Learnable Uncertainty under Laplace Approximations

Agustinus Kristiadi[1] Matthias Hein[1] Philipp Hennig[1,2]

[1]University of Tübingen, Tübingen, Germany
[2]Max Planck Institute for Intelligent Systems, Tübingen, Germany

Abstract

Laplace approximations are classic, computationally lightweight means for constructing Bayesian neural networks (BNNs). As in other approximate BNNs, one cannot necessarily expect the induced predictive uncertainty to be calibrated. Here we develop a formalism to explicitly "train" the uncertainty in a decoupled way to the prediction itself. To this end, we introduce *uncertainty units* for Laplace-approximated networks: Hidden units associated with a particular weight structure that can be added to any pre-trained, point-estimated network. Due to their weights, these units are inactive—they do not affect the predictions. But their presence changes the geometry (in particular the Hessian) of the loss landscape, thereby affecting the network's uncertainty estimates under a Laplace approximation. We show that such units can be trained via an uncertainty-aware objective, improving standard Laplace approximations' performance in various uncertainty quantification tasks.

1 INTRODUCTION

The point estimates of neural networks (NNs)—constructed as *maximum a posteriori* (MAP) estimates via regularized empirical risk minimization—empirically achieve high predictive performance. However, they tend to underestimate the uncertainty of their predictions and thus be overconfident [Nguyen et al., 2015, Guo et al., 2017], which could be disastrous in safety-critical applications such as autonomous driving. Bayesian inference offers a principled path to overcome this issue. The goal is to turn "vanilla" NNs into Bayesian neural networks (BNNs), i.e. equipping a NN with the posterior over its weights, inferred by Bayes' theorem and subsequently taken into account when making predictions [MacKay, 1992b, Neal, 1995].

Since the cost of exact posterior inference in a BNN is often prohibitive, approximate Bayesian methods are commonly employed instead. Laplace approximations (LAs) are classic methods for such a purpose [MacKay, 1992b]. Intuitively, the key idea is to obtain an approximate posterior by "surrounding" a MAP estimate of a network with a Gaussian, based on the loss landscape's geometry around it. More formally, they form a Gaussian approximation to the exact posterior, whose mean equals the network's MAP estimate and whose covariance equals the negative inverse Hessian (or approximations thereof) of the loss function, evaluated at the MAP estimate. LAs can thus be applied to any pre-trained, point-estimated network in a cost-efficient, *post-hoc* manner, especially thanks to recent advances in software toolkits for second-order optimization [Yao et al., 2019, Dangel et al., 2020]. This is in contrast to alternative approximate Bayesian methods such as variational Bayes [Hinton and Van Camp, 1993, Graves, 2011, Blundell et al., 2015] and Markov Chain Monte Carlo [Neal, 1993, Welling and Teh, 2011] which require either costly network re-training or posterior sampling.

A standard practice in contemporary LAs is to tune a single hyperparameter—the prior precision—to calibrate their predictive uncertainty [Ritter et al., 2018b]. However, this scalar parametrization allows only for a very limited form of uncertainty calibration. Below, 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, 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

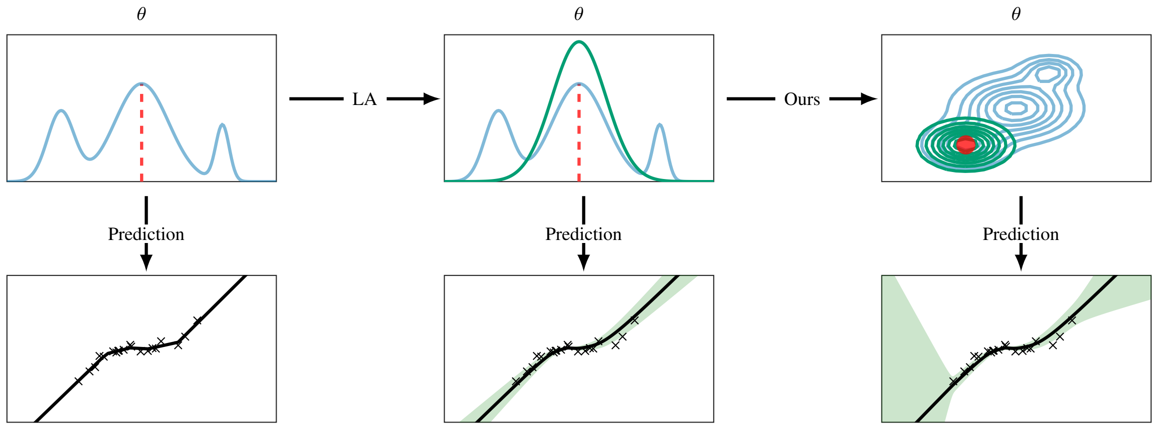


Figure 1: A schematic of our method. **Top row:** blue and green curves represent the true and the Laplace-approximated posteriors over the parameter space, respectively—the point estimates are in red. **Bottom row:** predictions induced by the respective Laplace approximation—lines and shades are predictive 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.

be trained via an uncertainty-aware objective [Lee et al., 2018, Hendrycks et al., 2019, etc.], such that they improve the predictive uncertainty quantification performance of the Laplace-approximated BNN. Figure 1 provides intuition.

In summary, we

(i) introduce uncertainty units: hidden units with a particular structure in their associated weights that can be applied to any MAP-trained network,

(ii) show that these units maintain the output of the network, while non-trivially affecting the loss landscape's curvature (the Hessian), thus also affecting predictive uncertainty under Laplace approximations, and

(iii) present a training method for the non-zero weights associated with these units via an uncertainty-aware objective so that they improve the uncertainty calibration of Laplace approximations.

2 BACKGROUND

2.1 BAYESIAN NEURAL NETWORKS

Let $f : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^k$ defined by $(x, \theta) \mapsto f(x; \theta)$ be an L-layer neural network. Here, θ is the vector of all the parameters of f. Suppose that the size of each layer of f is given by the sequence of $(n_l \in \mathbb{Z}_{>0})_{l=1}^L$. Then, for each $l = 1, \dots, L$, the l-th layer of f is defined by

$$a^{(l)} := W^{(l)} h^{(l-1)} + b^{(l)} \tag{1}$$

with

$$h^{(l)} := \begin{cases} \varphi(a^{(l)}) & \text{if } l < L \\ a^{(l)} & \text{if } l = L\,, \end{cases}$$

where $W^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ and $b^{(l)} \in \mathbb{R}^{n_l}$ are the weight matrix and bias vector of the layer, and φ is a component-wise activation function. We call the vector $h^{(l)} \in \mathbb{R}^{n_l}$ the l-th hidden units of f. Note that by convention, we consider $n_0 := n$ and $n_L := k$, while $h^{(0)} := x$ and $h^{(L)} := f(x; \theta)$.

From the Bayesian perspective, the ubiquitous training formalism of neural networks amounts to MAP estimation: The empirical risk and the regularizer are interpretable as the negative log-likelihood under an i.i.d. dataset $\mathcal{D} := \{x_i, y_i\}_{i=1}^m$ and the negative log-prior, respectively. That is, the loss function is interpreted as

$$\begin{aligned} \mathcal{L}(\theta) &:= -\sum_{i=1}^m \log p(y_i \mid f(x_i; \theta)) - \log p(\theta) \\ &= -\log p(\theta \mid \mathcal{D})\,. \end{aligned} \tag{2}$$

In this view, the *de facto* weight decay regularizer amounts to a zero-mean isotropic Gaussian prior $p(\theta) = \mathcal{N}(0, \lambda^{-1} I)$ with a scalar precision hyperparameter λ. Meanwhile, the usual softmax and quadratic output losses correspond to the Categorical and Gaussian distributions over y_i in the case of classification and regression, respectively.

MAP-trained neural networks have been shown to be overconfident [Hein et al., 2019] and BNNs can mitigate this issue [Kristiadi et al., 2020]. BNNs quantify epistemic uncertainty by inferring the full posterior distribution of the parameters θ, instead of just a single point estimate in MAP training. Given that $p(\theta \mid \mathcal{D})$ is the posterior, then the prediction for any test point $x \in \mathbb{R}^n$ is obtained via marginalization

$$p(y \mid x, \mathcal{D}) = \int p(y \mid f(x; \theta))\, p(\theta \mid \mathcal{D})\, d\theta\,, \tag{3}$$

which captures the uncertainty encoded in the posterior.

2.2 LAPLACE APPROXIMATIONS

In deep learning, since the exact Bayesian posterior is intractable, approximate Bayesian inference methods are used. Laplace approximations (LAs) are an important family of such methods. Let θ_{MAP} be the minimizer of (2), which corresponds to a mode of the posterior distribution. A LA locally approximates the posterior using a Gaussian

$$p(\theta \mid \mathcal{D}) \approx \mathcal{N}(\theta_{\text{MAP}}, \varSigma)\,,$$

where $\varSigma := (\nabla^2 \mathcal{L}|_{\theta_{\text{MAP}}})^{-1}$ is the inverse Hessian of the loss function, evaluated at the MAP estimate θ_{MAP}. Thus, LAs construct an approximate Gaussian posterior *around* θ_{MAP}, whose precision equals to the Hessian of the loss at θ_{MAP}—the "curvature" of the loss landscape at θ_{MAP}, cf. Fig. 1 (top) for an illustration.

While the covariance of a LA is tied to the weight decay of the loss, a common practice in LAs is to tune the prior precision under some objective in a *post-hoc* manner [Ritter et al., 2018b, Kristiadi et al., 2020]. In other words, the MAP estimation and the covariance inference are thought of as separate, independent processes. For example, given a fixed MAP estimate, one can maximize the log-likelihood of a LA w.r.t. the prior precision to obtain the covariance. This hyperparameter tuning can thus be thought of as an *uncertainty tuning*.

A recent example of LAs is the Kronecker-factored Laplace (KFL) [Ritter et al., 2018b]. The key idea is to approximate the Hessian matrix with the layer-wise Kronecker factorization scheme proposed by Heskes [2000], Martens and Grosse [2015]. That is, for each layer $l = 1, \dots, L$, KFL assumes that the Hessian corresponding to the l-th weight matrix $W^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ can be written as the Kronecker product $G^{(l)} \otimes A^{(l)}$ for some $G^{(l)} \in \mathbb{R}^{n_l \times n_l}$ and $A^{(l)} \in \mathbb{R}^{n_{l-1} \times n_{l-1}}$. This assumption brings the inversion cost of the Hessian down to $\Theta(n_l^3 + n_{l-1}^3)$, instead of the usual $\Theta(n_l^3 n_{l-1}^3)$ cost. Note that the approximate Hessian can easily be computed via tools such as BackPACK [Dangel et al., 2020].

Even in the case when a closed-form Laplace-approximated posterior can be obtained, the integral (3) in general does not have an analytic solution since f is nonlinear. To alleviate this, one can simply employ Monte-Carlo (MC) integration by sampling from the Gaussian:

$$p(y \mid x, \mathcal{D}) \approx \frac{1}{S} \sum_{s=1}^{S} p(y \mid f(x; \theta_s)) \quad \text{with } \theta_s \sim \mathcal{N}(\theta_{\text{MAP}}, \varSigma)\,,$$

for S number of samples.

Alternatively, a closed-form approximation to the predictive distribution—useful for analysis but has also been shown to be better than MC integration in practice [Foong et al., 2019, Immer et al., 2021]—can be obtained by linearizing the network w.r.t. its parameter at the MAP estimate.[1] That is, given any input $x \in \mathbb{R}^n$ and the Jacobian matrix $J(x) := \nabla_\theta f(x; \theta)|_{\theta_{\text{MAP}}} \in \mathbb{R}^{d \times k}$, we Taylor-approximate the network as

$$f(x; \theta) \approx f(x; \theta_{\text{MAP}}) + J(x)^\top (\theta - \theta_{\text{MAP}})\,. \tag{4}$$

Under this approximation, since θ is *a posteriori* distributed as Gaussian $\mathcal{N}(\theta_{\text{MAP}}, \varSigma)$, it follows that the marginal distribution over the network output $f(x)$ is also a Gaussian [Bishop, 2006, Sec. 5.7.3], given by

$$p(f(x) \mid x, \mathcal{D}) \sim \mathcal{N}(f(x; \theta_{\text{MAP}}), J(x)^\top \varSigma\, J(x))\,. \tag{5}$$

For classification, one can then use the so-called probit approximation [Spiegelhalter and Lauritzen, 1990, MacKay, 1992a] or its generalization [Gibbs, 1997] to obtain the predictive distribution. In the binary classification case, this is

$$\begin{aligned} p(y = 1 \mid x, \mathcal{D}) &= \int \sigma(f(x))\, p(f(x) \mid x, \mathcal{D})\, d(f(x)) \\ &\approx \sigma\left(\frac{f(x; \theta_{\text{MAP}})}{\sqrt{1 + \pi/8\, v(x)}} \right), \end{aligned} \tag{6}$$

where $v(x) := J(x)^\top \varSigma\, J(x)$ is the variance of $f(x)$ under (5). Using this approximation, we can clearly see the connection between output variance and predictive uncertainty: As $v(x)$ increases, the predictive probability becomes closer 0.5 and therefore the predictive entropy increases.

3 LEARNABLE UNCERTAINTY UNITS UNDER LAPLACE APPROXIMATIONS

In this section, we introduce *uncertainty units*, which can be added to the layers of any MAP-trained network (Section 3.1) and trained via an uncertainty-aware loss (Section 3.2) to improve uncertainty calibration under Laplace approximations. All proofs are in Appendix A.

3.1 CONSTRUCTION

Let $f : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^k$ be a MAP-trained L-layer neural network with parameters $\theta_{\text{MAP}} = (W^{(l)}_{\text{MAP}}, b^{(l)}_{\text{MAP}})_{l=1}^{L}$. The premise of our method is simple: At each hidden layer $l = 1, \dots, L-1$, we add $m_l \in \mathbb{Z}_{\geq 0}$ additional hidden units (under the original activation function) to $h^{(l)}$—as a consequence, the l-th weight matrix and bias vector need to be extended to accommodate them. Our 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.

[1]The resulting network is still non-linear in its input, but linear in its parameters.

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

$$\widetilde{W}^{(l)} := \begin{pmatrix} W^{(l)}_{\text{MAP}} & 0 \\ \widehat{W}^{(l)}_1 & \widehat{W}^{(l)}_2 \end{pmatrix} \in \mathbb{R}^{(n_l+m_l)\times(n_{l-1}+m_{l-1})} \, ,$$

$$\widetilde{b}^{(l)} := \begin{pmatrix} b^{(l)}_{\text{MAP}} \\ \widehat{b}^{(l)} \end{pmatrix} \in \mathbb{R}^{n_l+m_l} \, , \tag{7}$$

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 $l = L$, we define

$$\begin{aligned} \widetilde{W}^{(L)} &:= (W^{(L)}_{\text{MAP}}, 0) \in \mathbb{R}^{k \times (n_{L-1}+m_{L-1})} \, ; \\ \widetilde{b}^{(L)} &:= b^{(L)}_{\text{MAP}} \in \mathbb{R}^{k} \, , \end{aligned} \tag{8}$$

so that the output dimensionality is also unchanged. For brevity, we denote by $\widehat{\theta}^{(l)}$ the non-zero additional parameters in (7), i.e. we define $\widehat{\theta}^{(l)}$ to be the tuple $(\widehat{W}^{(l)}_1, \widehat{W}^{(l)}_2, \widehat{b}^{(l)})$. Altogether, considering all layers $l = 1, \dots, L-1$, we denote

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

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 $(\widetilde{W}^{(l)}, \widetilde{b}^{(l)})_{l=1}^{L}$—as $\widetilde{\theta}_{\text{MAP}} \in \mathbb{R}^{\widetilde{d}}$, where $\widetilde{d}$ is the resulting number of parameters. Refer to Fig. 2 for an illustration and Algorithm 2 in Appendix B for a step-by-step summary. Note that we can easily extend this construction to convolutional networks by expanding the "channel" of hidden convolution layers.[2]

Let us inspect the implication of this construction. Here for each $l = 1, \dots, L-1$, the sub-matrices $\widehat{W}^{(l)}_1$, $\widehat{W}^{(l)}_2$ and the sub-vector $\widehat{b}^{(l)}$ 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 $\widetilde{W}^{(l)}$, i.e. the zero part of the additional weights, *deactivates* the m_{l-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.

Proposition 1 (Properties). *Let $f : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^k$ be a MAP-trained L-layer network 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 $\widetilde{\theta}_{\text{MAP}} \in \mathbb{R}^{\widetilde{d}}$ are obtained via the previous construction, and $\widetilde{\mathcal{L}}$ is the resulting loss function under $\widetilde{f}$.*

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

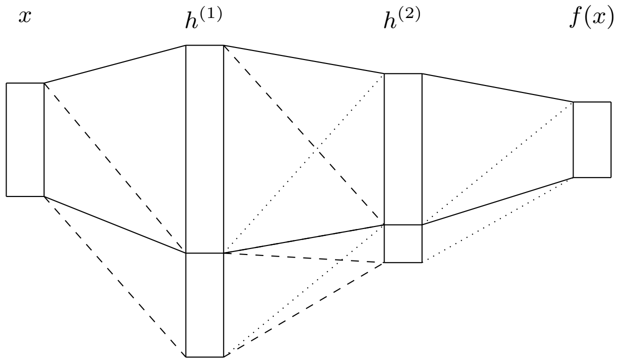


Figure 2: An illustration of the proposed construction. **Rectangles** represent layers, **solid lines** represent connection between layers, given by the original weight matrices $W^{(1)}_{\text{MAP}}, \dots, W^{(L)}_{\text{MAP}}$. The additional units are represented by the additional block at the bottom of each layer. **Dashed lines** correspond to the free parameters $\widehat{\theta}$, while **dotted lines** to the zero weights.

(a) For an arbitrary input $x \in \mathbb{R}^n$, we have $\widetilde{f}(x; \widetilde{\theta}_{\text{MAP}}) = f(x; \theta_{\text{MAP}})$.

(b) The gradient of $\widetilde{\mathcal{L}}$ w.r.t. the additional weights in $\widetilde{W}^{(L)}$ is non-linear in $\widehat{\theta}$.

Proof Sketch. Part (a) is straightforward. For part (b), we can show that the gradient of the network output w.r.t. the additional zero weight in (8) is given by the additional hidden units of the previous layer. Note that these hidden units are nonlinear in the additional weights induced by LULA, due to the structure (7). The result then follows immediately by the chain rule. The full proof is in Appendix A. □

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 (8) 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. (The usage of the network linearization below is necessary for analytical tractability.)

Proposition 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-approximated posteriors $\mathcal{N}(\theta_{\text{MAP}}, \text{diag}(\sigma))$, $\mathcal{N}(\widetilde{\theta}_{\text{MAP}}, \text{diag}(\widetilde{\sigma}))$ are employed for f and $\widetilde{f}$, respectively. Under the linearization (4), for any input $x \in \mathbb{R}^n$, the variance over the output $\widetilde{f}(x; \widetilde{\theta})$ is at least that of $f(x; \theta)$.*

In summary, the construction along with Propositions 1 and 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 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.

3.2 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., 2018, 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):

$$\begin{aligned}\mathcal{L}_{\text{LULA}}(\widetilde{\theta}_{\text{MAP}}) := \frac{1}{|\mathcal{D}_{\text{in}}|} \sum_{x_{\text{in}} \in \mathcal{D}_{\text{in}}} H[p(y \mid x_{\text{in}}, \mathcal{D}; \widetilde{\theta}_{\text{MAP}})] \\ - \frac{1}{|\mathcal{D}_{\text{out}}|} \sum_{x_{\text{out}} \in \mathcal{D}_{\text{out}}} H[p(y \mid x_{\text{out}}, \mathcal{D}; \widetilde{\theta}_{\text{MAP}})] \,,\end{aligned} \tag{9}$$

and minimize it w.r.t. 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 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 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 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 (9). These parameters, then, when used in a LA, improve the predictive uncertainty of standard non-LULA-augmented LAs.

Algorithm 1 Training LULA units.

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

 1: Construct $\widetilde{f}$ from $f$ by following Section 3.1.
 2: for $i = 1, \dots, E$ do
 3:     $q(\widetilde{\theta}) = \mathcal{N}(\widetilde{\theta}_{\text{MAP}}, \widetilde{\Sigma}(\widetilde{\theta}_{\text{MAP}}))$
 4:     Compute $\mathcal{L}_{\text{LULA}}(\widetilde{\theta}_{\text{MAP}})$ via (9) with $q(\widetilde{\theta})$, $\mathcal{D}$, $\mathcal{D}_{\text{out}}$
 5:     $g = \nabla \mathcal{L}_{\text{LULA}}(\widetilde{\theta}_{\text{MAP}})$
 6:     $\widehat{g} = \texttt{mask\_gradient}(g)$
 7:     $\widetilde{\theta}_{\text{MAP}} = \widetilde{\theta}_{\text{MAP}} - \alpha \widehat{g}$
 8: end for
 9: $p(\widetilde{\theta} \mid \mathcal{D}) \approx \mathcal{N}(\widetilde{\theta}_{\text{MAP}}, \widetilde{\Sigma}(\widetilde{\theta}_{\text{MAP}}))$
10: return $\widetilde{f}$ and $p(\widetilde{\theta} \mid \mathcal{D})$
```

3.2.1 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 (7) and (8). This can be enforced by gradient masking: For all $l = 1, \dots, L-1$, set the gradients of the blocks of $\widetilde{W}^{(l)}$ and $\widetilde{b}^{(l)}$ not corresponding to $\widehat{W}_1^{(l)}$, $\widehat{W}_2^{(l)}$, and $\widehat{b}^{(l)}$, to zero. Under this scheme, Proposition 1(a) will still hold for trained LULA units.

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, 2014], obtained from a single minibatch, irrespective of the Laplace approximation variant employed at test time—we show in Section 5 that this training scheme is both effective and efficient.[3] 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 1 provides a summary of LULA training in pseudocode. Code can be found in `https://github.com/wiseodd/lula`.

4 RELATED WORK

While traditionally hyperparameter optimization in LAs requires re-training the network (under type-II maximum likelihood or the evidence framework [MacKay, 1992b] or empirical Bayes [Robbins, 1956]), tuning it in a *post-hoc* manner has become increasingly common. Ritter et al. [2018a,b] tune the prior precision of a LA by maximizing the predictive log-likelihood. Kristiadi et al. [2020] extend this procedure by also using outliers to better calibrate the uncertainty. 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 and Hein [2020] use outliers to regularize the standard maximum-likelihood training. Malinin and 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*.

5 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.

5.1 SETUP

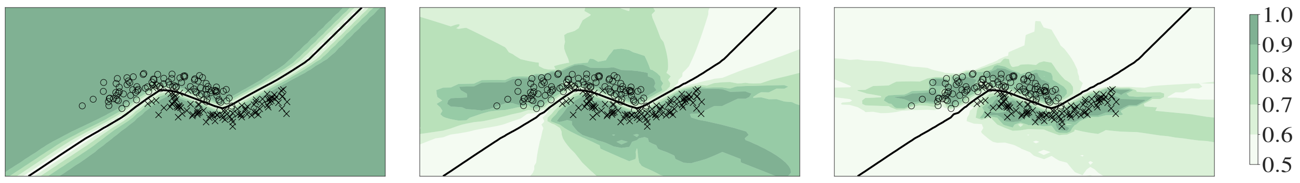
Toy experiments We use the "cubic" [Hernández-Lobato and Adams, 2015] and "two moons" datasets for regression and classification, respectively. For classification, we use a full Laplace with generalized Gauss-Newton Hessian approximation on a three-layer FC network. For regression, we apply the Kronecker-factored Laplace (KFL) [Ritter et al., 2018b] on a two-layer fully-connected network. In this particular case, we directly use the predictive variance instead of (differential) entropy for Eq. (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 and 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 and 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 and 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 Hendrycks et al. [2019], Meinke and 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

[3]The actual Laplace approximations used in all experiments are *non*-diagonal.



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

Figure 3: Predictive uncertainty estimates of a standard LA and the LULA-augmented LA. Black curves and shades are decision boundaries and confidence estimates, respectively.

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, 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.

5.2 TOY EXPERIMENTS

We begin with toy regression and classification results in Fig. 1 (bottom) and Fig. 3, 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 1: 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	$\underline{2.0}$±0.0	4.5±0.0	$\underline{4.1}$±0.0
DE	$\underline{13.2}$±0.0	4.3±0.0	6.1±0.0	5.4±0.0
DE-Temp	16.9±0.0	2.2±0.0	$\underline{3.8}$±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
Acc. ↑				
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

5.3 IMAGE CLASSIFICATIONS

5.3.1 Calibration

Table 1 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 6 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.

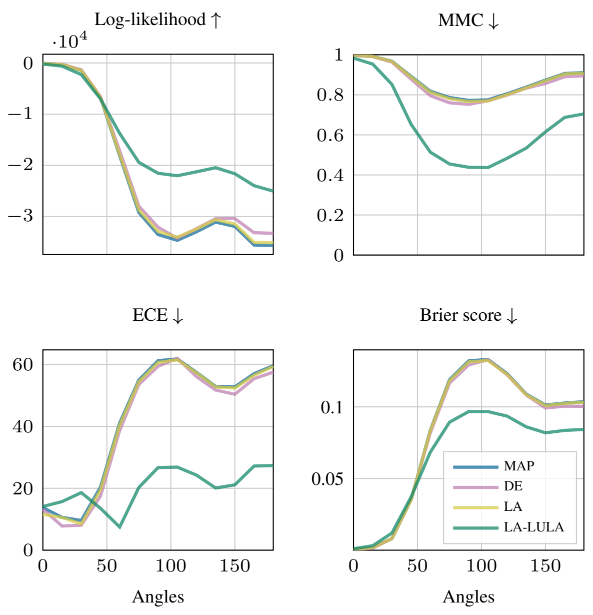


Figure 4: Values of various uncertainty metrics as the rotation angle on MNIST images increases.

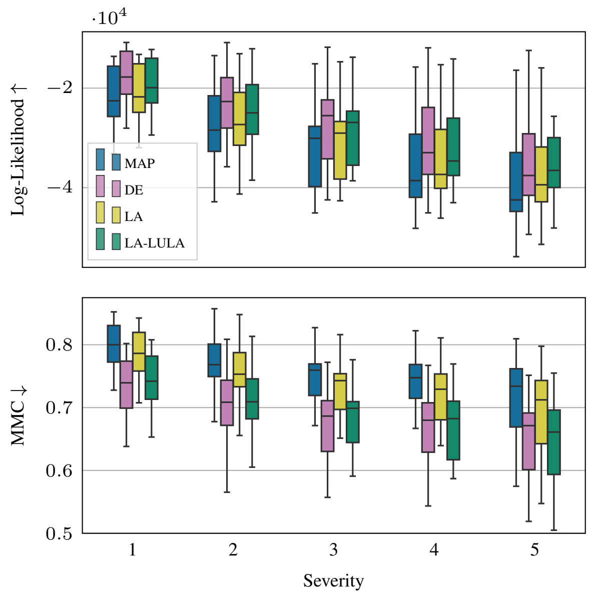


Figure 5: Summarized uncertainty quantification performance at each severity level of the CIFAR-10-C dataset.

5.3.2 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. 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. 5. 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].

5.3.3 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 2 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, LULA is orthogonal to prior variance tuning methods commonly done in Laplace approximations. Hence, in Table 2 we also show the OOD detection performance of LULA when applied to the LLLA baseline. 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.

5.4 COST

Table 3 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.

Table 2: 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 $\downarrow$				
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 $\downarrow$				
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

Table 3: Wall-clock time in seconds for augmenting the WideResNet-16-4 network with 512 LULA units and training them for ten epochs with a validation set of size 2000.

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

6 CONCLUSION

We have proposed LULA units: hidden units associated with partially zero weights that can be added to any pre-trained MAP network for the purpose of exclusively tuning the uncertainty of a Laplace approximation without affecting its predictive performance. The crux of LULA is the observation that these units induce additional dimensions and thus degrees of freedom in the network's parameter space that do not affect the network output. However, these additional parameters *do* non-trivially affect the curvature of the loss landscape and therefore the covariance matrices of Laplace approximations. Because of this, LULA units are indeed "uncertainty units". They can, moreover, be trained via an objective that depends on both inlier and outlier datasets to calibrate the network's predictive uncertainty estimates. We show empirically that LULA provides a cheap yet effective *post-hoc* uncertainty tuning for Laplace approximations.

Acknowledgements

The authors gratefully acknowledge financial support by the European Research Council through ERC StG Action 757275 / PANAMA; the DFG Cluster of Excellence "Machine Learning - New Perspectives for Science", EXC 2064/1, project number 390727645; the German Federal Ministry of Education and Research (BMBF) through the Tübingen AI Center (FKZ: 01IS18039A); and funds from the Ministry of Science, Research and Arts of the State of Baden-Württemberg. The authors are also grateful to all the anonymous reviewers for their critical and helpful feedback. AK is grateful to the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for support. AK also thanks all members of the Methods of Machine Learning group for helpful feedback.

References

Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

Julian Bitterwolf, Alexander Meinke, and Matthias Hein. Certifiably Adversarially Robust Detection of Out-of-Distribution Data. In *NeurIPS*, 2020.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight Uncertainty in Neural Networks. In *ICML*, 2015.

Glenn W Brier. Verification of Forecasts Expressed in Terms of Probability. *Monthly Weather Review*, 78(1), 1950.

Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A Downsampled Variant of ImageNet as an Alternative to the CIFAR Datasets. *arXiv preprint arXiv:1707.08819*, 2017.

Felix Dangel, Frederik Kunstner, and Philipp Hennig. BackPACK: Packing more into Backprop. In *ICLR*, 2020.

Andrew YK Foong, Yingzhen Li, José Miguel Hernández-Lobato, and Richard E Turner. 'In-Between' Uncertainty in Bayesian Neural Networks. *arXiv preprint arXiv:1906.11537*, 2019.

Mark N Gibbs. *Bayesian Gaussian Processes for Regression and Classification*. PhD thesis, Department of Physics, University of Cambridge, 1997.

Alex Graves. Practical Variational Inference for Neural Networks. In *NIPS*, 2011.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *ICML*, 2017.

Matthias Hein, Maksym Andriushchenko, and Julian Bitterwolf. Why ReLU Networks Yield High-confidence Predictions Far Away from the Training Data and How to Mitigate the Problem. In *CVPR*, 2019.

Dan Hendrycks, Mantas Mazeika, and Thomas Dietterich. Deep Anomaly Detection with Outlier Exposure. In *ICLR*, 2019.

José Miguel Hernández-Lobato and Ryan Adams. Probabilistic Backpropagation for Scalable Learning of Bayesian Neural Networks. In *ICML*, 2015.

Tom Heskes. On "Natural" Learning and Pruning in Multilayered Perceptrons. *Neural Computation*, 12(4), 2000.

Geoffrey E Hinton and Drew Van Camp. Keeping the Neural Networks Simple by Minimizing the Description Length of the Weights. In *COLT*, 1993.

Alexander Immer, Maciej Korzepa, and Matthias Bauer. Improving Predictions of Bayesian Neural Networks via Local Linearization. In *AISTATS*, 2021.

Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. Being Bayesian, Even Just a Bit, Fixes Overconfidence in ReLU Networks. In *ICML*, 2020.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In *NIPS*, 2017.

Kimin Lee, Honglak Lee, Kibok Lee, and Jinwoo Shin. Training Confidence-calibrated Classifiers for Detecting Out-of-Distribution Samples. In *ICLR*, 2018.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts. In *ICLR*, 2017.

David JC MacKay. The Evidence Framework Applied to Classification Networks. *Neural computation*, 1992a.

David JC MacKay. A Practical Bayesian Framework For Backpropagation Networks. *Neural computation*, 4(3), 1992b.

Andrey Malinin and Mark Gales. Predictive Uncertainty Estimation via Prior Networks. In *NIPS*, 2018.

Andrey Malinin and Mark Gales. Reverse KL-Divergence Training of Prior Networks: Improved Uncertainty and Adversarial Robustness. In *NIPS*, 2019.

James Martens. New Insights and Perspectives on the Natural Gradient Method. *arXiv preprint arXiv:1412.1193*, 2014.

James Martens and Roger Grosse. Optimizing Neural Networks With Kronecker-Factored Approximate Curvature. In *ICML*, 2015.

Alexander Meinke and Matthias Hein. Towards Neural Networks that Provably Know when They don't Know. In *ICLR*, 2020.

Mahdi Pakdaman Naeini, Gregory Cooper, and Milos Hauskrecht. Obtaining Well Calibrated Probabilities Using Bayesian Binning. In *AAAI*, 2015.

Radford M Neal. Bayesian Learning via Stochastic Dynamics. In *NIPS*, 1993.

Radford M Neal. *Bayesian Learning for Neural Networks*. PhD thesis, University of Toronto, 1995.

Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images. In *CVPR*, 2015.

Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, David Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can You Trust Your Model's Uncertainty? Evaluating Predictive Uncertainty under Dataset Shift. In *NeurIPS*, 2019.

Rahul Rahaman and Alexandre H Thiery. Uncertainty Quantification and Deep Ensembles. *arXiv preprint arXiv:2007.08792*, 2020.

Hippolyt Ritter, Aleksandar Botev, and David Barber. Online Structured Laplace Approximations for Overcoming Catastrophic Forgetting. In *NIPS*, 2018a.

Hippolyt Ritter, Aleksandar Botev, and David Barber. A Scalable Laplace Approximation for Neural Networks. In *ICLR*, 2018b.

Herbert E Robbins. An Empirical Bayes Approach to Statistics. In *Proceedings of the 3rd Berkeley Symposium on Mathematical Statistics and Probability*, 1956.

David J Spiegelhalter and Steffen L Lauritzen. Sequential Updating of Conditional Probabilities on Directed Graphical Structures. *Networks*, 1990.

Antonio Torralba, Rob Fergus, and William T Freeman. 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.

Max Welling and Yee W Teh. Bayesian Learning via Stochastic Gradient Langevin Dynamics. In *ICML*, 2011.

Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael Mahoney. PyHessian: Neural Networks Through the Lens of the Hessian. *arXiv preprint arXiv:1912.07145*, 2019.

Sergey Zagoruyko and Nikos Komodakis. Wide Residual Networks. In *BMVC*, 2016.