable Uncertainty under Laplace approximations (LULA)*** units.

4.2.2 Training

In this section, we discuss a way to train LULA units to improve predictive uncertainty under Laplace approximations. We follow a contemporary technique from the *non*-Bayesian robust learning literature which has been shown to be effective in improving uncertainty calibration of non-Bayesian networks (Lee et al., 2018b; Hendrycks et al., 2019; Bitterwolf et al., 2020, etc.).

Let $f : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^k$ be an L-layer neural network with a MAP-trained parameters θ_{MAP} and let $\widetilde{f} : \mathbb{R}^n \times \mathbb{R}^{\widetilde{d}} \to \mathbb{R}^k$ along with $\widetilde{\theta}_{\text{MAP}}$ be obtained by adding LULA units. Let $q(\widetilde{\theta}) := \mathcal{N}(\widetilde{\theta}_{\text{MAP}}, \widetilde{\Sigma})$ be the Laplace-approximated posterior and $p(y \mid x, \mathcal{D}; \widetilde{\theta}_{\text{MAP}})$ be the (approximate) predictive distribution under the LA. Furthermore, let us denote the dataset sampled i.i.d. from the data distribution as $\mathcal{D}_{\text{in}}$ and that from some outlier distribution as $\mathcal{D}_{\text{out}}$, and let H be the entropy functional. We construct the following loss function to induce high uncertainty on outliers while maintaining high confidence over the data (inliers):

$$\begin{aligned}\mathcal{L}_{\text{LULA}}(\widetilde{\theta}_{\text{MAP}}) :=& \frac{1}{|\mathcal{D}_{\text{in}}|} \sum_{x \in \mathcal{D}_{\text{in}}} \mathbb{H}[p(y \mid x, \mathcal{D}; \widetilde{\theta}_{\text{MAP}})] \\ &- \frac{1}{|\mathcal{D}_{\text{out}}|} \sum_{x \in \mathcal{D}_{\text{out}}} \mathbb{H}[p(y \mid x, \mathcal{D}; \widetilde{\theta}_{\text{MAP}})],\end{aligned} \tag{4.9}$$

and minimize it w.r.t. *only* the free parameters $\widehat{\theta}$. This objective is task agnostic—it can be used in regression and classification networks alike. Furthermore, the first term of this objective can alternatively be replaced with the standard negative log-likelihood loss. In our case, since by Proposition 4.1, predictions do not change under LULA, using the negative log-likelihood yields the same result as predictive entropy: they both only affect uncertainty and keep predictions over $\mathcal{D}_{\text{in}}$ confident. In any case, without this term, $\mathcal{L}_{\text{LULA}}$ potentially assigns the trivial solution of maximum uncertainty prediction everywhere in the input space.

The intuition of LULA training is as follows. By adding LULA units, we obtain a non-trivially augmented version of the network's loss landscape (Proposition 4.1(b)). The goal of LULA training is then to exploit the weight-space symmetry (i.e. different parameters that induce the same output) arising from the construction as shown by Proposition 4.1(a), and pick a point in the extended parameter space that is symmetric to the original parameters but has "better" curvatures, in the sense that they induce lower loss (4.9). These parameters, then, when used in a LA, improve the predictive uncertainty of standard non-LULA-augmented LAs.

4.2.2.1 Practical Matters

Datasets We can simply set $\mathcal{D}_{\text{in}}$ to be the validation set of the dataset $\mathcal{D}$. Meanwhile, $\mathcal{D}_{\text{out}}$ can be chosen depending on the task at hand, e.g. noise and large-scale natural image datasets can be used for regression and image classification tasks, respectively (Hendrycks et al., 2019).

Maintaining Weight Structures Since our aim is to improve predictive uncertainty by exploiting weight-space symmetries given by the structure of LULA weights, we must maintain the structure of all weights and biases in $\widetilde{\theta}_{\text{MAP}}$, in accordance to (4.4) and (4.5). This can be enforced by gradient masking: For all $\ell = 1, \ldots, L-1$, set the gradients of the blocks of $\widetilde{W}^{(\ell)}$ and $\widetilde{b}^{(\ell)}$ not corresponding to $\widehat{W}_1^{(\ell)}$, $\widehat{W}_2^{(\ell)}$, and $\widehat{b}^{(\ell)}$, to zero. Under this scheme, Proposition 4.1(a) will still hold for trained LULA units.

Algorithm 2 Training LULA units.

Input:

MAP-trained network f. Dataset $\mathcal{D}_{\text{in}}$, OOD dataset $\mathcal{D}_{\text{out}}$. Learning rate α. Number of epochs E.

Construct $\widetilde{f}$ from f by following Section 4.2.1.
for $i = 1, \ldots, E$ **do**
  $q(\widetilde{\theta}) = \mathcal{N}(\widetilde{\theta}_{\text{MAP}}, \widetilde{\Sigma}(\widetilde{\theta}_{\text{MAP}}))$
  Compute $\mathcal{L}_{\text{LULA}}(\widetilde{\theta}_{\text{MAP}})$ via (4.9) with $q(\widetilde{\theta})$, $\mathcal{D}$, $\mathcal{D}_{\text{out}}$
  $g = \nabla \mathcal{L}_{\text{LULA}}(\widetilde{\theta}_{\text{MAP}})$
  $\widehat{g} = \texttt{mask\_gradient}(g)$
  $\widetilde{\theta}_{\text{MAP}} = \widetilde{\theta}_{\text{MAP}} - \alpha \widehat{g}$
end for
$p(\widetilde{\theta} \mid \mathcal{D}) \approx \mathcal{N}(\widetilde{\theta}_{\text{MAP}}, \widetilde{\Sigma}(\widetilde{\theta}_{\text{MAP}}))$
return $\widetilde{f}$ and $p(\widetilde{\theta} \mid \mathcal{D})$

Laplace Approximations During Training Since the covariance matrix $\widetilde{\Sigma}$ of the Laplace-approximated posterior depends on $\widetilde{\theta}_{\text{MAP}}$, it needs to be updated at every iteration during the optimization of $\mathcal{L}_{\text{LULA}}$. This can be expensive for large networks depending on the Laplace approximation used, not to mention that one must use the entire dataset $\mathcal{D}_{\text{in}}$ to obtain this matrix. As a simple and much cheaper proxy to the true covariance, we employ a simple diagonal Fisher information matrix (Amari, 1998; Martens, 2020), obtained from a single minibatch, irrespective of the Laplace approximation variant employed at test time—we show in Section 4.2.4 that this training scheme is both effective and efficient.[4] Finally, we note that backpropagation through this diagonal matrix, which is fully determined by the network's gradient, does not pose a difficulty since modern deep learning libraries such as PyTorch and TensorFlow support "double backprop" efficiently. Algorithm 2 provides a summary of LULA training in pseudocode. Code can be found in `https://github.com/wiseodd/lula`.

4.2.3 Related Work

While traditionally hyperparameter optimization in LAs requires re-training the network (under type-II maximum likelihood or the evidence framework (MacKay, 1992c) or empirical Bayes (Robbins, 1956)), tuning it in a *post-hoc* manner has become increasingly common. Ritter et al. (2018b;a) tune the prior precision of a LA by maximizing the predictive log-likelihood. However, they are limited in terms of flexibility since the prior precision of the LAs constitutes a single scalar parameter. LULA can be seen as an extension of these approaches with greater flexibility and is complementary to them since it does not modify the prior precision used.

Confidence calibration via outliers has achieved state-of-the-art performance in non-Bayesian outlier detection. Hendrycks et al. (2019); Hein et al. (2019); Meinke & Hein (2020) use outliers to regularize the standard maximum-likelihood training. Malinin & Gales (2018; 2019) use outliers to train probabilistic models based on the Dirichlet distribution. In contrast to our approach, all these methods are neither Bayesian nor *post-hoc*.

[4] The actual Laplace approximations applied after LULA training can be *non*-diagonal.

4.2.4 Experiments

We empirically validate that LULA does improve vanilla LAs via toy and image classification experiments—results on UCI regression tasks are in the appendix. We expand the image classification experiment into *dataset shift robustness* and *out-of-distribution* (OOD) experiments to show LULA's performance over standard benchmark suites.

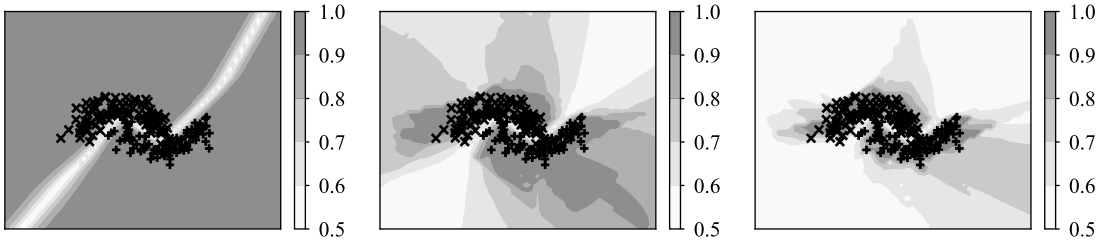
4.2.4.1 Setup

Toy experiments We use the "cubic" (Hernández-Lobato & Adams, 2015) and "two moons" datasets for regression and classification, respectively. For classification, we use a full Laplace with generalized Gauss-Newton Hessian approximation on a three-layer FC network. For regression, we apply the Kronecker-factored Laplace (KFL) (Ritter et al., 2018a) on a two-layer fully-connected network. In this particular case, we directly use the predictive variance instead of (differential) entropy for (4.9). The two are closely related, but in the case of regression with continuous output, the variance is easier to work with since it is lower-bounded by zero. Finally, the corresponding numbers of additional LULA units are 30 and 50, respectively.

Image classification We use the following standard datasets: MNIST, SVHN, CIFAR-10, and CIFAR-100. For each dataset, we split its test set to obtain a validation set of size 2000. On all datasets and all methods, we use the WideResNet-16-4 architecture (Zagoruyko & Komodakis, 2016) and optimize the network with Nesterov-SGD with weight decay 5×10^{-4} and initial learning rate 0.1 for 100 epochs. We anneal the learning rate with the cosine decay method (Loshchilov & Hutter, 2017).

Baselines We use the vanilla MAP-trained network (abbreviated as **MAP**), a last-layer KFL (**LA**), and Deep Ensemble (**DE**) (Lakshminarayanan et al., 2017) as baselines. For MAP and DE, we additionally use the temperature scaling post-processing scheme to improve their calibration (**Temp**) (Guo et al., 2017). Specifically for DE, a single temperature hyperparameter is used for all ensemble members (Rahaman & Thiery, 2020). Note that DE is used to represent the state-of-the-art uncertainty-quantification methods (Ovadia et al., 2019). For the Bayesian baseline (LA), we use a last-layer Laplace since it has been shown to be competitive to its all-layer counterpart while being much cheaper and thus more suitable for large networks (Kristiadi et al., 2020). We do not tune the prior variance of LA—it is obtained from the weight decay used during MAP training. Nevertheless, to show that LULA is also applicable to and can improve methods which their uncertainty is already explicitly tuned, we additionally use two OOD-trained/tuned baselines for the OOD-detection benchmark: (i) the last-layer Laplace where the prior variance is tuned via an OOD validation set (**LLLA**) (Kristiadi et al., 2020), and (ii) the outlier exposure method (**OE**) (Hendrycks et al., 2019) where OOD data is used during the MAP training itself. For the latter, we apply a standard last-layer KFL post-training (see (Kristiadi et al., 2020, Appendix D.6)).

LULA For the toy experiments, we use uniform noise as $\mathcal{D}_{\text{out}}$. We add 50 and 30 LULA units to each layer of the toy regression and classification networks, respectively. Meanwhile, we use the downscaled ImageNet dataset (Chrabaszcz et al., 2017) as $\mathcal{D}_{\text{out}}$ for the image classification experiments. We do not use the 80 Million Tiny Images dataset (Torralba et al., 2008) as used by



(a) MAP **(b)** LA **(c)** LA-LULA

Figure 4.10: Predictive uncertainty estimates of a standard LA and the LULA-augmented LA. Black curves and shades are decision boundaries and confidence estimates, respectively.

Hendrycks et al. (2019); Meinke & Hein (2020); Bitterwolf et al. (2020) since it is not available anymore. We use the aforementioned ImageNet dataset as the OOD dataset for training/tuning the LLLA and OE baselines. We put LULA units on top of the pre-trained LA baseline and optimize them using Adam for 10 epochs using the validation set. To pick the number of additional (last-layer) LULA units, we employ a grid search over the set $\{32, 64, 128, 256, 512, 1024\}$ and pick the one minimizing validation LULA loss $\mathcal{L}_{\text{LULA}}$ under the LA. Finally, note that we implement LULA on top of the KFL discussed above, thus by doing so, we show that LULA is generally applicable even though it is specifically trained via a proxy diagonal LA.

Benchmark For the dataset shift robustness experiment, we use the standard rotated-MNIST (MNIST-R) and corrupted-CIFAR-10 (CIFAR-10-C) datasets, which contain corrupted MNIST and CIFAR-10 test images with varying severity levels, respectively. Meanwhile, for the OOD experiment, we use 6 OOD datasets for each *in-distribution dataset* (i.e. the dataset the model is trained on).

Metrics First, we denote with "↓" next to the name of a metric to indicate that lower values are better, and vice versa for "↑". We use the standard uncertainty metrics: expected calibration error (ECE ↓) (Naeini et al., 2015), Brier score (↓) (Brier et al., 1950), test log-likelihood (↑), and average confidence (MMC ↓) (Hendrycks et al., 2019). Additionally, for OOD detection, we use the FPR95 (↓) metric which measures the false positive rate at a fixed true positive rate of 95% when discriminating between in- and out-of-distribution data, based on their confidence (maximum predictive probability) estimates.

4.2.4.2 Toy Experiments

We begin with toy regression and classification results in Fig. 4.8 (bottom) and Fig. 4.10, respectively. As expected, the MAP-trained networks produce overconfident predictions in both cases. While LA provides meaningful uncertainty estimates, it can still be overconfident near the data. The same can be seen in the regression case: LA's uncertainty outside the data region grows slowly. LULA improves both cases: it makes (i) the regression uncertainty grow faster far from the data and (ii) the classification confidence more compact around the data region. Notice that in both cases LULA does not change the prediction of LA.

Table 4.5: Calibration and generalization performance. All values are in percent and averages over five prediction runs. Best ECE values among each pair of the vanilla and LULA-equipped methods (e.g. LA and LA-LULA) are in bold. Best overall values are underlined.

	MNIST	**SVHN**	**CIFAR-10**	**CIFAR-100**
ECE ↓				
MAP	13.8±0.0	9.7±0.0	12.2±0.0	16.6±0.0
MAP-Temp	14.8±0.0	<u>2.0</u>±0.0	4.5±0.0	<u>4.1</u>±0.0
DE	<u>13.2</u>±0.0	4.3±0.0	6.1±0.0	5.4±0.0
DE-Temp	16.9±0.0	2.2±0.0	<u>3.8</u>±0.0	4.5±0.0
LA	**12.6**±0.1	9.3±0.0	10.9±0.3	7.0±0.1
LA-LULA	14.8±0.3	**3.3**±0.1	**7.5**±0.1	**5.3**±0.2
Accuracy ↑				
MAP	99.7±0.0	97.1±0.0	95.0±0.0	75.8±0.0
MAP-Temp	99.7±0.0	97.1±0.0	95.0±0.0	75.8±0.0
DE	99.7±0.0	97.6±0.0	95.5±0.0	79.0±0.0
DE-Temp	99.7±0.0	97.6±0.0	95.5±0.0	79.1±0.0
LA	99.7±0.0	97.1±0.0	95.0±0.0	75.8±0.0
LA-LULA	99.6±0.0	97.1±0.0	94.9±0.0	75.6±0.1

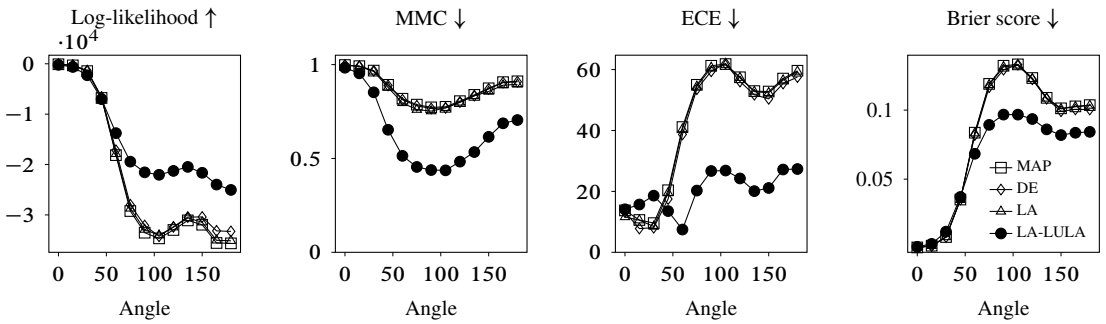


Figure 4.11: Values of various uncertainty metrics as the rotation angle on MNIST images increases.

4.2.4.3 Calibration

Table 4.5 summarizes the calibration and generalization performance of LULA in terms of ECE and test accuracy, respectively. We found that on "harder" datasets (SVHN, CIFAR-10, CIFAR-100), LULA consistently improves the vanilla LA's calibration, often even better than DE. However, on MNIST, both DE and LULA attain worse calibration than the vanilla LA. This might be because the accuracy of the network on MNIST is already almost perfect, thus even an overconfident classifier could yield a good ECE value—DE and LULA generally reduce confidence estimates (cf. Table F.3 in the appendix) and thus yielding higher ECE values. Nevertheless, as we shall see in the next section, LULA is in general better calibrated to outliers than the other baselines on MNIST. As a final note, we emphasize that LULA preserves the predictive performance of the base LA and thus MAP's. This is important in practice: The allure of deep networks is their high predictive performance, thus, "non-destructive" *post-hoc* methods are desirable.

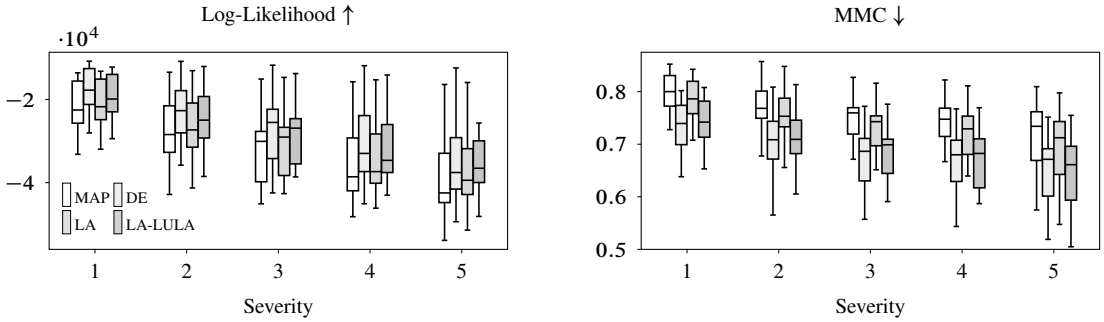


Figure 4.12: Calibration performance at each severity level of the CIFAR-10-C dataset.

4.2.4.4 Dataset Shift Robustness

Dataset shift robustness tasks benchmark uncertainty calibration of a predictive model on corruptions or perturbations of the true dataset. To this end, we present various uncertainty metrics of LULA on the MNIST-R dataset in Fig. 4.11. In all metrics considered, LULA improves not only the vanilla LA upon which LULA is implemented but also the state-of-the-art baseline in DE. Thus, even though LULA reduces calibration on the true MNIST dataset, it excels in making the network robust to outliers.

We furthermore present the results on the corrupted CIFAR-10 dataset in Fig. 4.12. It can be seen that on average, LULA improves the vanilla LA, making it competitive to DE. In fact, on higher corruption levels, LULA can achieve better performance than DE, albeit marginally so. Nevertheless, this is important since standard BNNs have been shown to underperform compared to DE (Ovadia et al., 2019).

4.2.4.5 OOD Detection

While dataset shift robustness tasks measure performance over outliers that are close to the true data, OOD detection tasks test performance on outliers that are far away from the data (e.g. SVHN images as outliers for the CIFAR-10 dataset). Table 4.6 summarizes results. For each in-distribution dataset, LULA consistently improves the base LA, both in terms of its confidence estimates on OOD data (MMC) and its detection performance (FPR95). Furthermore, LULA in general assigns lower confidence to OOD data than DE. This suggests that, far from the data, LULA is more calibrated than DE. While LULA is better than DE in the detection of OOD data on CIFAR-10, DE yields a stronger FPR95 performance than LULA in general. Nevertheless, we stress that LULA is more cost-efficient than DE since it can be applied to any MAP-trained network *post-hoc*. Moreover, unlike DE which requires us to train multiple (in our case, 5) independent networks, LULA training is far cheaper than even the training time of a single network—see next section.

As stated in Section 4.2.3, LULA is orthogonal to prior variance tuning methods commonly done in Laplace approximations. Hence, in Table 4.6 we also show the OOD detection performance of LULA when applied to the LLLA base-

Table 4.7: Wall-clock time in seconds for augmenting the WideResNet-16-4 network with 512 LULA units and training them for ten epochs with 2000 validation data.

	MNIST	**SVHN**	**CIFAR-10**	**CIFAR-100**
Construction	0.005	0.005	0.004	0.006
Training	20.898	22.856	22.222	21.648

Table 4.6: OOD detection performance for each in-distribution dataset in terms of MMC and FPR95. Values are averages over six OOD test sets and five prediction runs. Best values among each pair of the vanilla and LULA-equipped methods are in bold. Best overall values are underlined.

	MNIST	**SVHN**	**CIFAR-10**	**CIFAR-100**
MMC ↓				
MAP	80.4±0.0	72.9±0.1	74.2±0.1	64.5±0.1
MAP-Temp	82.2±0.0	63.4±0.0	60.5±0.0	48.2±0.1
DE	73.8±0.0	58.3±0.1	66.3±0.0	46.8±0.0
DE-Temp	84.1±0.0	59.0±0.1	62.0±0.0	46.5±0.1
LA	78.7±0.1	72.1±0.1	70.7±0.2	53.4±0.2
LA-LULA	**46.0**±0.8	**60.9**±0.2	**63.8**±0.4	**41.0**±0.5
LLLA	61.0±0.4	**47.3**±0.3	42.8±0.4	46.5±0.5
LLLA-LULA	**56.9**±0.8	52.1±0.4	**35.1**±0.3	**33.1**±0.7
OE	35.2±0.0	**18.0**±0.0	53.4±0.0	51.8±0.0
OE-LULA	**22.6**±0.2	20.1±0.2	**52.0**±0.1	**44.5**±0.2
FPR95 ↓				
MAP	5.0±0.0	25.9±0.1	53.1±0.2	80.1±0.1
MAP-Temp	5.0±0.0	25.6±0.1	47.0±0.2	77.1±0.1
DE	4.2±0.0	11.9±0.1	47.6±0.0	59.3±0.1
DE-Temp	4.5±0.0	16.4±0.1	44.8±0.0	72.3±0.1
LA	**4.9**±0.0	25.5±0.2	48.5±0.5	78.3±0.5
LA-LULA	5.8±0.5	**21.1**±0.4	**39.5**±1.4	**71.9**±1.3
LLLA	5.8±0.5	22.0±1.8	**23.7**±0.5	75.4±0.9
LLLA-LULA	**4.5**±0.1	**19.4**±0.5	**22.9**±0.8	**68.4**±1.7
OE	5.5±0.0	**1.7**±0.0	27.4±0.0	59.6±0.1
OE-LULA	**5.1**±0.3	**1.7**±0.0	**26.7**±0.2	**58.5**±0.4

line. We observe that LULA consistently improves LLLA. The same observation can also be seen when LULA is applied on top of a Laplace-approximated state-of-the-art OOD detector (OE): LULA also consistently improves OE even further.

4.2.4.6 Cost

Table 4.7 shows the computational overhead of LULA (wall-clock time, in seconds) on a single NVIDIA V100 GPU. The cost of augmenting the WideResNet-16-4 network with 512 LULA units is negligible. The training time of these units is around 20 seconds, which is also negligible compared to the time needed to do MAP training.

4.3 Out-of-Distribution Training for BNNs

Both the Bayesian and frequentist deep learning communities address similar UQ functionality, but it appears that even recent Bayesian neural networks (BNNs, Osawa et al., 2019; Tomczak et al., 2020; Dusenberry et al., 2020; Izmailov et al., 2021a;b, etc.) tend to underperform compared to the state-of-the-art frequentist UQ methods (Hendrycks et al., 2019; Lee et al., 2018b; Hein et al., 2019; Meinke & Hein, 2020; Bitterwolf et al., 2020, etc.). Figure 4.13 shows this

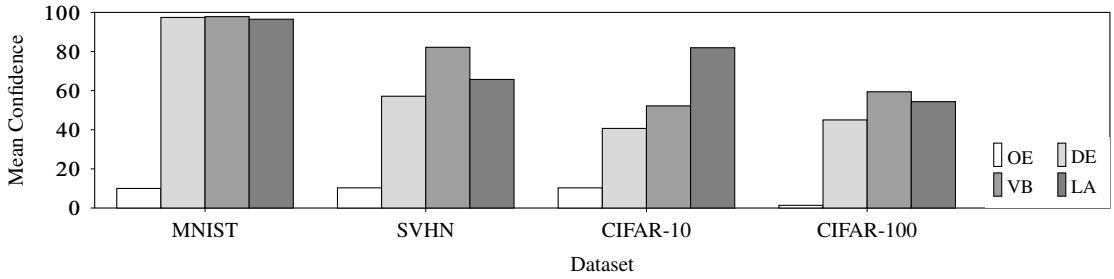


Figure 4.13: Average confidence on uniform OOD test data (lower is better). LA, VB, DE, and OE stand for the Laplace approximation, variational Bayes, Deep Ensemble, and Outlier Exposure, respectively. All methods have similar accuracy and confidence on all the in-distribution test sets, but OE is the only method that achieves low confidence on these outliers.

observation: the frequentist Outlier Exposure method (Hendrycks et al., 2019) performs much better than BNNs and even Deep Ensemble (Lakshminarayanan et al., 2017), which has been considered as a strong baseline in Bayesian deep learning (Ovadia et al., 2019).

This section thus seeks to answer the question of "how can we bring the performance of BNNs on par with that of recent frequentist UQ methods?" Our working hypothesis is that the disparity between them is *not* due to some fundamental advantage of the frequentist viewpoint. Rather, it is due to the more mundane practical fact that recent frequentist UQ methods leverage OOD data in their training process, via the so-called "OOD training" technique. The benefits of this technique are well-studied, both for improving generalization (Zhang & LeCun, 2017) and more recently, for OOD detection (Lee et al., 2018b; Hein et al., 2019; Meinke & Hein, 2020; Bitterwolf et al., 2020). But while OOD data have been used for tuning the hyperparameters of BNNs (Kristiadi et al., 2020), it appears that even recently proposed deep Bayesian methods have not considered OOD training. A reason for this may be that it is unclear how one can incorporate OOD data in the Bayesian inference itself.

We explore four options—some are philosophically clean, others are heuristics—of incorporating OOD data into Bayesian inference. These methods are motivated by the assumptions that the data (i) have an extra "none class", (ii) are entirely represented by "soft labels", or (iii) have mixed "hard" and "soft labels". Moreover, we also (iv) investigate an interpretation of the popular OE loss as a likelihood that can readily be used in Bayesian inference. We compare the four of them against strong baselines in various UQ tasks and show that BNNs equipped with these likelihoods can outperform a recent OOD-trained frequentist baseline. Our empirical findings thus validate the hypothesis that OOD training is the cause of BNNs underperformance. We hope that the proposed approaches, especially the simple yet best-performing "none-class" method, can serve as strong baselines in the Bayesian deep learning community.

4.3.1 Why Bayesian neural networks underperform

Suppose $f : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^k$ is a neural network with parameter $\theta \in \mathbb{R}^d$. Let U be the *data region*, i.e. a subset of the input space $\mathbb{R}^n$ where the distribution $p(x)$ assigns non-negligible mass. Suppose $V := \mathbb{R}^n \setminus U$ is the remaining subset of the input space $\mathbb{R}^n$ that has low mass under $p(x)$, i.e. it is the *OOD region*. It has recently been shown that any point estimate of F can induce an arbitrarily overconfident prediction on V (Hein et al., 2019; Nguyen et al., 2015). While Bayesian methods fixes this issue in the asymptotic regime (i.e. when the distance

between a test point $x \in V$ and the data region U tends to infinity; see Chapter 3), no such guarantee has been shown for *non-asymptotic* regime which contains outliers that are relatively close to U. In fact, empirical evidence shows that BNNs often yield suboptimal results in this regime, as Fig. 4.13 shows.

In an adjacent field, the frequentist community has proposed a technique—referred to here as ***OOD training***—to address this issue. The core idea is to "expose" the network to a particular kind of OOD data and let it generalize to unseen outliers. Suppose $\mathcal{D}_{\text{in}}$ is a dataset and $\mathcal{D}_{\text{out}}$ is a collection of m_{out} points sampled from some distribution on V. Then, one can incorporate these OOD samples into the standard MAP objective via an additional objective function $\mathcal{L}$ that depends on the network and $\mathcal{D}_{\text{out}}$, but not the dataset $\mathcal{D}$. We thus do the following:

$$\underset{\theta}{\operatorname{argmax}} \log p_{\text{Cat}}(\mathcal{D}|\theta) + \log p(\theta) + \mathcal{L}(\theta; \mathcal{D}_{\text{out}}), \tag{4.10}$$

where $\log p_{\text{Cat}}$ denotes the Categorical log-likelihood, i.e. the negative of the softmax cross-entropy loss and $\log p(\theta)$ is the log-prior. For instance, Hendrycks et al. (2019) define $\mathcal{L}$ to be the negative cross-entropy between the softmax output of f under $\mathcal{D}_{\text{out}}$ and the uniform discrete distribution. Intuitively, (4.10) tries to find a parameter vector of f that induces well-calibrated predictions both inside and outside of U. In particular, ideally, the network should retain the performance of the MAP estimate in U, while attaining the maximum entropy or uniform confidence prediction everywhere in V. Empirically, this frequentist robust training scheme obtains state-of-the-art performance in OOD detection benchmarks (Hendrycks et al., 2019; Meinke & Hein, 2020; Bitterwolf et al., 2020, etc.).

While some works have employed OOD data for tuning the hyperparameters of BNNs Chapter 3, ultimately OOD data are not considered as a first-class citizen in the Bayesian inference itself. Furthermore, while one can argue that theoretically, BNNs can automatically assign high uncertainty over V and thus robust to outliers, as we have previously discussed, empirical evidence suggests otherwise. Altogether, it thus now seems likely that indeed the fact that BNNs are not exposed to OOD data during training is a major factor contributing to the discrepancy in their UQ performance compared to that of the state-of-the-art frequentist methods.

4.3.2 OOD training for Bayesian neural networks

Motivated by the hypothesis laid out in the previous section, our goal here is to come up with an OOD training scheme for standard BNN inference while retaining a reasonable Bayesian interpretation. To this end, we explore four different ways of incorporating OOD data in Bayesian inference by making different assumptions about the data and hence the likelihood, starting from the most philosophically clean to the most heuristic.[5]

Method 1: Extra "None Class"

The most straightforward yet philosophically clean way to incorporate unlabeled OOD data is by adding an extra class, corresponding to the "none class"—also known as the "dustbin" or "garbage class" (Zhang & LeCun, 2017). That is, we redefine our network f as a function

[5] One might be tempted to treat $\mathcal{L}$ in (4.10) as a log-prior. However not only does this mean that we have a (controversial) data-dependent prior, but also it introduces implementation issues, e.g. the KL-divergence term in VB's objective cannot be computed easily anymore.

$\mathbb{R}^n \times \mathbb{R}^{d+\widehat{d}} \to \mathbb{R}^{k+1}$ where $\widehat{d}$ is the number of additional parameters in the last layer associated with the extra class. Note that this is different from the "background class" method (Zhang & LeCun, 2017; Wang & Aitchison, 2021) which assumes that the extra class is tied to the rest of the classes and thus does not have additional parameters. We choose to use the "dustbin class" since Zhang & LeCun (2017) showed that it is the better of the two.

Under this assumption, we only need to label all OOD data in $\mathcal{D}_{\text{out}}$ with the class $k+1$ and add them to the true dataset $\mathcal{D}$. That is, the new dataset is $\widetilde{\mathcal{D}} := \mathcal{D} \amalg \{(x_1^{\text{out}}, k+1), \ldots, (x_{m_{\text{out}}}^{\text{out}}, k+1)\}$, where $\amalg$ denotes disjoint union. Under this setting, we can directly use the Categorical likelihood, and thus, a BNN with this assumption has a sound Bayesian interpretation.

Method 2: Soft Labels

In this method, we simply assume that the data have "soft labels", i.e. the labels are treated as general probability vectors, instead of restricted to integer labels (Thiel, 2008).[6] Thus, we can assume that the target y is a $\Delta^{(k-1)}$-valued random variable. Under this assumption, since one-hot vectors are also elements of $\Delta^{(k-1)}$ (they represent the k corners of $\Delta^{(k-1)}$), we do not have to redefine $\mathcal{D}$ other than to one-hot encode the original integer labels.

Now let us turn our attention to the OOD training data. The fact that these data should be predicted with maximum entropy suggests that the suitable label for any $x^{\text{out}} \in \mathcal{D}_{\text{out}}$ is the uniform probability vector $u := (1/k, \ldots, 1/k)$ of length k—the center of $\Delta^{(k-1)}$. Thus, we can redefine $\mathcal{D}_{\text{out}}$ as the set $\{(x_i^{\text{out}}, u)\}_{i=1}^{m_{\text{out}}}$, and then define a new joint dataset $\widetilde{\mathcal{D}} := \mathcal{D} \amalg \mathcal{D}_{\text{out}}$ containing both the soft-labeled in- and out-distribution training data. Note that without the assumption that y is a simplex-valued random variable, we cannot assign the label u to the OOD training data, and thus we cannot naturally convey our intuition that we should be maximally uncertain over OOD data.

Under the previous assumption, we have to adapt the likelihood. A straightforward choice for simplex-valued random variables is the Dirichlet likelihood $p_{\text{Dir}}(y \mid x, \theta) := \text{Dir}(y \mid \alpha(f(x;\theta)))$ where we have made the dependence of α to the network output $f(x;\theta)$ explicit. So, we obtain the log-likelihood function

$$\begin{aligned} \log p_{\text{Dir}}(y \mid x, \theta) = \log \Gamma(\alpha_0) - \sum_{i=1}^{k} \log \Gamma(\alpha_i(f(x;\theta))) \\ + \sum_{i=1}^{k} (\alpha_i(f(x;\theta)) - 1) \log y_i, \end{aligned} \tag{4.11}$$

where $\alpha_0 := \sum_{i=1}^{k} \alpha_i(f(x;\theta))$ and Γ is the Gamma function. Therefore, the log-likelihood for $\widetilde{\mathcal{D}}$ is given by

$$\log p_{\text{Dir}}(\widetilde{\mathcal{D}} \mid x, \theta) = \sum_{i=1}^{m} \log p_{\text{Dir}}(y^{(i)} \mid x^{(i)}, \theta) + \sum_{i=1}^{m_{\text{out}}} \log p_{\text{Dir}}(u \mid x_i^{\text{out}}, \theta),$$

which can readily be used in Bayesian inference.

[6] The term "soft label" here is different than "fuzzy label" (Kuncheva, 2000; El Gayar et al., 2006) where it is not constrained to sum to one.

One thing left to discuss is the definition of $\alpha(f(x;\theta))$. An option is to decompose it into the mean and precision (Minka, 2000). We do so by writing $\alpha_i(f(x;\theta)) = \gamma \operatorname{softmax}_i(f(x;\theta))$ for each $i = 1,\dots,k$, where γ is the precision (treated as a hyperparameter) and the softmax output $\operatorname{softmax}(f(x;\theta))$ represents the mean—which is valid since it is an element of $\Delta^{(k-1)}$. The benefits are two-fold: First, since we focus solely on the mean, it is easier for optimization (Minka, 2000). Indeed, we found that the alternatives, such as $\alpha_i(f(x;\theta)) = \exp(f_i(x;\theta))$ yield worse results. Second, after training, we can use the softmax output of f as usual without additional steps, i.e. when making a prediction, we can treat the network as if it was trained using the standard softmax-Categorical likelihood.

Method 3: Mixed Labels

There is a technical issue when using the Dirichlet likelihood for the in-distribution data: It is known that the Dirichlet likelihood does not work well with one-hot encoded vectors and that it is harder to optimize than the Categorical likelihood (Malinin & Gales, 2018). To see this, notice in (4.11) that the logarithm is applied to y_i, in contrast to $\operatorname{softmax}_i(f(x;\theta))$ in the Categorical likelihood. If y is a one-hot encoded vector, this implies that for all but one $i \in \{1,\dots,k\}$, the expression $\log y_i$ is undefined and thus the entire log-likelihood also is. While one can mitigate this issue via e.g. label smoothing (Malinin & Gales, 2018; Szegedy et al., 2016), ultimately we found that models with the Dirichlet likelihood generalize worse than their Categorical counterparts (more in Section 4.3.4). Fortunately, the Dirichlet log-likelihood (4.11) does not suffer from this issue when used for OOD data because their label u is the uniform probability vector—in particular, all components of u are strictly larger than zero.

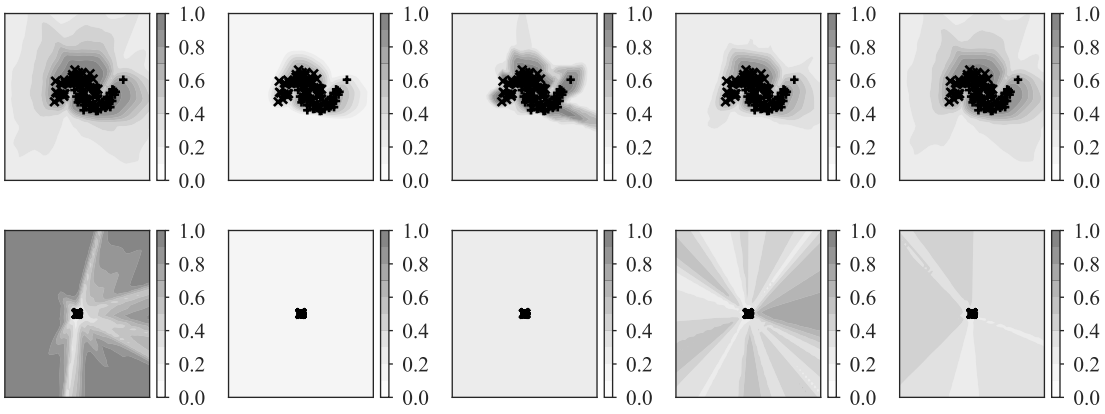
Motivated by these observations, we combine the best of both worlds in the stability of the Categorical likelihood in modeling "hard" one-hot encoded labels (or equivalently, integer labels) and the flexibility of the Dirichlet likelihood in modeling soft labels. To this end, we assume that all the in-distribution data in $\mathcal{D}$ have the standard integer labels, while all the OOD data in $\mathcal{D}_{\text{out}}$ have soft labels. Then, assuming $\widetilde{\mathcal{D}} = \mathcal{D} \sqcup \mathcal{D}_{\text{out}}$, we define the following "mixed" log-likelihood:

$$\log p(\widetilde{\mathcal{D}} \mid \theta) := \sum_{i=1}^{m} \log p_{\text{Cat}}(y^{(i)} \mid x^{(i)}, \theta) + \sum_{i=1}^{m_{\text{out}}} \log p_{\text{Dir}}(u \mid x_i^{\text{out}}, \theta).$$

The implicit assumption of this formulation is that, unlike the two previous methods, we have two distinct generative processes for generating the labels of input points in U and V. Data in $\mathcal{D}$ can thus have a different "data type" than data in $\mathcal{D}_{\text{out}}$. This method can therefore be interpreted as solving a multi-task or multi-modal learning problem.

Method 4: Frequentist-Loss Likelihood

Considering the effectiveness of frequentist methods, it is thus tempting to give a direct Bayesian treatment upon them. But to do so, we first have to find a sound probabilistic justification of $\mathcal{L}$ in (4.10) since not all loss functions can be interpreted as likelihood. We use the OE objective (Hendrycks et al., 2019) as a use case.



(a) Freq. OE **(b)** LA+NC **(c)** LA+SL **(d)** LA+ML **(e)** LA+OE

Figure 4.14: Confidence estimate of the frequentist OE baseline **(a)** and the discussed Bayesian OOD-training methods **(b-e)**. For the BNNs, we use the Laplace approximation (LA). The suffixes "+NC", "+SL", "+ML", and "+OE" refer to the "none class", "soft labels", "mixed labels", and "OE likelihoods" methods, respectively. The bottom row shows the zoomed-out versions of the top one. The OOD training data is sampled from $\mathrm{Unif}[-6, 6]^2$. While OE yields calibrated uncertainty near the training data (a), it is overconfident far from them. Meanwhile, Bayesian methods are less overconfident far away from the data (b-e).

First, recall that OE's OOD objective—the last term in (4.10)—is given by

$$\begin{aligned}\mathcal{L}_{\mathrm{OE}}(\theta; \mathcal{D}_{\mathrm{out}}) &:= -\mathbb{E}_{x^{\mathrm{out}} \sim \mathcal{D}_{\mathrm{out}}}(H(\mathrm{softmax}(f(x^{\mathrm{out}}; \theta)), u)) \\ &= \frac{1}{k\, m_{\mathrm{out}}} \sum_{i=1}^{m_{\mathrm{out}}} \sum_{j=1}^{k} \log \mathrm{softmax}_j(f(x_i^{\mathrm{out}}; \theta)),\end{aligned} \tag{4.12}$$

where $u := (1/k, \ldots, 1/k) \in \Delta^{(k-1)}$ is the uniform probability vector of length k and H is the functional measuring the *cross-entropy* between its two arguments. Our goal here is to interpret (4.12) as a log-likelihood function: we aim at finding a log-likelihood function $\log p(\mathcal{D}_{\mathrm{out}} \mid \theta)$ over $\mathcal{D}_{\mathrm{out}}$ that has the form of $\mathcal{L}_{\mathrm{OE}}$. This is sufficient for defining the overall likelihood over $\mathcal{D}$ and $\mathcal{D}_{\mathrm{out}}$ since given this function and assuming that these datasets are independent, we readily have a probabilistic interpretation of the log-likelihood terms in (4.10): $\log p(\mathcal{D}, \mathcal{D}_{\mathrm{out}} \mid \theta) = \log p_{\mathrm{Cat}}(\mathcal{D} \mid \theta) + \log p(\mathcal{D}_{\mathrm{out}} \mid \theta)$.

We begin with the assumption that the Categorical likelihood is used to model both the in- and out-of-distribution data—in particular, we use the standard integer labels for both $\mathcal{D}$ and $\mathcal{D}_{\mathrm{out}}$. Now, recall that the OOD data ideally have the uniform confidence, that is, they are equally likely under all possible labels. But since we have assumed hard labels, we cannot use u directly as the label for $x^{\mathrm{out}} \in \mathcal{D}_{\mathrm{out}}$. To circumvent this, we redefine the OOD dataset $\mathcal{D}_{\mathrm{out}}$ by assigning all k possible labels to each x^{out}, based on the intuition that f should be "maximally confused" about the correct label of x^{out}.

$$\mathcal{D}_{\mathrm{out}} := \{(x_1^{\mathrm{out}}, 1), \ldots, (x_1^{\mathrm{out}}, k), \ldots, (x_{m_{\mathrm{out}}}^{\mathrm{out}}, 1), \ldots, (x_{m_{\mathrm{out}}}^{\mathrm{out}}, k)\}. \tag{4.13}$$

Thus, given m_{out} unlabeled OOD data, we have $|\mathcal{D}_{\text{out}}| = k\, m_{\text{out}}$ OOD data points in our OOD training set. So, the negative log-Categorical likelihood over $\mathcal{D}_{\text{out}}$ is given by

$$-\log p(\mathcal{D}_{\text{out}} \mid \theta) = \sum_{i=1}^{m_{\text{out}}} \sum_{j=1}^{k} \log \text{softmax}_j(f(x_i^{\text{out}}; \theta)). \tag{4.14}$$

Comparing this to (4.12), we identify that $\log p(\mathcal{D}_{\text{out}} \mid \theta)$ is exactly $\mathcal{L}_{\text{OE}}$, up to a constant factor $1/(k\, m_{\text{out}})$, which can be thought of as a tempering factor to $p(\mathcal{D}_{\text{out}} \mid \theta)$. We have thus obtained the probabilistic interpretation of OE's objective—this likelihood can then be soundly used in a Bayesian inference—albeit arising from applying a heuristic (4.13) to the data.

Remark Here, we consider the question of whether there is an inherent advantage of using OOD-trained BNNs instead of the standard OOD-trained network. One answer to this question is given by the recent finding that Bayesian methods naturally yield low uncertainty in regions far away from the data Chapter 3. In contrast, OOD-trained point-estimated networks do not enjoy such a guarantee by default and must resort to e.g. generative models (Meinke & Hein, 2020). We illustrate this observation synthetically in Fig. 4.14.

4.3.3 Related Work

OOD training for BNNs has recently been used for tuning the hyperparameters of LAs Chapter 3 and Section 4.2. However, it appears that OOD training is not commonly utilized by BNNs in the Bayesian inference itself. Wang & Aitchison (2021), at the same time window as this work, also proposed OOD training for BNNs by justifying the presence of OOD training data as a consequence of the data curation process. Nevertheless, their method is different than all the methods proposed here and only validated on a single BNN. Meanwhile, we explore four distinct methods and extensively validate them on various BNNs, see Section 4.3.4.

From an adjacent field, adversarial training for BNNs has recently been studied. In particular, Liu et al. (2019) specifically employ VB and modify the first term of the ELBO to take into account the worst-case perturbation of each data point, which can be thought of as a particular type of OOD data. Unlike theirs, our methods are for general OOD data and are agnostic to the approximate inference method.

Non-Bayesian Dirichlet-based models have recently been studied for UQ (Sensoy et al., 2018). Similar to our proposed "soft labels" and "mixed labels" likelihoods, Malinin & Gales (2018; 2019); Nandy et al. (2020) use the Dirichlet distribution as the output of a non-Bayesian network and employ OOD training via a custom, non-standard loss. Their methods' Bayesian interpretation is therefore unclear. In contrast, for modeling soft labels, we simply use the standard Dirichlet log-likelihood function, which is well-studied in the context of generalized linear models (Gueorguieva et al., 2008). Our methods thus retain a clear Bayesian interpretation when used in BNNs.

4.3.4 Experiments

Baselines We use the following strong, recent baselines to represent non-Bayesian methods:[7] (i) standard MAP-trained network (**MAP**), (ii) Deep Ensemble (**DE**, Lakshminarayanan et al.,

[7] Deep Ensemble can also be seen as a Bayesian method, but it was originally proposed as a frequentist method.

Table 4.8: Test accuracy (↑) / ECE (↓), averaged over five prediction runs. Best values in each categories (separated by horizontal lines) are in bold.

	MNIST	**F-MNIST**	**SVHN**	**CIFAR-10**	**CIFAR-100**
MAP	99.4±0.0 / 6.4±0.0	92.4±0.0 / 13.9±0.0	97.4±0.0 / 8.9±0.0	94.8±0.0 / 10.0±0.0	76.7±0.0 / 14.3±0.0
DE	**99.5**±0.0 / 8.6±0.0	**93.6**±0.0 / 3.6±0.0	**97.6**±0.0 / **3.5**±0.0	**95.7**±0.0 / **4.5**±0.0	**80.0**±0.0 / **1.9**±0.0
OE	99.4±0.0 / **5.3**±0.0	92.3±0.0 / 12.1±0.0	97.4±0.0 / 10.6±0.0	94.6±0.0 / 13.2±0.0	76.7±0.0 / 15.0±0.0
VB	**99.5**±0.0 / 11.2±0.3	92.4±0.0 / 3.7±0.2	97.5±0.0 / 5.7±0.2	94.9±0.0 / 5.8±0.2	**75.4**±0.0 / **8.3**±0.0
+NC	99.4±0.0 / 12.6±0.3	92.2±0.0 / 3.3±0.1	97.5±0.0 / **4.1**±0.1	94.4±0.0 / 5.5±0.1	74.1±0.0 / 10.7±0.1
+SL	**99.5**±0.0 / 10.5±0.3	**93.1**±0.0 / 6.3±0.1	**97.6**±0.0 / 9.3±0.2	93.0±0.0 / 11.0±0.1	71.4±0.0 / 13.0±0.0
+ML	99.3±0.0 / 11.8±0.2	92.0±0.0 / **2.5**±0.1	**97.6**±0.0 / 4.2±0.0	**95.0**±0.0 / 4.9±0.2	**75.4**±0.0 / 10.4±0.0
+OE	99.4±0.0 / **10.0**±0.2	92.3±0.0 / 3.0±0.2	**97.6**±0.0 / 5.7±0.2	94.8±0.0 / **4.6**±0.2	74.2±0.0 / 8.9±0.0
LA	99.4±0.0 / 7.6±0.1	92.5±0.0 / 11.3±0.2	97.4±0.0 / 3.3±0.3	**94.8**±0.0 / 7.5±0.3	76.6±0.1 / 8.3±0.1
+NC	99.4±0.0 / 5.4±0.7	92.4±0.0 / 8.5±0.3	97.3±0.0 / 4.6±0.2	94.0±0.0 / **6.6**±0.3	76.2±0.0 / 6.1±0.0
+SL	**99.7**±0.0 / 12.1±1.1	**93.2**±0.0 / **3.2**±0.3	**97.5**±0.0 / 7.4±0.2	93.6±0.0 / 10.2±0.2	72.3±0.1 / 7.1±0.2
+ML	99.4±0.0 / 7.5±1.0	92.5±0.0 / 5.9±0.2	97.4±0.0 / **2.9**±0.2	**94.8**±0.0 / 6.9±0.3	76.5±0.1 / **4.4**±0.1
+OE	99.4±0.0 / **4.8**±0.7	92.3±0.0 / 7.4±0.1	97.4±0.0 / 3.2±0.1	94.6±0.0 / 8.8±0.1	**76.7**±0.1 / **4.4**±0.1

2017), and (iii) Outlier Exposure (**OE**, Hendrycks et al., 2019). Note that DE and OE are among the established state-of-the-art frequentist UQ methods.

For standard Bayesian methods, i.e. those considering only $\mathcal{D}$ in the inference, we use (iv) the all-layer diagonal Laplace approximation on top of the MAP network (**LA**) and (v) the last-layer mean-field VB (**VB**, Graves, 2011; Blundell et al., 2015). We mainly use only these simple Bayesian methods to validate that the proposed likelihood could make even these crudely approximated BNNs competitive to the strong baselines. Results with more advanced BNNs are in Table 4.12.

To represent our methods, we again use the same LA and VB but with the modifications proposed in Section 4.3.2. We denote these modified methods **LA+X** and **VB+X**, respectively. Here, **X** is the abbreviation for the proposed methods, i.e. **NC** for "none class" (Method 1), **SL** for "soft label" (Method 2), **ML** for "mixed label" (Method 3), and **OE** for "OE likelihood" (Method 4).

Finally, we use the LeNet and WideResNet-16-4 architectures, trained in the usual manner—see Appendix G.2. Source code is available at `https://github.com/wiseodd/bayesian_ood_training`.

Datasets As the in-distribution datasets, we use: (i) MNIST, (ii) Fashion-MNIST (F-MNIST), (iii) SVHN, (iv) CIFAR-10, and (v) CIFAR-100. For each of them, we obtain a validation set of size 2000 by randomly splitting the test set. For methods requiring OOD training data, i.e. OE, LA+X, and VB+X, we use the 32×32 downsampled ImageNet dataset (Chrabaszcz et al., 2017) as an alternative to the 80M Tiny Images dataset used by Hendrycks et al. (2019); Meinke & Hein (2020), since the latter is not available anymore.[8] For OOD detection tasks, we use various *unseen* (i.e. not used for training or tuning) OOD test sets as used in (Meinke & Hein, 2020; Hein et al., 2019), both real-world (e.g. E-MNIST) and synthetic (e.g. uniform noise). For text classification, we use the Stanford Sentiment Treebank (SST, Socher et al., 2013) and the TREC dataset (Voorhees, 2001). We detail of all OOD test sets in Appendix G.1. We furthermore test the methods in a dataset-shift robustness task using the corrupted CIFAR-10 (CIFAR-10-C) dataset (Ovadia et al., 2019; Hendrycks & Dietterich, 2019).

[8] `https://groups.csail.mit.edu/vision/TinyImages/`.

Table 4.9: OOD data detection in terms of FPR95. Values are averages over six OOD test sets and five prediction runs—lower is better. The best values of each category are in bold. Details are in Table G.1 in the appendix.

Methods	MNIST	F-MNIST	SVHN	CIFAR-10	CIFAR-100
MAP	17.7	69.4	22.4	52.4	81.0
DE	10.6	61.4	10.1	32.3	73.3
OE	**5.4**	**16.2**	**2.1**	**22.8**	**54.0**
VB	25.7	63.3	22.0	36.5	77.6
+NC	7.5	15.0	**1.4**	**28.0**	**49.9**
+SL	**2.7**	**4.2**	1.8	40.4	62.3
+ML	7.4	19.6	**1.4**	29.1	50.2
+OE	6.8	22.4	1.5	29.8	53.3
LA	19.4	68.7	17.1	53.6	81.3
+NC	6.6	8.3	1.5	**20.1**	**47.4**
+SL	**2.2**	**4.1**	**1.0**	38.5	60.9
+ML	5.5	14.3	1.1	21.8	52.5
+OE	5.4	17.0	1.1	23.3	53.9

Metrics To measure OOD detection performance, we use the standard FPR95 metric, which measures the false positive rate at 95% true positive rate. Other metrics such as average confidence and area under the ROC curve are presented in the appendix. Meanwhile, to measure dataset-shift robustness and predictive performance, we use test accuracy and expected calibration error (ECE) with 15 bins (Naeini et al., 2015).

4.3.4.1 Generalization and Calibration

We present the generalization and calibration performance in Table 4.8. We note that generally, all methods discussed in Section 4.3.2 attain comparable accuracy to, and are better calibrated than the vanilla MAP/OE models. However, the "soft label" method tends to underperform in both accuracy and ECE—this can be seen clearly on CIFAR-100. This issue appears to be because of the numerical issue we have discussed in Section 4.3.2. Note that this issue seems to also plague other Dirichlet-based methods (Malinin & Gales, 2018; 2019). Overall, it appears that Bayesian OOD training with NC, ML, and OE is not harmful to the in-distribution performance—they are even more calibrated than the frequentist OE.

4.3.4.2 OOD Detection

We present the OOD detection results on image classification datasets in Table 4.9.[9] As indicated in Fig. 4.13, OE is in general significantly better than even DE while retaining the computational efficiency of MAP. The vanilla Bayesian baselines, represented by VB and LA, achieve worse results than DE (and thus OE). But, when OOD training is employed to train these BNNs using the four methods we considered in Section 4.3.2, their performance improves. We observe that all Bayesian OOD training methods generally yield better results than DE and become competitive to OE. In particular, while the "soft label" method (SL) is best for "easy" datasets

[9]Refer to Appendix G.3 for the detailed, non-averaged results for Tables 4.9 to 4.12, along with additional metrics.

Table 4.10: OOD data detection on text classification tasks. Values are averages over five prediction runs and additionally, three OOD test sets for FPR95. Details are in the appendix (Tables G.7 and G.8).

	ECE		FPR95	
Methods	**SST**	**TREC**	**SST**	**TREC**
MAP	20.8	17.2	100.0	96.3
DE	**2.5**	10.6	100.0	24.2
OE	13.0	**9.4**	**0.0**	**0.0**
LA	21.0	17.3	100.0	96.4
+NC	17.9	18.6	**0.0**	**0.0**
+SL	17.5	10.4	95.3	0.8
+ML	**11.4**	11.5	84.6	**0.0**
+OE	12.8	**8.4**	**0.0**	**0.0**

Table 4.11: Average ECE and FPR95 under models trained with synthetic noises as $\mathcal{D}_{\text{out}}$. Both values are averaged over five prediction runs, and additionally six OOD test sets for FPR95. See Tables G.5 and G.6 in the appendix for details.

	ECE			FPR95		
Methods	**SVHN**	**CIFAR-10**	**CIFAR-100**	**SVHN**	**CIFAR-10**	**CIFAR-100**
MAP	**8.9**	**10.0**	**14.3**	22.4	52.4	81.0
OE	**8.9**	11.5	16.1	**11.4**	**31.0**	**60.1**
LA	**3.3**	7.5	8.3	17.1	53.6	81.3
+NC	5.0	8.3	**3.8**	10.5	**26.4**	64.5
+SL	13.5	16.0	4.0	93.7	37.9	68.6
+ML	7.4	**7.2**	3.3	14.4	28.4	61.0
+OE	3.8	**7.2**	8.6	**10.1**	35.3	**56.4**

(MNIST, F-MNIST), we found that the simplest "none class" method (NC) achieves the best results in general.

In Table 4.10, we additionally show the results on text classification datasets. We found that the OOD training methods consistently improve both the calibration and OOD-detection performance of the vanilla Bayesian methods, making them on par with OE. As before, the "none class" method performs well in OOD detection. This is a reassuring result since NC is also the most philosophically clean (i.e. requires fewer heuristics) than the other three methods considered.

A common concern regarding OOD training is the choice of $\mathcal{D}_{\text{out}}$. As an attempt to address this, in Table 4.11 we provide results on OOD detection when the model is trained using a synthetic noise dataset. The noise dataset used here is the "smooth noise" dataset (Hein et al., 2019), obtained by permuting, blurring, and contrast-rescaling the original training dataset. We found that even with such a simple OOD dataset, we can still generally obtain better OOD detection results than OE, as shown by the FPR95 values. Moreover, the combination of Bayesian formalism and OOD training is beneficial in calibrating the in-distribution uncertainty: We found that using this $\mathcal{D}_{\text{out}}$ to train OE yields worse-calibrated results than even the vanilla MAP model, as the ECE values show. In contrast, OOD-trained LA yields better ECE results in general.

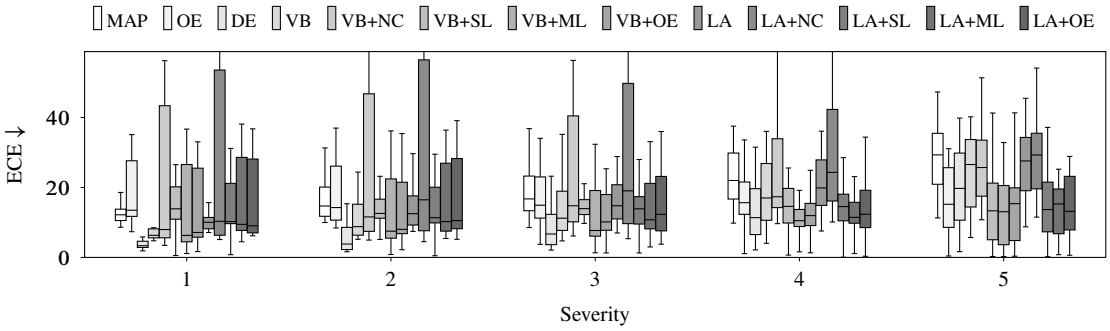


Figure 4.15: Dataset shift performance on the corrupted-CIFAR-10-C dataset, following Ovadia et al. (2019). Values are ECE—lower is better.

Finally, we show that OOD training is beneficial for other BNNs. In Table 4.12, we consider two recent (all-layer) BNNs: a VB with the flipout estimator (Flipout, Wen et al., 2018) and the cyclical stochastic-gradient Hamiltonian Monte Carlo (CSGHMC, Zhang et al., 2020). Evidently, OOD training improves their OOD detection performance by a large margin. Moreover, OOD training also improves the performance of DE.

4.3.4.3 Dataset-Shift Robustness

In this UQ task, OOD training is beneficial for both MAP and the vanilla Bayesian methods (VB, LA), making them competitive to the state-of-the-art DE's performance in larger severity levels, see Fig. 4.15. Moreover, the OOD-trained VB and LA are in general more calibrated than OE, which shows the benefit of the Bayesian formalism vis-à-vis the point-estimated OE. This indicates that *both* being Bayesian and considering OOD data during training are beneficial.

Table 4.12: Average ECE and FPR95 with more sophisticated base models. "C-10" stands for CIFAR-10, while "C-100" for CIFAR-100

	ECE		**FPR95**	
Methods	**C-10**	**C-100**	**C-10**	**C-100**
Flipout	10.9	19.8	65.0	85.4
+NC	**8.2**	**13.8**	**40.9**	**56.2**
CSGHMC	**1.7**	4.0	60.3	81.0
+NC	6.2	**2.4**	**25.0**	**43.0**
DE	**4.5**	1.9	32.3	73.3
+NC	4.8	**1.7**	**17.0**	**44.4**

Even though it is the best in OOD detection, here we observe that NC is less calibrated in terms of ECE than its counterparts. This might be due to the incompatibility of calibration metrics with the additional class: When the data are corrupted, they become closer to the OOD data, and thus NC tends to assign higher probability mass to the last class which does not correspond to any of the true classes (contrast this to other the approaches). Therefore, in this case, the confidence over the true class becomes necessarily lower—more so than the other approaches. Considering that calibration metrics depend on the confidence of the true class, the calibration of NC thus suffers. One way to overcome this issue is to make calibration metrics aware of the "none class", e.g. by measuring calibration only on data that have low "none class" probability. We leave the investigation for future research.

4.3.4.4 Costs

The additional costs associated with all the OOD-training methods presented here are negligible: Like other non-Bayesian OOD-training methods, the only overhead is the additional minibatch of OOD training data at each training iteration—the costs are similar to when considering a standard training procedure with double the minibatch size. Additionally for LA, in its Hessian computation, one effectively computes it with twice the number of the original data. However, this only needs to be done *once* post-training.

Chapter 5
Conclusion

Uncertainty estimation and calibration are important functionalities for safe predictive systems, including neural networks. Yet, despite their ubiquitousness, they generally yield badly calibrated, overconfident predictions. This thesis offered several low-cost Bayesian solutions for improving the predictive uncertainty quantification performance of neural networks.

Chapter 2 showed that it is now easier than ever to apply the Bayesian paradigm in deep neural networks, thanks to recent advances in Laplace approximations. This is facilitated further by the simple, easy-to-use `laplace-torch` library.

Chapter 3 provided an application of Laplace approximations (and Gaussian-based posterior approximations in general) for mitigating overconfidence in ReLU networks that occur far away from the training data. One can show that even a simple Laplace approximation applied only at the last layer of a deep network helps. Furthermore, the latter half of Chapter 3 shows that one can completely fix the asymptotic overconfidence problem by essentially adding infinitely many additional ReLU features to the network, which can be done efficiently thanks to its Gaussian process formulation.

Finally, Chapter 4 proposed low-cost methods that can be applied on top of Laplace approximations. First, one can improve the weight-space approximation quality of a pre-trained Laplace approximation by refining it with simple normalizing flows. While cheap, the resulting refined posterior can be competitive with the more expensive Hamiltonian Monte Carlo method. Second, one can also train *only* the predictive uncertainty of a Laplace approximation by introducing further degrees of freedom to the original parameter space, which in turn gives the possibility to find a point estimate that induces the same predictive mean, but with more a desirable local geometry around it. Last but not least, Bayesian neural network classifiers can be improved further by including outliers during training. This can be done by simply assuming an extra class to the underlying network. All these methods were able to improve the calibration and out-of-distribution data detection of standard Laplace approximations.

The fact that these simple, low-cost methods can mitigate overconfidence in deep networks and improve Bayesian neural networks further is encouraging: It gives us more confidence in our journey towards safe and reliable, yet high-performing predictive systems.

Appendix A
Derivations

A.1 The Probit Approximation

The ***probit function*** Φ is the cumulative distribution function of the standard Gaussian distribution $\mathcal{N}(x \mid 0, 1)$ on $\mathbb{R}$, i.e., $\Phi(z) := \int_{-\infty}^{z} \mathcal{N}(x \mid 0, 1)\, dx$. It can conveniently be written in terms of the ***error function***

$$\operatorname{erf}(z) := \frac{2}{\sqrt{\pi}} \int_0^z \exp(-x^2)\, dx \tag{A.1}$$

by

$$\Phi(z) = \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{z}{\sqrt{2}}\right)\right). \tag{A.2}$$

Proposition A.1 (The Probit Integral). *If $\mathcal{N}(x \mid \mu, \sigma^2)$ be a Gaussian on $\mathbb{R}$ and $a, b \in \mathbb{R}$ then*

$$\int_{\mathbb{R}} \Phi(ax + b)\, \mathcal{N}(x \mid \mu, \sigma^2)\, dx = \Phi\left(\frac{a\mu + b}{\sqrt{1 + a^2\sigma^2}}\right). \tag{A.3}$$

Proof. The standard property of the error function says that

$$\int_{\mathbb{R}} \operatorname{erf}(ax + b)\, \mathcal{N}(x \mid \mu, \sigma^2)\, dx = \operatorname{erf}\left(\frac{a\mu + b}{\sqrt{1 + 2a^2\sigma^2}}\right).$$

So,

$$\begin{aligned}
&\int_{\mathbb{R}} \left(\frac{1}{2} + \frac{1}{2}\operatorname{erf}\left(\frac{ax + b}{\sqrt{2}}\right)\right) \mathcal{N}(x \mid \mu, \sigma^2)\, dx \\
&\quad = \frac{1}{2} + \frac{1}{2}\int_{\mathbb{R}} \operatorname{erf}\left(\left(\frac{a}{\sqrt{2}}\right)x + \left(\frac{b}{\sqrt{2}}\right)\right) \mathcal{N}(x \mid \mu, \sigma^2)\, dx \\
&\quad = \frac{1}{2} + \frac{1}{2}\operatorname{erf}\left(\frac{(a\mu + b)/\sqrt{2}}{\sqrt{1 + 2(a/\sqrt{2})^2\sigma^2}}\right) \\
&\quad = \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{a\mu + b}{\sqrt{2}\sqrt{1 + a^2\sigma^2}}\right)\right) \\
&\quad = \Phi\left(\frac{a\mu + b}{\sqrt{1 + a^2\sigma^2}}\right),
\end{aligned}$$

and the proof is complete. □

This integral is very useful for Bayesian inference since it enables us to approximate the following integral that is ubiquitous in Bayesian binary classifications

$$\int_{\mathbb{R}} \sigma(z)\,\mathcal{N}(z \mid m, s^2)\,dz,$$

where $\sigma(z) := 1/(1+\exp(-z))$ is the logistic function.

The key idea is to notice that the probit and logistic function are both *sigmoid* functions. That is, their graphs have a similar "S-shape". Moreover, their images are both $[0, 1]$. However, they are a bit different—the probit function is more "horizontally stretched" compared to the logistic function.

So, the strategy to approximate the integral above is as follows: (i) horizontally "contract" the probit function and then (ii) use Proposition 3 to get an analytic approximation to the integral. For the first step, this can be done by a simple change of coordinate: stretch the domain of the probit function with a constant λ, i.e., $z \mapsto \lambda z$. There are several "good" values for λ, but commonly it is chosen to be $\lambda = \sqrt{\pi/8}$, which makes the probit function have the same derivative as the logistic function at zero. That is, we have the approximation $\sigma(z) \approx \Phi(\lambda z) = \Phi(\sqrt{\pi/8}\,z)$.

Corollary A.2. *If $\mathcal{N}(z \mid m, s^2)$ is a Gaussian on $\mathbb{R}$, then*

$$\int_{\mathbb{R}} \Phi(\lambda z)\,\mathcal{N}(z \mid m, s^2)\,dz = \Phi\left(\frac{m}{\sqrt{\lambda^{-2}+s^2}}\right).$$

Proof. By Proposition 3, we have

$$\begin{aligned}\int_{\mathbb{R}} \Phi(\lambda\, z)\,\mathcal{N}(z \mid m, s^2)\,dz &= \Phi\left(\frac{\lambda\mu}{\sqrt{1+\lambda^2 s^2}}\right)\\ &= \Phi\left(\frac{\chi\mu}{\chi\sqrt{\lambda^{-2}+s^2}}\right).\end{aligned}$$

We have thus arrived at the claim. □

Now we are ready to obtain the probit approximation.

Proposition A.3 (The Probit Approximation). *If $\mathcal{N}(z \mid m, s^2)$ is a Gaussian on $\mathbb{R}$ and $\sigma(z) \approx \Phi\left(\sqrt{\pi/8}\,z\right)$, then*

$$\int_{\mathbb{R}} \sigma(z)\,\mathcal{N}(z \mid m, s^2)\,dz \approx \sigma\left(\frac{m}{\sqrt{1+\pi/8\,s^2}}\right).$$

Proof. Let $\lambda = \sqrt{\pi/8}$. Using Corollary 4 and substituting $\Phi(z) \approx \sigma\left(\lambda^{-1}\,z\right)$:

$$\int_{\mathbb{R}} \sigma(z)\,\mathcal{N}(z \mid m, s^2)\,dz \approx \Phi\left(\frac{m}{\sqrt{\lambda^{-2}+s^2}}\right)$$

$$= \sigma\left(\frac{\lambda^{-1} m}{\sqrt{\lambda^{-2} + s^2}}\right)$$
$$= \sigma\left(\frac{\cancel{\lambda^{-1}} m}{\cancel{\lambda^{-1}} \sqrt{1 + \lambda^2 s^2}}\right).$$

Substituting $\lambda^2 = \pi/8$ into the last equation yields the desired result. □

A.2 The Multiclass Probit Approximation

Let $p(z) := \mathcal{N}(z \mid \mu, \Sigma)$ be the distribution a random variable $z \in \mathbb{R}^c$. The multi-class probit approximation (MPA) is defined by

$$\int_{\mathbb{R}^c} \text{softmax}(z)\, p(z)\, dz \approx \text{softmax}\left(\frac{\mu}{\sqrt{1 + \pi/8 \,\text{diag}(\Sigma)}}\right),$$

where the vector division is defined component-wise. Its derivation, based on Lu et al. (2020) is as follows.

Notice that we can write the i-th component of softmax(z) as $1/(1 + \sum_{j \neq i} \exp(-(z_i - z_j)))$. So, for each $i = 1, \ldots, c$, using $z_{ij} := z_i - z_j$, we can write

$$\frac{1}{1 + \sum_{j \neq i} \exp(-z_{ij})} = \frac{1}{1 - (K-1) + \sum_{j \neq i} \frac{1}{1 + \exp(-z_{ij})}}$$
$$= \frac{1}{2 - K + \sum_{j \neq i} \frac{1}{\sigma(z_{ij})}}.$$

Then, we use the following approximations:

1. $\mathbb{E}(f(x)) \approx f(\mathbb{E}(x))$ for a non-zero function f,
2. the mean-field approximation $\mathcal{N}(z \mid \mu, \Sigma) \approx \mathcal{N}(z \mid \mu, \text{diag}(\Sigma))$, and thus we have $z_{ij} := z_i - z_j \sim \mathcal{N}(z_{ij} \mid \mu_i - \mu_j, \Sigma_{ii} + \Sigma_{jj})$, and
3. using the (binary) probit approximation (see Appendix A.1), with a further approximation:

$$\int_{\mathbb{R}} \sigma(z_{ij})\, \mathcal{N}(z_{ij} \mid \mu_i - \mu_j, \Sigma_{ii} + \Sigma_{jj})\, dz_{ij} \approx \sigma\left(\frac{\mu_i - \mu_j}{\sqrt{1 + \pi/8\, \Sigma_{ii} + \Sigma_{jj}}}\right)$$
$$\approx \sigma\left(\frac{\mu_i}{\sqrt{1 + \pi/8\, \Sigma_{ii}}} - \frac{\mu_j}{\sqrt{1 + \pi/8\, \Sigma_{jj}}}\right),$$

we obtain

$$\begin{aligned}
\int_{\mathbb{R}^c} \operatorname{softmax}_i(z)\,\mathcal{N}(z \mid \mu, \Sigma) &\approx \frac{1}{2 - K + \sum_{j \neq i} \frac{1}{\mathbb{E}\sigma(z_{ij})}} \\
&\approx \frac{1}{2 - K + \sum_{j \neq i} \frac{1}{\sigma\left(\frac{\mu_i}{\sqrt{1+\pi/8\,\Sigma_{ii}}} - \frac{\mu_j}{\sqrt{1+\pi/8\,\Sigma_{jj}}}\right)}} \\
&= \frac{1}{1 + \sum_{j \neq i} \exp\left(-\left(\frac{\mu_i}{\sqrt{1+\pi/8\,\Sigma_{ii}}} - \frac{\mu_j}{\sqrt{1+\pi/8\,\Sigma_{jj}}}\right)\right)} \\
&= \frac{\exp\left(\mu_i / \sqrt{1 + \pi/8\,\Sigma_{ii}}\right)}{\sum_{j=1}^{k} \exp\left(\mu_j / \sqrt{1 + \pi/8\,\Sigma_{jj}}\right)}
\end{aligned}$$

We identify the last equation above as the i-th component of $\operatorname{softmax}\left(\frac{\mu}{\sqrt{1+\pi/8\,\operatorname{diag}(\Sigma)}}\right)$.

Appendix B
Appendix of Chapter 2

B.1 Components of Laplace-Approximated Neural Networks

Recall from Section 1.2.1 and Section 1.3.1 that the goal of a Laplace approximation on a BNN f_θ is to approximate the intractable posterior

$$p(\theta \mid \mathcal{D}) = \frac{1}{\int p(\mathcal{D} \mid \theta) p(\theta)\, d\theta}\, p(\mathcal{D} \mid \theta) p(\theta) =: \frac{1}{Z} h(\theta; \mathcal{D}),$$

given a dataset $\mathcal{D} := \{(x_i, y_i)\}_{i=1}^m$, where $p(\mathcal{D} \mid \theta) := \prod_{i=1}^m p(y_i \mid f_\theta(x_i))$ and $p(\theta)$ are a likelihood and a prior, respectively. This is done by first finding the mode θ_{MAP} and then "surrounding" it with a Gaussian with covariance equals the inverse of the Hessian $\Lambda := \nabla^2_\theta \mathcal{L}(\theta; \mathcal{D})|_{\theta_{\text{MAP}}}$, where $\mathcal{L}(\theta; \mathcal{D}) := -\log h(\theta; \mathcal{D})$. Generally, any prior with twice differentiable log-density can be used. But, due to the popularity of the weight decay regularizer (Section 1.2.1), we assume that the prior is a zero-mean Gaussian $p(\theta) = \mathcal{N}(\theta \mid 0, \gamma^2 I)$ unless stated otherwise.[1] The Hessian $\nabla^2_\theta \mathcal{L}(\mathcal{D}; \theta)|_{\theta_{\text{MAP}}}$ then depends both on the (simple) log-prior and the (complicated) log-likelihood:

$$\Lambda = \nabla^2_\theta \mathcal{L}(\mathcal{D}; \theta)|_{\theta_{\text{MAP}}} = -\gamma^{-2} I_d - \sum_{i=1}^{n} \nabla^2_\theta \log p(y_i \mid f_\theta(x_i))|_{\theta_{\text{MAP}}}. \tag{B.1}$$

A naive implementation of the Hessian is infeasible because the second term in (B.1) scales quadratically with the number of network parameters, which can be in the millions (He et al., 2016). In recent years, several works have addressed scalability, as well as other factors that affect approximation quality and predictive performance of the LA. In the following, we identify, review, and discuss three key components that allow LAs to scale and perform well on modern deep architectures. See Fig. 2.2 for an overview—note that the fourth component is discussed in Section 1.3.4.

B.1.1 Subset of Weights

The naïve impelementation of a LA on all weights of a deep network is often intractable. Fortunately, recent works (Kristiadi et al., 2020; Daxberger et al., 2021b, see also Section 2.3.2) show that the LA can be just as effective when applied only on a subset of the network's weights.

[1] One can also consider a per-layer or even per-parameter weight decay, which corresponds to a more general, but still comparably simple Gaussian prior. In particular, the Hessian of this prior is still diagonal and constant.

B.1.1.1 Last-Layer Laplace

Let $f_\theta : \mathbb{R}^n \to \mathbb{R}^k$ be an L-layer NN, and assume that the first $L-1$ layers of f_θ is a feature map, denoted by $\phi : \mathbb{R}^n \to \mathbb{R}^{n_{L-1}}$ where $\phi := f^{(L-1)}_{\theta^{(L-1)}} \circ \cdots \circ f^{(1)}_{\theta^{(1)}}$. Given MAP-trained parameters θ_{MAP}, we fix $\theta^{(1)}, \ldots, \theta^{(L-1)}$ to their MAP-estimated values. This yields a linear model

$$f_{\theta^{(L)}}(x) = W^{(L)}\phi(x) + b^{(L)} \tag{B.2}$$

where now $\theta^{(L)}$ the only parameter of this model. Note that, this model can also be seen as a linear model with nonlinear feature extractor ϕ.

Under the previous assumption, we can perform a Laplace approximation:

$$p(\theta^{(L)} \mid \mathcal{D}) \approx \mathcal{N}(\theta^{(L)} \mid \theta^{(L)}_{\text{MAP}}, \Sigma^{(L)}), \tag{B.3}$$

where the covariance matrix $\Sigma^{(L)}$ in this case is generally much smaller than the standard Laplace approximation on the whole θ. See Appendix B.1.2 for the detail on how to compute $\Sigma^{(L)}$.

B.1.1.2 Subnetwork Laplace

The last-layer Laplace can be generalized by considering a Laplace approximation of an arbitrary subset of the network's parameter. This is motivated by recent findings that neural networks can be heavily pruned without sacrificing test accuracy (Frankle & Carbin, 2019), and that in the neighborhood of a local optimum, there are many directions that leave the predictions unchanged (Maddox et al., 2020).

Let $\theta^{(s)}$ be an arbitrary subset of θ and let $\theta^{(r)} := \theta \setminus \theta^{(s)}$. Given a pre-trained parameter θ_{MAP}, we fix $\theta^{(r)}$ to its MAP-estimated value $\theta^{(r)}_{\text{MAP}}$ and perform a Laplace approximation only on $\theta^{(s)}$. In other words, we have the following posterior approximation over the whole θ:

$$p(\theta \mid \mathcal{D}) \approx \mathcal{N}(\theta^{(s)} \mid \theta^{(s)}_{\text{MAP}}, \Sigma^{(s)}) \prod_{\hat{\theta} \in \theta^{(r)}} \delta(\hat{\theta} - \hat{\theta}_{\text{MAP}}) =: q_s(\theta), \tag{B.4}$$

where $\delta(z-a)$ denotes the Dirac delta function centered at a.

The importance of this construction is that the size of the subset $\theta^{(s)}$—i.e. the subnetwork size—can be considered as a hyperparameter that can be controlled. Typically, $\theta^{(s)}$ will be set such that its size is much smaller than the d, the size of θ. In particular,$\theta^{(s)}$ can be set such that it is tractable to compute and store the full covariance matrix over the subnetwork. This allows us to capture rich dependencies across the weights within the subnetwork. However, in principle one could also employ one of the (less expressive) factorizations of the Hessian described in Appendix B.1.2.

Daxberger et al. (2021b) propose to choose the subnetwork such that the subnetwork posterior in (B.4) is as close as possible (w.r.t. some discrepancy measure) to the Laplace approximation over the full parameter $q(\theta) = \mathcal{N}(\theta \mid \theta_{\text{MAP}}, \Sigma)$. As the subnetwork posterior is degenerate due to the involved Dirac delta functions, common discrepancy measures such as the KL di-

vergence are not well defined. Therefore, Daxberger et al. (2021b) propose to use the squared 2-Wasserstein distance, which in this case takes the following form:

$$W_2(q(\theta), q_s(\theta))^2 = \mathrm{Tr}\left(\Sigma + \Sigma^{(s)} - 2\left(\Sigma^{(s)1/2} \, \Sigma \, \Sigma^{(s)1/2} \right)^{1/2} \right), \tag{B.5}$$

where the (degenerate) subnetwork covariance matrix $\Sigma^{(s)}$ is equal to the full covariance matrix Σ but with zeros at the positions corresponding to the weights in $\theta^{(r)}$.

However, finding the subset of weights $\theta^{(s)}$ of size d_s that minimizes (B.5) is combinatorially hard, as the contribution of each weight depends on every other weight. Daxberger et al. (2021b) therefore assume that the weights are independent, resulting in the following simplified objective:

$$W_2(q(\theta), q_s(\theta))^2 \approx \sum_{i=1}^{d} \sigma_i^2 (1 - m_i) \, , \tag{B.6}$$

where $\sigma_i^2 = \Sigma_{ii}$ is the marginal variance of the i-th weight under $q(\theta)$, and $m_i = 1$ if $\theta_i \in \theta^{(s)}$ (with slight abuse of notation) or 0 otherwise is a binary mask indicating which weights are part of the subnetwork (see Daxberger et al. (2021b) for details). The objective in (B.6) is trivially minimized by choosing a subnetwork containing the d_s weights with the highest σ_i^2 values—i.e. those with largest marginal variances.

B.1.2 Hessian Factorization

Practically speaking, the Hessian $\nabla^2_\theta \mathcal{L}(\theta; \mathcal{D})|_{\theta_{\mathrm{MAP}}}$ at the MAP estimate θ_{MAP} is not guranteed to be positive-semidefinite—the requirement for the Laplace approximation—due to the imperfection in neural network optimization. To alleviate this issue, one can use a positive-semidefinite approximation to the Hessian, called the ***Fisher information matrix*** (or simply the ***Fisher***). For brevity, given a datum (x, y), we denote $s(x, y)$ to be the gradient of the log-likelihood at θ_{MAP}, i.e.

$$s(x, y) := \nabla_\theta \log p(y \mid f_\theta(x))|_{\theta_{\mathrm{MAP}}}.$$

The Fisher is then defined by

$$F := \sum_{i=1}^{n} \mathbb{E}_{p(y|f_\theta(x_i))} \left(s(x_i, y) s(x_i, y)^\top \right). \tag{B.7}$$

To distinguish between this matrix and its variants, we shall refer to it as the ***full Fisher***.

Note that the full Fisher matrix F is as large as the exact Hessian of the network, so it is costly. Thus, here, we review several factorization schemes that makes the computation and storage of the Fisher efficient, starting from the simplest.

B.1.2.1 Diagonal

In this factorization, we simply assume that the Hessian Λ is simply a diagonal matrix with diagonal elements equal the diagonal of the Fisher, i.e. $\Lambda \approx -\text{diag}(F)^\top I - \lambda I$. Since we can write[2]

$$\text{diag}(F) = \sum_{i=1}^{m} \mathbb{E}_{p(y|f_{\theta_{\text{MAP}}}(x_i))}(s(x_i, y) \odot s(x_i, y)),$$

this factorization is efficient: Not only does it require only a vector of length D to represent F but also it incurs only a $O(d)$ cost when inverting Λ—down from $O(d^3)$.

B.1.2.2 KFAC

The KFAC factorization can be seen as a midpoint between the two extremes: diagonal factorization, which might be too restrictive, and the full Fisher, which is computationally infeasible. The key idea is to model the correlation between weights in the same layer but assume that any pair of weights from two different layers are independent—this is a more sophisticated assumption compared to the diagonal factorization since there, it is assumed that *all* weights are independent of each other. For any layer $\ell = 1, \dots, L$, denoting n_ℓ as the number of hidden units at the ℓ-th layer, let $W^{(\ell)} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ be the weight matrix of the ℓ-th layer of the network, $a^{(\ell)}$ the ℓ-th hidden vector, and $g^{(\ell)} \in \mathbb{R}^{n_\ell}$ the log-likelihood gradient w.r.t. $a^{(\ell)}$. For each $\ell = 1, \dots, L$, we can then write the outer product inside expectation in (B.7) as $s(x_i, y)s(x_i, y)^\top = a^{(\ell-1)}a^{(\ell)\top} \otimes g^{(\ell)}g^{(\ell)\top}$. Furthermore, assuming that $a^{(\ell-1)}$ is independent of $g^{(\ell)}$, we obtain the approximation of the ℓ-th diagonal block of F, which we denote by $F^{(\ell)}$:

$$F^{(\ell)} \approx \mathbb{E}\left(a^{(\ell-1)}a^{(\ell-1)\top}\right) \otimes \mathbb{E}\left(g^{(\ell)}g^{(\ell)\top}\right) =: A^{(\ell-1)} \otimes G^{(\ell)}, \tag{B.8}$$

where we represent both the sum and the expectation in (B.7) as $\mathbb{E}$ for brevity.

From the previous expression we can see that the space complexity for storing $F^{(\ell)}$ is reduced to $O(n_\ell^2 + n_{\ell-1}^2)$, down from $O(n_\ell^2 n_{\ell-1}^2)$. Considering all L layers of the network, we obtain the layer-wise Kronecker factors $\{A^{(\ell)}\}_{\ell=0}^{L-1}$ and $\{G^{(\ell)}\}_{\ell=1}^{L}$ of the log-likelihood's Hessian. This corresponds to the block-diagonal approximation of the full Hessian.

One can then readily use these Kronecker factors in a Laplace approximation. For each layer ℓ, we obtain the ℓ-th diagonal block of Λ—denoted $\Lambda^{(\ell)}$—by

$$\Lambda^{(\ell)} \approx \left(A^{(\ell-1)} + \sqrt{\lambda} I\right) \otimes \left(G^{(\ell)} + \sqrt{\lambda} I\right) =: V^{(\ell)} \otimes U^{(\ell)}.$$

Note that we take the square root of the prior precision to avoid "double-counting" the effect of the prior. Nonetheless, this can still be a crude approximation (Martens & Grosse, 2015; Immer et al., 2021b). This particular Laplace approximation has been studied by Ritter et al. (2018a;b) and can be seen as approximating the posterior of each $W^{(\ell)}$ with the matrix-variate Gaussian distribution (Gupta & Nagar, 1999): $p(W^{(\ell)} \mid \mathcal{D}) \approx \mathcal{MN}(W^{(\ell)} \mid W^{(\ell)}_{\text{MAP}}, U^{(\ell)-1}, V^{(\ell)-1})$. Hence, sampling can be done easily in a layer-wise manner:

$$W^{(\ell)} \sim p\left(W^{(\ell)} \mid \mathcal{D}\right) \iff W^{(\ell)} = W^{(\ell)}_{\text{MAP}} + U^{(\ell)-\frac{1}{2}} E V^{(\ell)-\frac{1}{2}}$$

[2]The operator $\odot$ denotes the component-wise product.

$$\text{where} \quad E \sim \mathcal{MN}(0, I_{n_\ell}, I_{n_{\ell-1}}),$$

where we have denoted by I_b the identity $b \times b$ matrix, for $b \in \mathbb{N}$. Note that the above matrix inversions and square-root are in general much cheaper than those involving the entire Λ. Sampling E is not a problem either since $\mathcal{MN}(0, I_{n_\ell}, I_{n_{\ell-1}})$ is equivalent to the standard $(n_\ell n_{\ell-1})$-variate Normal distribution. As an alternative, Immer et al. (2021b) suggest to incorporate the prior exactly using an eigendecomposition of the individual Kronecker factors, which can improve performance.

B.1.2.3 Low Rank

Instead of only approximating each block by a low-rank structure, the entire Hessian or GGN can also be approximated by a low-rank structure (Sharma et al., 2021; Maddox et al., 2020). Eigendecomposition of F is a convenient way to obtain a low-rank approximation. The eigendecomposition of F is given by $QLQ^\top$ where the columns of $Q \in \mathbb{R}^{D \times D}$ are eigenvectors of F and $L = \text{diag}(l)$ is a D-dimensional diagonal matrix of eigenvalues. Assuming the eigenvalues in ℓ are arranged in a descending order, the optimal k-rank approximation in Frobenius or spectral norm is given by truncation (Eckart & Young, 1936): let $\widehat{Q} \in \mathbb{R}^{D \times k}$ be the matrix of the first k eigenvectors corresponding to the largest k eigenvalues $\widehat{l} \in \mathbb{R}^k$. That is, we truncate all eigenvectors and eigenvalues after the k largest eigenvalues. The low-rank approximation is then given by

$$F \approx \widehat{Q}\, \text{diag}(\widehat{l})\, \widehat{Q}^\top.$$

The rank k can be chosen based on the eigenvalues so as to retain as much information of the Hessian (approximation) as possible. Further, sampling and computation of the log-determinant can be carried out efficiently.

B.1.3 Hyperparameter Tuning

In this section we focus on tuning the prior variance/precision hyperparameter for simplicity. The same principle can be used for other hyperparameters of the Laplace approximation such that observation noise in the case of regression.

Post-Hoc Here, we assume that the steps of the Laplace approximation—MAP training and forming the Gaussian approximation—as two independent steps. As such, we are free to choose different prior variance γ^2 in the latter part, irrespective to the weight decay hyperparameter used in the former. Here, we review several ways to optimize γ^2 *post-hoc*. Ritter et al. (2018a) proposes to tune γ^2 by maximizing the posterior-predictive over a validation set $\mathcal{D}_{\text{val}} := (x_n, y_n)_{n=1}^{n_{\text{val}}}$. That is we solve the following one-parameter optimization problem:

$$\gamma_*^2 = \underset{\gamma^2}{\text{argmax}} \sum_{i=1}^{n_{\text{val}}} \log p(y_i \mid x_i, \mathcal{D}). \tag{B.9}$$

Table B.1: Qualitative comparison of different Hessian approximations. The KFAC Hessian approximation performs similar to FULL Gauss-Newton but is almost as fast as DIAG. We use online marginal likelihood method (Immer et al., 2021a) to train a small convolutional network on FMNIST and measure performance at test time. We repeat for three seeds to estimate the standard error. The OOD-AUROC is averaged over EMNIST, MNIST, and KMNIST. The prediction time is taken as the average over all in and out-of-distribution data sets. We use the MC predictive with 100 samples.

	test log likelihood	test accuracy	OOD-AUROC	prediction time (s)
DIAG	-0.302±0.005	0.894±0.002	0.832±0.011	29.5±0.2
KFAC	-0.282±0.004	0.899±0.002	0.836±0.004	30.6±0.1
FULL	-0.285±0.004	0.898±0.002	0.876±0.003	62.8±1.1

However, Kristiadi et al. (2020) found that the previous objective tends to make the Laplace approximation overconfident to outliers. Hence, they proposed to add an auxiliary term that depends on an OOD dataset $\mathcal{D}_{\text{out}} := \left(x_i^{(\text{out})}\right)_{i=1}^{n_{\text{out}}}$ to (B.9), as follows

$$\gamma_*^2 = \underset{\gamma^2}{\operatorname{argmax}} \sum_{i=1}^{n_{\text{val}}} \log p(y_i \mid x_i, \mathcal{D}) + \lambda \sum_{i=1}^{n_{\text{out}}} \mathbb{H}\Big(p(y_i \mid x_i^{(\text{out})}, \mathcal{D})\Big), \tag{B.10}$$

where $\mathbb{H}$ is the entropy functional and $\lambda \in (0, 1]$ is a trade-off hyperparameter. Intuitively, we choose γ^2 that balances the calibration on the true dataset and the low-confidence on outliers. Moreover, other losses could be constructed to tune the prior precision for optimal performance w.r.t. some desired quantity. Finally, inspired by Immer et al. (2021a) (further details below in *Online*) one can also maximize the Laplace-approximated marginal likelihood (2.6) to obtain γ_*^2, which eliminates the need for the validation data.

Online Contrary to the *post-hoc* tuning above, here we perform a Laplace approximation and tune the prior variance simultaneously as we perform a MAP training (Immer et al., 2021a). The key is to form a Laplace-approximated posterior every B epochs of a gradient descent, and use this posterior to approximate the marginal likelihood, cf. (2.6). By maximizing this marginal likelihood, we can find the best hyperparameters. Thus, once the MAP training has finished, we automatically obtain a prior variance that is already suitable for the Laplace approximation. Note that, this way, only a single MAP training needs to be done. This is in contrast to the classic, offline evidence framework (MacKay, 1992b) where the marginal likelihood maximization is performed only when the MAP estimation is done, and these steps need to be iteratively done until convergence. As a final note, similar to the *post-hoc* marginal likelihood above, this *online Laplace* does not require a validation set and has an additional benefit of improving the network's generalization performance (Immer et al., 2021a).

B.2 Predictive Uncertainty Quantification Experiments

B.2.1 Training Details

We use LeNet (LeCun et al., 1998) and WideResNet-16-4 (WRN, Zagoruyko & Komodakis, 2016) architectures for the MNIST and CIFAR-10 experiments, respectively. We adopt the commonly-used training procedure and hyperparameter values.

MAP We use Adam and Nesterov-SGD to train LeNet and WRN, respectively. The initial learning rate is 0.1 and annealed via the cosine decay method (Loshchilov & Hutter, 2017) over 100 epochs. The weight decay is set to 5×10^{-4}. Unless stated otherwise, all methods below use these training parameters.

DE We train five MAP network (see above) independently to form the ensemble.

VB We use the Bayesian-Torch library (Krishnan & Esposito, 2020) to train the network. Tha variational posterior is chosen to be the diagonal Gaussian (Graves, 2011; Blundell et al., 2015) and the flipout estimator (Wen et al., 2018) is employed. The prior precision is set to 5×10^{-4} to match the MAP-trained network, while the KL-term downscaling factor is set to 0.1, following (Osawa et al., 2019).

CSGHMC We use the publicly available code provided by the original authors (Zhang et al., 2020).[3] We use their default (i.e. recommended) hyperparameters.

SWAG For the SWAG baseline, we follow Maddox et al. (2019a) and run stochastic gradient descent with a constant learning rate on the pre-trained models to collect one model snapshot per epoch, for a total of 40 snapshots. At test time, we then make predictions by using 30 Monte Carlo samples from the posterior distribution; we correct the batch normalization statistics of each sample as described in Maddox et al. (2019a). To tune the constant learning rate, we used the same approach as in Eschenhagen et al. (2021), combining a grid search with a threshold on the mean confidence. For MNIST, we defined the grid to be the set { 1e-1, 5e-2, 1e-2, 5e-3, 1e-3 }, yielding an optimal value of 1e-2. For CIFAR-10, searching over the same grid suggested that the optimal value lies between 5e-3 and 1e-3; another, finer-grained grid search over the set { 5e-3, 4e-3, 3e-3, 2e-3, 1e-3 } then revealed the best value to be 2e-3.

Other baselines Our choice of baselines is based on the most common and best performing methods of recent Bayesian DL papers. Despite its popularity, **Monte Carlo (MC) dropout** (Gal & Ghahramani, 2016) has been shown to underperform compared to more recent methods (see e.g. Ovadia et al. (2019)). A recent VI method called **Variational Online Gauss-Newton (VOGN)** (Osawa et al., 2019) also seems to underperform. For example, Fig. 5 of Osawa et al. (2019) shows that on OOD detection with CIFAR-10 vs. SVHN, MC-dropout and VOGN only achieve AUROC↑ values of 81.9 and 80.0, respectively, while last-layer-LA obtains a substantially better value of 91.9 (they use ResNet-18, which is comparable to our model).

[3] `https://github.com/ruqizhang/csgmcmc`

Table B.2: MNIST OOD detection results.

	Confidence ↓			AUROC ↑		
Methods	**EMNIST**	**FMNIST**	**KMNIST**	**EMNIST**	**FMNIST**	**KMNIST**
MAP	83.6±0.3	64.2±0.5	77.3±0.3	93.5±0.3	98.9±0.0	97.0±0.1
DE	75.8±0.2	55.4±0.4	65.9±0.3	95.1±0.0	99.2±0.0	98.3±0.0
BBB	79.1±0.4	67.5±1.6	73.1±0.4	92.3±0.2	98.2±0.2	97.0±0.2
CSGHMC	76.2±1.6	63.6±1.9	67.9±1.5	93.4±0.2	97.7±0.2	97.1±0.1
SWAG	64.9±0.3	84.0±0.2	78.5±0.3	98.9±0.0	93.6±0.3	97.1±0.1
LA	74.8±0.4	58.8±0.5	69.0±0.4	93.4±0.3	98.5±0.1	96.6±0.1
LA*	62.0±0.5	49.6±0.6	56.7±0.5	94.3±0.2	98.3±0.1	96.6±0.2

Table B.3: CIFAR-10 OOD detection results.

	Confidence ↓			AUROC ↑		
Methods	**SVHN**	**LSUN**	**CIFAR-100**	**SVHN**	**LSUN**	**CIFAR-100**
MAP	77.5±2.9	71.3±0.6	79.3±0.1	91.8±1.2	94.5±0.2	90.1±0.1
DE	62.8±0.7	62.6±0.4	70.8±0.0	95.4±0.2	95.3±0.1	91.4±0.1
BBB	60.2±0.7	53.8±1.1	63.8±0.2	88.5±0.4	91.9±0.4	84.9±0.1
CSGHMC	69.8±0.8	65.2±0.8	73.1±0.1	91.2±0.3	92.6±0.3	87.9±0.1
SWAG	69.3±4.0	62.2±2.3	73.0±0.4	91.6±1.3	94.0±0.7	88.2±0.5
LA	70.6±3.2	63.8±0.5	72.6±0.1	92.0±1.2	94.6±0.2	90.1±0.1
LA*	58.0±3.1	50.0±0.5	59.0±0.1	91.9±1.3	95.0±0.2	90.2±0.1

B.2.2 Detailed Results

We provide the detailed (i.e. non-averaged) OOD detection results in Tables B.2 and B.3.

B.2.3 Additional Details on Wall-clock Time Comparison

Concerning the wall-clock time comparison in Fig. 2.6, we would like to clarify that for LA, we consider the default configuration of `laplace-torch`. As the default LA variant uses the closed-form probit approximation to the predictive distribution and therefore neither requires Monte Carlo (MC) sampling nor multiple forward passes, the wall-clock time for making predictions is essentially the same as for MAP. This is contrast to the baseline methods, which are significantly more expensive at prediction time due to the need for MC sampling (VB, SWAG) or forward passes through multiple model snapshots (DE, CSGHMC).

Importantly, note that is an advantage exclusive to our implementation of LA (i.e. with a GGN/Fisher Hessian approximation or with the last-layer LA) that it can be used without sampling (i.e. using the probit or Laplace bridge predictive approximations). This kind of approximation is incompatible with the other baselines (i.e. DE, CSGHMC, SWAG, and VB) since these methods just yield samples/distributions over weights while our LA variants implicitly yield a Gaussian distribution over logits due to the linearization of the NN induced by the use of the GGN/Fisher (see Immer et al. (2021b) for details) or the use of only the last layer. While one could still apply linearization to other methods, this would not be theoretically justified, in contrast to GGN-/last-layer-LA.

Finally, the reason we benchmark our deterministic, probit-based version is that we found it to consistently perform on par or better than MC sampling. If we predict with the LA using MC samples on the logits, the runtime is only around 20% slower than the deterministic probit approximation, which is still significantly faster than all other methods.

In summary, we believe that the ability to obtain calibrated predictions with a single forward-pass is a critical and distinctive advantage of the LA over almost all other Bayesian deep learning and ensemble methods.

B.3 WILDS Experiments

For this set of experiments, we use WILDS (Koh et al., 2020), a recently proposed benchmark of realistic distribution shifts encompassing a variety of real-world datasets across different data modalities and application domains. In particular, we consider the following WILDS datasets:

- CAMELYON17: Tumor classification (binary) of histopathological tissue images across different hospitals (ID vs. OOD) using a DenseNet-121 model (10 seeds).
- FMOW: Building / land use classification (62 classes) of satellite images across different times and regions (ID vs. OOD) using a DenseNet-121 model (3 seeds).
- CIVILCOMMMENTS: Toxicity classification (binary) of online text comments across different demographic identities (ID vs. OOD) using a DistilBERT-base-uncased model (5 seeds).
- AMAZON: Sentiment classification (5 classes) of product reviews across different reviewers (ID vs. OOD) using a DistilBERT-base-uncased model (3 seeds).
- POVERTYMAP: Asset wealth index regression (real-valued) across different countries and rural/urban areas (ID vs. OOD) using a ResNet-18 model (5 seeds).

Please refer to the original paper for more details on this benchmark and the above-mentioned datasets. All reported results in Fig. 2.7 show the mean and standard error across as many seeds as there are provided with the original paper (see the list of datasets above for the exact numbers).

For the last-layer Laplace method, we use either a KFAC or full covariance matrix (depending on the size of the last layer; in particular, we use a KFAC covariance for FMOW and full covariances for all other datasets) and the linearized Monte Carlo predictive distribution with 10,000 samples.

For the deep ensemble, we simply the aggregate the pre-trained models provided by the original paper. This yields ensembles of 5 neural network models, which is a commly-used ensemble size (Ovadia et al., 2019). Since these models were trained in different ways (e.g. using different domain generalization methods, see Koh et al. (2020) for details), their combinations can be viewed as *hyperparameter ensembles* (Wenzel et al., 2020b).

Note that the temperature scaling baseline is only applicable for classification tasks, and therefore we do not report it for the POVERTYMAP regression dataset.

We tune the temperature parameter for temperature scaling, the prior precision parameter for Laplace, and the noise standard deviation parameter for regression (i.e. for the POVERTYMAP dataset) by minimizing the negative log-likelihood on the in-distribution validation sets provided with WILDS.

B.4 Continual Learning Experiments

We benchmark Laplace approximations in the Bayesian continual learning setting on the *permuted MNIST* benchmark which consists of 10 consecutive tasks where each task is a permutation of the pixels of the MNIST images. Following common practice (Ritter et al., 2018b; Nguyen et al., 2018; Osawa et al., 2019), we use a 2-hidden layer MLP with 100 hidden units each and $28 \times 28 = 784$ input dimensions and 10 output dimensions for the MNIST classes. We adopt the implementation of the continual learning task and the model by Pan et al. (2020). In the following, we will briefly outline the Bayesian approach to continual learning (Nguyen et al., 2018) and explain how a diagonal and KFAC Laplace approximation can be employed in this setting. Further, we describe how this can be combined with the evidence framework to update the prior online alleviating the need for a validation set, which is unlikely to be available in real continual learning scenarios.

B.4.1 Bayesian Continual Learning

The Bayesian approach to continual learning can be simply described as iteratively updating the posterior after each task. We are given t data sets $\mathcal{D} := \{\mathcal{D}_i\}_{i=1}^t$ and have a neural network with parameters θ. In line with the standard supervised learning setting outlined in Section 2.1, we have a prior on parameters $p(\theta) = \mathcal{N}(\theta \mid 0, \gamma^2 I)$ and a likelihood $p(\mathcal{D} \mid \theta)$ realized by a neural network. The posterior on the parameters after all tasks is then

$$p(\theta \mid \mathcal{D}) \propto p(\mathcal{D}_t \mid \theta) \dots p(\mathcal{D}_2 \mid \theta) p(\mathcal{D}_1 \mid \theta) p(\theta). \tag{B.11}$$

This factorization gives rise to a recursion to update the posterior after $i-1$ data sets to the posterior after i data sets:

$$p(\theta \mid \mathcal{D}_1, \dots, \mathcal{D}_i) \propto p(\mathcal{D}_i \mid \theta) p(\theta \mid \mathcal{D}_1, \dots, \mathcal{D}_{i-1}). \tag{B.12}$$

The normalizer in (B.12) is given by the marginal likelihood $p(\mathcal{D}_i \mid \mathcal{D}_1, \dots, \mathcal{D}_{i-1})$ and we will use it for optimizing the variance γ^2 of $p(\theta)$. Incorporating a new task is the same as Bayesian inference in the supervised case but with an updated prior, i.e., the prior is the previous posterior distribution on θ. The Laplace approximation provides one way to approximately infer the posterior distributions after each task (Huszár, 2018; Ritter et al., 2018b; Pan et al., 2020). Alternatively, variational inference can be used (Nguyen et al., 2018; Osawa et al., 2019).

B.4.2 The Laplace Approximation for Continual Learning

The Laplace approximation facilitates the recursive updates ((B.12)) that arise in continual learning. In this context, it was first suggested with a diagonal Hessian approximation by Kirkpatrick et al. (2017, EWC) and Huszár (2018) corrected their updates. Ritter et al. (2018b) greatly improved the performance by using a KFAC Hessian approximation instead of a diagonal. The Laplace approximation to the posterior after observing task t is a Gaussian $\mathcal{N}(\theta_{\text{MAP}}^{(i)}, \Sigma^{(i)})$ We obtain θ_{MAP} by optimizing the unnormalized log posterior distribution on θ as annotated in

(B.11) for every task, one after another. The Hessian of the same unnormalized log posterior also specifies the posterior covariance $\Sigma^{(i)}$:

$$\Sigma^{(i)} = \Big(\underbrace{- \nabla^2_\theta \log p(\mathcal{D}_i \mid \theta)|_{\theta^{(i)}_{\text{MAP}}}}_{\text{log-likelihood Hessian}} \underbrace{- \sum_{j=1}^{i-1} \nabla^2_\theta \log p(\mathcal{D}_j \mid \theta)|_{\theta^{(j)}_{\text{MAP}}}}_{\text{previous log-likelihood Hessians}} + \underbrace{\gamma^{-2} I}_{\text{log prior Hessian}} \Big)^{-1}. \tag{B.13}$$

This summation over Hessians is typically intractable for neural networks with large parameter vectors θ and hence diagonal or KFAC approximations are used (Kirkpatrick et al., 2017; Huszár, 2018; Ritter et al., 2018b). For the diagonal version, the addition of Hessians and log prior is exact. For the KFAC version, we follow the alternative suggestion by Ritter et al. (2018b) and add up Kronecker factors which is an approximation to the sum of Kronecker products. However, this approximation is what underlies KFAC even in the supervised learning case where we add up factors per data point over the entire data set. Lastly, we adapt γ during training on each task t by optimizing the marginal likelihood $p(\mathcal{D}_i \mid \mathcal{D}_1, \dots \mathcal{D}_{i-1})$, i.e., by differentiating it with respect to γ. This can be done by computing the eigendecomposition of the summed Kronecker factors (Immer et al., 2021a) and allows us to 1) adjust the regularization suitably per task and 2) avoid setting a hyperparameter thereby alleviating the need for validation data.

Appendix C
Appendix of Section 3.2

C.1 Adversarial Examples

The adversarial datasets ("Adversarial" and "FarAwayAdv", cf. Table 3.2) are constructed as follows. For "Adversarial": We use the standard PGD attack (Madry et al., 2018) on a uniform noise dataset of size 2000. The objective is to maximize the confidence of the MAP model (resp. ACET and OE below) inside of an ℓ^∞ ball with radius $\epsilon = 0.3$. The optimization is carried out for 40 iterations with a step size of 0.1. We ensure that the resulting adversarial examples are in the image space. For "FarAwayAdv": We use the same construction, but start from the "far-away" Noise datasets as used in Table 3.2 and we do not project the resulting adversarial examples onto the image space.

C.2 Bayesian Methods on Top of State-of-the-art OOD Detectors

We can also apply all methods we are considering here on top of the state-of-the-art models that are specifically trained to mitigate the overconfidence problem, namely ACET (Hein et al., 2019) and outlier exposure (OE) (Hendrycks et al., 2019). The results are presented in Tables C.3 and C.4. In general, applying the Bayesian methods improves the models further, especially in the asymptotic regime.

C.3 Frequentist Calibration

Although calibration is a frequentist approach for predictive uncertainty quantification, it is nevertheless interesting to get an insight on whether the properties of the Bayesian predictive distribution lead to a better calibration. To answer this, we use a standard metric (Naeini et al., 2015; Guo et al., 2017): the expected calibration error (ECE). We use the same models along with the same hyperparameters as we have used in the previous OOD experiments. We present the results in Table C.1. We found that all the Bayesian methods are competitive to the temperature scaling method, which is specifically constructed for improving the frequentist calibration.

Table C.1: Expected calibration errors (ECE).

	MNIST	**CIFAR10**	**SVHN**	**CIFAR100**
MAP	**6.7±0.3**	13.1±0.2	10.1±0.2	8.1±0.3
+Temp.	11.4±2.2	**3.6±0.6**	**2.1±0.5**	6.4±0.5
+LLLA	6.9±0.3	**3.6±0.6**	5.2±0.8	4.8±0.3
+DLA	15.5±0.2	6.9±0.1	8.3±0.0	**4.7±0.3**
+KFLA	9.7±0.3	7.9±0.1	6.5±0.1	5.6±0.4
ACET	**5.9±0.2**	15.8±0.4	11.9±0.2	10.1±0.4
+Temp.	11.0±1.5	**3.7±0.8**	2.3±0.4	6.4±0.4
+LLLA	6.1±0.2	12.3±0.7	9.3±0.5	6.9±0.3
+DLA	6.2±0.3	4.3±0.3	**2.0±0.1**	6.0±0.3
+KFLA	6.1±0.3	4.3±0.2	2.1±0.1	**4.6±0.2**
OE	14.7±1.2	15.8±0.3	11.0±0.1	25.0±0.2
+Temp.	9.0±2.3	23.3±0.7	**3.7±0.7**	**19.4±0.2**
+LLLA	**6.5±0.6**	**14.6±0.2**	4.1±0.3	24.9±0.4
+DLA	9.1±0.6	15.8±0.3	7.2±0.1	29.0±0.2
+KFLA	10.1±0.9	15.9±0.3	6.4±0.1	29.0±0.2

Table C.2: Adversarial OOD detection results.

	MAP		**+Temp.**		**+LLLA**		**+DLA**		**+KFLA**	
	MMC	AUR	MMC	AUR	MMC	AUR	MMC	AUR	MMC	AUR
MNIST - Adversarial	100.0±0.0	0.3±0.0	100.0±0.0	6.8±4.1	100.0±0.0	5.3±0.1	99.6±0.2	2.0±0.9	**91.3±1.2**	**69.2±3.5**
MNIST - FarAwayAdv	100.0±0.0	0.1±0.0	100.0±0.0	6.8±4.1	99.9±0.0	9.3±0.6	85.3±1.4	53.0±3.8	**55.6±2.0**	**97.4±0.3**
CIFAR10 - Adversarial	100.0±0.0	0.0±0.0	100.0±0.0	0.0±0.0	99.7±0.0	**9.1±0.1**	99.3±0.1	9.0±1.0	**99.2±0.0**	5.8±0.4
CIFAR10 - FarAwayAdv	99.5±0.0	8.8±0.0	99.2±0.0	7.9±0.1	**17.4±0.1**	**100.0±0.0**	61.3±2.4	89.4±1.0	61.2±1.3	87.8±0.8
SVHN - Adversarial	100.0±0.0	0.0±0.0	100.0±0.0	0.0±0.0	**97.6±0.0**	**32.5±0.3**	98.6±0.0	6.8±0.3	98.6±0.1	9.6±0.4
SVHN - FarAwayAdv	99.7±0.0	7.7±0.0	99.5±0.0	6.9±0.1	**27.5±0.1**	**99.6±0.0**	61.7±1.4	92.4±0.9	61.0±1.2	94.4±0.3
CIFAR100 - Adversarial	**100.0±0.0**	0.0±0.0	**100.0±0.0**	0.0±0.0	**100.0±0.0**	**0.2±0.0**	**100.0±0.0**	0.1±0.0	**100.0±0.0**	0.0±0.0
CIFAR100 - FarAwayAdv	100.0±0.0	1.3±0.0	99.9±0.0	1.2±0.0	**5.9±0.0**	**99.9±0.0**	42.0±1.5	83.9±0.9	42.3±1.8	80.8±1.2

Table C.3: OOD detection results when applying post-hoc Bayesian methods on top of models trained with ACET (Hein et al., 2019).

	MAP		+Temp.		+LLLA		+DLA		+KFLA	
	MMC	AUR	MMC	AUR	MMC	AUR	MMC	AUR	MMC	AUR
MNIST - MNIST	98.9±0.0	-	99.5±0.0	-	98.9±0.0	-	98.9±0.0	-	98.9±0.0	-
MNIST - EMNIST	59.1±0.0	**96.9±0.0**	70.9±1.8	96.5±0.1	**59.0±0.0**	**96.9±0.0**	**59.0±0.0**	**96.9±0.0**	59.1±0.0	**96.9±0.0**
MNIST - FMNIST	**10.2±0.0**	**100.0±0.0**	10.3±0.0	**100.0±0.0**	**10.2±0.0**	**100.0±0.0**	**10.2±0.0**	**100.0±0.0**	**10.2±0.0**	**100.0±0.0**
MNIST - Noise ($\delta = 2000$)	100.0±0.0	0.0±0.0	100.0±0.0	**21.9±9.2**	100.0±0.0	0.3±0.2	**99.9±0.0**	0.3±0.2	100.0±0.0	0.2±0.1
MNIST - Adversarial	**10.0±0.0**	**100.0±0.0**	**10.0±0.0**	**100.0±0.0**	10.1±0.0	**100.0±0.0**	**10.0±0.0**	**100.0±0.0**	**10.0±0.0**	**100.0±0.0**
MNIST - FarAwayAdv	**100.0±0.0**	0.0±0.0	**100.0±0.0**	**21.9±9.2**	**100.0±0.0**	0.1±0.0	**100.0±0.0**	0.2±0.0	**100.0±0.0**	0.1±0.0
CIFAR10 - CIFAR10	97.3±0.0	-	95.2±0.2	-	96.7±0.1	-	94.7±0.0	-	94.9±0.0	-
CIFAR10 - SVHN	62.8±0.0	96.1±0.0	**52.9±0.7**	**96.5±0.0**	59.5±0.5	96.1±0.1	53.1±0.1	96.2±0.1	53.7±0.1	96.2±0.0
CIFAR10 - LSUN	72.1±0.0	92.8±0.0	62.6±0.7	93.2±0.1	68.9±0.6	92.8±0.1	**59.9±0.6**	**93.8±0.2**	60.4±0.2	93.7±0.1
CIFAR10 - Noise ($\delta = 2000$)	100.0±0.0	0.0±0.0	100.0±0.0	0.0±0.0	**16.0±0.0**	**100.0±0.0**	71.7±1.7	92.3±0.7	65.8±1.8	94.4±0.5
CIFAR10 - Adversarial	78.1±0.0	83.1±0.1	71.1±0.5	84.1±0.1	76.8±0.0	83.2±0.1	67.9±0.4	88.5±0.2	**67.7±0.3**	**88.8±0.2**
CIFAR10 - FarAwayAdv	100.0±0.0	0.0±0.0	100.0±0.0	0.0±0.0	**16.0±0.0**	**100.0±0.0**	72.5±2.4	92.1±0.7	70.7±1.9	93.1±0.5
SVHN - SVHN	98.5±0.0	-	97.3±0.2	-	98.3±0.0	-	96.7±0.0	-	96.2±0.0	-
SVHN - CIFAR10	65.9±0.0	95.6±0.0	58.5±0.8	95.7±0.0	64.0±0.3	95.7±0.0	49.8±0.1	**97.5±0.0**	**48.3±0.1**	97.4±0.0
SVHN - LSUN	28.0±0.0	99.3±0.0	24.6±0.3	99.4±0.0	27.8±0.1	99.3±0.0	22.8±0.6	**99.6±0.0**	**21.7±0.6**	**99.6±0.0**
SVHN - Noise ($\delta = 2000$)	17.9±0.2	**100.0±0.0**	16.0±0.3	**100.0±0.0**	**15.0±0.0**	**100.0±0.0**	45.1±2.1	99.0±0.2	41.6±1.3	99.1±0.1
SVHN - Adversarial	10.4±0.0	**100.0±0.0**	**10.3±0.0**	**100.0±0.0**	10.8±0.0	**100.0±0.0**	10.4±0.0	**100.0±0.0**	10.4±0.0	**100.0±0.0**
SVHN - FarAwayAdv	17.6±0.0	**100.0±0.0**	15.7±0.2	**100.0±0.0**	**15.0±0.0**	**100.0±0.0**	44.9±2.6	99.0±0.2	44.8±1.5	98.8±0.1
CIFAR100 - CIFAR100	82.0±0.1	-	78.1±0.5	-	79.6±0.1	-	78.7±0.1	-	76.3±0.1	-
CIFAR100 - SVHN	57.1±0.0	77.8±0.1	**49.5±0.8**	**78.7±0.1**	52.7±0.1	78.4±0.1	52.4±0.0	77.7±0.1	**49.5±0.0**	77.4±0.2
CIFAR100 - LSUN	55.1±0.0	78.8±0.1	48.3±0.7	79.0±0.1	50.6±0.1	**79.5±0.1**	49.8±0.1	79.3±0.1	**46.8±0.2**	79.2±0.2
CIFAR100 - Noise ($\delta = 2000$)	99.3±0.1	4.2±0.2	99.2±0.1	3.8±0.2	**5.4±0.0**	**100.0±0.0**	58.5±1.3	76.1±0.9	51.8±1.1	77.6±0.9
CIFAR100 - Adversarial	1.5±0.0	**100.0±0.0**	**1.4±0.0**	**100.0±0.0**	1.5±0.0	**100.0±0.0**	**1.4±0.0**	**100.0±0.0**	**1.4±0.0**	**100.0±0.0**
CIFAR100 - FarAwayAdv	99.7±0.0	3.4±0.0	99.6±0.0	3.1±0.0	**5.4±0.0**	**100.0±0.0**	57.7±1.5	76.6±1.0	57.9±1.4	73.4±1.0

Table C.4: OOD detection results when applying post-hoc Bayesian methods on top of models trained with outlier exposure (OE) (Hendrycks et al., 2019).

	MAP		+Temp.		+LLLA		+DLA		+KFLA	
	MMC	AUR	MMC	AUR	MMC	AUR	MMC	AUR	MMC	AUR
MNIST - MNIST	99.6±0.0	-	99.4±0.1	-	97.8±0.8	-	99.4±0.0	-	99.4±0.0	-
MNIST - EMNIST	84.2±0.0	96.0±0.1	77.1±2.6	**96.3±0.1**	**67.3±3.1**	94.3±0.7	79.6±0.0	95.6±0.1	79.1±0.0	95.9±0.1
MNIST - FMNIST	27.9±0.0	**99.9±0.0**	**22.8±1.5**	**99.9±0.0**	25.6±1.6	**99.9±0.0**	27.5±0.1	**99.9±0.0**	27.3±0.0	**99.9±0.0**
MNIST - Noise ($\delta = 2000$)	99.9±0.0	26.4±0.2	99.9±0.0	5.0±2.4	66.0±0.6	95.9±0.4	58.4±0.3	97.6±0.3	**49.8±0.3**	**99.3±0.1**
MNIST - Adversarial	40.5±0.0	98.8±0.0	**35.2±1.1**	99.1±0.0	38.7±0.0	98.1±0.0	38.1±0.0	98.7±0.0	35.8±0.1	**99.2±0.0**
MNIST - FarAwayAdv	100.0±0.0	25.5±0.2	100.0±0.0	3.6±2.4	66.6±0.1	95.7±0.1	59.2±0.3	97.2±0.2	**50.5±0.3**	**99.3±0.1**
CIFAR10 - CIFAR10	89.4±0.1	-	92.5±0.4	-	89.2±0.1	-	89.3±0.1	-	89.3±0.1	-
CIFAR10 - SVHN	**10.8±0.0**	**98.8±0.0**	11.2±0.1	**98.8±0.0**	10.9±0.0	98.7±0.0	**10.8±0.0**	**98.8±0.0**	**10.8±0.0**	**98.8±0.0**
CIFAR10 - LSUN	**10.4±0.0**	**98.6±0.0**	10.7±0.1	**98.6±0.0**	10.6±0.0	98.5±0.1	**10.4±0.0**	**98.6±0.0**	**10.4±0.0**	**98.6±0.0**
CIFAR10 - Noise ($\delta = 2000$)	99.1±0.1	6.5±0.6	99.4±0.1	7.6±0.7	**25.0±0.1**	**93.6±0.1**	77.9±1.0	79.5±2.0	72.7±1.5	84.6±1.2
CIFAR10 - Adversarial	**98.5±0.0**	2.4±0.0	98.8±0.0	**2.6±0.2**	**98.5±0.0**	2.4±0.0	**98.5±0.0**	2.4±0.0	**98.5±0.0**	2.4±0.0
CIFAR10 - FarAwayAdv	99.5±0.0	5.2±0.0	99.8±0.0	6.2±0.3	**25.1±0.1**	**93.6±0.1**	79.4±1.1	78.4±1.9	78.1±1.4	79.6±1.7
SVHN - SVHN	97.4±0.0	-	95.8±0.3	-	95.7±0.2	-	92.5±0.0	-	93.5±0.0	-
SVHN - CIFAR10	10.2±0.0	**100.0±0.0**	**10.1±0.0**	**100.0±0.0**	14.3±0.6	99.9±0.0	10.8±0.0	**100.0±0.0**	10.8±0.0	**100.0±0.0**
SVHN - LSUN	**10.1±0.0**	**100.0±0.0**	**10.1±0.0**	**100.0±0.0**	14.2±0.6	99.9±0.0	10.8±0.0	**100.0±0.0**	10.9±0.1	**100.0±0.0**
SVHN - Noise ($\delta = 2000$)	99.7±0.0	3.0±0.2	99.6±0.1	2.7±0.2	**16.2±0.0**	**99.7±0.0**	31.5±1.4	98.4±0.2	33.0±1.3	98.4±0.2
SVHN - Adversarial	44.9±0.0	98.2±0.0	34.4±0.7	98.5±0.0	34.2±0.0	98.5±0.0	**17.9±0.2**	99.5±0.0	18.2±0.2	**99.6±0.0**
SVHN - FarAwayAdv	99.9±0.0	2.4±0.0	99.8±0.0	2.2±0.0	**16.3±0.0**	**99.7±0.0**	32.1±1.2	98.3±0.2	31.7±1.6	98.6±0.2
CIFAR100 - CIFAR100	59.6±0.2	-	71.8±0.5	-	54.9±0.2	-	51.5±0.2	-	52.0±0.2	-
CIFAR100 - SVHN	**3.6±0.0**	**93.5±0.1**	7.2±0.2	93.4±0.1	**3.6±0.2**	92.9±0.5	**3.6±0.0**	93.2±0.1	**3.6±0.0**	93.4±0.1
CIFAR100 - LSUN	2.6±0.0	95.4±0.1	5.0±0.1	95.3±0.1	2.9±0.1	94.6±0.2	**2.5±0.1**	**95.9±0.2**	**2.5±0.1**	**95.9±0.2**
CIFAR100 - Noise ($\delta = 2000$)	100.0±0.0	1.3±0.0	100.0±0.0	7.3±0.7	**25.3±0.1**	**64.5±0.2**	89.7±1.9	25.0±2.7	82.4±1.4	33.5±1.3
CIFAR100 - Adversarial	95.6±0.0	21.7±0.1	96.7±0.0	24.6±0.4	**67.3±0.1**	**40.2±0.2**	89.8±0.3	23.9±0.3	89.8±0.2	24.9±0.2
CIFAR100 - FarAwayAdv	100.0±0.0	1.3±0.0	100.0±0.0	7.3±0.7	**25.3±0.1**	**64.5±0.2**	89.4±1.6	25.6±2.2	89.1±1.9	27.2±2.2

Appendix D
Appendix of Section 3.3

D.1 Training Details

For LeNet, we use Adam optimizer with an initial learning rate 1×10^{-3} while for ResNet, we use SGD with an initial learning rate of 0.1 and momentum 0.9. In both cases, the optimization is carried out for 100 epochs using weight decay 5×10^{-4} on a single GPU. We also reduce the learning rate by a factor of 10 at epochs 50, 75, and 90. Test accuracies are in Table D.5.

D.2 Experiments

D.2.1 Asymptotic Regime

As a gold standard GP baseline, we compare against the method of Qiu et al. (2020) (with our DSCS kernel). We refer to this baseline simply as GP-DSCS. The base methods, which RGPR is implemented on, are the following recently-proposed BNNs: (i) Kronecker-factored Laplace (KFL, Ritter et al., 2018a), (ii) stochastic weight averaging-Gaussian (SWAG, Maddox et al., 2019a), and (iii) stochastic variational deep kernel learning (SVDKL, Wilson et al., 2016b). All the kernel hyperparameters for RGPR are set to a constant value of 1×10^{-10} since we focus on the asymptotic regime. In all cases, MC-integral with 10 posterior samples is used for making predictions. We construct a test dataset artificially by sampling 2000 uniform noises in $[0, 1]^N$ and scale them with a scalar $\alpha = 2000$. The goal is to achieve low confidence over these far-away points.

The results are presented in Table D.1. We observe that the RGPR-augmented methods are significantly better than their respective base methods. In particular, their confidence estimates are significantly lower than those of the vanilla methods, becoming closer to the confidence of the gold-standard GP-DSCS baseline. This indicates that RGPR makes BNNs better calibrated in the asymptotic regime.

D.2.2 Non-Asymptotic Regime

D.2.2.1 Dataset shift

In Table D.2 we present the non-normalized numerical results to complement Fig. 3.11. RGPR in general improves the vanilla LLL.

Table D.1: RGPRs compared to their respective base methods on the detection of far-away outliers. Values are average confidences. Error bars are standard errors over three prediction runs. For each dataset, the best value over each vanilla and RGPR-imbued method (e.g. KFL against KFL-RGPR) are in bold.

Methods	CIFAR10	SVHN
GP-DSCS	22.0±0.2	22.1±0.3
KFL	64.5±0.7	63.4±1.5
KFL-RGPR	**29.9**±0.3	**27.5**±0.0
SWAG	63.5±1.8	50.2±4.2
SWAG-RGPR	**29.3**±0.2	**27.5**±0.0
SVDKL	46.4±0.3	49.1±0.2
SVDKL-RGPR	**22.0**±0.1	**22.1**±0.1

Table D.2: CIFAR10-C results. Values are mean over all corruptions.

	NLL	ECE	Brier	Confidence	Accuracy
MAP	1.066	0.226	0.402	0.887	0.739
Temp.	0.914	0.147	0.378	0.842	0.739
DE	0.909	0.110	**0.354**	0.840	**0.752**
GP-DSCS	1.096	0.232	0.413	0.888	0.734
LLL	0.872	0.080	0.363	0.800	0.739
LLL-RGPR-LL	0.870	**0.079**	0.363	0.796	0.738
LLL-RGPR-OOD	**0.869**	0.095	0.363	**0.717**	0.738

D.2.2.2 OOD detection

We expand Table 3.5 in Table D.6. In the same table, we additionally show the mean confidence values (Hendrycks & Gimpel, 2017, MMC,). For CIFAR10, SVHN, and CIFAR100, we test each model against FMNIST (called FMNIST3D) to measure the performance on grayscale OOD images. Finally, we also show the OOD detection performance via additional AUROC and area under precision-recall curve (AUPRC) metrics in Table D.7.

Additionally, we compare RGPR with recent non-Bayesian baselines: (i) the Mahalanobis detector (Lee et al., 2018c) and (ii) deterministic uncertainty quantification (DUQ) (Van Amersfoort et al., 2020). Values are taken directly from the original papers—they used the same architecture as in this paper. Table D.3 shows that a RGPR-equipped BNN is better than the Mahalanobis detector. Moreover, LLL-RGPR-OOD is competitive to DUQ, but without the drawback of reducing test accuracy.

D.2.2.3 Hyperparameter tuning

We present the optimal hyperparameters $(\sigma_l^2)_{l=0}^{L-1}$ in Table D.8. We observe that using higher representations of the data is beneficial, as indicated by non-trivial hyperparameter values on all layers across all networks and datasets.

Table D.3: RGPR against recent non-Bayesian baselines. The OOD detection metric is AUROC.

	CIFAR10 vs. LSUN	CIFAR10 vs. SVHN
Mahalanobis	89.2	91.5
LLL-RGPR-OOD	**92.6**	**95.8**

	Test Acc.	CIFAR10 vs. SVHN
DUQ ($\lambda = 0$)	94.2	86.1
DUQ ($\lambda = 0.5$)	93.2	**92.7**
LLL-RGPR-OOD	**94.3**	92.6

Table D.4: Expected calibration errors (ECE).

	MNIST	CIFAR10	SVHN	CIFAR100
MAP	6.7	13.1	10.1	8.1
Temp. Scaling	11.4	3.6	2.1	6.4
ACET	5.9	15.8	11.9	10.1
OE	14.7	15.8	11.0	25.0

D.2.2.4 Natural images for tuning

We present OOD detection results via different $\mathcal{D}_{\text{our}}$ for tuning $\boldsymbol{\sigma}^2$, in Table D.9. Specifically, we use the ImageNet32×32 dataset (Chrabaszcz et al., 2017), which represents natural image datasets, and is thus more sophisticated than the noise dataset used in the main text. Nevertheless, we observe that the OOD detection performance is comparable to that of the noise dataset, justifying the choice of $\mathcal{D}_{\text{out}}$ we have made in the main text.

D.2.2.5 Calibration is at odds with OOD detection

As noted in the main text, we observe that employing OOD data for tuning $\boldsymbol{\sigma}^2$ degrades the in-distribution calibration (as measured by the ECE metric) of RGPR. In Table D.4 (taken from Table 5 of Kristiadi et al. (2020)), we can see that even recent OOD training methods with many more parameters than RGPR such as ACET (Hein et al., 2019) and OE (Hendrycks et al., 2019) degrade the in-distribution ECE. However, note that ACET and OE represent state-of-the-art OOD detectors. Hence, it is reasonable to conclude that this issue does not seem to be inherent to RGPR.

D.2.2.6 Regression

To empirically validate our method and analysis (esp. Proposition 3.13), we present a toy regression results. The outlier dataset is constructed by sampling 1000 points from the standard Gaussian and scale them with $\alpha = 2000$. The metric used is the predictive error bar (standard deviation). Following the standard practice (see e.g. Sun et al. (2019)), we use a two-layer ReLU network with 50 hidden units. The Bayesian methods used are LLL, KFL, SWAG, and stochastic variational GP (SVGP, Hensman et al., 2015) using 50 inducing points. Finally, we standardize the data and the hyperparameter for RGPR is set to 0.001 so that Proposition 3.12 is satisfied.

Table D.5: Accuracy and calibration error.

Methods	**MNIST**	**CIFAR10**	**SVHN**	**CIFAR100**
Acc. ↑				
MAP	99.4	94.3	97.1	76.7
Temp. Scaling	99.4	94.3	97.1	76.7
Deep Ens.	99.6	95.3	97.4	79.5
GP-DSCS	99.3	93.9	97.0	76.6
LLL	99.4	94.3	97.0	76.7
LLL-RGPR-LL	99.2	94.4	97.0	76.7
LLL-RGPR-OOD	99.1	94.3	96.9	76.6
ECE ↓				
MAP	5.4	13.9	13.3	6.4
Temp. Scaling	9.9	6.7	7.5	4.7
Deep Ens.	12.5	2.8	1.3	1.9
GP-DSCS	4.5	14.4	13.6	8.2
LLL	14.0	2.8	12.9	4.7
LLL-RGPR-LL	15.8	3.6	13.1	5.7
LLL-RGPR-OOD	19.6	12.5	15.9	15.8

The results are presented in Table D.10. We can observe that RGPR retain high confidence estimates over inlier data and yield much larger error bars compared to the base methods.

Table D.6: OOD data detection results in terms of MMC and FPR@95 metrics. All values are averages and standard errors over 10 prediction trials.

	MAP		Temp. Scaling		Deep Ens.		GP-DSCS		LLL		LLL-RGPR-LL		LLL-RGPR-OOD	
Datasets	**MMC ↓**	**FPR ↓**	**MMC ↓**	**FPR ↓**	**MMC ↓**	**FPR ↓**	**MMC ↓**	**FPR ↓**	**MMC ↓**	**FPR ↓**	**MMC ↓**	**FPR ↓**	**MMC ↓**	**FPR ↓**
MNIST	99.2	-	99.5±0.0	-	99.1	-	99.2±0.0	-	97.4±0.0	-	97.0±0.0	-	96.1±0.0	-
EMNIST	78.1	24.5	83.4±0.0	24.9±0.0	74.1	**21.4**	77.6±0.0	24.7±0.0	62.7±0.0	23.3±0.1	55.7±0.0	21.9±0.1	**49.4**±0.0	21.7±0.1
KMNIST	73.1	14.3	79.3±0.0	14.1±0.0	63.1	5.6	72.2±0.0	13.2±0.0	52.7±0.0	6.3±0.0	17.1±0.0	0.4±0.0	**15.6**±0.0	**0.0**±0.0
FMNIST	79.8	26.8	85.0±0.0	27.3±0.0	71.7	11.3	79.1±0.0	25.5±0.1	64.6±0.0	19.1±0.2	18.1±0.0	1.3±0.0	**15.5**±0.0	**0.0**±0.0
GrayCIFAR10	85.7	3.6	93.4±0.0	4.3±0.0	72.7	**0.0**	85.2±0.0	3.5±0.0	61.1±0.0	0.5±0.0	**15.1**±0.0	**0.0**±0.0	**15.1**±0.0	**0.0**±0.0
UniformNoise	100.0	100.0	100.0±0.0	100.0±0.0	99.9	100.0	100.0±0.0	100.0±0.0	95.7±0.0	99.7±0.0	**15.1**±0.0	**0.0**±0.0	**15.1**±0.0	**0.0**±0.0
CIFAR10	97.0	-	95.0±0.0	-	95.6	-	96.9±0.0	-	93.4±0.0	-	93.1±0.0	-	85.9±0.0	-
SVHN	62.5	29.3	53.7±0.0	25.6±0.0	59.7	37.0	69.0±0.0	40.0±0.1	47.0±0.0	24.8±0.1	46.7±0.0	25.1±0.1	**40.6**±0.0	**23.3**±0.2
LSUN	74.5	52.7	65.9±0.0	48.7±0.0	65.6	50.3	76.6±0.0	55.1±0.3	58.5±0.1	44.1±0.7	57.4±0.1	42.9±0.6	**48.5**±0.1	**40.0**±0.5
CIFAR100	79.4	61.5	72.4±0.0	59.4±0.0	70.7	58.0	80.0±0.0	62.5±0.1	66.0±0.0	58.2±0.2	65.3±0.0	58.2±0.2	**55.6**±0.0	**54.7**±0.2
FMNIST3D	71.4	45.3	62.8±0.0	41.0±0.0	63.0	44.1	72.6±0.0	47.9±0.2	53.4±0.0	34.7±0.2	52.6±0.0	34.5±0.2	**36.6**±0.0	**16.4**±0.3
UniformNoise	64.7	26.2	54.7±0.1	19.5±0.3	73.9	86.0	75.8±0.1	55.3±0.4	39.1±0.1	2.8±0.1	37.9±0.1	2.2±0.2	**32.0**±0.1	**1.7**±0.3
SVHN	98.5	-	97.6±0.0	-	97.8	-	98.5±0.0	-	92.4±0.0	-	92.2±0.0	-	88.0±0.0	-
CIFAR10	70.4	18.3	64.7±0.0	18.0±0.0	57.2	**11.9**	70.9±0.0	19.8±0.0	41.7±0.0	15.0±0.1	41.2±0.0	14.9±0.1	**34.9**±0.0	14.7±0.1
LSUN	71.7	18.7	66.0±0.0	19.0±0.0	56.0	**10.0**	72.2±0.0	20.1±0.2	42.9±0.1	16.2±0.5	42.0±0.1	15.5±0.2	**32.3**±0.1	11.9±0.3
CIFAR100	71.3	20.4	65.7±0.0	20.1±0.0	57.6	**12.6**	71.8±0.0	22.2±0.0	43.2±0.0	17.7±0.1	42.5±0.0	17.5±0.1	**35.2**±0.0	16.0±0.1
FMNIST3D	72.5	21.9	66.9±0.0	21.7±0.0	61.9	20.0	72.8±0.0	22.9±0.0	45.3±0.0	21.5±0.1	38.9±0.0	12.6±0.1	**16.8**±0.0	**0.0**±0.0
UniformNoise	68.9	14.0	62.7±0.1	13.6±0.2	48.1	**3.8**	68.8±0.1	14.9±0.2	41.0±0.1	12.5±0.5	39.5±0.1	11.4±0.4	**27.3**±0.1	**4.1**±0.2
CIFAR100	81.3	-	78.9±0.0	-	80.2	-	82.2±0.0	-	74.4±0.0	-	73.4±0.0	-	62.8±0.0	-
SVHN	53.5	78.9	49.1±0.0	78.3±0.0	44.7	**65.5**	46.8±0.0	68.2±0.0	42.6±0.0	77.4±0.2	42.0±0.0	78.2±0.3	**34.9**±0.0	79.7±0.2
LSUN	50.7	74.7	46.6±0.0	75.0±0.0	47.1	76.0	53.6±0.0	76.8±0.1	39.6±0.1	**73.5**±0.5	38.0±0.1	**73.7**±0.3	**30.3**±0.0	75.7±0.6
CIFAR10	53.3	78.3	49.3±0.0	78.0±0.0	51.3	**76.9**	56.0±0.0	78.8±0.0	44.1±0.0	77.9±0.2	43.0±0.0	78.3±0.3	**34.9**±0.0	79.1±0.2
FMNIST3D	38.9	60.8	34.8±0.0	60.0±0.0	38.1	59.6	44.3±0.0	65.5±0.1	30.0±0.0	58.6±0.2	29.0±0.0	58.6±0.3	**16.8**±0.0	**38.7**±0.3
UniformNoise	29.4	55.8	25.7±0.1	55.5±0.4	45.1	94.9	31.6±0.1	49.9±0.1	22.0±0.1	47.0±0.4	17.1±0.1	**24.0**±0.8	**14.3**±0.0	29.6±0.5

Table D.7: OOD data detection results in terms of AUROC and AUPRC metrics. All values are averages and standard errors over 10 prediction trials.

	MAP		Temp. Scaling		Deep Ens.		GP-DSCS		LLL		LLL-RGPR-LL		LLL-RGPR-OOD	
Datasets	**AUROC ↓**	**AUPRC ↓**	**AUROC ↓**	**AUPRC ↓**	**AUROC ↓**	**AUPRC ↓**	**AUROC ↓**	**AUPRC ↓**	**AUROC ↓**	**AUPRC ↓**	**AUROC ↓**	**AUPRC ↓**	**AUROC ↓**	**AUPRC ↓**
MNIST	-	-	-	-	-	-	-	-	-	-	-	-	-	-
EMNIST	95.0	89.6	94.9±0.0	89.5±0.0	**95.7**	**91.2**	94.8±0.0	89.0±0.0	94.2±0.0	86.8±0.0	94.5±0.0	87.6±0.0	94.5±0.0	87.8±0.0
KMNIST	96.0	93.0	96.1±0.0	93.5±0.0	98.3	97.6	96.4±0.0	93.7±0.0	98.4±0.0	98.3±0.0	**99.8**±0.0	**99.8**±0.0	**99.8**±0.0	**99.8**±0.0
FMNIST	92.2	85.8	92.2±0.0	86.2±0.0	96.6	94.0	92.7±0.0	86.5±0.0	96.8±0.0	96.9±0.0	99.7±0.0	99.7±0.0	**99.8**±0.0	**99.8**±0.0
GrayCIFAR10	98.0	98.5	97.8±0.0	98.4±0.0	99.0	99.4	98.0±0.0	98.6±0.0	98.5±0.0	99.0±0.0	**99.9**±0.0	**100.0**±0.0	99.8±0.0	99.9±0.0
UniformNoise	0.1	59.8	0.4±0.0	60.1±0.0	42.6	76.5	0.1±0.0	59.8±0.0	84.6±0.1	96.3±0.0	**99.9**±0.0	**100.0**±0.0	99.8±0.0	**100.0**±0.0
CIFAR10	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SVHN	95.7	91.0	96.1±0.0	91.2±0.0	95.2	92.0	93.6±0.0	85.6±0.0	**96.3**±0.0	**92.1**±0.0	96.2±0.0	91.9±0.0	95.8±0.0	90.2±0.0
LSUN	91.8	99.6	92.2±0.0	99.6±0.0	**92.8**	**99.7**	90.7±0.0	99.6±0.0	92.7±0.0	**99.7**±0.0	**92.8**±0.0	**99.7**±0.0	92.6±0.0	**99.7**±0.0
CIFAR100	87.3	83.7	87.4±0.0	83.4±0.0	**90.1**	**89.5**	86.3±0.0	82.4±0.0	88.0±0.0	84.7±0.0	87.9±0.0	84.5±0.0	87.0±0.0	82.9±0.0
FMNIST3D	92.9	92.2	93.3±0.0	92.5±0.0	94.0	94.5	92.3±0.0	91.6±0.0	94.7±0.0	94.5±0.0	94.7±0.0	94.5±0.0	**97.4**±0.0	**97.5**±0.0
UniformNoise	96.7	99.2	97.1±0.0	99.3±0.0	92.8	98.4	94.2±0.0	98.7±0.0	98.8±0.0	99.7±0.0	**98.9**±0.0	99.7±0.0	**98.9**±0.0	**99.8**±0.0
SVHN	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CIFAR10	95.4	97.0	95.4±0.0	96.9±0.0	**97.5**	98.9	95.0±0.0	96.7±0.0	97.3±0.0	98.9±0.0	97.3±0.0	98.9±0.0	97.4±0.0	**99.0**±0.0
LSUN	95.6	99.9	95.6±0.0	99.9±0.0	**98.0**	**100.0**	95.1±0.0	99.9±0.0	97.4±0.0	**100.0**±0.0	97.4±0.0	**100.0**±0.0	**98.0**±0.0	**100.0**±0.0
CIFAR100	94.5	96.4	94.5±0.0	96.4±0.0	**97.3**	98.7	94.1±0.0	96.1±0.0	96.8±0.0	98.7±0.0	96.9±0.0	98.7±0.0	97.1±0.0	**98.8**±0.0
FMNIST3D	94.2	96.4	94.2±0.0	96.4±0.0	96.5	98.5	94.1±0.0	96.4±0.0	96.0±0.0	98.2±0.0	97.8±0.0	99.2±0.0	**99.9**±0.0	**100.0**±0.0
UniformNoise	96.8	99.7	96.9±0.1	99.7±0.0	**98.9**	**99.9**	96.7±0.1	99.7±0.0	97.7±0.0	99.8±0.0	97.9±0.0	99.8±0.0	98.8±0.0	**99.9**±0.0
CIFAR100	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SVHN	78.8	63.7	79.3±0.0	64.2±0.0	**84.6**	73.2	84.4±0.0	**73.3**±0.0	80.3±0.0	66.6±0.0	79.9±0.0	65.7±0.0	78.0±0.0	58.7±0.0
LSUN	81.1	99.1	81.2±0.0	99.1±0.0	**83.2**	**99.2**	80.3±0.0	99.1±0.0	82.5±0.1	**99.2**±0.0	82.9±0.1	**99.2**±0.0	82.3±0.0	**99.2**±0.0
CIFAR10	78.7	77.8	78.9±0.0	77.9±0.0	**80.1**	**79.6**	78.1±0.0	77.2±0.0	78.9±0.0	77.6±0.0	78.9±0.0	77.7±0.0	77.9±0.0	75.6±0.0
FMNIST3D	87.4	86.9	87.8±0.0	87.3±0.0	89.0	89.5	85.7±0.0	85.4±0.0	88.5±0.0	88.1±0.0	88.6±0.0	88.2±0.0	**93.3**±0.0	**93.1**±0.0
UniformNoise	93.4	98.5	93.5±0.0	98.5±0.0	86.4	96.9	93.3±0.0	98.5±0.0	94.2±0.0	98.7±0.0	**96.3**±0.0	**99.2**±0.0	95.8±0.0	99.1±0.0

Table D.8: Optimal hyperparameter for each layer (or residual block for ResNet) on LLL.

Datasets	Input	Layer 1	Layer 2	Layer 3	Layer 4
$\mathcal{L}_{\text{LL}}$					
MNIST	3.3939e-08	5.4485e-07	1.1377e-07	2.3509e-03	-
SVHN	9.3995e-04	1.3767e-04	1.1347e-04	2.2835e-04	3.9480e-05
CIFAR10	0.0036	0.0005	0.0008	0.0018	0.0028
CIFAR100	0.0094	0.0093	0.0019	0.0049	0.0144
$\mathcal{L}_{\text{OOD}}$ **(Synthetic)**					
MNIST	1.7384e-05	1.6409e-06	1.3555e-07	2.5206e-03	-
SVHN	8.2850e+00	6.2021e-03	9.1418e-03	4.7633e-03	1.3424e-02
CIFAR10	4.6957e+01	8.4602e-04	1.3050e-03	5.9322e-03	1.9222e-03
CIFAR100	2.6372e+01	2.8527e-03	8.7588e-04	4.5595e-03	2.5490e-01
$\mathcal{L}_{\text{OOD}}$ **(32x32 ImageNet)**					
MNIST	3.5457e-08	5.9255e-07	1.1685e-07	2.4544e-03	-
SVHN	1.1849e-03	1.3038e-01	3.5909e-04	3.8309e-04	8.2367e-05
CIFAR10	0.0236	0.9079	0.0030	0.0049	0.0053
CIFAR100	0.0152	0.9533	0.0051	0.0094	0.2049

Table D.9: UQ performance with ImageNet32x32 as $\mathcal{D}_{\text{out}}$.

Methods	MNIST	CIFAR10	SVHN	CIFAR100
ECE $\downarrow$				
LLL-RGPR-LL	15.8	3.6	13.1	5.7
LLL-RGPR-OOD	19.6	12.5	15.9	15.8
LLL-RGPR-OOD ImageNet	15.8	20.3	18.8	19.3
FPR@95 $\downarrow$				
LLL-RGPR-LL	3.9	29.6	13.8	65.8
LLL-RGPR-OOD	3.6	24.2	9.6	63.0
LLL-RGPR-OOD ImageNet	3.9	39.5	7.3	61.0

Table D.10: Regression far-away outlier detection. Values correspond to predictive error bars (averaged over ten prediction trials), similar to what shades represent in Fig. 3.6. "In" and "Out" correspond to inliers and outliers, respectively.

	housing		concrete		energy		wine	
Methods	**In** $\downarrow$	**Out** $\uparrow$	**In** $\downarrow$	**Out** $\uparrow$	**In** $\downarrow$	**Out** $\uparrow$	**In** $\downarrow$	**Out** $\uparrow$
LLL	0.405	823.215	0.324	580.616	0.252	319.890	0.126	24.176
LLL-RGPR	0.407	**2504.325**	0.329	**3394.466**	0.253	**2138.909**	0.129	**1948.813**
KFL	1.171	2996.606	1.281	2518.338	0.651	1486.748	0.291	475.141
KFL-RGPR	1.165	**3909.140**	1.264	**4258.177**	0.656	**2681.780**	0.292	**2031.481**
SWAG	0.181	440.085	1.192	2770.455	0.418	1066.044	0.181	77.357
SWAG-RGPR	0.186	**2403.366**	1.146	**4693.273**	0.428	**2647.922**	0.187	**1947.677**
SVGP	0.641	2.547	0.845	3.100	0.367	2.237	0.092	0.983
SVGP-RGPR	0.641	**1973.506**	0.845	**1932.061**	0.367	**1931.299**	0.095	**1956.027**

Appendix E
Appendix of Section 4.1

E.0.1 Training

We use the Pyro library (Bingham et al., 2019) to implement the normalizing flow (NF) used for the refinement. The NF is trained by maximizing the evidence lower bound using the Adam optimizer (Kingma & Ba, 2015) and the cosine learning rate decay (Loshchilov & Hutter, 2017) for 20 epochs, with an initial learning rate of 0.001. Following (Izmailov et al., 2021b), we do not use data augmentation.

For the HMC baseline, we use the default implementation of NUTS in Pyro. We confirm that the HMC used in our experiments are well-converged: The average Gelman-Rubin $\widehat{R}$'s are 0.998, 0.999, 0.997, and 1.096—below the standard threshold of 1.1—for the last-layer F-MNIST, last-layer CIFAR-10, last-layer CIFAR-100, and all-layer F-MNIST experiments, respectively.

For the MAP, VB, and CSGHMC baselines, we use the same settings as Daxberger et al. (2021a): We train them for 100 epochs with an initial learning rate of 0.1, annealed via the cosine decay method (Loshchilov & Hutter, 2017). The minibatch size is 128, and data augmentation is employed. For MAP, we use weight decay of 5×10^{-4}. For VB and CSGHMC, we use the prior precision corresponding to the previous weight decay value.

For the LA baseline, we use the `laplace-torch` library (Daxberger et al., 2021a). The diagonal Hessian is used for CIFAR-100 and all-layer F-MNIST, while the full Hessian is used for other cases. Following the current best-practice in LA, we tune the prior precision with *post hoc* marginal likelihood maximization (Daxberger et al., 2021a).

Finally, for methods which require validation data, e.g. HMC (for finding the optimal prior precision), we obtain a validation test set of size 2000 by randomly splitting a test set. Note that, these validation data are not used for testing.

E.0.2 Datasets

For the dataset-shift experiment, we use the following test sets: Rotated F-MNIST and Corrupted CIFAR-10 (Hendrycks & Dietterich, 2019; Ovadia et al., 2019). Meanwhile, we use the following OOD test sets for each the in-distribution training set:

- **F-MNIST:** MNIST, K-MNIST, E-MNIST.
- **CIFAR-10:** LSUN, SVHN, CIFAR-100.
- **CIFAR-100:** LSUN, SVHN, CIFAR-10.

Table E.1: In-distribution calibration performance.

	F-MNIST			CIFAR-10			CIFAR-100		
Methods	**Acc.** ↑	**Brier** ↓	**ECE** ↓	**Acc.** ↑	**Brier** ↓	**ECE** ↓	**Acc.** ↑	**Brier** ↓	**ECE** ↓
MAP	90.4±0.1	0.1445±0.0008	11.7±0.3	94.8±0.1	0.0790±0.0004	10.5±0.2	76.5±0.1	0.3396±0.0012	13.7±0.2
LA	90.4±0.0	0.1439±0.0008	11.1±0.2	94.8±0.0	0.0785±0.0004	9.8±0.3	75.6±0.1	0.3529±0.0009	9.7±0.1
LA-Refine-1	90.4±0.1	0.1386±0.0007	5.2±0.2	94.7±0.0	0.0776±0.0003	4.3±0.2	75.9±0.1	0.3445±0.0010	8.0±0.2
LA-Refine-5	90.4±0.1	0.1375±0.0009	3.2±0.1	94.8±0.1	0.0768±0.0004	4.3±0.2	76.2±0.1	0.3311±0.0007	4.5±0.2
LA-Refine-10	90.5±0.1	0.1376±0.0008	3.6±0.1	94.9±0.1	0.0765±0.0004	4.4±0.2	76.1±0.1	0.3312±0.0008	4.4±0.1
LA-Refine-30	90.4±0.1	0.1376±0.0009	3.5±0.1	94.9±0.1	0.0765±0.0004	4.4±0.1	76.1±0.1	0.3315±0.0007	4.2±0.2
HMC	90.4±0.1	0.1375±0.0009	3.4±0.0	94.9±0.1	0.0765±0.0004	4.3±0.1	76.4±0.1	0.3283±0.0007	4.6±0.1

Table E.2: Calibration of all-layer BNNs on F-MNIST. The architecture is two-layer ReLU fully-connected network with 50 hidden units.

Methods	**MMD** ↓	**Acc.** ↑	**NLL** ↓	**Brier** ↓	**ECE** ↓
LA	0.278±0.003	88.0±0.1	0.3597±0.0009	0.18±0.0006	7.7±0.1
LA-Refine-1	0.194±0.006	87.6±0.1	0.3564±0.0015	0.1807±0.0006	6.1±0.1
LA-Refine-5	0.19±0.006	87.6±0.1	0.3483±0.0012	0.1781±0.0005	4.9±0.3
LA-Refine-10	0.186±0.006	87.7±0.1	0.3459±0.0008	0.1771±0.0004	4.7±0.3
LA-Refine-30	0.183±0.006	87.8±0.1	0.3432±0.0014	0.176±0.0007	4.6±0.3
HMC	-	89.7±0.0	0.2908±0.0002	0.1502±0.0001	4.5±0.1

E.0.3 Image classification

To complement Table 4.3 in the main text, we present results for additional metrics (accuracy, Brier score, and ECE) in Table E.1. We see that the trend Table 4.3 is also observable here. In Table E.2, we observe that refining an *all-layer* posterior improves its predictive quality further.[1]

In Table E.3, we present the detailed, non-averaged results to complement Table 4.4. Moreover, we also present dataset-shift results on standard benchmark problems (Rotated F-MNIST and Corrupted CIFAR-10). In both cases, we observe that the performance of the refined posterior approaches HMC's.

[1] The network is a two-layer fully-connected ReLU network with 50 hidden units.

E Appendix of Section 4.1

Table E.3: Detailed OOD detection results. Values are FPR95. "LA-R" stands for "LA-Refine".

Datasets	VB*	CSGHMC*	LA	LA-R-1	LA-R-5	LA-R-10	LA-R-30	HMC
FMNIST								
EMNIST	83.4±0.6	86.5±0.5	84.7±0.7	85.4±0.8	87.6±0.6	87.6±0.6	87.6±0.6	87.2±0.6
MNIST	76.0±1.6	75.8±1.8	77.9±0.8	77.5±0.9	79.6±1.0	79.6±1.0	79.6±1.1	79.0±0.9
KMNIST	71.3±0.9	74.4±0.5	78.5±0.6	78.3±0.8	79.9±0.9	79.9±0.9	79.9±0.9	79.4±0.9
CIFAR-10								
SVHN	66.1±1.2	59.8±1.4	38.3±2.9	40.1±3.4	38.2±3.2	36.5±3.0	35.8±2.9	36.0±3.0
LSUN	53.3±2.5	51.7±1.5	51.1±1.1	46.9±0.5	46.7±0.6	46.9±0.7	47.1±0.5	46.7±0.8
CIFAR-100	69.3±0.2	64.6±0.3	58.2±0.8	56.1±0.6	55.7±0.8	55.3±0.6	55.2±0.5	55.3±0.4
CIFAR-100								
SVHN	81.7±0.7	75.9±1.5	82.2±0.8	77.7±1.2	78.1±1.3	77.9±1.5	78.3±1.6	78.2±1.6
LSUN	76.6±1.8	79.3±1.8	75.5±1.6	75.1±1.2	75.7±1.4	75.9±1.2	75.8±1.3	75.5±1.7
CIFAR-10	84.2±0.4	82.8±0.3	81.0±0.3	79.1±0.4	79.5±0.4	79.5±0.4	79.6±0.4	79.7±0.2

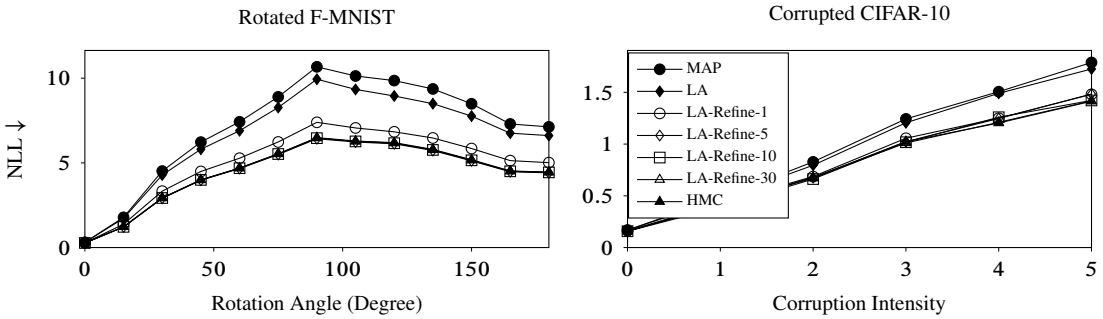


Figure E.1: Calibration under dataset shifts in terms of NLL—lower is better.

Appendix F
Appendix of Section 4.2

F.1 UCI Regression

To validate the performance of LULA in regressions, we employ a subset of the UCI regression benchmark datasets. Following previous works, the network architecture used here is a single-hidden-layer ReLU network with 50 hidden units. The data are standardized to have zero mean and unit variance. We use 50 LULA units and optimize them for 40 epochs using OOD data sampled uniformly from $[-10, 10]^n$. For LA and LULA, each prediction is done via MC-integration with 100 samples. For the evaluation of each dataset, we use a 60-20-20 train-validation-test split. We repeat each train-test process 10 times and take the average.

In Table F.1 we report the average predictive standard deviation for each dataset. Note that this metric is the direct generalization of the 1D uncertainty estimates in Fig. 4.8 to multi-dimension. The test outliers are sampled uniformly from $[-10, 10]^n$. Note that since the inlier data are centered around the origin and have unit variance, they lie approximately in a Euclidean ball with a radius of 2. Therefore, these outliers are far away from them. Thus, naturally, high uncertainty values over these outliers are desirable. Uncertainties over the test sets are generally low for all methods, although LULA has slightly higher uncertainties compared to the base LA. However, LULA yield much higher uncertainties over outliers across all datasets, significantly more than the baselines. Moreover, in Table F.2, we show that LULA maintains the predictive performance of the base LA. Altogether, they imply that LULA can detect outliers better than other methods without costing the predictive performance.

F.2 Image Classification

We present the detailed results on OOD detection in terms of MMC, FPR95 (Tables F.3 and F.4), and additionally area under ROC (AUROC) and precision-recall (AUPRC) curves (Tables F.5 and F.6). We use standard datasets: EMNIST, KMNIST, FMNIST, and LSUN. Furthermore, we use the following artificial datasets:

- GRAYCIFAR10: obtained by converting CIFAR-10 test data into grayscale images.
- UNIFORMNOISE: obtained by uniformly sampling from the hypercube $[0, 1]^n$.
- SMOOTHEDNOISE: obtained by permuting, blurring, and contrast re-scaling the original test images (Hein et al., 2019).
- FMNIST3D: obtained by converting the grayscale FMNIST images into 3-channel images.

Table F.1: UQ performances on UCI datasets. Values are the average (over all data points and ten training-prediction trials) predictive standard deviations. Lower is better for test data and vice-versa for outliers. By definition, MAP does not have (epistemic) uncertainty.

	Test set ↓			Outliers ↑		
Dataset	**DE**	**LA**	**LA-LULA**	**DE**	**LA**	**LA-LULA**
Housing	5.82	1.26	1.37	145.33	222.76	377.92
Concrete	8.11	10.44	16.89	964.63	30898.92	83241.42
Energy	4.40	1.05	1.08	126.11	1070.09	5163.53
Kin8nm	0.10	0.14	0.18	2.12	0.80	2.12
Power	19.85	2.85	3.20	12235.87	4148.98	221287.80
Wine	0.64	1.15	1.22	28.57	186.76	21383.17
Yacht	5.17	2.08	2.78	187.41	5105.69	13119.99

Table F.2: Predictive performances on UCI regression datasets in terms of average test log-likelihood. The numbers reported are averages over ten training-prediction runs along with the corresponding standard deviations. The performances of LULA are similar to LA's. The differences between their exact values are likely due to MC-integration.

Dataset	**MAP**	**DE**	**LA**	**LA-LULA**
Housing	-2.794±0.012	-3.045±0.009	-3.506±0.055	-3.495±0.047
Concrete	-3.409±0.036	-3.951±0.062	-4.730±0.205	-4.365±0.094
Energy	-2.270±0.128	-2.673±0.015	-2.707±0.030	-2.698±0.014
Kin8nm	-0.923±0.000	1.086±0.022	-0.965±0.003	-0.969±0.003
Power	-3.154±0.002	-54.804±7.728	-3.273±0.015	-3.277±0.024
Wine	-1.190±0.014	-1.038±0.018	-1.624±0.075	-1.630±0.092
Yacht	-1.835±0.053	-3.272±0.079	-2.509±0.367	-2.663±0.276

We observe that LULA consistently improves the base LA. Especially, LULA makes the confidence estimates over OOD data lower without introducing underconfidence on in-distribution data.

Table F.3: Detailed MMC results. Values are averages over five prediction runs.

Dataset	MAP	MAP-Temp	DE	DE-Temp	LA	LA-LULA	LLLA	LLLA-LULA	OE	OE-LULA
MNIST	99.8	99.8±0.0	99.7	99.8±0.0	99.7±0.0	98.3±0.0	99.3±0.0	99.2±0.0	99.4±0.0	71.5±0.4
EMNIST	84.6	86.2±0.0	82.8	87.4±0.0	84.1±0.0	56.6±0.5	73.7±0.2	67.4±0.3	81.2±0.0	31.7±0.2
KMNIST	71.3	73.8±0.0	67.8	76.1±0.0	70.5±0.0	34.7±0.4	56.4±0.3	45.6±0.5	66.5±0.0	23.7±0.1
FMNIST	76.7	79.0±0.0	69.6	80.1±0.0	75.7±0.0	37.6±0.9	57.3±0.3	50.3±1.2	32.6±0.0	22.4±0.1
GrayCIFAR10	68.2	71.0±0.0	55.4	66.7±0.0	66.9±0.0	32.4±0.6	46.2±0.2	42.5±0.7	10.2±0.0	22.1±0.2
UniformNoise	82.0	83.7±0.1	67.4	94.6±0.1	75.7±0.4	29.4±0.7	36.0±0.9	39.6±1.2	10.1±0.0	21.7±0.7
Noise	99.4	99.7±0.0	99.5	99.9±0.0	99.4±0.0	85.6±1.5	96.4±0.2	95.9±0.6	10.4±0.0	14.2±0.1
SVHN	98.5	97.1±0.0	98.1	97.5±0.0	98.5±0.0	97.5±0.0	91.8±0.5	95.9±0.1	98.4±0.0	98.4±0.0
CIFAR10	72.5	62.4±0.0	58.7	58.1±0.0	71.8±0.0	60.8±0.1	48.5±0.2	52.3±0.4	10.7±0.0	13.3±0.2
LSUN	73.7	63.9±0.0	59.0	59.6±0.0	73.0±0.0	61.5±0.2	48.2±0.3	52.5±0.5	10.3±0.0	12.8±0.3
CIFAR100	73.4	63.5±0.0	60.0	59.6±0.0	72.7±0.0	61.6±0.1	48.9±0.2	52.9±0.4	11.3±0.0	14.0±0.3
FMNIST3D	74.6	64.8±0.0	64.1	61.4±0.0	74.0±0.0	65.2±0.2	53.3±0.4	57.6±0.4	10.6±0.0	13.7±0.2
UniformNoise	79.1	70.8±0.1	54.6	63.8±0.2	77.8±0.2	62.5±0.6	43.9±0.5	51.6±0.3	10.0±0.0	12.4±0.3
Noise	64.2	55.1±0.2	53.3	51.7±0.2	63.5±0.2	53.6±0.1	41.3±0.2	45.8±0.4	55.3±0.1	54.3±0.1
CIFAR10	97.2	94.8±0.0	96.1	95.7±0.0	96.9±0.0	96.2±0.0	90.6±0.0	83.4±0.2	97.3±0.0	97.0±0.0
SVHN	70.6	57.2±0.0	57.2	52.6±0.0	67.7±0.1	63.2±0.3	42.1±0.5	35.0±0.5	56.1±0.0	53.3±0.1
LSUN	74.8	61.5±0.0	65.6	61.8±0.0	73.4±0.0	68.7±0.2	51.3±0.3	40.5±0.4	66.2±0.0	64.4±0.1
CIFAR100	78.7	67.1±0.0	71.2	68.3±0.0	77.3±0.0	73.4±0.1	56.6±0.1	46.6±0.2	78.1±0.0	76.6±0.0
FMNIST3D	68.8	53.7±0.0	60.7	54.7±0.0	66.5±0.1	61.2±0.1	40.4±0.4	32.9±0.3	61.4±0.0	59.2±0.0
UniformNoise	88.0	71.5±0.1	89.3	82.2±0.0	79.5±0.6	62.6±1.6	30.7±0.6	25.2±0.2	10.1±0.0	12.2±0.1
Noise	64.5	52.2±0.2	53.7	52.6±0.1	59.6±0.2	53.8±0.3	35.5±0.4	30.3±0.4	48.6±0.3	46.4±0.2
CIFAR100	85.7	76.8±0.0	81.5	80.6±0.0	80.4±0.0	72.6±0.1	75.7±0.1	63.8±0.2	86.5±0.0	81.2±0.1
SVHN	61.3	42.0±0.0	47.5	42.2±0.0	52.9±0.1	40.7±0.5	46.8±0.6	33.0±0.8	63.7±0.0	54.6±0.2
LSUN	64.9	47.8±0.0	51.7	49.3±0.0	56.0±0.2	46.1±0.1	49.1±0.4	37.5±0.8	58.4±0.0	50.8±0.3
CIFAR10	67.2	51.8±0.0	56.1	54.4±0.0	58.9±0.1	49.8±0.1	52.6±0.1	41.6±0.2	68.8±0.0	59.7±0.1
FMNIST3D	56.4	35.7±0.0	45.8	39.2±0.0	49.0±0.1	40.1±0.3	42.7±0.2	32.6±0.3	53.9±0.0	46.2±0.2
UniformNoise	68.3	56.5±0.1	29.5	43.7±0.1	45.3±0.5	33.0±0.8	36.5±0.9	24.7±0.8	1.7±0.0	1.7±0.0
Noise	68.7	55.3±0.2	50.5	50.2±0.3	58.1±0.2	36.3±1.0	51.2±1.0	29.0±1.4	64.5±0.2	53.8±0.4

Table F.4: Detailed FPR95 results. Values are averages over five prediction runs.

Dataset	MAP	MAP-Temp	DE	DE-Temp	LA	LA-LULA	LLLA	LLLA-LULA	OE	OE-LULA
MNIST	-	-	-	-	-	-	-	-	-	-
EMNIST	23.9	24.0±0.0	22.3	22.4±0.0	23.9±0.0	23.6±0.2	24.0±0.2	23.5±0.1	27.5±0.0	23.5±0.6
KMNIST	2.4	2.4±0.0	1.8	2.3±0.0	2.4±0.0	0.8±0.0	1.8±0.2	1.0±0.1	5.1±0.0	3.6±0.4
FMNIST	2.4	2.4±0.0	1.1	1.8±0.0	2.3±0.0	0.8±0.0	1.5±0.1	0.9±0.1	0.2±0.0	1.8±0.2
GrayCIFAR10	0.1	0.0±0.0	0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.8±0.2
UniformNoise	1.1	1.0±0.0	0.0	0.2±0.0	0.3±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.9±0.5
Noise	0.1	0.1±0.0	0.2	0.2±0.1	0.1±0.0	9.4±3.0	7.6±2.2	1.4±0.3	0.0±0.0	0.2±0.2
SVHN	-	-	-	-	-	-	-	-	-	-
CIFAR10	24.0	23.2±0.0	11.3	14.1±0.0	23.8±0.1	20.7±0.2	23.7±1.8	19.6±0.4	0.0±0.0	0.0±0.0
LSUN	25.7	25.3±0.0	11.0	16.3±0.0	25.5±0.2	21.3±0.5	22.2±2.2	19.7±0.8	0.0±0.0	0.0±0.0
CIFAR100	25.5	24.8±0.0	13.3	16.8±0.0	25.3±0.1	21.9±0.2	24.3±1.6	20.5±0.4	0.2±0.0	0.1±0.0
FMNIST3D	29.7	28.9±0.0	22.5	22.5±0.0	29.8±0.1	29.4±0.3	33.0±1.6	29.2±0.3	0.0±0.0	0.0±0.0
UniformNoise	33.2	34.1±0.3	5.4	18.4±0.3	31.7±0.3	19.5±1.1	14.0±2.1	15.6±0.6	0.0±0.0	0.0±0.0
Noise	17.5	17.0±0.5	7.8	10.0±0.5	17.1±0.5	13.6±0.3	14.7±1.7	12.1±0.7	10.3±0.1	10.2±0.2
CIFAR10	-	-	-	-	-	-	-	-	-	-
SVHN	41.7	35.4±0.0	25.0	20.1±0.0	38.9±0.2	37.6±0.5	19.6±0.8	20.4±1.3	22.8±0.0	20.9±0.1
LSUN	50.7	45.7±0.0	45.3	39.3±0.0	50.9±0.2	48.9±0.5	41.9±0.2	37.4±1.1	38.3±0.0	38.4±0.2
CIFAR100	60.1	55.9±0.0	54.6	51.7±0.0	59.7±0.3	58.7±0.2	51.4±0.4	50.4±0.4	58.0±0.0	57.5±0.2
FMNIST3D	40.4	31.3±0.0	35.2	27.0±0.0	39.0±0.3	36.3±0.3	19.1±0.6	17.4±0.4	30.0±0.0	29.2±0.2
UniformNoise	89.0	81.6±0.5	99.9	99.3±0.1	73.9±1.7	31.3±5.9	0.1±0.1	0.7±0.3	0.0±0.0	0.0±0.0
Noise	36.6	31.8±0.5	25.7	31.6±0.2	28.9±0.4	24.3±0.7	9.9±0.7	11.0±1.0	15.4±0.2	14.0±0.3
CIFAR100	-	-	-	-	-	-	-	-	-	-
SVHN	73.8	67.9±0.0	62.1	58.2±0.0	73.3±0.3	68.8±0.6	72.4±0.9	67.5±0.9	75.9±0.0	74.1±0.3
LSUN	81.7	81.7±0.0	73.0	75.3±0.0	82.4±0.6	82.1±0.4	81.7±0.6	81.0±0.8	69.7±0.0	71.0±0.8
CIFAR10	83.0	81.5±0.0	77.2	78.2±0.0	82.9±0.2	82.8±0.3	82.3±0.2	82.7±0.2	82.4±0.0	81.5±0.2
FMNIST3D	70.2	59.5±0.0	64.3	58.8±0.0	70.6±0.2	69.1±0.6	69.1±0.3	67.8±0.9	63.1±0.0	62.9±0.4
UniformNoise	97.7	100.0±0.0	15.7	99.5±0.1	89.1±1.1	71.2±4.3	76.9±3.1	57.5±5.6	0.0±0.0	0.0±0.0
Noise	74.2	72.0±0.5	63.3	63.8±0.4	71.6±0.5	57.1±1.8	69.9±0.5	54.3±1.6	66.6±0.3	61.6±0.8

F Appendix of Section 4.2

Table F.5: Detailed AUROC results. Values are averages over five prediction runs.

Dataset	MAP	MAP-Temp	DE	DE-Temp	LA	LA-LULA	LLLA	LLLA-LULA	OE	OE-LULA
MNIST	-	-	-	-	-	-	-	-	-	-
EMNIST	89.5	89.5±0.0	89.8	89.6±0.0	89.5±0.0	90.6±0.2	89.6±0.1	90.3±0.0	92.9±0.0	93.2±0.1
KMNIST	98.9	98.9±0.0	99.1	98.9±0.0	98.9±0.0	99.5±0.0	99.3±0.0	99.5±0.0	98.6±0.0	98.9±0.1
FMNIST	98.8	98.8±0.0	99.2	99.0±0.0	98.9±0.0	99.3±0.0	99.3±0.0	99.3±0.1	99.7±0.0	99.2±0.0
GrayCIFAR10	99.7	99.6±0.0	99.8	99.8±0.0	99.7±0.0	99.6±0.0	99.8±0.0	99.7±0.0	100.0±0.0	99.3±0.0
UniformNoise	99.1	99.2±0.0	99.8	99.1±0.0	99.5±0.0	99.8±0.0	100.0±0.0	99.8±0.0	100.0±0.0	99.4±0.1
Noise	97.4	97.3±0.0	96.9	96.8±0.0	97.4±0.0	96.3±0.3	96.7±0.1	96.7±0.1	100.0±0.0	99.9±0.0
SVHN	-	-	-	-	-	-	-	-	-	-
CIFAR10	95.2	95.3±0.0	97.7	97.2±0.0	95.3±0.0	96.2±0.1	95.5±0.3	96.5±0.0	100.0±0.0	100.0±0.0
LSUN	94.9	94.9±0.0	97.9	96.9±0.0	94.9±0.0	96.0±0.1	95.8±0.3	96.5±0.1	100.0±0.0	100.0±0.0
CIFAR100	94.6	94.6±0.0	97.2	96.5±0.0	94.7±0.0	95.8±0.0	95.3±0.3	96.2±0.0	100.0±0.0	100.0±0.0
FMNIST3D	94.2	94.3±0.0	96.2	96.0±0.0	94.2±0.0	94.4±0.1	93.0±0.5	94.3±0.1	100.0±0.0	100.0±0.0
UniformNoise	93.8	93.4±0.1	98.5	96.5±0.0	94.1±0.1	96.6±0.2	97.4±0.2	97.3±0.1	100.0±0.0	100.0±0.0
Noise	96.6	96.6±0.1	98.3	97.9±0.1	96.6±0.1	97.4±0.0	97.2±0.3	97.7±0.1	97.9±0.1	97.9±0.1
CIFAR10	-	-	-	-	-	-	-	-	-	-
SVHN	94.6	95.3±0.0	96.6	97.1±0.0	94.9±0.0	95.0±0.1	96.9±0.1	96.6±0.2	97.0±0.0	97.2±0.0
LSUN	92.5	93.5±0.0	93.7	94.3±0.0	92.5±0.0	92.8±0.1	93.2±0.1	93.9±0.2	94.9±0.0	94.9±0.0
CIFAR100	90.0	90.6±0.0	91.1	91.6±0.0	90.1±0.0	90.1±0.0	90.2±0.1	90.0±0.1	90.1±0.0	90.2±0.0
FMNIST3D	94.7	95.8±0.0	95.3	96.3±0.0	94.9±0.0	95.3±0.0	97.0±0.1	97.2±0.1	95.9±0.0	96.0±0.0
UniformNoise	91.5	92.6±0.0	88.6	91.0±0.0	93.6±0.1	96.2±0.3	99.4±0.1	99.3±0.0	100.0±0.0	100.0±0.0
Noise	95.2	95.7±0.1	96.6	95.9±0.1	96.0±0.1	96.7±0.1	98.1±0.1	98.0±0.1	97.1±0.1	97.4±0.1
CIFAR100	-	-	-	-	-	-	-	-	-	-
SVHN	80.2	83.9±0.0	85.0	86.7±0.0	80.5±0.1	83.5±0.4	80.7±0.4	84.1±0.7	80.1±0.0	80.2±0.2
LSUN	78.1	80.1±0.0	82.5	82.7±0.0	78.5±0.2	79.1±0.1	79.4±0.4	79.8±0.9	83.7±0.0	83.2±0.2
CIFAR10	75.4	76.4±0.0	78.7	78.6±0.0	75.5±0.1	75.4±0.2	75.8±0.1	75.3±0.2	75.4±0.0	75.8±0.0
FMNIST3D	84.1	88.2±0.0	86.6	89.1±0.0	83.7±0.1	84.1±0.2	84.3±0.2	84.5±0.3	86.4±0.0	86.2±0.1
UniformNoise	78.7	75.5±0.1	96.8	88.5±0.0	88.1±0.4	91.2±0.6	90.4±0.6	93.0±0.6	100.0±0.0	100.0±0.0
Noise	69.3	71.5±0.2	80.9	78.1±0.2	74.3±0.3	86.2±0.9	75.5±0.9	87.0±1.4	75.1±0.2	78.5±0.4

Table F.6: Detailed AUPRC results. Values are averages over five prediction runs.

Dataset	MAP	MAP-Temp	DE	DE-Temp	LA	LA-LULA	LLLA	LLLA-LULA	OE	OE-LULA
MNIST	-	-	-	-	-	-	-	-	-	-
EMNIST	67.3	67.2±0.0	67.0	66.5±0.0	67.3±0.2	69.7±0.7	67.9±0.4	69.1±0.3	84.5±0.0	81.0±0.3
KMNIST	97.9	97.9±0.0	98.4	98.0±0.0	98.0±0.0	99.5±0.0	99.2±0.1	99.4±0.0	98.6±0.0	99.0±0.0
FMNIST	98.3	98.4±0.0	98.9	98.5±0.0	98.4±0.0	99.3±0.0	99.1±0.0	99.3±0.1	99.7±0.0	99.3±0.0
GrayCIFAR10	99.7	99.7±0.0	99.9	99.8±0.0	99.7±0.0	99.7±0.0	99.8±0.0	99.7±0.0	100.0±0.0	99.4±0.0
UniformNoise	99.8	99.8±0.0	100.0	99.8±0.0	99.9±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	99.9±0.0
Noise	99.5	99.4±0.0	99.4	99.3±0.0	99.5±0.0	99.2±0.1	99.3±0.0	99.3±0.0	100.0±0.0	100.0±0.0
SVHN	-	-	-	-	-	-	-	-	-	-
CIFAR10	97.8	97.8±0.0	99.1	98.8±0.0	97.8±0.0	98.3±0.0	98.1±0.2	98.5±0.0	100.0±0.0	100.0±0.0
LSUN	99.9	99.9±0.0	100.0	100.0±0.0	99.9±0.0	99.9±0.0	99.9±0.0	100.0±0.0	100.0±0.0	100.0±0.0
CIFAR100	97.4	97.3±0.0	98.8	98.4±0.0	97.4±0.0	98.0±0.0	97.9±0.2	98.3±0.0	100.0±0.0	100.0±0.0
FMNIST3D	97.4	97.4±0.0	98.5	98.3±0.0	97.3±0.0	97.5±0.0	96.8±0.3	97.4±0.0	100.0±0.0	100.0±0.0
UniformNoise	99.4	99.3±0.0	99.9	99.7±0.0	99.4±0.0	99.7±0.0	99.8±0.0	99.8±0.0	100.0±0.0	100.0±0.0
Noise	99.7	99.7±0.0	99.8	99.8±0.0	99.7±0.0	99.8±0.0	99.8±0.0	99.8±0.0	99.8±0.0	99.8±0.0
CIFAR10	-	-	-	-	-	-	-	-	-	-
SVHN	91.5	92.2±0.0	94.5	95.0±0.0	91.9±0.0	92.0±0.1	94.3±0.2	93.9±0.3	94.3±0.0	94.6±0.0
LSUN	99.7	99.7±0.0	99.7	99.8±0.0	99.7±0.0	99.7±0.0	99.7±0.0	99.7±0.0	99.8±0.0	99.8±0.0
CIFAR100	90.3	90.6±0.0	91.2	91.6±0.0	90.3±0.0	90.3±0.0	89.6±0.1	89.1±0.1	90.3±0.0	90.3±0.0
FMNIST3D	95.3	96.1±0.0	95.7	96.6±0.0	95.5±0.0	95.7±0.0	97.0±0.1	97.2±0.1	96.1±0.0	96.2±0.0
UniformNoise	98.1	98.4±0.0	97.5	98.1±0.0	98.6±0.0	99.2±0.1	99.9±0.0	99.8±0.0	100.0±0.0	100.0±0.0
Noise	98.8	98.9±0.0	99.1	99.0±0.0	99.0±0.0	99.2±0.0	99.5±0.0	99.5±0.0	99.2±0.0	99.3±0.0
CIFAR100	-	-	-	-	-	-	-	-	-	-
SVHN	67.8	72.3±0.0	73.1	75.4±0.0	67.4±0.2	71.7±1.0	66.5±0.9	71.9±1.6	69.4±0.0	68.3±0.3
LSUN	99.0	99.1±0.0	99.2	99.2±0.0	99.0±0.0	99.0±0.0	99.0±0.0	99.0±0.1	99.3±0.0	99.2±0.0
CIFAR10	74.7	75.2±0.0	77.8	77.6±0.0	74.4±0.1	73.8±0.2	74.6±0.3	73.1±0.4	75.3±0.0	75.3±0.1
FMNIST3D	85.0	88.5±0.0	87.5	89.6±0.0	84.3±0.1	84.2±0.2	84.5±0.2	84.3±0.3	87.1±0.0	86.6±0.1
UniformNoise	94.7	94.0±0.0	99.3	97.4±0.0	97.2±0.1	98.0±0.2	97.8±0.2	98.4±0.1	100.0±0.0	100.0±0.0
Noise	90.2	91.0±0.1	94.2	92.8±0.1	92.2±0.1	96.2±0.3	92.5±0.4	96.4±0.5	92.1±0.1	93.3±0.2

Appendix G
Appendix of Section 4.3

G.1 OOD Test Sets

For image-based OOD detection tasks, we use the following test sets on top of MNIST, F-MNIST, SVHN, CIFAR-10, and CIFAR-100:

- E-MNIST: Contains handwritten letters ("a"-"z")—same format as MNIST (Cohen et al., 2017).
- K-MNIST: Contains handwritten Hiragana scripts—same format as MNIST (Clanuwat et al., 2018).
- LSUN-CR: Contains real-world images of classrooms (Yu et al., 2015).
- CIFAR-GR: Obtained by converting CIFAR-10 test images to grayscale.
- F-MNIST-3D: Obtained by converting single-channel F-MNIST images into images of three channels—all these three channels have identical values.
- UNIFORM: Obtained by drawing independent uniformly-distributed random pixel.
- SMOOTH: Obtained by permuting, smoothing, and contrast-rescaling the original (i.e. the respective in-distribution) test images (Hein et al., 2019).

Meanwhile, for text classification, we use the following OOD test set, following (Hendrycks et al., 2019):

- MULTI30K: Multilingual English-German image description dataset (Elliott et al., 2016).
- WMT16: Machine-translation dataset, avaliable at `http://www.statmt.org/wmt14/translation-task.html`.
- SNLI: Collection of human-written English sentence pairs manually labeled for balanced classification with the labels entailment, contradiction, and neutral (Bowman et al., 2015).

Finally, for dataset-shift robustness tasks, we use the standard dataset:

- CIFAR-10-C: Contains 19 different perturbations—e.g. snow, motion blur, brightness rescaling—with 5 level of severity for a total of 95 distinct shifts (Hendrycks & Dietterich, 2019; Ovadia et al., 2019).

G.2 Training Details

Non-Bayesian For MNIST and F-MNIST, we use a five-layer LeNet architecture. Meanwhile, for SVHN, CIFAR-10, and CIFAR-100, we use WideResNet-16-4 (Zagoruyko & Komodakis, 2016). For all methods, the training procedures are as follows. For LeNet, we use Adam with initial learning rate of 1×10^{-3} and annealed it using the cosine decay method (Loshchilov &

Hutter, 2017) along with weight decay of 5×10^{-4} for 100 epochs. We use a batch size of 128 for both in- and out-distribution batches, amounting to an effective batch size of 256 in the case of OOD training. The standard data augmentation pipeline (random crop and horizontal flip) is applied to both in-distribution and OOD data. For WideResNet-16-4, we use SGD instead with an initial learning rate of 1×10^{-1} and Nesterov momentum of 0.9 along with the dropout regularization with rate 0.3—all other hyperparameters are identical to LeNet. Finally, we use 5 ensemble members for DE.

Bayesian For both LA, VB, and their variants (i.e. LA+X and VB+X), we use the identical setup as in the non-Bayesian training above. Additionally, for LA and LA+X, we use the diagonal Fisher matrix as the approximate Hessian. Moreover, we tune prior variance by minimizing the validation Brier score. All predictions are done using 20 MC samples. For VB and VB+X, we use a diagonal Gaussian variational posterior for both the last-layer weight matrix and bias vector. Moreover, the prior is a zero-mean isotropic Gaussian with prior precision 5×10^{-4} (to emulate the choice of the weight decay in the non-Bayesian training). The trade-off hyperparameter τ of the ELBO is set to the standard value of 0.1 (Osawa et al., 2019; Zhang et al., 2018). We do not use weight decay on the last layer since the regularization of its parameters is done by the KL-term of the ELBO. Lastly, we use 5 and 200 MC samples for computing the ELBO and for making predictions, respectively.

Text Classification The network used is a two-layer Gated Recurrent Unit (GRU, Cho et al., 2014) with 128 hidden units on each layer. The word-embedding dimension is 50 and the maximum vocabulary size is 10000. We put an affine layer on top of the last GRU output to translate the hidden units to output units. Both the LA and VB are applied only on this layer. We use a batch size of 64 and Adam optimizer with a learning rate of 0.01 without weight decay, except for LA in which case we use weight decay of 5×10^{-4}. The optimization is done for 5 epochs, following (Hendrycks et al., 2019).

G.3 Additional Results

The detailed, non-averaged results for the FPR95 metric are in Table G.1. Furthermore, additional results with the area under ROC curve (AUROC), area under precision-recall curve (AUPRC), and mean confidence (MMC) metrics are in Tables G.1 to G.4, respectively. For the full results for models trained with the SMOOTH noise dataset as $\mathcal{D}_{\text{out}}$ are in Tables G.5 and G.6. Furthermore, the full results of the NLP experiment is in Tables G.7 and G.8. Finally, detailed, non-averaged results for sophisticated models (Flipout and CSGHMC) are in Tables G.9 and G.10.

Table G.1: OOD data detection in terms of FPR95. Lower is better. Values are averages over five prediction runs.

				VB					LA				
Datasets	**MAP**	**OE**	**DE**	**Plain**	**NC**	**SL**	**ML**	**OE**	**Plain**	**NC**	**SL**	**ML**	**OE**
MNIST													
F-MNIST	11.8±0.0	0.0±0.0	5.3±0.0	12.5±0.1	0.1±0.0	0.0±0.0	0.4±0.0	1.1±0.0	12.0±0.0	0.2±0.0	0.0±0.0	0.1±0.0	0.0±0.0
E-MNIST	35.6±0.0	26.4±0.0	30.4±0.0	34.5±0.1	34.7±0.1	14.3±0.1	34.2±0.1	31.4±0.1	35.8±0.1	30.6±0.0	12.6±0.1	26.8±0.1	26.7±0.1
K-MNIST	14.4±0.0	5.9±0.0	7.7±0.0	14.0±0.1	10.5±0.1	2.1±0.0	9.7±0.1	8.5±0.0	14.5±0.1	8.9±0.0	0.7±0.0	5.8±0.0	5.9±0.0
CIFAR-Gr	0.2±0.0	0.0±0.0	0.0±0.0	0.2±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.2±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Uniform	44.3±0.0	0.0±0.0	19.8±0.0	93.1±0.2	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	54.2±0.4	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Smooth	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
F-MNIST													
MNIST	73.5±0.0	38.5±0.0	65.8±0.0	66.8±0.1	43.5±0.1	9.5±0.0	50.1±0.1	57.2±0.1	72.2±0.2	25.6±0.1	11.5±0.1	38.9±0.3	39.9±0.1
E-MNIST	73.6±0.0	21.0±0.0	58.6±0.0	68.1±0.1	18.7±0.0	5.0±0.0	34.0±0.0	40.6±0.0	72.2±0.2	6.0±0.0	4.6±0.1	14.7±0.1	23.1±0.1
K-MNIST	73.7±0.0	37.4±0.0	47.2±0.0	62.6±0.1	28.0±0.1	10.6±0.0	33.4±0.0	36.7±0.0	71.5±0.2	18.2±0.1	8.7±0.1	32.5±0.2	38.7±0.3
CIFAR-Gr	87.2±0.0	0.0±0.0	86.6±0.0	75.3±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	87.7±0.1	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Uniform	81.3±0.0	0.0±0.0	86.3±0.0	87.3±0.1	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	81.0±0.2	0.0±0.0	0.0±0.0	0.0±0.0	0.1±0.0
Smooth	26.8±0.0	0.0±0.0	24.2±0.0	19.6±0.0	0.0±0.0	0.0±0.0	0.2±0.0	0.1±0.0	27.3±0.1	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
SVHN													
CIFAR-10	18.9±0.0	0.1±0.0	9.5±0.0	15.0±0.0	0.3±0.0	0.1±0.0	0.0±0.0	0.1±0.0	15.4±0.1	0.4±0.0	0.0±0.0	0.0±0.0	0.1±0.0
LSUN-CR	19.7±0.0	0.0±0.0	8.3±0.0	17.2±0.2	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	15.5±0.1	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
CIFAR-100	21.8±0.0	0.2±0.0	11.6±0.0	18.1±0.0	0.5±0.0	0.5±0.0	0.1±0.0	0.2±0.0	17.6±0.1	0.6±0.0	0.2±0.0	0.2±0.0	0.1±0.0
FMNIST-3D	26.7±0.0	0.0±0.0	17.5±0.0	24.5±0.1	0.0±0.0	0.6±0.0	0.0±0.0	0.0±0.0	27.2±0.1	0.1±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Uniform	30.0±0.0	0.0±0.0	6.4±0.0	48.2±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	17.0±0.1	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Smooth	17.3±0.0	12.0±0.0	6.9±0.0	9.1±0.0	7.7±0.0	9.5±0.0	8.3±0.0	8.4±0.0	10.1±0.1	8.1±0.1	5.9±0.0	6.6±0.1	6.4±0.0
CIFAR-10													
SVHN	34.5±0.0	10.0±0.0	33.9±0.0	33.5±0.0	30.6±0.1	59.4±0.0	18.3±0.1	33.9±0.1	35.5±0.1	12.7±0.2	47.2±0.3	8.7±0.1	10.8±0.0
LSUN-CR	53.3±0.0	28.0±0.0	44.0±0.0	49.4±0.4	25.9±0.2	43.7±0.1	36.8±0.1	34.8±0.2	53.8±0.6	17.5±0.3	41.2±0.9	30.1±0.5	28.4±0.4
CIFAR-100	61.2±0.0	57.8±0.0	52.5±0.0	58.4±0.1	58.5±0.1	63.3±0.0	56.8±0.1	57.1±0.1	61.4±0.1	59.6±0.2	62.2±0.2	60.4±0.2	57.9±0.1
FMNIST-3D	42.4±0.0	26.8±0.0	30.7±0.0	37.4±0.0	19.0±0.1	43.9±0.0	32.2±0.1	29.6±0.1	43.2±0.2	15.4±0.1	36.8±0.2	24.2±0.1	27.8±0.1
Uniform	87.7±0.0	0.0±0.0	0.0±0.0	13.8±0.1	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	92.8±0.1	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Smooth	35.1±0.0	14.2±0.0	32.9±0.0	26.4±0.0	34.0±0.2	31.9±0.0	30.3±0.0	23.1±0.1	34.9±0.1	15.5±0.2	43.6±0.2	7.5±0.0	14.9±0.1
CIFAR-100													
LSUN-CR	82.0±0.0	64.3±0.0	75.3±0.0	73.8±0.2	62.3±0.5	76.3±0.1	65.3±0.1	67.6±0.3	82.8±0.5	55.9±0.5	75.6±0.7	65.3±0.5	64.1±1.0
CIFAR-10	79.8±0.0	81.9±0.0	76.4±0.0	78.2±0.1	81.4±0.1	82.8±0.0	79.5±0.1	79.0±0.0	79.5±0.1	80.9±0.2	81.7±0.1	80.8±0.1	80.0±0.1
FMNIST-3D	65.8±0.0	58.5±0.0	61.8±0.0	57.1±0.1	41.0±0.2	72.0±0.1	51.7±0.1	56.0±0.1	66.1±0.1	58.6±0.3	69.0±0.2	59.2±0.1	59.3±0.3
Uniform	97.6±0.0	0.0±0.0	94.3±0.0	100.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	98.8±0.1	0.0±0.0	0.0±0.0	0.1±0.0	0.0±0.0
Smooth	79.5±0.0	65.2±0.0	58.7±0.0	79.1±0.0	64.8±0.1	80.2±0.1	54.4±0.1	64.0±0.0	79.2±0.1	41.6±0.2	78.0±0.1	57.1±0.3	66.2±0.1

Table G.2: OOD data detection in terms of AUROC. Higher is better. Values are averages over five prediction runs.

				VB					LA				
Datasets	**MAP**	**OE**	**DE**	**Plain**	**NC**	**SL**	**ML**	**OE**	**Plain**	**NC**	**SL**	**ML**	**OE**
MNIST													
F-MNIST	97.3±0.0	99.9±0.0	98.7±0.0	97.4±0.0	99.9±0.0	99.9±0.0	99.8±0.0	99.6±0.0	97.4±0.0	99.9±0.0	99.9±0.0	99.9±0.0	99.9±0.0
E-MNIST	89.1±0.0	93.7±0.0	90.4±0.0	89.9±0.1	90.4±0.0	95.7±0.0	91.1±0.0	92.1±0.1	89.1±0.0	91.2±0.0	94.9±0.0	93.3±0.0	93.6±0.0
K-MNIST	96.9±0.0	98.5±0.0	98.1±0.0	96.9±0.0	97.8±0.0	98.8±0.0	98.0±0.0	98.2±0.0	96.9±0.0	97.9±0.0	99.2±0.0	98.5±0.0	98.5±0.0
CIFAR-Gr	99.6±0.0	100.0±0.0	99.8±0.0	99.6±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	99.6±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
Uniform	95.0±0.0	100.0±0.0	95.8±0.0	90.5±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	94.6±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
Smooth	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
F-MNIST													
MNIST	79.7±0.0	92.9±0.0	83.0±0.0	85.3±0.0	87.9±0.0	98.6±0.0	86.7±0.0	86.2±0.0	80.3±0.0	94.2±0.0	98.2±0.0	92.9±0.0	92.5±0.0
E-MNIST	81.8±0.0	96.5±0.0	87.5±0.0	85.1±0.0	95.6±0.0	99.2±0.0	92.0±0.0	91.3±0.0	82.3±0.0	98.9±0.0	99.3±0.0	97.6±0.0	96.1±0.0
K-MNIST	83.1±0.0	94.4±0.0	91.7±0.0	86.9±0.0	94.3±0.0	98.4±0.0	93.5±0.0	93.3±0.0	83.9±0.0	96.9±0.0	98.7±0.0	94.9±0.0	94.1±0.0
CIFAR-Gr	82.2±0.0	100.0±0.0	83.6±0.0	87.5±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	81.4±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
Uniform	85.5±0.0	100.0±0.0	85.7±0.0	85.8±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	85.3±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
Smooth	95.7±0.0	100.0±0.0	96.4±0.0	97.2±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	95.5±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
SVHN													
CIFAR-10	96.2±0.0	100.0±0.0	97.9±0.0	95.6±0.0	99.9±0.0	99.9±0.0	100.0±0.0	100.0±0.0	97.1±0.0	99.9±0.0	100.0±0.0	100.0±0.0	100.0±0.0
LSUN-CR	95.7±0.0	100.0±0.0	97.7±0.0	95.9±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	97.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
CIFAR-100	95.5±0.0	99.9±0.0	97.4±0.0	94.7±0.0	99.9±0.0	99.8±0.0	100.0±0.0	99.9±0.0	96.5±0.0	99.9±0.0	99.9±0.0	100.0±0.0	100.0±0.0
FMNIST-3D	95.5±0.0	100.0±0.0	97.1±0.0	91.4±0.0	100.0±0.0	99.8±0.0	100.0±0.0	100.0±0.0	95.6±0.0	100.0±0.0	99.9±0.0	100.0±0.0	100.0±0.0
Uniform	94.3±0.0	100.0±0.0	98.2±0.0	80.2±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	96.8±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
Smooth	96.5±0.0	97.6±0.0	98.4±0.0	97.5±0.0	97.7±0.0	95.9±0.0	97.5±0.0	97.7±0.0	97.7±0.0	98.4±0.0	98.1±0.0	98.5±0.0	98.7±0.0
CIFAR-10													
SVHN	95.6±0.0	98.2±0.0	95.6±0.0	95.7±0.0	95.8±0.0	89.0±0.0	97.2±0.0	95.6±0.0	95.5±0.0	97.8±0.0	92.7±0.0	98.7±0.0	98.1±0.0
LSUN-CR	91.8±0.0	95.9±0.0	93.7±0.0	91.5±0.0	96.1±0.0	92.5±0.0	94.1±0.0	94.4±0.0	92.0±0.0	97.5±0.0	93.8±0.1	95.7±0.0	96.0±0.1
CIFAR-100	89.8±0.0	90.1±0.0	91.3±0.0	88.6±0.0	88.2±0.0	85.9±0.0	88.3±0.0	88.9±0.0	89.9±0.0	89.8±0.0	86.2±0.0	89.7±0.0	90.0±0.0
FMNIST-3D	94.4±0.0	96.2±0.0	95.8±0.0	94.5±0.0	97.2±0.0	92.9±0.0	95.1±0.0	95.9±0.0	94.3±0.0	97.7±0.0	94.0±0.0	96.6±0.0	96.1±0.0
Uniform	93.0±0.0	100.0±0.0	99.5±0.0	97.6±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	92.2±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
Smooth	94.3±0.0	97.6±0.0	95.6±0.0	96.0±0.0	95.3±0.0	94.8±0.0	95.4±0.0	96.2±0.0	94.6±0.0	97.6±0.0	93.9±0.0	98.8±0.0	97.6±0.0
CIFAR-100													
LSUN-CR	78.4±0.0	85.3±0.0	83.8±0.0	81.3±0.1	87.9±0.0	80.1±0.0	85.6±0.0	85.3±0.1	78.8±0.1	89.1±0.0	82.0±0.3	86.3±0.1	85.9±0.2
CIFAR-10	77.4±0.0	77.1±0.0	79.8±0.0	77.7±0.0	76.8±0.0	76.3±0.0	77.8±0.0	77.3±0.0	77.8±0.0	77.5±0.0	76.6±0.0	77.6±0.0	77.7±0.0
FMNIST-3D	85.3±0.0	86.2±0.0	87.6±0.0	87.7±0.0	91.5±0.0	84.0±0.0	89.6±0.0	87.9±0.0	85.2±0.0	87.0±0.0	85.7±0.0	86.3±0.0	86.0±0.0
Uniform	80.1±0.0	100.0±0.0	87.7±0.0	64.8±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	82.6±0.1	100.0±0.0	100.0±0.0	99.9±0.0	100.0±0.0
Smooth	77.0±0.0	76.6±0.0	82.9±0.0	69.4±0.0	74.5±0.0	80.3±0.0	83.2±0.0	78.3±0.0	79.4±0.1	91.7±0.0	78.1±0.0	86.4±0.1	77.7±0.1

G Appendix of Section 4.3

Table G.3: OOD data detection in terms of AUPRC. Higher is better. Values are averages over five prediction runs.

				VB					LA				
Datasets	**MAP**	**OE**	**DE**	**Plain**	**NC**	**SL**	**ML**	**OE**	**Plain**	**NC**	**SL**	**ML**	**OE**
MNIST													
F-MNIST	96.9±0.0	99.9±0.0	98.7±0.0	97.5±0.0	99.9±0.0	99.8±0.0	99.8±0.0	99.6±0.0	97.0±0.0	99.9±0.0	99.9±0.0	99.9±0.0	99.9±0.0
E-MNIST	74.2±0.0	86.7±0.0	77.2±0.0	76.3±0.2	77.8±0.1	83.3±0.0	80.6±0.2	82.5±0.2	74.1±0.1	79.6±0.0	78.8±0.1	85.5±0.0	86.4±0.0
K-MNIST	96.5±0.0	98.5±0.0	98.0±0.0	96.4±0.0	97.5±0.0	97.2±0.0	97.8±0.0	98.1±0.0	96.6±0.0	97.8±0.0	98.9±0.0	98.4±0.0	98.4±0.0
CIFAR-Gr	99.7±0.0	100.0±0.0	99.8±0.0	99.6±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	99.6±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
Uniform	96.7±0.0	100.0±0.0	97.3±0.0	93.4±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	96.5±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
Smooth	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
F-MNIST													
MNIST	75.3±0.0	92.2±0.0	79.0±0.0	83.1±0.1	84.0±0.1	98.4±0.0	83.9±0.1	84.1±0.1	76.3±0.0	92.5±0.0	98.0±0.0	92.3±0.0	91.9±0.0
E-MNIST	66.9±0.0	92.7±0.0	76.4±0.0	74.0±0.3	88.7±0.1	98.2±0.0	82.8±0.0	82.5±0.1	67.8±0.0	96.8±0.0	98.4±0.0	94.8±0.0	92.0±0.0
K-MNIST	81.7±0.0	94.4±0.0	91.1±0.0	85.2±0.1	93.1±0.0	98.1±0.0	92.5±0.0	92.6±0.0	82.8±0.0	96.2±0.0	98.6±0.0	94.7±0.0	94.2±0.0
CIFAR-Gr	85.5±0.0	100.0±0.0	87.2±0.0	89.6±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	84.9±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
Uniform	88.1±0.0	100.0±0.0	88.9±0.0	89.2±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	87.9±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
Smooth	95.3±0.0	100.0±0.0	96.1±0.0	96.9±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	95.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
SVHN													
CIFAR-10	98.3±0.0	100.0±0.0	99.1±0.0	96.9±0.0	99.9±0.0	99.9±0.0	100.0±0.0	100.0±0.0	98.8±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
LSUN-CR	99.9±0.0	100.0±0.0	100.0±0.0	99.9±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
CIFAR-100	97.7±0.0	100.0±0.0	98.8±0.0	96.4±0.0	99.9±0.0	99.9±0.0	100.0±0.0	100.0±0.0	98.3±0.0	99.9±0.0	100.0±0.0	100.0±0.0	100.0±0.0
FMNIST-3D	98.1±0.0	100.0±0.0	98.8±0.0	93.5±0.0	100.0±0.0	99.9±0.0	100.0±0.0	100.0±0.0	98.2±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
Uniform	97.3±0.0	100.0±0.0	99.3±0.0	82.8±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	98.7±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
Smooth	98.5±0.0	98.9±0.0	99.4±0.0	98.6±0.0	98.6±0.0	96.5±0.0	98.4±0.0	98.7±0.0	99.1±0.0	99.3±0.0	98.7±0.0	99.4±0.0	99.5±0.0
CIFAR-10													
SVHN	93.3±0.0	96.5±0.0	93.3±0.0	93.3±0.0	92.8±0.0	77.6±0.0	94.5±0.0	92.8±0.0	93.3±0.0	95.9±0.0	85.2±0.0	97.0±0.0	96.4±0.0
LSUN-CR	99.7±0.0	99.8±0.0	99.7±0.0	99.6±0.0	99.8±0.0	99.7±0.0	99.7±0.0	99.7±0.0	99.7±0.0	99.9±0.0	99.7±0.0	99.8±0.0	99.8±0.0
CIFAR-100	89.9±0.0	90.0±0.0	91.3±0.0	86.7±0.0	85.7±0.0	82.8±0.0	85.3±0.0	86.9±0.0	90.0±0.0	90.0±0.0	82.4±0.0	89.8±0.0	90.0±0.0
FMNIST-3D	95.1±0.0	96.4±0.0	96.1±0.0	94.7±0.0	97.0±0.0	92.8±0.0	95.0±0.0	96.1±0.0	95.0±0.0	97.6±0.0	93.4±0.0	96.7±0.0	96.3±0.0
Uniform	95.5±0.0	100.0±0.0	99.6±0.0	98.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	95.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
Smooth	94.4±0.0	97.5±0.0	95.8±0.0	95.7±0.0	95.5±0.0	93.9±0.0	95.1±0.0	95.9±0.0	94.7±0.0	97.6±0.0	93.9±0.0	98.6±0.0	97.4±0.0
CIFAR-100													
LSUN-CR	99.0±0.0	99.4±0.0	99.3±0.0	99.0±0.0	99.5±0.0	99.1±0.0	99.3±0.0	99.3±0.0	99.0±0.0	99.5±0.0	99.2±0.0	99.4±0.0	99.4±0.0
CIFAR-10	77.2±0.0	77.0±0.0	79.3±0.0	76.8±0.0	77.1±0.0	75.4±0.0	77.4±0.0	77.1±0.0	77.3±0.0	77.1±0.0	75.8±0.0	77.2±0.0	77.1±0.0
FMNIST-3D	85.7±0.0	86.0±0.0	88.2±0.0	87.6±0.0	90.9±0.0	84.8±0.0	89.3±0.0	87.6±0.0	85.6±0.0	87.1±0.0	86.5±0.0	86.3±0.0	85.7±0.0
Uniform	84.6±0.0	100.0±0.0	91.5±0.0	73.3±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	87.1±0.1	100.0±0.0	100.0±0.0	99.9±0.0	100.0±0.0
Smooth	75.4±0.0	71.0±0.0	80.3±0.0	67.4±0.1	69.8±0.1	81.4±0.0	79.9±0.1	74.9±0.0	78.5±0.1	90.9±0.0	78.3±0.0	85.4±0.1	73.9±0.1

Table G.4: Confidences in terms of MMC, averaged over five prediction runs. Lower is better for OOD datasets.

				VB					LA				
Datasets	**MAP**	**OE**	**DE**	**Plain**	**NC**	**SL**	**ML**	**OE**	**Plain**	**NC**	**SL**	**ML**	**OE**
MNIST	99.1	99.4	99.3	98.7	98.2	99.1	98.2	98.6	99.0	99.3	99.2	99.2	99.3
F-MNIST	66.3±0.0	22.0±0.0	64.9±0.0	70.4±0.0	6.1±0.0	20.4±0.0	21.2±0.0	27.3±0.0	65.2±0.0	7.7±0.0	21.0±0.0	20.7±0.0	22.2±0.0
E-MNIST	82.3±0.0	79.3±0.0	80.9±0.0	78.6±0.0	74.0±0.0	75.9±0.0	73.0±0.0	73.9±0.0	81.1±0.0	79.6±0.0	80.4±0.0	76.5±0.0	78.1±0.0
K-MNIST	73.3±0.0	65.7±0.0	69.7±0.0	67.6±0.0	51.8±0.0	61.2±0.0	50.3±0.0	52.4±0.0	71.8±0.0	64.3±0.0	66.3±0.0	62.3±0.0	64.3±0.0
CIFAR-Gr	48.0±0.0	10.0±0.0	43.1±0.0	45.2±0.0	0.0±0.0	10.0±0.0	10.1±0.0	10.1±0.0	47.3±0.0	0.0±0.0	10.1±0.0	10.4±0.0	10.2±0.0
Uniform	96.8±0.0	10.0±0.0	97.4±0.0	97.8±0.0	0.1±0.0	10.0±0.0	10.1±0.0	10.1±0.0	96.5±0.0	0.0±0.0	10.2±0.0	10.3±0.0	10.2±0.0
Smooth	12.9±0.0	10.1±0.0	12.6±0.0	12.7±0.0	0.3±0.0	21.6±0.0	10.2±0.0	10.2±0.0	12.8±0.0	0.5±0.0	20.5±0.0	10.1±0.0	10.1±0.0
F-MNIST	96.1	96.0	94.8	93.6	91.0	95.0	91.2	92.9	95.7	94.9	93.5	94.5	94.7
MNIST	82.8±0.0	60.1±0.0	74.3±0.0	70.9±0.0	50.0±0.0	32.9±0.0	57.2±0.0	63.7±0.0	80.9±0.0	33.8±0.1	33.9±0.0	55.9±0.1	57.9±0.0
E-MNIST	82.5±0.0	45.0±0.0	70.0±0.0	71.5±0.0	25.0±0.0	25.5±0.0	44.4±0.0	50.7±0.0	80.6±0.0	9.4±0.0	24.3±0.0	36.2±0.0	44.7±0.0
K-MNIST	82.5±0.0	60.1±0.0	64.0±0.0	68.4±0.0	37.0±0.0	33.5±0.0	44.8±0.0	48.0±0.0	80.0±0.0	28.1±0.0	32.0±0.0	52.6±0.0	57.3±0.0
CIFAR-Gr	89.1±0.0	10.0±0.0	84.6±0.0	73.1±0.0	0.0±0.0	10.0±0.0	10.1±0.0	10.2±0.0	88.6±0.0	0.0±0.0	10.8±0.0	10.3±0.0	10.3±0.0
Uniform	85.5±0.0	13.2±0.0	82.3±0.0	79.4±0.0	0.6±0.0	10.2±0.0	10.6±0.0	11.1±0.0	84.2±0.0	0.0±0.0	10.9±0.0	12.2±0.0	15.0±0.0
Smooth	48.0±0.0	10.5±0.0	44.0±0.0	42.0±0.0	1.0±0.0	11.0±0.0	11.9±0.0	10.7±0.0	47.4±0.0	0.3±0.0	11.6±0.0	10.4±0.0	10.8±0.0
SVHN	98.6	98.6	98.1	97.7	97.3	98.4	97.1	97.6	97.9	98.0	97.9	97.8	97.3
CIFAR-10	69.0±0.0	11.6±0.0	57.3±0.0	61.3±0.0	3.7±0.0	15.4±0.0	11.1±0.0	11.7±0.0	60.7±0.0	3.6±0.0	12.4±0.0	12.1±0.0	12.5±0.0
LSUN-CR	69.8±0.0	10.2±0.0	57.7±0.0	63.9±0.0	0.1±0.0	10.8±0.0	10.2±0.0	10.2±0.0	61.5±0.1	0.0±0.0	11.0±0.0	10.3±0.0	10.7±0.0
CIFAR-100	70.3±0.0	12.4±0.0	58.8±0.0	63.5±0.0	4.3±0.0	17.3±0.0	11.6±0.0	12.3±0.0	62.3±0.0	4.1±0.0	13.4±0.0	12.7±0.0	13.4±0.0
FMNIST-3D	73.6±0.0	11.0±0.0	62.5±0.0	67.8±0.0	1.5±0.0	17.6±0.0	10.8±0.0	11.5±0.0	68.0±0.0	1.8±0.0	15.5±0.0	10.9±0.0	12.0±0.0
Uniform	77.5±0.0	10.3±0.0	57.1±0.0	82.2±0.0	0.1±0.0	10.0±0.0	10.1±0.0	10.2±0.0	65.7±0.1	0.0±0.0	10.4±0.0	10.3±0.0	10.7±0.0
Smooth	68.8±0.0	52.1±0.0	51.3±0.0	55.5±0.0	42.3±0.0	62.7±0.0	45.6±0.0	44.0±0.0	58.3±0.1	43.9±0.0	44.9±0.0	44.6±0.1	40.8±0.0
CIFAR-10	96.7	96.9	95.9	95.7	94.5	96.0	95.1	95.5	96.4	94.9	95.8	96.1	96.1
SVHN	65.2±0.0	45.7±0.0	60.8±0.0	59.5±0.0	49.7±0.0	73.3±0.0	45.0±0.0	55.8±0.0	63.5±0.0	36.8±0.1	64.7±0.0	33.0±0.0	44.0±0.0
LSUN-CR	73.9±0.0	58.7±0.0	65.2±0.0	68.2±0.0	41.1±0.1	64.3±0.0	56.7±0.0	57.1±0.0	71.6±0.0	29.2±0.2	61.7±0.4	53.6±0.3	55.2±0.3
CIFAR-100	77.2±0.0	76.1±0.0	69.8±0.0	72.4±0.0	67.7±0.0	75.9±0.0	69.8±0.0	70.5±0.0	75.4±0.0	67.3±0.0	74.3±0.0	73.2±0.0	72.6±0.0
FMNIST-3D	67.7±0.0	55.9±0.0	58.5±0.0	60.4±0.0	32.7±0.0	64.3±0.0	54.5±0.0	52.3±0.0	66.0±0.0	27.4±0.0	57.6±0.0	48.5±0.0	53.1±0.1
Uniform	82.1±0.0	10.3±0.0	40.7±0.0	52.2±0.0	0.1±0.0	10.2±0.0	10.7±0.0	10.1±0.0	81.9±0.1	0.0±0.0	10.2±0.0	10.3±0.0	10.3±0.0
Smooth	64.3±0.0	45.2±0.0	57.2±0.0	54.0±0.0	51.4±0.0	53.3±0.0	53.0±0.0	49.3±0.0	62.5±0.0	32.6±0.0	59.5±0.1	26.4±0.0	43.5±0.0
CIFAR-100	84.7	85.1	80.9	69.0	66.2	80.1	67.1	67.1	81.6	80.3	77.0	79.1	79.4
SVHN	52.8±0.0	43.9±0.0	43.8±0.0	27.3±0.0	31.0±0.0	46.2±0.0	24.9±0.0	26.9±0.0	46.9±0.0	33.7±0.1	42.1±0.0	34.8±0.0	37.9±0.1
LSUN-CR	62.7±0.0	51.3±0.0	48.8±0.0	33.3±0.1	21.1±0.0	49.7±0.0	26.5±0.0	26.4±0.1	57.0±0.1	36.4±0.1	43.5±0.5	40.9±0.1	42.6±0.3
CIFAR-10	62.9±0.0	63.7±0.0	53.4±0.0	39.0±0.0	36.9±0.0	55.3±0.0	37.2±0.0	37.4±0.0	57.3±0.0	55.7±0.0	50.8±0.0	53.8±0.0	54.0±0.0
FMNIST-3D	51.8±0.0	48.2±0.0	42.5±0.0	24.8±0.0	15.9±0.0	44.5±0.0	20.6±0.0	22.6±0.0	47.1±0.0	40.7±0.0	38.4±0.0	40.3±0.0	41.4±0.0
Uniform	64.2±0.0	1.4±0.0	45.0±0.0	59.4±0.0	0.0±0.0	2.2±0.0	1.2±0.0	1.2±0.0	54.4±0.1	0.0±0.0	1.5±0.0	4.4±0.1	1.5±0.0
Smooth	61.7±0.0	58.9±0.0	47.3±0.0	49.3±0.0	38.4±0.0	50.6±0.0	28.7±0.0	34.8±0.0	54.8±0.1	30.4±0.1	49.1±0.0	40.2±0.1	51.5±0.1

Table G.5: Test accuracy (↑) / ECE (↓) of models trained with random noises (Hein et al., 2019) as $\mathcal{D}_{\text{out}}$, averaged over five prediction runs.

	MNIST	F-MNIST	SVHN	CIFAR-10	CIFAR-100
MAP	99.4±0.0 / 6.4±0.0	92.4±0.0 / 13.9±0.0	97.4±0.0 / 8.9±0.0	94.8±0.0 / 10.0±0.0	76.7±0.0 / 14.3±0.0
DE	99.5±0.0 / 8.6±0.0	93.6±0.0 / 3.6±0.0	97.6±0.0 / 3.5±0.0	95.7±0.0 / 4.5±0.0	80.0±0.0 / 1.9±0.0
OE	99.6±0.0 / 6.4±0.0	92.6±0.0 / 12.7±0.0	97.5±0.0 / 8.9±0.0	94.7±0.0 / 11.5±0.0	76.5±0.0 / 16.1±0.0
VB	99.5±0.0 / 11.2±0.3	92.4±0.0 / 3.7±0.2	97.5±0.0 / 5.7±0.2	94.9±0.0 / 5.8±0.2	75.4±0.0 / 8.3±0.0
+NC	99.4±0.0 / 10.6±0.1	92.3±0.0 / 3.0±0.1	97.4±0.0 / 4.2±0.2	94.9±0.0 / 5.1±0.1	74.0±0.1 / 8.8±0.1
+SL	99.6±0.0 / 12.4±0.1	93.2±0.0 / 12.4±0.1	97.3±0.0 / 12.8±0.0	91.5±0.0 / 18.5±0.1	1.0±0.1 / 0.1±0.0
+ML	99.4±0.0 / 11.6±0.2	92.1±0.0 / 2.4±0.1	97.6±0.0 / 3.3±0.1	95.1±0.0 / 3.6±0.2	75.2±0.0 / 9.8±0.0
+OE	99.5±0.0 / 10.2±0.2	92.5±0.0 / 3.2±0.1	97.5±0.0 / 5.0±0.1	94.9±0.0 / 6.8±0.2	74.1±0.0 / 8.0±0.0
LA	99.4±0.0 / 7.6±0.1	92.5±0.0 / 11.3±0.2	97.4±0.0 / 3.3±0.3	94.8±0.0 / 7.5±0.3	76.6±0.1 / 8.3±0.1
+NC	99.3±0.0 / 10.1±0.8	92.5±0.0 / 2.8±0.1	96.3±0.0 / 5.0±0.1	94.9±0.0 / 8.3±0.3	75.9±0.1 / 3.8±0.1
+SL	99.6±0.0 / 11.1±0.4	93.0±0.0 / 9.1±0.1	18.8±0.0 / 13.5±0.1	91.5±0.0 / 16.0±0.2	72.2±0.0 / 4.0±0.1
+ML	99.5±0.0 / 5.5±0.2	92.3±0.0 / 9.3±0.1	97.5±0.0 / 7.4±0.3	94.8±0.0 / 7.2±0.3	76.8±0.1 / 3.3±0.2
+OE	99.6±0.0 / 6.6±0.3	92.3±0.0 / 2.0±0.1	97.5±0.0 / 3.8±0.2	94.6±0.0 / 7.2±0.3	76.3±0.0 / 8.6±0.1

Table G.6: OOD data detection under models trained with random noises (Hein et al., 2019) as $\mathcal{D}_{\text{out}}$. Values are FPR95, averaged over five prediction runs—lower is better.

				VB					LA				
Datasets	**MAP**	**OE**	**DE**	**Plain**	**NC**	**SL**	**ML**	**OE**	**Plain**	**NC**	**SL**	**ML**	**OE**
MNIST													
F-MNIST	11.8±0.0	6.8±0.0	5.3±0.0	12.5±0.1	6.5±0.0	0.7±0.0	11.9±0.1	10.3±0.1	12.0±0.0	8.2±0.0	0.0±0.0	6.3±0.0	6.8±0.0
E-MNIST	35.6±0.0	30.7±0.0	30.4±0.0	34.5±0.1	35.1±0.1	17.9±0.0	37.3±0.1	34.3±0.1	35.8±0.1	34.2±0.0	15.3±0.1	31.0±0.0	30.7±0.0
K-MNIST	14.4±0.0	7.8±0.0	7.7±0.0	14.0±0.1	14.5±0.1	1.1±0.0	15.8±0.1	14.0±0.1	14.5±0.1	10.6±0.0	0.7±0.0	8.5±0.0	7.8±0.0
CIFAR-Gr	0.2±0.0	0.0±0.0	0.0±0.0	0.2±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.1±0.0	0.2±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Uniform	44.3±0.0	0.7±0.0	19.8±0.0	93.1±0.2	0.0±0.0	0.0±0.0	1.9±0.1	1.8±0.0	54.2±0.4	0.7±0.0	0.0±0.0	0.6±0.0	0.8±0.0
F-MNIST													
MNIST	73.5±0.0	62.2±0.0	65.8±0.0	66.8±0.1	62.2±0.0	30.4±0.1	59.1±0.1	60.4±0.1	72.2±0.2	60.8±0.2	24.3±0.3	55.4±0.2	57.4±0.3
E-MNIST	73.6±0.0	50.2±0.0	58.6±0.0	68.1±0.1	43.9±0.0	25.7±0.0	54.7±0.1	54.6±0.1	72.2±0.2	44.2±0.1	22.5±0.2	39.6±0.2	48.3±0.3
K-MNIST	73.7±0.0	47.4±0.0	47.2±0.0	62.6±0.1	31.4±0.1	19.1±0.0	35.6±0.1	38.1±0.1	71.5±0.2	33.9±0.2	20.9±0.2	31.7±0.2	43.0±0.4
CIFAR-Gr	87.2±0.0	0.5±0.0	86.6±0.0	75.3±0.0	0.1±0.0	0.2±0.0	0.8±0.0	1.1±0.0	87.7±0.1	0.7±0.0	0.2±0.0	0.7±0.0	1.0±0.0
Uniform	81.3±0.0	26.0±0.0	86.3±0.0	87.3±0.1	47.1±0.2	0.0±0.0	0.1±0.0	4.9±0.0	81.0±0.2	43.4±0.7	0.0±0.0	22.0±0.2	38.1±0.9
SVHN													
CIFAR-10	18.9±0.0	13.8±0.0	9.5±0.0	15.0±0.0	13.0±0.1	16.5±0.0	8.4±0.0	11.5±0.0	15.4±0.1	8.4±0.0	94.8±1.6	14.9±0.1	11.4±0.1
LSUN-CR	19.7±0.0	9.0±0.0	8.3±0.0	17.2±0.2	10.5±0.1	8.8±0.1	5.4±0.1	9.1±0.1	15.5±0.1	8.3±0.3	95.4±4.0	12.6±0.2	8.2±0.1
CIFAR-100	21.8±0.0	15.6±0.0	11.6±0.0	18.1±0.0	14.8±0.1	17.9±0.0	10.2±0.0	12.4±0.0	17.6±0.1	11.7±0.0	93.9±0.7	16.6±0.1	13.4±0.1
FMNIST-3D	26.7±0.0	29.8±0.0	17.5±0.0	24.5±0.1	31.1±0.0	30.4±0.0	30.0±0.1	25.3±0.0	27.2±0.1	34.6±0.1	95.1±0.8	23.3±0.1	27.7±0.1
Uniform	30.0±0.0	0.0±0.0	6.4±0.0	48.2±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	17.0±0.1	0.0±0.0	90.2±0.6	19.0±0.1	0.0±0.0
CIFAR-10													
SVHN	34.5±0.0	7.3±0.0	33.9±0.0	33.5±0.0	11.1±0.0	20.6±0.0	11.0±0.0	9.7±0.0	35.5±0.1	6.6±0.0	26.0±0.3	7.5±0.0	8.3±0.1
LSUN-CR	53.3±0.0	49.0±0.0	44.0±0.0	49.4±0.4	46.7±0.1	61.7±0.0	45.7±0.2	47.6±0.3	53.8±0.6	48.9±0.5	54.9±0.4	51.9±0.3	48.3±0.5
CIFAR-100	61.2±0.0	58.2±0.0	52.5±0.0	58.4±0.1	57.7±0.1	71.1±0.0	56.3±0.1	56.6±0.1	61.4±0.1	59.3±0.2	70.7±0.3	57.6±0.1	59.0±0.3
FMNIST-3D	42.4±0.0	44.9±0.0	30.7±0.0	37.4±0.0	39.5±0.1	62.7±0.0	43.3±0.1	44.0±0.1	43.2±0.2	40.2±0.1	57.8±0.4	40.8±0.1	46.4±0.3
Uniform	87.7±0.0	26.7±0.0	0.0±0.0	13.8±0.1	100.0±0.0	57.3±0.1	98.0±0.0	100.0±0.0	92.8±0.1	3.5±0.1	17.7±0.4	12.9±0.3	49.8±0.9
CIFAR-100													
LSUN-CR	82.0±0.0	79.7±0.0	75.3±0.0	73.8±0.2	80.9±0.3	91.5±8.5	77.9±0.1	71.1±0.4	82.8±0.5	82.0±0.8	78.5±0.8	72.8±0.9	79.7±0.8
CIFAR-10	79.8±0.0	80.5±0.0	76.4±0.0	78.2±0.1	81.3±0.0	93.9±1.1	80.0±0.1	81.4±0.1	79.5±0.1	80.6±0.2	82.1±0.1	78.8±0.2	80.2±0.2
FMNIST-3D	65.8±0.0	66.9±0.0	61.8±0.0	57.1±0.1	69.3±0.1	93.6±1.1	63.1±0.1	61.9±0.1	66.1±0.1	71.2±0.2	82.0±0.1	64.4±0.4	67.9±0.2
Uniform	97.6±0.0	73.3±0.0	94.3±0.0	100.0±0.0	99.7±0.0	95.4±1.4	99.5±0.0	99.8±0.0	98.8±0.1	88.4±0.5	100.0±0.0	88.9±0.5	54.0±0.6

Table G.7: Accuracy and ECE on text classification tasks, averaged over five prediction runs.

Methods	**SST**	**TREC**
MAP	78.1±0.0 / 20.8±0.0	76.0±0.0 / 17.2±0.0
DE	82.9±0.0 / 2.5±0.0	80.6±0.0 / 10.6±0.0
OE	78.6±0.0 / 13.0±0.0	68.8±0.0 / 9.4±0.0
LA	78.0±0.0 / 21.0±0.4	75.9±0.1 / 17.3±0.3
+NC	78.0±0.0 / 17.9±0.1	43.4±0.0 / 18.6±0.2
+DL	69.2±0.2 / 17.5±0.7	45.0±0.2 / 10.4±0.8
+ML	48.9±0.3 / 11.4±0.2	55.8±0.1 / 11.5±0.3
+OE	78.5±0.0 / 12.8±0.4	68.1±0.1 / 8.4±0.7

G Appendix of Section 4.3

Table G.8: OOD data detection on text classification tasks. Values are FPR95, averaged over five prediction runs—lower is better.

				LA				
Datasets	**MAP**	**OE**	**DE**	**Plain**	**NC**	**SL**	**ML**	**OE**
SST								
SNLI	100.0±0.0	0.0±0.0	100.0±0.0	100.0±0.0	0.0±0.0	97.0±0.3	89.6±0.7	0.0±0.0
Multi30k	100.0±0.0	0.0±0.0	100.0±0.0	100.0±0.0	0.0±0.0	99.5±0.0	83.5±1.5	0.0±0.0
WMT16	100.0±0.0	0.0±0.0	100.0±0.0	100.0±0.0	0.0±0.0	89.3±0.6	80.7±1.4	0.0±0.0
TREC								
SNLI	99.7±0.0	0.0±0.0	31.0±0.1	99.7±0.0	0.0±0.0	0.7±0.3	0.0±0.0	0.0±0.0
Multi30k	100.0±0.0	0.0±0.0	14.2±0.0	100.0±0.0	0.0±0.0	0.8±0.5	0.0±0.0	0.0±0.0
WMT16	89.2±0.0	0.0±0.0	27.3±0.0	89.3±0.0	0.0±0.0	0.8±0.7	0.0±0.0	0.0±0.0

Table G.9: Test accuracy (↑) / ECE (↓) of more sophisticated base models, averaged over five prediction runs.

	CIFAR-10	**CIFAR-100**
Flipout	91.3±0.0 / 10.9±0.2	70.4±0.1 / 19.8±0.2
+NC	89.7±0.1 / 8.2±0.2	67.1±0.1 / 13.8±0.1
CSGHMC	93.9±0.0 / 1.7±0.0	74.0±0.0 / 4.0±0.0
+NC	92.2±0.0 / 6.2±0.0	71.6±0.0 / 2.4±0.0
DE	95.7±0.0 / 4.5±0.0	80.0±0.0 / 1.9±0.0
+NC	94.9±0.0 / 4.8±0.0	79.0±0.0 / 1.7±0.0

Table G.10: OOD data detection with more sophisticated base models. Values are FPR95, averaged over five prediction runs.

	Flipout		**CSGHMC**		**DE**	
Datasets	**Plain**	**NC**	**Plain**	**NC**	**Plain**	**NC**
CIFAR-10						
SVHN	72.1±0.3	39.6±0.2	56.8±0.0	16.4±0.0	33.9±0.0	8.1±0.0
LSUN-CR	63.7±1.0	37.5±0.2	56.7±0.0	24.0±0.0	44.0±0.0	18.3±0.0
CIFAR-100	74.5±0.2	70.4±0.1	63.4±0.0	63.1±0.0	52.5±0.0	51.7±0.0
FMNIST-3D	65.0±0.2	38.2±0.1	51.0±0.0	14.8±0.0	30.7±0.0	10.3±0.0
Uniform	53.8±0.6	0.0±0.0	87.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Smooth	61.1±0.2	59.6±0.2	47.1±0.0	31.9±0.0	32.9±0.0	13.6±0.0
CIFAR-100						
LSUN-CR	85.8±0.6	55.6±0.4	79.3±0.0	38.0±0.0	75.3±0.0	54.0±0.0
CIFAR-10	86.1±0.3	87.0±0.2	82.1±0.0	84.2±0.0	76.4±0.0	78.5±0.0
FMNIST-3D	73.4±0.4	65.8±0.2	67.0±0.0	45.5±0.0	61.8±0.0	50.0±0.0
Uniform	99.7±0.0	0.0±0.0	93.8±0.0	0.0±0.0	94.3±0.0	0.0±0.0
Smooth	82.0±0.3	72.5±0.3	83.0±0.0	47.2±0.0	58.7±0.0	39.3±0.0

Reference

Amari, S.-I. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

Arora, R., Basu, A., Mianjy, P., and Mukherjee, A. Understanding deep neural networks with rectified linear units. In *ICLR*, 2018.

Barber, D. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

Bernardo, J. M. and Smith, A. F. *Bayesian Theory*. John Wiley & Sons, 2009.

Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P., Horsfall, P., and Goodman, N. D. Pyro: Deep universal probabilistic programming. *JMLR*, 20(1), 2019.

Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006.

Bitterwolf, J., Meinke, A., and Hein, M. Certifiably adversarially robust detection of out-of-distribution data. In *NeurIPS*, 2020.

Blight, B. and Ott, L. A Bayesian approach to model inadequacy for polynomial regression. *Biometrika*, 62, 1975.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. Weight uncertainty in neural networks. In *ICML*, 2015.

Botev, A., Ritter, H., and Barber, D. Practical gauss-newton optimisation for deep learning. In *ICML*, 2017.

Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. A large annotated corpus for learning natural language inference. In *EMNLP*, 2015.

Brier, G. W. et al. Verification of forecasts expressed in terms of probability. *Monthly weather review*, 78(1), 1950.

Brosse, N., Riquelme, C., Martin, A., Gelly, S., and Moulines, É. On last-layer algorithms for classification: Decoupling representation from uncertainty estimation. *arXiv preprint arXiv:2001.08049*, 2020.

Chapelle, O. and Li, L. An empirical evaluation of thompson sampling. In *NIPS*, 2011.

Chen, T., Fox, E., and Guestrin, C. Stochastic gradient hamiltonian monte carlo. In *ICML*, 2014.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Cho, Y. and Saul, L. K. Kernel methods for deep learning. In *NIPS*, 2009.

Chrabaszcz, P., Loshchilov, I., and Hutter, F. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017.

Clanuwat, T., Bober-Irizar, M., Kitamoto, A., Lamb, A., Yamamoto, K., and Ha, D. Deep learning for classical Japanese literature. *arXiv preprint arXiv:1812.01718*, 2018.

Cohen, G., Afshar, S., Tapson, J., and Van Schaik, A. EMNIST: Extending MNIST to handwritten letters. In *International Joint Conference on Neural Networks*, 2017.

Dangel, F., Kunstner, F., and Hennig, P. BackPACK: Packing more into backprop. In *ICLR*, 2020.

Daxberger, E., Kristiadi, A., Immer, A., Eschenhagen, R., Bauer, M., and Hennig, P. Laplace redux–effortless Bayesian deep learning. In *NeurIPS*, 2021a.

Daxberger, E., Nalisnick, E., Allingham, J. U., Antorán, J., and Hernández-Lobato, J. M. Bayesian deep learning via subnetwork inference. In *ICML*, 2021b.

Denker, J. S. and LeCun, Y. Transforming neural-net output levels to probability distributions. In *NIPS*, 1990.

Dinh, L., Krueger, D., and Bengio, Y. NICE: Non-linear independent components estimation. In *ICLR Workshop*, 2015.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

Dusenberry, M., Jerfel, G., Wen, Y., Ma, Y., Snoek, J., Heller, K., Lakshminarayanan, B., and Tran, D. Efficient and scalable Bayesian neural nets with rank-1 factors. In *ICML*, 2020.

Eckart, C. and Young, G. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3), 1936.

El Gayar, N., Schwenker, F., and Palm, G. A study of the robustness of KNN classifiers trained using soft labels. In *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*, 2006.

Elliott, D., Frank, S., Sima'an, K., and Specia, L. Multi30K: Multilingual English-German image descriptions. In *ACL Workshop on Vision and Language*, 2016.

Eschenhagen, R., Daxberger, E., Hennig, P., and Kristiadi, A. Mixtures of Laplace approximations for improved post-hoc uncertainty in deep learning. In *NeurIPS Workshop of Bayesian Deep Learning*, 2021.

Foong, A. Y., Li, Y., Hernández-Lobato, J. M., and Turner, R. E. In-between uncertainty in Bayesian neural networks. *arXiv*, 2019.

Franchi, G., Bursuc, A., Aldea, E., Dubuisson, S., and Bloch, I. TRADI: Tracking deep neural network weight distributions. *arXiv preprint arXiv:1912.11316*, 2019.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019.

Gal, Y. and Ghahramani, Z. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *ICML*, 2016.

Gardner, J., Pleiss, G., Weinberger, K. Q., Bindel, D., and Wilson, A. G. GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. In *NIPS*, 2018.

Garriga-Alonso, A. and Fortuin, V. Exact Langevin dynamics with stochastic gradients. *arXiv preprint arXiv:2102.01691*, 2021.

Gast, J. and Roth, S. Lightweight probabilistic deep networks. In *CVPR*, 2018.

Gelman, A. and Rubin, D. B. Inference from iterative simulation using multiple sequences. *Statistical Science*, 7(4), 1992.

Gelman, A., Jakulin, A., Pittau, M. G., Su, Y.-S., et al. A weakly informative default prior distribution for logistic and other regression models. *The annals of applied statistics*, 2(4), 2008.

George, T., Laurent, C., Bouthillier, X., Ballas, N., and Vincent, P. Fast approximate natural gradient descent in a Kronecker factored eigenbasis. In *NIPS*, 2018.

Ghahramani, Z. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553), 2015.

Gibbs, M. *Bayesian Gaussian Processes for Regression and Classification*. PhD thesis, University of Cambridge, 1998.

Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., and Bengio, Y. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

Graves, A. Practical variational inference for neural networks. In *NIPS*, 2011.

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. A kernel two-sample test. *JMLR*, 13(1), 2012.

Gueorguieva, R., Rosenheck, R., and Zelterman, D. Dirichlet component regression and its applications to psychiatric data. *Computational Statistics & Data Analysis*, 52(12), 2008.

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. On calibration of modern neural networks. In *ICML*, 2017.

Gupta, A. K. and Nagar, D. K. *Matrix variate distributions*. Chapman and Hall/CRC, 1999.

Havasi, M., Snoek, J., Tran, D., Gordon, J., and Hernández-Lobato, J. M. Refining the variational posterior through iterative optimization. *Entropy*, 23(1475), 2021.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *CVPR*, 2016.

Hein, M., Andriushchenko, M., and Bitterwolf, J. Why ReLU networks yield high-confidence predictions far away from the training data and how to mitigate the problem. In *CVPR*, 2019.

Hendrycks, D. and Dietterich, T. Benchmarking neural network robustness to common corruptions and perturbations. In *ICLR*, 2019.

Hendrycks, D. and Gimpel, K. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *ICLR*, 2017.

Hendrycks, D., Mazeika, M., and Dietterich, T. Deep anomaly detection with outlier exposure. In *ICLR*, 2019.

Hennig, P., Stern, D., Herbrich, R., and Graepel, T. Kernel topic models. In *AISTATS*, 2012.

Hensman, J., Matthews, A., and Ghahramani, Z. Scalable variational Gaussian process classification. In *AISTATS*, 2015.

Hernández-Lobato, J. M. and Adams, R. Probabilistic backpropagation for scalable learning of Bayesian neural networks. In *International Conference on Machine Learning*, pp. 1861–1869, 2015.

Heskes, T. On "natural" learning and pruning in multilayered perceptrons. *Neural Computation*, 12(4), 2000.

Higham, N. J. *A Survey of Componentwise Perturbation Theory*, volume 48. American Mathematical Society, 1994.

Hinton, G., Srivastava, N., and Swersky, K. RMSProp: Divide the gradient by a running average of its recent magnitude. *Neural networks for machine learning, Coursera lecture 6e*, 2012.

Hinton, G. E. and Van Camp, D. Keeping the neural networks simple by minimizing the description length of the weights. In *COLT*, 1993.

Hobbhahn, M., Kristiadi, A., and Hennig, P. Fast predictive uncertainty for classification with Bayesian deep networks. 2022.

Hoffman, M. D., Gelman, A., et al. The No-U-Turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *JMLR*, 15(1), 2014.

Humt, M., Lee, J., and Triebel, R. Bayesian optimization meets laplace approximation for robotic introspection. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) Long-Term Autonomy Workshop*, 2020.

Huszár, F. Note on the quadratic penalties in elastic weight consolidation. In *Proceedings of the National Academy of Sciences*, 2018.

Hutter, F., Kotthoff, L., and Vanschoren, J. *Automated Cachine Learning: Methods, Systems, Challenges*. Springer Nature, 2019.

Immer, A., Bauer, M., Fortuin, V., Rätsch, G., and Khan, M. E. Scalable marginal likelihood estimation for model selection in deep learning. In *ICML*, 2021a.

Immer, A., Korzepa, M., and Bauer, M. Improving predictions of Bayesian neural nets via local linearization. In *AISTATS*, 2021b.

Izmailov, P., Maddox, W. J., Kirichenko, P., Garipov, T., Vetrov, D., and Wilson, A. G. Subspace inference for Bayesian deep learning. In *UAI*, 2019.

Izmailov, P., Nicholson, P., Lotfi, S., and Wilson, A. G. Dangers of Bayesian model averaging under covariate shift. In *NeurIPS*, 2021a.

Izmailov, P., Vikram, S., Hoffman, M. D., and Wilson, A. G. What are Bayesian neural network posteriors really like? In *ICML*, 2021b.

Khan, M. E. E., Immer, A., Abedi, E., and Korzepa, M. Approximate inference turns deep networks into Gaussian processes. In *NeurIPS*, 2019.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13), 2017.

Koh, P. W., Sagawa, S., Marklund, H., Xie, S. M., Zhang, M., Balsubramani, A., Hu, W., Yasunaga, M., Phillips, R. L., Gao, I., et al. WILDS: A benchmark of in-the-wild distribution shifts. *arXiv preprint arXiv:2012.07421*, 2020.

Korattikara, A., Rathod, V., Murphy, K., and Welling, M. Bayesian dark knowledge. In *NIPS*, 2015.

Krishnan, R. and Esposito, P. Bayesian-Torch: Bayesian neural network layers for uncertainty estimation. `https://github.com/IntelLabs/bayesian-torch`, 2020.

Kristiadi, A. Last-layer Laplace approximation code examples. `https://github.com/wiseodd/last_layer_laplace`, 2020.

Kristiadi, A., Hein, M., and Hennig, P. Being Bayesian, even just a bit, fixes overconfidence in ReLU networks. In *ICML*, 2020.

Kristiadi, A., Hein, M., and Hennig, P. Learnable uncertainty under Laplace approximations. In *UAI*, 2021.

Kuleshov, V., Fenner, N., and Ermon, S. Accurate uncertainties for deep learning using calibrated regression. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Kuncheva, L. *Fuzzy Classifier Design*. Springer Science & Business Media, 2000.

Kunstner, F., Balles, L., and Hennig, P. Limitations of the empirical fisher approximation for natural gradient descent. In *NeurIPS*, 2019.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. Simple and scalable predictive uncertainty estimation using deep ensembles. In *NIPS*, 2017.

Laplace, P.-S. Mémoires de mathématique et de physique, tome sixieme. 1774.

LeCun, Y., Denker, J. S., and Solla, S. A. Optimal brain damage. In *NIPS*, 1990.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998.

Lee, J. and Humt, M. Official code: Estimating model uncertainty of neural networks in sparse information form, icml2020. `https://github.com/DLR-RM/curvature`, 2020.

Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J., and Sohl-Dickstein, J. Deep neural networks as Gaussian processes. In *ICLR*, 2018a.

Lee, J., Humt, M., Feng, J., and Triebel, R. Estimating model uncertainty of neural networks in sparse information form. In *ICML*, 2020.

Lee, K., Lee, H., Lee, K., and Shin, J. Training confidence-calibrated classifiers for detecting out-of-distribution samples. In *ICLR*, 2018b.

Lee, K., Lee, K., Lee, H., and Shin, J. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *NIPS*, 2018c.

Liang, S., Li, Y., and Srikant, R. Enhancing the reliability of out-of-distribution image detection in neural networks. In *ICLR*, 2018.

Liu, X., Li, Y., Wu, C., and Hsieh, C.-J. Adv-BNN: Improved adversarial defense through robust Bayesian neural network. In *ICLR*, 2019.

Loshchilov, I. and Hutter, F. SGDR: Stochastic gradient descent with warm restarts. In *ICLR*, 2017.

Lotfi, S., Izmailov, P., Benton, G., Goldblum, M., and Wilson, A. G. Bayesian model selection, the marginal likelihood, and generalization. *arXiv preprint arXiv:2202.11678*, 2022.

Louizos, C. and Welling, M. Structured and efficient variational deep learning with matrix Gaussian posteriors. In *ICML*, 2016.

Louizos, C. and Welling, M. Multiplicative normalizing flows for variational Bayesian neural networks. In *ICML*, 2017.

Lu, Z., Ie, E., and Sha, F. Uncertainty estimation with infinitesimal jackknife, its distribution and mean-field approximation. *arXiv preprint arXiv:2006.07584*, 2020.

MacKay, D. J. Bayesian interpolation. *Neural computation*, 4(3), 1992a.

MacKay, D. J. The evidence framework applied to classification networks. *Neural Computation*, 4(5), 1992b.

MacKay, D. J. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3), 1992c.

MacKay, D. J. Probable networks and plausible predictions—a review of practical Bayesian methods for supervised neural networks. *Network: computation in neural systems*, 1995.

MacKay, D. J. Choice of basis for laplace approximation. *Machine Learning*, 33(1), 1998.

Maddox, W. J., Izmailov, P., Garipov, T., Vetrov, D. P., and Wilson, A. G. A simple baseline for Bayesian uncertainty in deep learning. In *NeurIPS*, 2019a.

Maddox, W. J., Izmailov, P., Garipov, T., Vetrov, D. P., and Wilson, A. G. Code repo for "A Simple Baseline for Bayesian Deep Learning". `https://github.com/wjmaddox/swa_gaussian`, 2019b.

Maddox, W. J., Benton, G., and Wilson, A. G. Rethinking parameter counting in deep models: Effective dimensionality revisited. *arXiv preprint arXiv:2003.02139*, 2020.

Madras, D., Atwood, J., and D'Amour, A. Detecting extrapolation with local ensembles. In *ICLR*, 2020.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018.

Malinin, A. and Gales, M. Predictive uncertainty estimation via prior networks. In *NIPS*, 2018.

Malinin, A. and Gales, M. Reverse KL-divergence training of prior networks: Improved uncertainty and adversarial robustness. In *NIPS*, 2019.

Maroñas, J., Hamelijnck, O., Knoblauch, J., and Damoulas, T. Transforming Gaussian processes with normalizing flows. In *AISTATS*, 2021.

Martens, J. New insights and perspectives on the natural gradient method. *JMLR*, 21(146), 2020.

Martens, J. and Grosse, R. Optimizing neural networks with Kronecker-factored approximate curvature. In *International conference on machine learning*, pp. 2408–2417, 2015.

Meinke, A. and Hein, M. Towards neural networks that provably know when they don't know. In *ICLR*, 2020.

Miller, A. C., Foti, N. J., and Adams, R. P. Variational boosting: Iteratively refining posterior approximations. In *ICML*, 2017.

Minka, T. Estimating a Dirichlet distribution, 2000.

Naeini, M. P., Cooper, G., and Hauskrecht, M. Obtaining well calibrated probabilities using Bayesian binning. In *AAAI*, 2015.

Nair, V. and Hinton, G. E. Rectified linear units improve restricted Boltzmann machines. In *ICML*, 2010.

Nandy, J., Hsu, W., and Lee, M. L. Towards maximizing the representation gap between in-domain & out-of-distribution examples. In *NeurIPS*, 2020.

Neal, R. M. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

Neal, R. M. et al. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2(11), 2011.

Nguyen, A., Yosinski, J., and Clune, J. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *CVPR*, 2015.

Nguyen, C. V., Li, Y., Bui, T. D., and Turner, R. E. Variational continual learning. In *ICLR*, 2018.

Ober, S. W. and Rasmussen, C. E. Benchmarking the neural linear model for regression. *arXiv preprint arXiv:1912.08416*, 2019.

O'Hagan, A. Curve fitting and optimal design for prediction. *Journal of the Royal Statistical Society: Series B (Methodological)*, 40, 1978.

Osawa, K. ASDL: Automatic second-order differentiation (for Fisher, gradient covariance, Hessian, Jacobian, and kernel) library. `https://github.com/kazukiosawa/asdfghjkl`, 2021a.

Osawa, K. ASDL: Automatic second-order differentiation (for Fisher, gradient covariance, Hessian, Jacobian, and kernel) library. `https://github.com/kazukiosawa/asdfghjkl`, 2021b.

Osawa, K., Swaroop, S., Khan, M. E. E., Jain, A., Eschenhagen, R., Turner, R. E., and Yokota, R. Practical deep learning with Bayesian principles. In *NeurIPS*, 2019.

Ovadia, Y., Fertig, E., Ren, J., Nado, Z., Sculley, D., Nowozin, S., Dillon, J., Lakshminarayanan, B., and Snoek, J. Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift. In *NeurIPS*, 2019.

Pan, P., Swaroop, S., Immer, A., Eschenhagen, R., Turner, R. E., and Khan, M. E. Continual deep learning by functional regularisation of memorable past. In *NeurIPS*, 2020.

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., and Lakshminarayanan, B. Normalizing flows for probabilistic modeling and inference. *JMLR*, 22(57), 2021.

Park, M., Horwitz, G., and Pillow, J. W. Active learning of neural response functions with Gaussian processes. In *NIPS*, 2011.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

Platt, J. et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

Qiu, X., Meyerson, E., and Miikkulainen, R. Quantifying point-prediction uncertainty in neural networks via residual estimation with an I/O kernel. In *ICLR*, 2020.

Rahaman, R. and Thiery, A. H. Uncertainty quantification and deep ensembles. *arXiv preprint arXiv:2007.08792*, 2020.

Rasmussen, C. E. and Williams, C. K. I. *Gaussian processes in machine learning*. 2005.

Rezende, D. and Mohamed, S. Variational inference with normalizing flows. In *ICML*, 2015.

Rippel, O. and Adams, R. P. High-dimensional probability estimation with deep density models. *arXiv preprint arXiv:1302.5125*, 2013.

Riquelme, C., Tucker, G., and Snoek, J. Deep Bayesian bandits showdown: An empirical comparison of Bayesian deep networks for thompson sampling. In *ICLR*, 2018.

Ritter, H., Botev, A., and Barber, D. A scalable Laplace approximation for neural networks. In *ICLR*, 2018a.

Ritter, H., Botev, A., and Barber, D. Online structured Laplace approximations for overcoming catastrophic forgetting. In *NIPS*, 2018b.

Robbins, H. E. An empirical bayes approach to statistics. In *Proceedings of the 3rd Berkeley Symposium on Mathematical Statistics and Probability*, 1956.

Schraudolph, N. N. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14(7), 2002.

Sensoy, M., Kaplan, L., and Kandemir, M. Evidential deep learning to quantify classification uncertainty. In *NIPS*, 2018.

Sharma, A., Azizan, N., and Pavone, M. Sketching curvature for efficient out-of-distribution detection for deep neural networks. *arXiv preprint arXiv:2102.12567*, 2021.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., and Adams, R. Scalable Bayesian optimization using deep neural networks. In *ICML*, 2015.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 2013.

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

Sun, S., Zhang, G., Shi, J., and Grosse, R. Functional variational Bayesian neural networks. In *ICLR*, 2019.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the Inception architecture for computer vision. In *CVPR*, 2016.

Thiel, C. Classification on soft labels is robust against label noise. In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, 2008.

Titsias, M. K., Schwarz, J., Matthews, A. G. d. G., Pascanu, R., and Teh, Y. W. Functional regularisation for continual learning with Gaussian processes. In *ICLR*, 2020.

Tomczak, M., Swaroop, S., and Turner, R. Efficient low rank Gaussian variational inference for neural networks. In *NeurIPS*, 2020.

Torralba, A., Fergus, R., and Freeman, W. T. 80 Million Tiny Images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11), 2008.

Van Amersfoort, J., Smith, L., Teh, Y. W., and Gal, Y. Uncertainty estimation using a single deep deterministic neural network. In *ICML*, 2020.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *NIPS*, 2017.

Voorhees, E. M. Overview of the TREC-9 question answering track. In *Text REtrieval Conference (TREC)*, 2001.

Wahba, G. Improper priors, spline smoothing and the problem of guarding against model errors in regression. *Journal of the Royal Statistical Society: Series B (Methodological)*, 40, 1978.

Wahba, G. *Spline Models for Observational Data*. SIAM, 1990.

Wang, K.-C., Vicol, P., Lucas, J., Gu, L., Grosse, R., and Zemel, R. Adversarial distillation of Bayesian neural network posteriors. In *ICML*, 2018.

Wang, X. and Aitchison, L. Bayesian OOD detection with aleatoric uncertainty and outlier exposure. *arXiv preprint arXiv:2102.12959v2*, 2021.

Welling, M. and Teh, Y. W. Bayesian learning via stochastic gradient langevin dynamics. In *ICML*, 2011.

Wen, Y., Vicol, P., Ba, J., Tran, D., and Grosse, R. Flipout: Efficient pseudo-independent weight perturbations on mini-batches. In *ICLR*, 2018.

Wenger, J., Kjellström, H., and Triebel, R. Non-parametric calibration for classification. *arXiv preprint arXiv:1906.04933*, 2019.

Wenzel, F., Roth, K., Veeling, B. S., Świątkowski, J., Tran, L., Mandt, S., Snoek, J., Salimans, T., Jenatton, R., and Nowozin, S. How good is the Bayes posterior in deep neural networks really? *ICML*, 2020a.

Wenzel, F., Snoek, J., Tran, D., and Jenatton, R. Hyperparameter ensembles for robustness and uncertainty quantification. In *NeurIPS*, 2020b.

Williams, C. K. and Barber, D. Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12), 1998.

Wilson, A. G. and Izmailov, P. Bayesian deep learning and a probabilistic perspective of generalization. In *NeurIPS*, 2020.

Wilson, A. G., Hu, Z., Salakhutdinov, R., and Xing, E. P. Deep kernel learning. In *AISTATS*, 2016a.

Wilson, A. G., Hu, Z., Salakhutdinov, R. R., and Xing, E. P. Stochastic variational deep kernel learning. In *NIPS*, 2016b.

Wilson, J. T., Borovitskiy, V., Terenin, A., Mostowsky, P., and Deisenroth, M. P. Efficiently sampling functions from Gaussian process posteriors. In *ICML*, 2020.

Wu, A., Nowozin, S., Meeds, E., Turner, R. E., Hernandez-Lobato, J. M., and Gaunt, A. L. Deterministic variational inference for robust Bayesian neural networks. In *ICLR*, 2019.

Yu, F., Seff, A., Zhang, Y., Song, S., Funkhouser, T., and Xiao, J. LSUN: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

Zagoruyko, S. and Komodakis, N. Wide residual networks. In *BMVC*, 2016.

Zhang, G., Sun, S., Duvenaud, D., and Grosse, R. Noisy natural gradient as variational inference. In *ICML*, 2018.

Zhang, R., Li, C., Zhang, J., Chen, C., and Wilson, A. G. Cyclical stochastic gradient MCMC for Bayesian deep learning. In *ICLR*, 2020.

Zhang, X. and LeCun, Y. Universum prescription: Regularization using unlabeled data. In *AAAI*, 2017.

Index