Laplace Redux – Effortless Bayesian Deep Learning

Erik Daxberger[*,c,m] **Agustinus Kristiadi**[*,t] **Alexander Immer**[*,e,p] **Runa Eschenhagen**[*,t]
Matthias Bauer[d] **Philipp Hennig**[t,m]

[c]University of Cambridge
[m]MPI for Intelligent Systems, Tübingen
[t]University of Tübingen
[e]Department of Computer Science, ETH Zurich
[p]Max Planck ETH Center for Learning Systems
[d]DeepMind, London

Abstract

Bayesian formulations of deep learning have been shown to have compelling theoretical properties and offer practical functional benefits, such as improved predictive uncertainty quantification and model selection. The Laplace approximation (LA) is a classic, and arguably the simplest family of approximations for the intractable posteriors of deep neural networks. Yet, despite its simplicity, the LA is not as popular as alternatives like variational Bayes or deep ensembles. This may be due to assumptions that the LA is expensive due to the involved Hessian computation, that it is difficult to implement, or that it yields inferior results. In this work we show that these are misconceptions: we (i) review the range of variants of the LA including versions with minimal cost overhead; (ii) introduce `laplace`, an easy-to-use software library for PyTorch offering user-friendly access to all major flavors of the LA; and (iii) demonstrate through extensive experiments that the LA is competitive with more popular alternatives in terms of performance, while excelling in terms of computational cost. We hope that this work will serve as a catalyst to a wider adoption of the LA in practical deep learning, including in domains where Bayesian approaches are not typically considered at the moment.

`laplace` **code:** `https://github.com/AlexImmer/Laplace`

1 Introduction

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 [1], (ii) catastrophic forgetting of previously learned tasks when continuously trained on new tasks [2], and (iii) the difficulty of selecting suitable NN architectures and hyperparameters [3]. Bayesian modeling [4, 5] provides a principled and unified approach to tackle these issues by (i) equipping models with robust uncertainty estimates [6], (ii) enabling models to learn continually by capturing past information [7], and (iii) allowing for automated model selection by optimally trading off data fit and model complexity [8].

Even though this provides compelling motivation for using *Bayesian neural networks* (BNNs) [9], 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.

[*]Equal contributors; author ordering sampled uniformly at random. Correspondence to: `ead54@cam.ac.uk`, `agustinus.kristiadi@uni-tuebingen.de`, `alexander.immer@inf.ethz.ch`, `runa.eschenhagen@student.uni-tuebingen.de`.

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

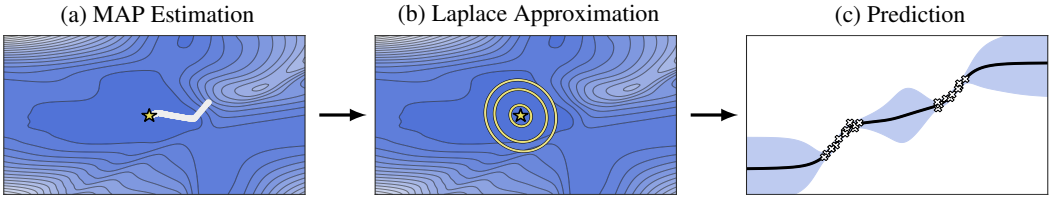


Figure 1: Probabilistic predictions with the Laplace approximation in three steps. (a) We find a MAP estimate (yellow star) via standard training (background contours = log-posterior landscape on the two-dimensional PCA subspace of the SGD trajectory [30]). **(b)** We locally approximate the posterior landscape by fitting a Gaussian centered at the MAP estimate (yellow contours), with covariance matrix equal to the negative inverse Hessian of the loss at the MAP—this is the Laplace approximation (LA). **(c)** We use the LA to make predictions with *predictive uncertainty estimates*—here, the black curve is the predictive mean, and the shading covers the 95% confidence interval.

For instance, popular variational Bayesian methods [10–12, etc.] require considerable changes to the training procedure and model architecture. Also, their optimization process is slower and typically more unstable unless carefully tuned [13]. Other methods, such as deep ensembles [14], Monte Carlo dropout [6], and SWAG [15] 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 paper 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 [16], the LA dates back to the 18th century [17]. 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, 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 [18–20] and easy-to-use software libraries [21]. 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 LA to the marginal likelihood may even be used for Bayesian model selection or NN training [8, 22]. Figure 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 [23–29], 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. To this end, our work makes the following contributions:

1. We first survey recent advances and present the key components of scalable and practical Laplace approximations in deep learning (Section 2).
2. We then introduce `laplace`, an easy-to-use PyTorch-based library for "turning a NN into a BNN" via the LA (Section 3). `laplace` implements a wide range of different LA variants.
3. Lastly, using `laplace`, we show in an extensive empirical study that the LA is competitive to alternative approaches, especially considering how simple and cheap it is (Section 4).

2 The Laplace Approximation 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* (see Eq. (5) below) to enable *probabilistic predictions* (as also illustrated in Fig. 1). Secondly, we can use the LA to approximate the *model evidence* (see Eq. (6)) 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 $\mathcal{D} := \{(x_n \in \mathbb{R}^M, y_n \in \mathbb{R}^C)\}_{n=1}^N$, the weights $\theta \in \mathbb{R}^D$ of an L-layer NN $f_\theta : \mathbb{R}^M \to \mathbb{R}^C$ are trained to minimize the (regularized) empirical risk, which typically decomposes into a sum over empirical loss terms $\ell(x_n, y_n; \theta)$ and a regularizer $r(\theta)$,

$$\theta_{\text{MAP}} = \arg\min_{\theta \in \mathbb{R}^D} \mathcal{L}(\mathcal{D}; \theta) = \arg\min_{\theta \in \mathbb{R}^D} \left(r(\theta) + \sum_{n=1}^N \ell(x_n, y_n; \theta) \right). \tag{1}$$

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:

$$\ell(x_n, y_n; \theta) = -\log p(y_n \mid f_\theta(x_n)) \qquad \text{and} \qquad r(\theta) = -\log p(\theta) \tag{2}$$

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 $p(\theta) = \mathcal{N}(\theta; 0, \gamma^2 I)$, 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

$$p(\theta \mid \mathcal{D}) = \frac{1}{Z} p(\mathcal{D} \mid \theta)\, p(\theta) = \frac{1}{Z} \exp(-\mathcal{L}(\mathcal{D}; \theta)), \qquad Z := \int p(\mathcal{D} \mid \theta)\, p(\theta)\, d\theta \tag{3}$$

with an intractable ***normalizing constant*** Z. ***Laplace approximations*** [17] 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:

$$\mathcal{L}(\mathcal{D}; \theta) \approx \mathcal{L}(\mathcal{D}; \theta_{\text{MAP}}) + \tfrac{1}{2}(\theta - \theta_{\text{MAP}})^\intercal \left(\nabla^2_\theta \mathcal{L}(\mathcal{D}; \theta)|_{\theta_{\text{MAP}}} \right) (\theta - \theta_{\text{MAP}}), \tag{4}$$

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

Laplace posterior approximation

$$p(\theta \mid \mathcal{D}) \approx \mathcal{N}(\theta; \theta_{\text{MAP}}, \Sigma) \qquad \text{with} \qquad \Sigma := -\left(\nabla^2_\theta \mathcal{L}(\mathcal{D}; \theta)|_{\theta_{\text{MAP}}} \right)^{-1}. \tag{5}$$

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

Laplace approximation of the evidence

$$Z \approx \exp(-\mathcal{L}(\mathcal{D}; \theta_{\text{MAP}}))\, (2\pi)^{D/2}\, (\det \Sigma)^{1/2}. \tag{6}$$

See Appendix A for more details. 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 Figure 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 [21]. LAs are widely used to approximate the posterior distribution in logistic regression [31], Gaussian process classification [32, 33], and also for Bayesian neural networks (BNNs), both shallow [34] and deep [23]. 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 $p(\theta) = \mathcal{N}(\theta; 0, \gamma^2 I)$ 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:

$$\nabla^2_\theta \mathcal{L}(\mathcal{D}; \theta)|_{\theta_{\text{MAP}}} = -\gamma^{-2} I - \sum_{n=1}^N \nabla^2_\theta \log p(y_n \mid f_\theta(x_n))|_{\theta_{\text{MAP}}}. \tag{7}$$

A naive implementation of the Hessian is infeasible because the second term in Eq. (7) scales quadratically with the number of network parameters, which can be in the millions or even billions [35, 36]. 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 for an overview and Appendix B for a more detailed version of the review and discussion.

Four Components of Scalable Laplace Approximations for Deep Neural Networks

① Inference on 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 networks is

[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.

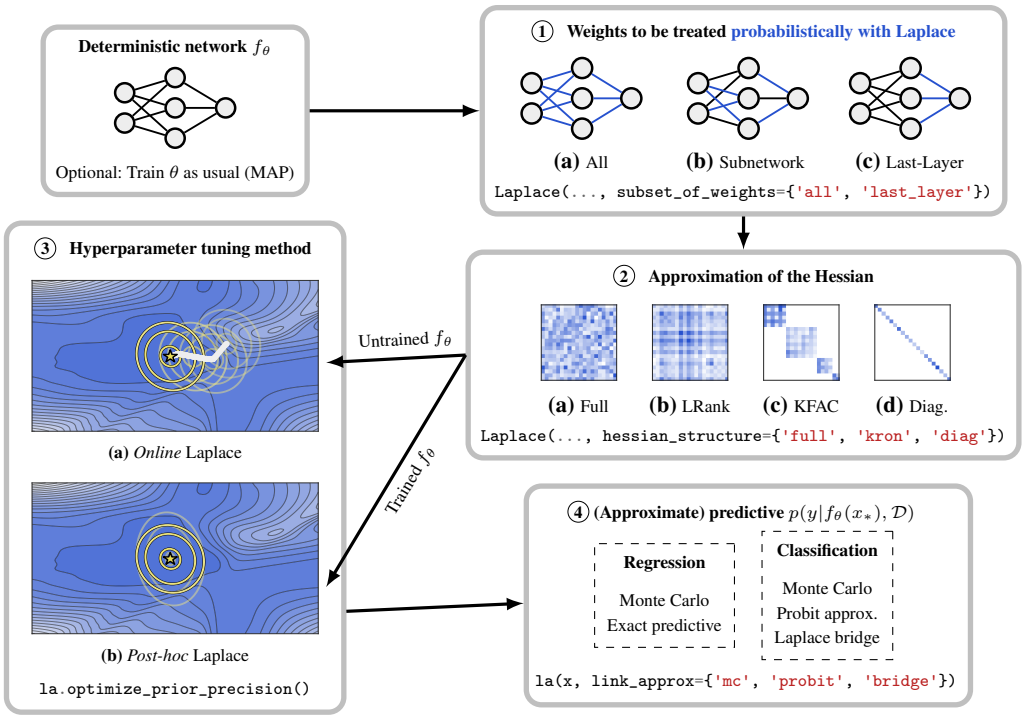


Figure 2: Four key components to scale and apply the LA to a network f_θ (with randomly-initialized or pre-trained weights θ), with corresponding `laplace` code. ① We first choose which part of the model we want to perform inference over with the LA. ② We then select how to to approximate the Hessian. 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 yellow contours). ④ Finally, to make predictions for a new input x_*, we have several options for computing/approximating the predictive distribution $p(y|f_\theta(x_*), \mathcal{D})$.

to treat only a subset of weights probabilistically with the LA and to leave the remaining weights at their MAP-estimated values. One way is to partition an L-layer deep net into a fixed feature extractor, comprising its first $L-1$ layers, and its last linear layer [37, 28]. This ***last-layer LA*** is cost-effective yet compelling both theoretically and in practice [28]. Alternatively, one can also consider a *general subset* of θ to yield a ***subnetwork LA*** [27], which is intuitively motivated by recent findings that NNs can be heavily pruned without sacrificing test accuracy[3] [38].

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 Eq. (7) [39]. The ***Fisher information matrix*** [40], abbreviated as *the Fisher* and defined by

$$F := \sum_{n=1}^{N} \mathbb{E}_{\widehat{y}\sim p(y \mid f_\theta(x_n))} \left[(\nabla_\theta \log p(\widehat{y} \mid f_\theta(x_n))|_{\theta_{\text{MAP}}}) (\nabla_\theta \log p(\widehat{y} \mid f_\theta(x_n))|_{\theta_{\text{MAP}}})^\intercal \right], \tag{8}$$

is one such choice.[4] One can also use the ***generalized Gauss-Newton matrix (GGN)*** matrix [42]

$$G := \sum_{n=1}^{N} J(x_n) \left(\nabla_f^2 \log p(y_n \mid f)|_{f=f_{\theta_{\text{MAP}}}(x_n)} \right) J(x_n)^\intercal, \tag{9}$$

[3] Note that Daxberger et al. [27] do *not* prune weights, but only consider a subset of weights *to do inference over*. So instead of zeroing out weights themselves, they zero out (co-)variances between/of weights.

[4] If, instead of taking expectation in (8), we use the training label y_n, we call the matrix the ***empirical Fisher***, which is distinct from the Fisher [39, 41].

where $J(x_n) := \nabla_\theta f_\theta(x_n)|_{\theta_{\text{MAP}}}$ is the NN's Jacobian matrix. As the Fisher and GGN are equivalent for common log-likelihoods [39], we will henceforth refer to them interchangeably. In deep LAs, they have emerged as the default choice [23, 24, 28, 29, 27, 26, 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 [43, 44]. More expressive alternatives are block-diagonal factorizations such as ***Kronecker-factored approximate curvature (KFAC)*** [18–20], which factorizes each within-layer Fisher[5] as a Kronecker product of two smaller matrices. KFAC has been successfully applied to the LA [23, 24] and can be improved by low-rank approximations of the KFAC factors [29] by leveraging their eigendecompositions [45]. Finally, recent work has studied/enabled ***low-rank approximations*** of the Hessian/Fisher [46–48].

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 [23, 28, 27, 26, 22]. Commonly, this is done through ***cross-validation***, e.g. by maximizing the validation log-likelihood [23, 49] or, additionally, using out-of-distribution data [28, 50]. When using the LA, however, ***marginal likelihood maximization*** (a.k.a. ***empirical Bayes*** or ***the evidence framework*** [34, 51]) constitutes a more principled alternative to tune these hyperparameters, and requires no validation data. Immer et al. [22] 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 [52] or the addition of dedicated, trainable hidden units for the sole purpose of uncertainty tuning [50].

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 $(\theta_s)_{s=1}^S$ from $p(\theta \mid \mathcal{D})$: $p(y \mid f(x_*), \mathcal{D}) \approx S^{-1} \sum_{s=1}^S p(y \mid f_{\theta_s}(x_*))$. However, for LAs with GGN and Fisher Hessian approximations Monte Carlo integration can perform poorly [49, 26]. Immer et al. [26] 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 [28]. 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 $p(\theta \mid \mathcal{D}) \approx \mathcal{N}(\theta; \theta_{\text{MAP}}, \Sigma)$ 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 $p(f_* \mid x_*, \mathcal{D}) \approx \mathcal{N}(f_*; f_{\theta_{\text{MAP}}}(x_*), J(x_*)^\intercal \Sigma J(x_*))$ and is typically significantly lower-dimensional (number of outputs C instead of parameters D) [25]. Given the distribution on neural network 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: $p(y \mid x_*, \mathcal{D}) \approx \mathcal{N}(y; f_{\theta_{\text{MAP}}}(x_*), J(x_*)^\intercal \Sigma J(x_*) + \sigma^2 I)$. 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 approximation*** [31, 16] which approximates the logistic function with the probit function. In the multi-class case, we can use its generalization, the ***extended probit approximation*** [53]. Finally, first proposed for non-BNN applications [54, 55], the ***Laplace bridge*** approximates the softmax-Gaussian integral via a Dirichlet distribution [56]. The key advantage is that it yields a *distribution* of the integral solutions.

3 `laplace`: A Toolkit for Deep Laplace Approximations

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 im-

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

```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)
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 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.

```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)

# Differentiate marginal likelihood w.r.t.
# prior precision and observation noise
ml = la.marglik(prior_precision=prior_prec,
                sigma_noise=obs_noise)
ml.backward()
```

Listing 2: Fit KFAC LA over the last layer of a pre- or un-trained regression model and differentiate its marginal likelihood w.r.t. some hyperparameters for *post-hoc* hyperparameter tuning or online empirical Bayes (see Immer et al. [22]).

plementation of various LA flavors—instead, it is common for deep learning researchers to repeatedly re-implement the LA and Hessian computation with varying efficiency [57–59, 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`: a simple, easy-to-use, extensible library for scalable LAs of deep NNs in PyTorch [60]. `laplace` enables *all* possible combinations of the four components discussed in Section 2—see Fig. 2 for details. Listings 1 and 2 show code examples.

The core of `laplace` 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 [21] and ASDL [61]. Moreover, we design `laplace` in a modular manner that makes it easy to add new backends and approximations in the future. For (ii), we follow Immer et al. [22] 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. Note that `laplace` also supports standard cross-validation for hyperparameter tuning [23, 28], as shown in Listing 1. Finally, for (iii), `laplace` supports all approximations to the posterior predictive distribution discussed in Section 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 (Section 2), we provide the following default choices based on our extensive experiments (Section 4); 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 1 and 2).

Limitations Because `laplace` employs external libraries (BackPACK [21] and ASDL [61]) as backends, it inherits the available choices of Hessian factorizations from these libraries. For instance, the LA variant proposed by Lee et al. [29] can currently not be implemented via `laplace`, because neither backend supports eigenvalue-corrected KFAC [45] (yet). Similarly, the first version of `laplace` does not yet support the subnetwork LA [27]—we plan to add it in the next iteration of the library.

4 Experiments

We benchmark various LAs implemented via `laplace`. Section 4.1 addresses the question of "which are the best design choices for the LA", in light of Figure 2. Section 4.2 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 4.3 demonstrates the applicability of the (last-layer) LA on various data modalities and NN architectures (including transformers [62])—settings where other Bayesian methods are challenging to implement.

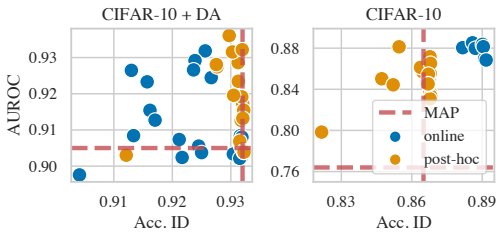


Figure 3: In- vs. out-of-distribution (ID and OOD, resp.) performance on CIFAR-10 of different LA configurations (dots), each being a combination of settings for 1) subset-of-weights, 2) covariance structure, 3) hyperparameter tuning, and 4) predictive approximation (see Appendix C.1 for details). "DA" 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.

Table 1: OOD detection performance averaged over all test sets (see Appendix C.2 for details). Confidence is defined as the max. of the predictive probability vector [63] (e.g. Confidence$([0.7, 0.2, 0.1]) = 0.7$). LA and especially LA* reduce the overconfidence of MAP and achieve better results than the VB, CSGHMC (`HMC`), and SWAG (`SWG`) 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
HMC	69.2±1.7	69.4±0.6	96.1±0.2	90.6±0.2
SWG	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

Section 4.4 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.

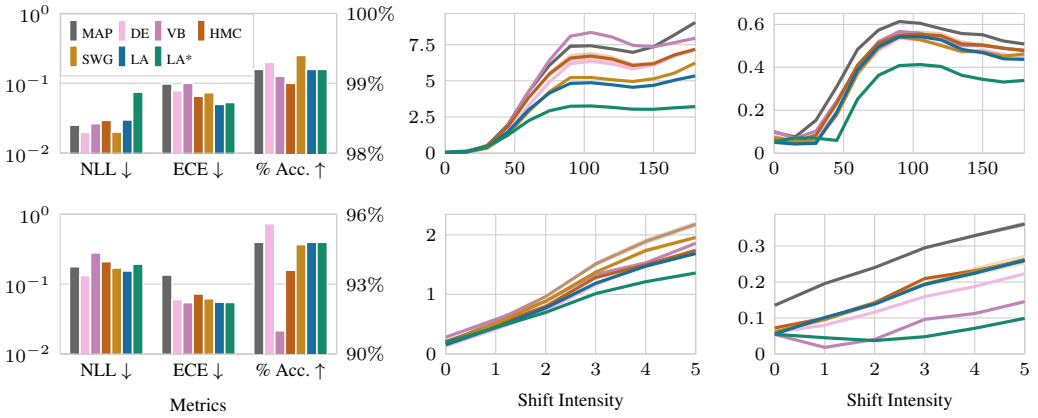
4.1 Choosing the Right Laplace Approximation

In Section 2 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`. 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. We provide details of the comparison and a list of the considered variants in Appendix C.1 and summarize the main arguments and take-aways in the following.

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. [22]). 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. 5) while performing on par or better than state-of-the-art approaches (cf. Fig. 4). 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. [22] (cf Section 4.4). Figure 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.

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



(a) In-Distribution **(b)** Distribution-shift NLL ↓ **(c)** Distribution-shift ECE ↓

Figure 4: Assessing model calibration **(a)** on in-distribution data and **(b&c)** under distribution shift, for the MNIST (top row) and CIFAR-10 (bottom row) datasets. For **(b&c)**, we use the Rotated-MNIST (top) and Corrupted-CIFAR-10 (bottom) benchmarks [64, 65]. In **(a)**, we report accuracy and, to measure calibration, negative log likelihood (NLL) and expected calibration error (ECE), all evaluated on the standard test sets. In **(b)** and **(c)**, we plot shift intensities against NLL and ECE, respectively. For Rotated-MNIST (top), shift intensities denote degrees of rotation of the images, while for Corrupted-CIFAR-10 (bottom), they denote the amount of image distortion (see [64, 65] for details). **(a)** On in-distribution data, LA is the best-calibrated method in terms of ECE, while also retaining the accuracy of MAP (unlike VB and CSGHMC). **(b&c)** On corrupted data, all Bayesian methods improve upon MAP significantly. Even though *post-hoc*, all LAs achieve competitive results, even to DE. In particular, LA* achieves the best results, at the expense of slightly worse in-distribution calibration—this trade-off between in- and out-of-distribution performance has been observed previously [66].

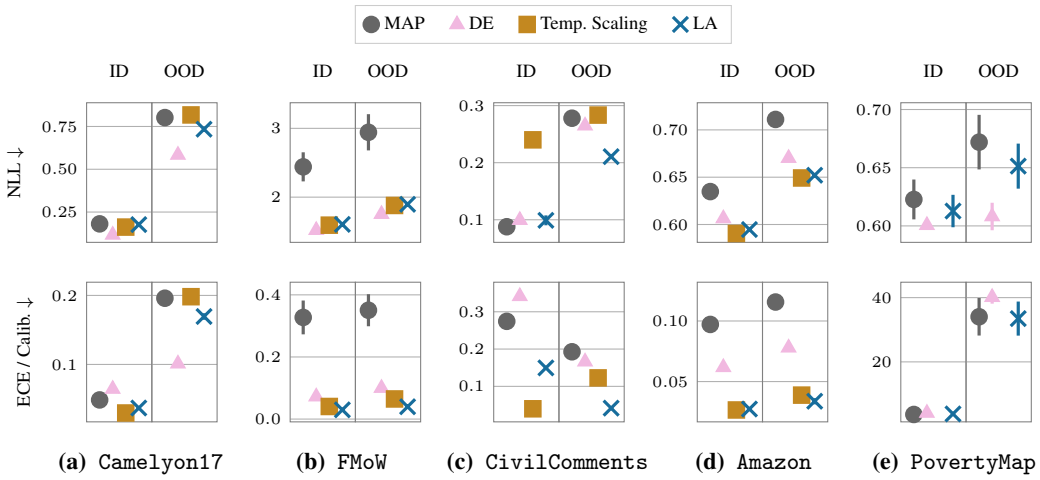
can sometimes be superior for OOD detection but at increased computational cost. The Laplace bridge has the same cost as the probit approximation but typically provides inferior results in our experiments. When using the LA online to optimize hyperparameters, we find that the resulting MAP predictive provides good performance in-distribution, but a probit or MC predictive improve OOD performance.

Overall recommendation. Following the experimental evidence, the default in `laplace` 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.

4.2 Predictive Uncertainty Quantification

We consider two flavors of LAs: the default flavor of `laplace` (**LA**) and the most robust one in terms of distribution shift found in Section 4.1 (**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 [**DE**, 14], mean-field variational Bayes [**VB**, 11, 12] with the flipout estimator [67], cyclical stochastic-gradient Hamiltonian Monte Carlo [**CSGHMC / HMC**, 68], and SWAG [**SWG**, 15]. For each baseline, we use the hyperparameters recommended in the original paper—see Appendix A for details. First, Fig. 4 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 1 shows that **LA** and **LA*** achieve better results on out-of-distribution (OOD) detection than even **VB**, **CSGHMC**, and **SWG**.

The LA shines even more when we consider its (time *and* memory) cost relative to the other, more complex baselines. In Fig. 5 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*, **SWG** is almost twice as expensive as **MAP** during training due to the need of sampling



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

Figure 6: Assessing real-world distribution shift robustness on five datasets from the `WILDS` benchmark [69], covering different data modalities, model architectures, and output types. `Camelyon17`: Tissue slide image tumor classification across hospitals (DenseNet-121 [70]). `FMoW`: Satellite image land use classification across regions/years (DenseNet-121). `CivilCommments`: Online comment toxicity classification across demographics (DistilBERT [71]). `Amazon`: Product review sentiment classification across users (DistilBERT). `PovertyMap`: Satellite image asset wealth regression across countries (ResNet-18 [35]). We plot means ± standard errors of the NLL (top) and ECE (for classification) or regression calibration error [72] (bottom). The in-distribution (left panels) and OOD (right panels) dataset splits correspond to different domains (e.g. hospitals for `Camelyon17`). LA is much better calibrated than MAP, and competitive with temp. scaling and DE, especially on the OOD splits.

and updating its batch normalization statistics. Moreover, with 30 samples, as recommended by its authors [15], 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 considered: it only incurs a negligible overhead on top of the costs of **MAP**. This is similar for the memory consumption (see Table 5 in Appendix C.5). This shows that the LA is significantly more memory- and compute-efficient than all the other methods, adding minimal overhead over MAP inference and prediction. This makes the LA particularly attractive for practitioners, especially in low-resource environments. Together with Fig. 4 and Table 1, this justifies our default flavor in `laplace`, and importantly, shows that Bayesian deep learning does not have to be expensive.

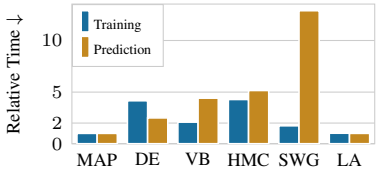


Figure 5: Wall-clock time costs relative to MAP. LA introduces negligible overhead over MAP, while all other baselines are significantly more expensive.

4.3 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 [62], 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` can improve the distribution shift robustness of complex pre-trained models in large-scale settings. To this end, we use `WILDS` [69], 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 [69], 2) *post-hoc* temperature scaling of the MAP models (for classification tasks) [1], and 3) deep ensembles [14].[6]

[6]We simply construct deep ensembles from the various pre-trained models provided by Koh et al. [69].

More details on the experimental setup are provided in Appendix C.3. Fig. 6 shows the results on five different `WILDS` datasets (see caption for details). Overall, Laplace is significantly better calibrated than MAP, and competitive with temperature scaling and ensembles, especially on the OOD splits.

4.4 Further Applications

Beyond predictive uncertainty quantification, the LA is useful in wide range of applications such as Bayesian optimization [37], bandits [73], active learning [34, 74], and continual learning [24]. The `laplace` 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 [75]. Figure 7 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. [24] and improve upon their result by tuning the prior precision through marginal likelihood optimization during training, following Immer et al. [22] (details in Appendix C.4). Using this scheme, the performance after 10 tasks is at around 96% accuracy, outperforming other Bayesian approaches for continual learning [7, 76, 77]. Concretely, we show that the KFAC LA, while much simpler when applied via `laplace`, can achieve better performance to a recent VB baseline [VOGN, 13]. Our library thus provides an easy and quick way of constructing a strong baseline for this application.

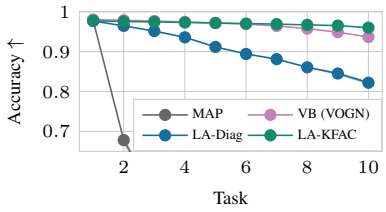


Figure 7: Continual learning results on Permuted-MNIST. MAP fails catastrophically as more tasks are added. The Bayesian approaches substantially outperform MAP, with LA-KFAC performing the best, closely followed by VOGN.

5 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 [11–13] or SWAG [15] also only capture 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 [78, 79, 68, 80, 81, etc.] and deep ensembles [14] can explore several modes. Nevertheless, prior works—also validated in our experiments in Section 4—indicate that using a single mode might not be as limiting in practice as one might think. Wilson and Izmailov [82] 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 [83, 84]. Finally, note that both the LA as well as SWAG can be extended to ensembles of modes in a *post-hoc* manner [85, 82].

6 Conclusion

In this paper, we argued that the Laplace approximation is a simple yet competitive and versatile method for Bayesian deep learning that deserves wider adoption. To this end, we reviewed many recent advances to and variants of the Laplace approximation, including versions with minimal cost overhead that can be applied *post-hoc* to pre-trained off-the-shelf models. In a comprehensive evaluation we demonstrated that the Laplace approximation is on par with other approaches that approximate the intractable network posterior, but at typically much lower computational cost. A particularly simple variant that only treats some weights probabilistically can even be used in the context of pre-trained transformer models to improve predictive uncertainty. As an efficient implementation is not straightforward, we introduced `laplace`, a modular and extensible software library for PyTorch offering user-friendly access to all major flavors of the Laplace approximation. In this way, Laplace approximations provide drop-in Bayesian functionality for most types of deep neural networks.

Acknowledgments and Disclosure of Funding

We thank Kazuki Osawa for providing early access to his automatic second-order differentiation (ASDL) library for PyTorch and Alex Botev for feedback on the manuscript. We also thank the anonymous reviewers for their helpful suggestions for our paper.

E.D. acknowledges funding from the EPSRC and Qualcomm. A.I. gratefully acknowledges funding by the Max Planck ETH Center for Learning Systems (CLS). R.E., A.K. and P.H. 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. A.K. is grateful to the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for support.

References

[1] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On Calibration of Modern Neural Networks. In *ICML*, 2017.

[2] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming Catastrophic Forgetting in Neural Networks. *Proceedings of the National Academy of Sciences*, 114(13), 2017.

[3] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated Cachine Learning: Methods, Systems, Challenges*. Springer Nature, 2019.

[4] David Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[5] Zoubin Ghahramani. Probabilistic Machine Learning and Artificial Intelligence. *Nature*, 521(7553), 2015.

[6] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *ICML*, 2016.

[7] Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. Variational Continual Learning. In *ICLR*, 2018.

[8] David JC MacKay. Probable Networks and Plausible Predictions—a Review of Practical Bayesian Methods for Supervised Neural Networks. *Network: Computation in Neural Systems*, 1995.

[9] Yarin Gal. Uncertainty in deep learning. *University of Cambridge*, 2016.

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

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

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

[13] Kazuki Osawa, Siddharth Swaroop, Mohammad Emtiyaz E Khan, Anirudh Jain, Runa Eschenhagen, Richard E Turner, and Rio Yokota. Practical Deep Learning with Bayesian Principles. In *NeurIPS*, 2019.

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

[15] Wesley J Maddox, Pavel Izmailov, Timur Garipov, Dmitry P Vetrov, and Andrew Gordon Wilson. A Simple Baseline for Bayesian Uncertainty in Deep Learning. In *NeurIPS*, 2019.

[16] David JC MacKay. Bayesian Interpolation. *Neural computation*, 4(3), 1992.

[17] Pierre-Simon Laplace. Mémoires de Mathématique et de Physique, Tome Sixieme. 1774.

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

[19] James Martens and Roger Grosse. Optimizing Neural Networks with Kronecker-Factored Approximate Curvature. In *ICML*, 2015.

[20] Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical Gauss-Newton Optimisation for Deep Learning. In *ICML*, 2017.

[21] Felix Dangel, Frederik Kunstner, and Philipp Hennig. Backpack: Packing More into Backprop. In *ICLR*, 2020.

[22] Alexander Immer, Matthias Bauer, Vincent Fortuin, Gunnar Rätsch, and Mohammad Emtiyaz Khan. Scalable Marginal Likelihood Estimation for Model Selection in Deep Learning. In *ICML*, 2021.

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

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

[25] Mohammad Emtiyaz E Khan, Alexander Immer, Ehsan Abedi, and Maciej Korzepa. Approximate Inference Turns Deep Networks Into Gaussian Processes. In *NeurIPS*, 2019.

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

[27] Erik Daxberger, Eric Nalisnick, James Urquhart Allingham, Javier Antorán, and José Miguel Hernández-Lobato. Bayesian Deep Learning via Subnetwork Inference. In *ICML*, 2021.

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

[29] Jongseok Lee, Matthias Humt, Jianxiang Feng, and Rudolph Triebel. Estimating Model Uncertainty of Neural Networks in Sparse Information Form. In *ICML*, 2020.

[30] Pavel Izmailov, Wesley J Maddox, Polina Kirichenko, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Subspace Inference for Bayesian Deep Learning. In *UAI*, 2019.

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

[32] Christopher KI Williams and David Barber. Bayesian Classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12), 1998.

[33] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes in Machine Learning*. The MIT Press, 2005.

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

[35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016.

[36] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[37] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *ICML*, 2015.

[38] Jonathan Frankle and Michael Carbin. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In *ICLR*, 2019.

[39] James Martens. New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, 21(146):1–76, 2020.

[40] Shun-Ichi Amari. Natural Gradient Works Efficiently in Learning. *Neural computation*, 10(2), 1998.

[41] Frederik Kunstner, Lukas Balles, and Philipp Hennig. Limitations of the Empirical Fisher Approximation for Natural Gradient Descent. In *NeurIPS*, 2019.

[42] Nicol N Schraudolph. Fast Curvature Matrix-Vector Products for Second-Order Gradient Descent. *Neural computation*, 14(7), 2002.

[43] Yann LeCun, John S Denker, and Sara A Solla. Optimal Brain Damage. In *NIPS*, 1990.

[44] John S Denker and Yann LeCun. Transforming Neural-Net Output Levels to Probability Distributions. In *NIPS*, 1990.

[45] Thomas George, César Laurent, Xavier Bouthillier, Nicolas Ballas, and Pascal Vincent. Fast Approximate Natural Gradient Descent in a Kronecker Factored Eigenbasis. In *NIPS*, 2018.

[46] David Madras, James Atwood, and Alex D'Amour. Detecting extrapolation with local ensembles. In *ICLR*, 2020.

[47] Wesley J Maddox, Gregory Benton, and Andrew Gordon Wilson. Rethinking parameter counting in deep models: Effective dimensionality revisited. *arXiv preprint arXiv:2003.02139*, 2020.

[48] Apoorva Sharma, Navid Azizan, and Marco Pavone. Sketching curvature for efficient out-of-distribution detection for deep neural networks. *arXiv preprint arXiv:2102.12567*, 2021.

[49] 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.

[50] Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. Learnable Uncertainty under Laplace Approximations. In *UAI*, 2021.

[51] José M Bernardo and Adrian FM Smith. *Bayesian Theory*. John Wiley & Sons, 2009.

[52] Matthias Humt, Jongseok Lee, and Rudolph Triebel. Bayesian Optimization Meets Laplace Approximation for Robotic Introspection. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) Long-Term Autonomy Workshop*, 2020.

[53] Mark N Gibbs. Bayesian Gaussian Processes for Regression and Classification. *Ph. D. Thesis, Department of Physics, University of Cambridge*, 1997.

[54] David JC MacKay. Choice of Basis for Laplace Approximation. *Machine learning*, 33(1), 1998.

[55] Philipp Hennig, David Stern, Ralf Herbrich, and Thore Graepel. Kernel Topic Models. In *AISTATS*, 2012.

[56] Marius Hobbhahn, Agustinus Kristiadi, and Philipp Hennig. Fast Predictive Uncertainty for Classification with Bayesian Deep Networks. *arXiv preprint arXiv:2003.01227*, 2020.

[57] Wesley J Maddox, Pavel Izmailov, Timur Garipov, Dmitry P Vetrov, and Andrew Gordon Wilson. Code repo for "A Simple Baseline for Bayesian Deep Learning". `https://github.com/wjmaddox/swa_gaussian`, 2019.

[58] Agustinus Kristiadi. Last-layer Laplace approximation code examples. `https://github.com/wiseodd/last_layer_laplace`, 2020.

[59] Jongseok Lee and Matthias Humt. Official Code: Estimating Model Uncertainty of Neural Networks in Sparse Information Form, ICML2020. `https://github.com/DLR-RM/curvature`, 2020.

[60] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS*, 2019.

[61] Kazuki Osawa. ASDL: Automatic second-order differentiation (for fisher, gradient covariance, hessian, jacobian, and kernel) library. `https://github.com/kazukiosawa/asdfghjkl`, 2021.

[62] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All You Need. In *NIPS*, 2017.

[63] Dan Hendrycks and Kevin Gimpel. A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks. In *ICLR*, 2017.

[64] Dan Hendrycks and Thomas Dietterich. Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. In *ICLR*, 2019.

[65] 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.

[66] Zhiyun Lu, Eugene Ie, and Fei Sha. Uncertainty Estimation with Infinitesimal Jackknife, Its Distribution and Mean-Field Approximation. *arXiv preprint arXiv:2006.07584*, 2020.

[67] Yeming Wen, Paul Vicol, Jimmy Ba, Dustin Tran, and Roger Grosse. Flipout: Efficient Pseudo-Independent Weight Perturbations on Mini-Batches. In *ICLR*, 2018.

[68] Ruqi Zhang, Chunyuan Li, Jianyi Zhang, Changyou Chen, and Andrew Gordon Wilson. Cyclical Stochastic Gradient MCMC for Bayesian Deep Learning. In *ICLR*, 2020.

[69] Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, et al. WILDS: A Benchmark of In-The-Wild Distribution Shifts. In *arXiv preprint arXiv:2012.07421*, 2020.

[70] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely Connected Convolutional Networks. In *CVPR*, 2017.

[71] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a Distilled Version of Bert: Smaller, Faster, Cheaper and Lighter. In *5th Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS*, 2019.

[72] Volodymyr Kuleshov, Nathan Fenner, and Stefano Ermon. Accurate Uncertainties for Deep Learning Using Calibrated Regression. In *ICML*, 2018.

[73] Olivier Chapelle and Lihong Li. An Empirical Evaluation of Thompson Sampling. In *NIPS*, 2011.

[74] Mijung Park, Greg Horwitz, and Jonathan W Pillow. Active Learning of Neural Response Functions with Gaussian Processes. In *NIPS*, 2011.

[75] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks. *arXiv preprint arXiv:1312.6211*, 2013.

[76] Michalis K Titsias, Jonathan Schwarz, Alexander G de G Matthews, Razvan Pascanu, and Yee Whye Teh. Functional Regularisation for Continual Learning with Gaussian Processes. In *ICLR*, 2020.

[77] Pingbo Pan, Siddharth Swaroop, Alexander Immer, Runa Eschenhagen, Richard E Turner, and Mohammad Emtiyaz Khan. Continual Deep Learning by Functional Regularisation of Memorable Past. In *NeurIPS*, 2020.

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

[79] Florian Wenzel, Kevin Roth, Bastiaan S Veeling, Jakub Świątkowski, Linh Tran, Stephan Mandt, Jasper Snoek, Tim Salimans, Rodolphe Jenatton, and Sebastian Nowozin. How Good is the Bayes Posterior in Deep Neural Networks Really? *ICML*, 2020.

[80] Pavel Izmailov, Sharad Vikram, Matthew D Hoffman, and Andrew Gordon Wilson. What Are Bayesian Neural Network Posteriors Really Like? In *ICML*, 2021.

[81] Adrià Garriga-Alonso and Vincent Fortuin. Exact langevin dynamics with stochastic gradients. *arXiv preprint arXiv:2102.01691*, 2021.

[82] Andrew G Wilson and Pavel Izmailov. Bayesian Deep Learning and a Probabilistic Perspective of Generalization. In *NeurIPS*, 2020.

[83] Anoop Korattikara, Vivek Rathod, Kevin Murphy, and Max Welling. Bayesian Dark Knowledge. In *NIPS*, 2015.

[84] Kuan-Chieh Wang, Paul Vicol, James Lucas, Li Gu, Roger Grosse, and Richard Zemel. Adversarial Distillation of Bayesian Neural Network Posteriors. In *ICML*, 2018.

[85] Runa Eschenhagen, Erik Daxberger, Philipp Hennig, and Agustinus Kristiadi. Mixtures of Laplace Approximations for Improved *Post-Hoc* Uncertainty in Deep Learning. *NeurIPS Workshop on Bayesian Deep Learning*, 2021.

[86] David JC MacKay. A Practical Bayesian Framework For Backpropagation Networks. *Neural computation*, 1992.

[87] Sebastian Farquhar, Lewis Smith, and Yarin Gal. Liberty or Depth: Deep Bayesian Neural Nets Do Not Need Complex Weight Posterior Approximations. In *NeurIPS*, 2020.

[88] Arjun K Gupta and Daya K Nagar. *Matrix Variate Distributions*. Chapman and Hall, 1999.

[89] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1 (3):211–218, 1936.

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

[91] Anqi Wu, Sebastian Nowozin, Edward Meeds, Richard E. Turner, Jose Miguel Hernandez-Lobato, and Alexander L. Gaunt. Deterministic Variational Inference for Robust Bayesian Neural Networks. In *ICLR*, 2019.

[92] Amr Ahmed and Eric P Xing. Seeking The Truly Correlated Topic Posterior—On Tight Approximate Inference of Logistic-Normal Admixture Model. In *AISTATS*, 2007.

[93] Michael Braun and Jon McAuliffe. Variational Inference for Large-Scale Models of Discrete Choice. *Journal of the American Statistical Association*, 105(489), 2010.

[94] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

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

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

[97] Ranganath Krishnan and Piero Esposito. Bayesian-Torch: Bayesian Neural Network Layers for Uncertainty Estimation. `https://github.com/IntelLabs/bayesian-torch`, 2020.

[98] Florian Wenzel, Jasper Snoek, Dustin Tran, and Rodolphe Jenatton. Hyperparameter Ensembles for Robustness and Uncertainty Quantification. In *NeurIPS*, 2020.

[99] Ferenc Huszár. Note on the quadratic penalties in elastic weight consolidation. *Proceedings of the National Academy of Sciences*, page 201717042, 2018.

Appendix A Derivation

A.1 The Derivation of the Laplace Approximation

Let $p(\theta \mid \mathcal{D})$ be an intractable posterior, written as

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

Our goal is to approximate this distribution with a Gaussian arising from the Laplace approximation. The key observation is that we can rewrite the normalizing constant Z as the integral $\int \exp(\log h(\theta))\, d\theta$. Let $\theta_{\text{MAP}} := \arg\max_\theta \log p(\theta \mid \mathcal{D}) = \arg\max_\theta \log h(\theta)$ be a (local) maximum of the posterior—the so-called ***maximum a posteriori (MAP)*** estimate. Taylor-expanding $\log h$ around θ_{MAP} up to the second order yields

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

where $\Lambda := -\nabla^2 \log h(\theta)|_{\theta_{\text{MAP}}}$ is the negative Hessian matrix of the log-joint in (1), evaluated at θ_{MAP}. Similar to its original formulation, here we again obtain a (multivariate) Gaussian integral, the analytic solution of which is readily available:

$$\begin{aligned} Z &\approx \exp(\log h(\theta_{\text{MAP}})) \int \exp\left(-\frac{1}{2}(\theta - \theta_{\text{MAP}})^\top \Lambda\, (\theta - \theta_{\text{MAP}})\right) d\theta \\ &= h(\theta_{\text{MAP}}) \frac{(2\pi)^{\frac{d}{2}}}{(\det \Lambda)^{\frac{1}{2}}}. \end{aligned} \tag{3}$$

Plugging the approximations (2) and (3) back into the expression of $p(\theta \mid \mathcal{D})$, we obtain

$$p(\theta \mid \mathcal{D}) = \frac{1}{Z} h(\theta) \approx \frac{(\det \Lambda)^{\frac{1}{2}}}{(2\pi)^{\frac{d}{2}}} \exp\left(-\frac{1}{2}(\theta - \theta_{\text{MAP}})^\top \Lambda\, (\theta - \theta_{\text{MAP}})\right), \tag{4}$$

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

Appendix B Details on the Four Components

① Inference over Subsets of Weights

B.1.1 Subnetwork

Storing the full $D \times D$ covariance matrix Σ of the weight posterior in Eq. (4) is computationally intractable for a modern neural networks. One approach to reduce this computational burden is to perform inference over only a small *subset* of the model parameters θ [27]. This is motivated by recent findings that neural nets can be heavily pruned without sacrificing test accuracy [38], and that in the neighborhood of a local optimum, there are many directions that leave the predictions unchanged [47].

This *subnetwork inference* approach uses the following approximation to the posterior in Eq. (4):

$$p(\theta \mid \mathcal{D}) \;\approx\; p(\theta_S \mid \mathcal{D}) \prod_r \delta(\theta_r - \widehat{\theta}_r) \;=\; q_S(\theta)\,, \tag{5}$$

where $\delta(x - a)$ denotes the Dirac delta function centered at a. The approximation $q_S(\theta)$ in Eq. (5) simply decomposes the full neural network posterior $p(\theta \mid \mathcal{D})$ into a Laplace posterior $p(\theta_S \mid \mathcal{D})$ over the subnetwork $\theta_S \in \mathbb{R}^S$, and fixed, deterministic values $\widehat{\theta}_r$ to the $D - S$ remaining weights θ_r. In practice, the remaining weights θ_r are simply set to their MAP estimates, i.e. $\widehat{\theta}_r = \theta_r^{\text{MAP}}$, requiring no additional computation. Importantly, note that the subnetwork size S is in practice a hyperparameter that can be controlled by the user. Typically, S will be set such that the subnetwork is much smaller than the full network, i.e. $S \ll D$. In particular, S can be set such that it is tractable to compute and store the full $S \times S$ 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/Fisher described in Section B.1.2.

Daxberger et al. [27] propose to choose the subnetwork such that the subnetwork posterior $q_S(\theta)$ in Eq. (5) is as close as possible (w.r.t. some discrepancy measure) to the full posterior $p(\theta \mid \mathcal{D})$ in Eq. (4). As the subnetwork posterior is degenerate due to the involved Dirac delta functions, common discrepancy measures such as the KL divergence are not well defined. Therefore, Daxberger et al. [27] propose to use the squared 2-Wasserstein distance, which in this case takes the following form:

$$W_2(p(\theta \mid \mathcal{D}), q_S(\theta))^2 = \text{Tr}\left(\Sigma + \Sigma_S - 2\left(\Sigma_S^{1/2}\,\Sigma\,\Sigma_S^{1/2}\right)^{1/2}\right), \tag{6}$$

where the (degenerate) subnetwork covariance matrix Σ_S is equal to the full covariance matrix Σ but with zeros at the positions corresponding to the weights θ_r (i.e. those *not* part of the subnetwork).

Unfortunately, finding the subset of weights $\theta_S \in \mathbb{R}^S$ of size S that minimizes Eq. (6) is combinatorially hard, as the contribution of each weight depends on every other weight. Daxberger et al. [27] therefore assume that the weights are independent, resulting in the following simplified objective:

$$W_2(p(\theta \mid \mathcal{D}), q_S(\theta))^2 \approx \sum_{d=1}^{D} \sigma_d^2(1 - m_d)\,, \tag{7}$$

where $\sigma_d^2 = \Sigma_{dd}$ is the marginal variance of the d^{th} weight, and $m_d = 1$ if $\theta_d \in \theta_S$ (with slight abuse of notation) or 0 otherwise is a binary mask indicating which weights are part of the subnetwork (see Daxberger et al. [27] for details). The objective in Eq. (7) is trivially minimized by choosing a subnetwork containing the S weights with the highest σ_d^2 values (i.e. with largest marginal variances).

In practice, even computing the marginal variances (i.e. the diagonal of Σ) is intractable, as it requires storing and inverting the Hessian/Fisher Λ. To approximate the marginal variances, one could use a diagonal Laplace approximation [44, 2] that assumes $\text{diag}(\Sigma) \approx \text{diag}(\Lambda)^{-1}$. Alternatively, one could use diagonal SWAG [15]. For more details on subnetwork inference, refer to Daxberger et al. [27].

B.1.2 Last-Layer

The last-layer Laplace [37, 28] is a special variant of the subnetwork Laplace where θ_S in (5) is assumed to equal the last-layer weight matrix $W^{(L)}$ of the network. That is, we let $f_\theta : \mathbb{R}^M \to \mathbb{R}^C$ is an L-layer NN, and assume that the first $L-1$ layers of f_θ is a feature map. Given MAP-trained parameters θ_{MAP}, we define a Laplace-approximated posterior over $W^{(L)}$

$$p(W^{(L)} \mid \mathcal{D}) \approx \mathcal{N}(W^{(L)} \mid W^{(L)}_{\text{MAP}}, \Sigma^{(L)}), \tag{8}$$

and we leave the rest of the parameters with their MAP-estimated values. Since this matrix is small relative to the entire network, the last-layer Laplace can be implemented efficiently.

② Hessian Factorization

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.

$$s(x, y) := \nabla_\theta p(y \mid f_\theta(x))|_{\theta_{\text{MAP}}}.$$

Using this notation, we can write the Fisher compactly by

$$F := \textstyle\sum_{n=1}^{N} \mathbb{E}_{p(y \mid f_\theta(x_n))}\left(s(x_n, y)s(x_n, y)^\intercal\right), \tag{9}$$

We shall refer to this matrix as the *full Fisher*. Recall that F is as large as the exact Hessian of the network, so its computation is often infeasible. Thus, here, we review several factorization schemes that makes the computation (and storage) of the Fisher efficient, starting from the simplest.

Diagonal Although MacKay recommended to not use the diagonal factorization of the Hessian [86], a recent work has indicated this factorization is usable for sufficiently deep NNs [87]. In this factorization, we simply assume that the negative-log-posterior's 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 $\text{diag}(F) = \sum_{n=1}^{N} \mathbb{E}_{p\left(y \mid f_{\theta_{\text{MAP}}}(x_n)\right)}(s(x_n, y) \odot s(x_n, y))$,[7] 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 $O(D^3)$.

[7] The operator $\odot$ denotes the Hadamard product.

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 $l = 1, \dots, L$, denoting N_l as the number of hidden units at the l-th layer, let $W^{(l)} \in \mathbb{R}^{N_l \times N_{l-1}}$ be the weight matrix of the l-th layer of the network, $a^{(l)}$ the l-th hidden vector, and $g^{(l)} \in \mathbb{R}^{N_l}$ the log-likelihood gradient w.r.t. $a^{(l)}$. For each $l = 1, \dots, L$, we can then write the outer product inside expectation in (8) as $s(x_i, y)s(x_i, y)^\top = a^{(l-1)}a^{(l)\top} \otimes g^{(l)}g^{(l)\top}$. Furthermore, assuming that $a^{(l-1)}$ is independent of $g^{(l)}$, we obtain the approximation of the l-th diagonal block of F, which we denote by $F^{(l)}$:

$$F^{(l)} \approx \mathbb{E}\left(a^{(l-1)}a^{(l-1)\top}\right) \otimes \mathbb{E}\left(g^{(l)}g^{(l)\top}\right) =: A^{(l-1)} \otimes G^{(l)}, \tag{10}$$

where we represent both the sum and the expectation in (9) as $\mathbb{E}$ for brevity.

From the previous expression we can see that the space complexity for storing $F^{(l)}$ is reduced to $O(N_l^2 + N_{l-1}^2)$, down from $O(N_l^2 N_{l-1}^2)$. Considering all L layers of the network, we obtain the layer-wise Kronecker factors $\{A^{(l)}\}_{l=0}^{L-1}$ and $\{G^{(l)}\}_{l=1}^{L}$ 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 $\Lambda^{(l)}$—by

$$\begin{aligned}\Lambda^{(l)} &\approx \left(A^{(l-1)} + \sqrt{\lambda}I\right) \otimes \left(G^{(l)} + \sqrt{\lambda}I\right) \\ &=: V^{(l)} \otimes U^{(l)}.\end{aligned}$$

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 [19, 26]. This particular Laplace approximation has been studied by Ritter et al. [23, 24] and can be seen as approximating the posterior of each $W^{(l)}$ with the matrix-variate Gaussian distribution [88]: $p(W^{(l)} \mid \mathcal{D}) \approx \mathcal{MN}(W^{(l)} \mid W^{(l)}_{\text{MAP}}, U^{(l)-1}, V^{(l)-1})$. Hence, sampling can be done easily in a layer-wise manner:

$$W^{(l)} \sim p\left(W^{(l)} \mid \mathcal{D}\right) \iff W^{(l)} = W^{(l)}_{\text{MAP}} + U^{(l)-\frac{1}{2}} E V^{(l)-\frac{1}{2}}$$

where

$$E \sim \mathcal{MN}(0, I_{N_l}, I_{N_{l-1}}),$$

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 $\mathcal{MN}(0, I_{N_l}, I_{N_{l-1}})$ is equivalent to the standard $(N_l N_{l-1})$-variate Normal distribution. As an alternative, Immer et al. [26] suggest to incorporate the prior exactly using an eigendecomposition of the individual Kronecker factors, which can improve performance.

Low-rank block-diagonal We can improve KFAC's efficiency by considering its low-rank factorization [29]. The key idea is to eigendecompose the Kronecker factors in (10) and keep only the eigenvectors corresponding to the first k largest eigenvalues. This can be done employing the eigenvalue-corrected KFAC [45]. That is, for each layer $l = 1, \dots, L$:

$$\begin{aligned}F^{(l)} &\approx \left(U_A^{(l-1)} S_A^{(l-1)} U_A^{(l-1)\top}\right) \otimes \left(U_G^{(l)} S_G^{l} U_G^{(l)\top}\right) \\ &= \left(U_A^{(l-1)} \otimes U_G^{(l)}\right)\left(S_A^{(l-1)} \otimes S_G^{(l)}\right)\left(U_A^{(l-1)} \otimes U_G^{(l)}\right)^\top.\end{aligned}$$

Under this decomposition, one can the easily obtain the optimal rank-k approximation of $F^{(l)}$, denoted by $F_k^{(l)}$, by selecting the top-k eigenvalues. However, the diagonal of this rank-k matrix can deviate too far from the exact diagonal elements of $F^{(l)}$. Hence, one can make the diagonal of this low rank matrix exact replacing $\operatorname{diag}(F_k^l)$ with $\operatorname{diag}(F^{(l)})$, and obtain the following rank-k-plus-diagonal approximation of $F^{(l)}$:

$$F^{(l)} \approx F_k^{(l)} + \operatorname{diag}(F^{(l)}) - \operatorname{diag}(F_k^{(l)}).$$

This factorization can be seen as a combination of the previous two approximations: For each diagonal block of F, we use the exact diagonal elements of F and approximate the off-diagonal elements with a rank-k matrix arising from KFAC. Both the space and computational complexities are lower than those of KFAC since here we work exclusively with truncated and diagonal matrices.

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 [48, 47]. 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 $L = \operatorname{diag}(l)$ 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 [89]: 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

$$F \approx \widehat{Q} \operatorname{diag}(\widehat{l})\, \widehat{Q}^\top.$$

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.

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. [23] 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:

$$\gamma_*^2 = \arg\max_{\gamma^2} \sum_{n=1}^{N_{\text{val}}} \log p(y_n \mid x_n, \mathcal{D}). \tag{11}$$

However, Kristiadi et al. [28] 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 $\mathcal{D}_{\text{out}} := (x_n^{(\text{out})})_{n=1}^{N_{\text{out}}}$ to (11), as follows

$$\gamma_*^2 = \arg\max_{\gamma^2} \sum_{n=1}^{N_{\text{val}}} \log p(y_n \mid x_n, \mathcal{D}) + \lambda \sum_{n=1}^{N_{\text{out}}} H\left[p(y_n \mid x_n^{(\text{out})}, \mathcal{D})\right], \tag{12}$$

where 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. [22] (further details below in *Online*) one can also maximize the Laplace-approximated marginal likelihood (3) 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 [22]. 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. (3). 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 [34] 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 [22]. We refer the reader to Algorithm 1 for an overview.

Algorithm 1 Online Laplace (adapted from Immer et al. [22, Algorithm 1])

Input:
NN f_θ; training set $\mathcal{D}$; learning rate α_0 and number of epochs T_0 for MAP estimation; learning rate α_1 and number of epochs T_1 for hyperparameter tuning; marginal likelihood maximization frequency F.

Initialize θ_0
for $t = 1, \dots, T_0$ **do**
  $g_t \leftarrow \nabla_\theta \mathcal{L}(\mathcal{D}; \theta)|_{\theta_{t-1}}$
  $\theta_t \leftarrow \theta_{t-1} - \alpha_0 \, g_t$
  if $t \mod F = 0$ **then**
    $p(\theta \mid \mathcal{D}) \approx \mathcal{N}(\theta \mid \theta_t, (\nabla^2 \mathcal{L}(\mathcal{D}; \theta)|_{\theta_t})^{-1})$ ▷ Perform a Laplace approximation
    for $\widetilde{t} = 1, \dots, T_1$ **do** ▷ Hyperparameter optimization
      $h_{\widetilde{t}} \leftarrow \nabla_{\gamma^2} \log p(\mathcal{D} \mid \gamma^2)|_{\gamma^2_{\widetilde{t}-1}}$ ▷ The marginal likelihood follows from (3)
      $\gamma^2_{\widetilde{t}} \leftarrow \gamma^2_{\widetilde{t}-1} + \alpha_1 \, h_{\widetilde{t}}$
    end for
  end if
end for
return θ_{T_0}; $\nabla^2 \mathcal{L}(\mathcal{D}; \theta)|_{\theta_{T_0}}$

④ Approximate Predictive Distribution

Here, we denote $x_* \in \mathbb{R}^N$ to be a test point, and f_* be the network output at this point. We will review different way to approximate the predictive distribution $p(y \mid x_*, \mathcal{D})$ given a Gaussian approximate posterior, starting from the most general.

B.4.3 General

Monte Carlo Integration The simplest but general and unbiased approximation is the Monte Carlo (MC) integration, which can be performed by sampling an approximate posterior $q(\theta \mid \mathcal{D})$ repeatedly:

$$p(y \mid x_*, \mathcal{D}) \approx \frac{1}{S} \sum_{s=1}^{S} p(y \mid f_{\theta_s}(x_*)), \qquad \text{where } \theta_s \sim q(\theta \mid \mathcal{D}).$$

While the error of this approximation decays like $1/\sqrt{S}$ and thus requires many samples to be accurate, for practical BNNs, it is standard to use 10 or 20 samples of $q(\theta \mid \mathcal{D})$ [23, 28, 12, etc.]. Note that this approximation can be used regardless the form of the likelihood $p(y \mid f_\theta(x))$, in particular it can be used to directly obtain the predictive distribution in both the regression and classification alike.

B.4.4 Distribution of Network Outputs

Here, we are concerned in approximating the marginal distribution of $f(x_*)$, where θ has been integrated out.

Linearization In this approximation, we linearize the network to obtain

$$f_\theta(x_*) \approx f_{\theta_{\text{MAP}}}(x_*) + J_*^\top (\theta - \theta_{\text{MAP}}),$$

where $J_* := \nabla_\theta f_\theta(x_*)|_{\theta_{\text{MAP}}} \in \mathbb{R}^{d \times c}$ is the Jacobian matrix of the network output. This way, under a Gaussian approximate posterior $q(\theta \mid \mathcal{D})$, the marginal distribution over the network output $f_* := f(x_*)$ is again a Gaussian, given by[8]

$$\begin{aligned} p(f_* \mid f_\theta(x_*), x_*, \mathcal{D}) &= \int \delta(f_* - f_\theta(x_*)) \, q(\theta \mid \mathcal{D}) \, d\theta \\ &\approx \mathcal{N}(f_* \mid f_{\theta_{\text{MAP}}}(x_*), J_*^\top \Sigma J_*) \end{aligned}$$

This approximation has been extensively used for small networks [34], but it has since gone out of favor in deep learning due to its cost—the Jacobian J_* needs to be computed *per input point*.

[8] See Bishop [90, Sec. 4.5.2].

Nevertheless, this approximation is still useful in theoretical works due to its analytical nature [28, 50, 85] . Moreover, in problems where it can be efficiently use in practice, it offers a better approximation than MC-integral [26, 49].

B.4.5 Regression

Assume that we already have a Gaussian approximation to $p(f_* \mid x_*, \mathcal{D}) \approx \mathcal{N}(f_* \mid \mu_*, \Sigma_*)$ via the linearization above. In regression, we still need to incorporate the observation noise β encoded in the (usually) Gaussian likelihood $\mathcal{N}(y_* \mid f_*, \beta I)$[9] to make prediction. This can be easily done in an exact manner:

$$\begin{aligned} p(y_* \mid x_*) &= \int_{\mathbb{R}^C} \mathcal{N}(y_* \mid f_*, \beta I)\, \mathcal{N}(f_* \mid \mu_*, \Sigma_*)\, df_* \\ &= \mathcal{N}(y_* \mid \mu_*, \Sigma_* + \beta I), \end{aligned}$$

since the integral above is just a convolution of two Gaussian r.v.s.

B.4.6 Classification and Generalized Regression

Since unlike the regression case, the classification likelihood $p(y_* \mid f_*)$ is non-Gaussian, we cannot analytically obtain $p(y_* \mid x_*)$ given a Gaussian approximation $p(f_* \mid x_*, \mathcal{D}) \approx \mathcal{N}(f_* \mid \mu_*, \Sigma_*)$. So, in this case we are interested in approximating the intractable integral

$$p(y_* \mid x_*) = \int p(y_* \mid f_*)\, \mathcal{N}(f_* \mid \mu_*, \Sigma_*)\, df_*,$$

where $p(y_* \mid f_*)$ is constructed via an inverse-link function. Here we will review the usual case of classification, i.e. when $p(y_* \mid f_*) = \sigma(f_*)$ where σ is the logistic-sigmoid function, or $p(y_* \mid f_*) = \mathrm{softmax}(f_*)$.

Delta Method The crux of the delta method [91–93] is a Taylor-expansion of the softmax function around μ_* up to the second order. Then, since $p(f_* \mid x_*, \mathcal{D})$ is assumed to be Gaussian, the integral $\mathbb{E}_{p(f_* \mid x_*, \mathcal{D})}(\mathrm{softmax}(f_*))$ can be computed easily, resulting in an analytic expression $\mathrm{softmax}(\mu_*) + 1/2\,\mathrm{tr}(B\Sigma_*)$, where B is the Hessian matrix of the softmax at μ_*.

Probit Approximations The essence of the (binary) probit approximation [31, 34] is to approximate σ with the probit function Φ—the standard Normal c.d.f.—which makes the integral solvable analytically. Using this approximation, one can then obtain the closed-form approximation

$$\begin{aligned} p(y_* \mid x_*) &\approx \int_{\mathbb{R}} \Phi(f_*)\, \mathcal{N}(f_* \mid \mu_*, \sigma_*^2)\, df_* \\ &= \sigma\left(\frac{\mu_*}{\sqrt{1 + \frac{\pi}{8}\,\sigma_*^2}}\right). \end{aligned}$$

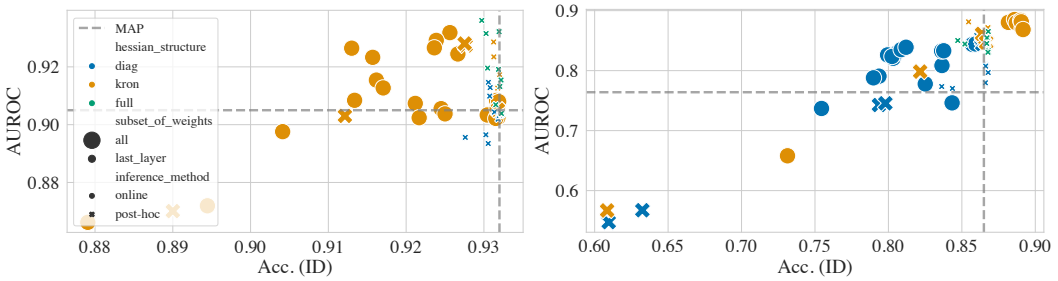
It has a generalization to multi-class classification, due to Gibbs [53], i.e. for approximating

$$p(y_* \mid x_*) = \int_{\mathbb{R}^C} \mathrm{softmax}(f_*)\, \mathcal{N}(f_* \mid \mu_*, \Sigma_*)\, df_*. \tag{13}$$

In this case, we approximate the resulting probability vector of length C with a vector which i-th component is given by $\exp(\tau_i)/\sum_{j=1}^{C} \exp(\tau_j)$, where $\tau_j = \mu_{*j}/\sqrt{1 + \pi/8\,\Sigma_{*jj}}$ for each $j = 1, \ldots, C$. This approximation ignores the correlation between logits since it only depends on the diagonal of Σ_*. Nevertheless, it yields good results even in deep learning [66], and are invaluable tools for theoretical work [85].

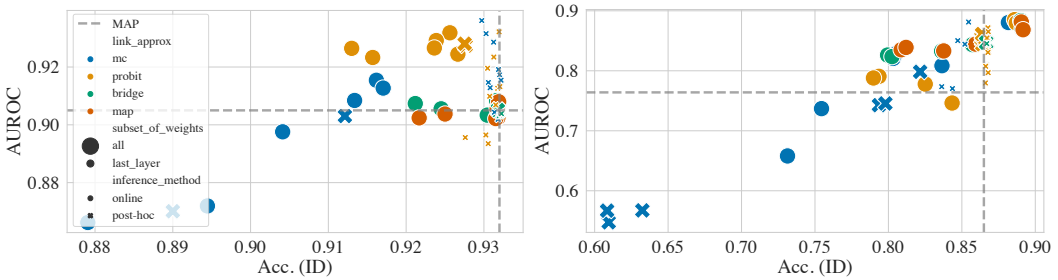
Laplace Bridge The main idea of the Laplace bridge is to perform a Laplace approximation to the Dirichlet distribution by first writing it as a distribution over $\mathbb{R}^C$ with the help of the softmax function [54, 55]. This way, Laplace approximation can be reasonably applied to approximate the Dirichlet, which can be thought as mapping the Dirichlet $\mathrm{Dir}(\alpha_*)$ to a Gaussian $\mathcal{N}(\mu_*, \Sigma_*)$. The

[9] We assume a multivariate output $y_* \in \mathbb{R}^C$ for full generality.



(a) Hessian structure (CIFAR-10 + DA)

(b) Hessian structure (CIFAR-10)



(c) Predictive approximation (CIFAR-10 + DA)

(d) Predictive approximation (CIFAR-10)

Figure 8: Comparison of variations of the LA on the CIFAR-10 OOD experiment with ((a) and (c)) and without ((b) and (d)) data augmentation (DA).

pseudo-inverse of this map, mapping (μ_*, Σ_*) to α_* where for each $i = 1, \ldots, C$, the i-th component α is given by the simple closed-form expression

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

is the *Laplace bridge*. Just like the probit approximation, the Laplace bridge ignores the correlation between logits. But, unlike all the previous approximations, it yields a *full distribution* over the solutions of the softmax-Gaussian integral (13). So, the Laplace bridge is a richer yet comparably simple approximation to the integral and is useful for many applications in deep BNNs [56].

Appendix C Further Experiments Details and Results

C.1 Laplace Comparison

Here, we present more detailed results of our comparison of the different variations of the Laplace approximation. We show in-distribution accuracy for CIFAR-10 using a model trained with and without data augmentation, and AUROC values averaged over the out-of-distribution datasets SVHN, LSUN, and CIFAR-100. In the first row of Figure 8, we highlight the different Hessian structures with different colors; in the second row, we use color to highlight the different link approximations in the predictive distribution. We considered most combinations of the different choices for the components discussed in Section 2, but exclude some combinations which we have found to not work well at all, e.g. online Laplace when performing a Laplace approximation over the weights of only the last layer. In Table 2, we compare the predictive performance and runtime when using differently structured Hessian approximations. We find that the Kronecker-factored Hessian approximations provides a good trade-off between runtime and performance.

	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

Table 2: 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 [22] 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.

C.2 Predictive Uncertainty Quantification

C.2.1 Training Details

We use LeNet [94] and WideResNet-16-4 [WRN, 95] 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 [96] 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 [97] to train the network. Tha variational posterior is chosen to be the diagonal Gaussian [11, 12] and the flipout estimator [67] 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 [13].

CSGHMC We use the publicly available code provided by the original authors [68].[10] We use their default (i.e. recommended) hyperparameters.

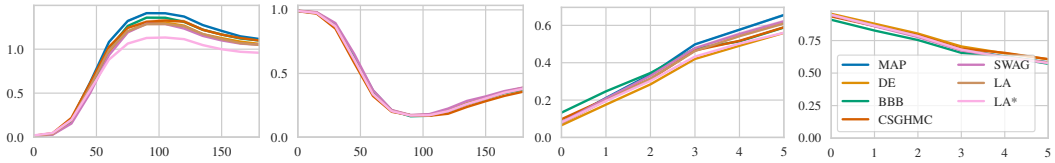
SWAG For the SWAG baseline, we follow Maddox et al. [15] and run stochastic gradient descent with a constant learning rate on the pre-trained models to collect one model snapshot per epoch, for a total of 40 snapshots. At test time, we then make predictions by using 30 Monte Carlo samples from the posterior distribution; we correct the batch normalization statistics of each sample as described in Maddox et al. [15]. To tune the constant learning rate, we used the same approach as in Eschenhagen et al. [85], 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** [6] has been shown to underperform compared to more recent methods (see e.g. Ovadia et al. [65]). A recent VI method called **Variational Online Gauss-Newton (VOGN)** [13] also seems to underperform. For example, Fig. 5 of Osawa et al. [13] 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).

C.2.2 Detailed Results

We show the Brier score and accuracy as a function of shift intensity in Fig. 9. Moreover, we provide the detailed (i.e. non-averaged) OOD detection results in Tables 3 and 4.

[10] https://github.com/ruqizhang/csgmcmc



(a) MNIST-R Brier ↓ **(b)** MNIST-R Acc. ↑ **(c)** CIFAR10-C Brier ↓ **(d)** CIFAR10-C Acc. ↑

Figure 9: Dataset shift on the Rotated-MNIST (top) and Corrupted-CIFAR-10 datasets (bottom).

Table 3: 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

C.2.3 Additional Details on Wall-clock Time Comparison

Concerning the wall-clock time comparison in Fig. 5, we would like to clarify that for LA, we consider the default configuration of `laplace`. 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. [26] 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.

C.3 `WILDS` Experiments

For this set of experiments, we use `WILDS` [69], 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).

Table 4: 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

- `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. 6 and Fig. 10 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[11] This yields ensembles of 5 neural network models, which is a commly-used ensemble size [65]. Since these models were trained in different ways (e.g. using different domain generalization methods, see [69] for details), their combinations can be viewed as *hyperparameter ensembles* [98].

Note that the temperature scaling baseline is only applicable for classification tasks, and therefore we do not report it for the `PovertyMap` regression dataset.

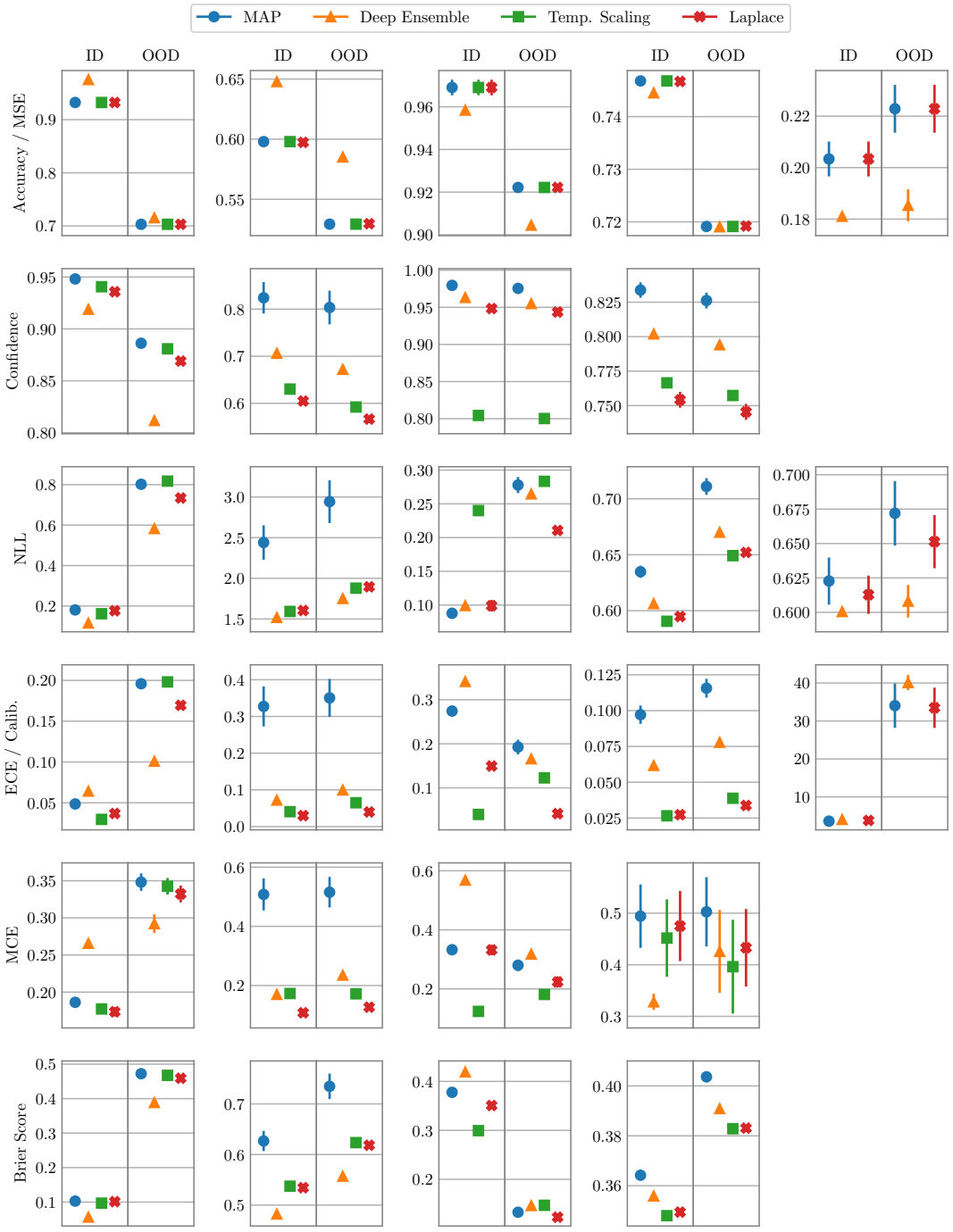
We tune the temperature parameter for temperature scaling, the prior precision parameter for Laplace, and the noise standard deviation parameter for regression (i.e. for the `PovertyMap` dataset) by minimizing the negative log-likelihood on the in-distribution validation sets provided with WILDS.

Finally, Fig. 10 shows an extended version of the results reported in Fig. 6, which additionally reports the following metrics: accuracy (for classification) or mean squared error (for regression), confidence (only for classification), mean calibration error (only for classification), and Brier score (only for classification). The overall conclusion here is the same as for Fig. 6, namely that Laplace is significantly better calibrated than MAP, and competitive with temperature scaling and ensembles, especially on the OOD splits. Note that the differences in accuracies of the ensemble stem from the different training procedures of the ensemble members (which sometimes achieve higher and sometimes lower accuracy), as mentioned above.

C.4 Further Details on the Continual Learning Experiment

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 [24, 7, 13], 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.

[11] See `https://worksheets.codalab.org/worksheets/0x52cea64d1d3f4fa89de326b4e31aa50a` for the complete list of models.



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

Figure 10: Assessing real-world distribution shift robustness on five datasets from the `WILDS` benchmark [69], covering different data modalities, model architectures, and output types; see text for details. We report means $\pm$ standard errors of several metrics (from top to bottom): accuracy (for classification) or mean squared error (for regression), confidence (only for classification), negative log-likelihood, ECE (for classification) or regression calibration error [72], mean calibration error (only for classification), and Brier score (only for classification). The in-distribution (left panels) and OOD (right panels) dataset splits correspond to different domains (e.g. hospitals for `Camelyon17`).

[77].[12] In the following, we will briefly outline the Bayesian approach to continual learning [7] 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.

C.4.1 Bayesian Approach to 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 $\mathcal{D} := \{\mathcal{D}_t\}_{t=1}^T$ and have a neural network with parameters θ. In line with the standard supervised learning setting outlined in Section 2, we have a prior on parameters $p(\theta) = \mathcal{N}(\theta; 0, \gamma^2 I)$ and a likelihood $p(\mathcal{D} \mid \theta)$ realized by a neural network. The posterior on the parameters after all tasks is then

$$p(\theta \mid \mathcal{D}) \propto p(\mathcal{D}_T \mid \theta) \times \ldots \times \underbrace{p(\mathcal{D}_2 \mid \theta) \times \underbrace{p(\mathcal{D}_1 \mid \theta) \times p(\theta)}_{\propto p(\theta \mid \mathcal{D}_1)}}_{\propto p(\theta \mid \mathcal{D}_1, \mathcal{D}_2)}. \tag{14}$$

This factorization gives rise to a recursion to update the posterior after $t-1$ data sets to the posterior after t data sets:

$$p(\theta \mid \mathcal{D}_1, \ldots, \mathcal{D}_t) \propto p(\mathcal{D}_t \mid \theta) p(\theta \mid \mathcal{D}_1, \ldots, \mathcal{D}_{t-1}). \tag{15}$$

The normalizer for each update in Eq. (15) is given by the marginal likelihood $p(\mathcal{D}_t \mid \mathcal{D}_1, \ldots, \mathcal{D}_{t-1})$ 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 [99, 24, 77]. Alternatively, variational inference can be used [7, 13].

C.4.2 The Laplace Approximation for Continual Learning

The Laplace approximation facilitates the recursive updates (Eq. (15)) that arise in continual learning. In this context, it was first suggested with a diagonal Hessian approximation by Kirkpatrick et al. [2, EWC] and Huszár [99] corrected their updates. Ritter et al. [24] 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 $\mathcal{N}(\theta^{(t)}_{\text{MAP}}, \Sigma^{(t)})$ We obtain θ_{MAP} by optimizing the unnormalized log posterior distribution on θ as annotated in Eq. (14) for every task, one after another. The Hessian of the same unnormalized log posterior also specifies the posterior covariance $\Sigma^{(t)}$:

$$\Sigma^{(t)} = \Big(\underbrace{\nabla^2_\theta \log p(\mathcal{D}_t \mid \theta)|_{\theta^{(t)}_{\text{MAP}}}}_{\text{log likelihood Hessian}} + \underbrace{\textstyle\sum_{t'=1}^{t-1} \nabla^2_\theta \log p(\mathcal{D}_{t'} \mid \theta)|_{\theta^{(t')}_{\text{MAP}}}}_{\text{previous log likelihood Hessians}} + \underbrace{\gamma^{-2} I}_{\text{log prior Hessian}} \Big)^{-1}. \tag{16}$$

This summation over Hessians is typically intractable for neural networks with large parameter vectors θ and hence diagonal or KFAC approximations are used [2, 99, 24]. 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. [24] 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 $p(\mathcal{D}_t \mid \mathcal{D}_1, \ldots \mathcal{D}_{t-1})$, i.e., by differentiating it with respect to γ. This can be done by computing the eigendecomposition of the summed Kronecker factors [22] and allows us to 1) adjust the regularization suitably per task and 2) avoid setting a hyperparameter thereby alleviating the need for validation data.

C.5 Comparison of Memory Complexity

Table 5 compares the theoretical memory complexity and actual memory footprint (of a Wide ResNet 16-4 on CIFAR-10) of the different methods.

[12] The code is avilable at `https://github.com/team-approx-bayes/fromp`.

Table 5: The memory complexities of all methods in $\mathcal{O}$ 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 Wide ResNet 16-4 (WRN) on CIFAR-10. Here, 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 (DE) members. Mean-field variational inference (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, $S = 12$ CSGHMC samples, and $N = 5$ ensemble members, which are the hyperparameters recommended in the original papers (and therefore also used in our experiments). It can be seen that the proposed default KFAC-last-layer approximation poses a small memory overhead of $\mathcal{O}(H^2 + K^2)$ on top of the MAP estimate.

METHOD	MEM. COMPLEXITY	WRN ON CIFAR-10
MAP	M	11MB
LA	$\mathbf{M + H^2 + K^2}$	**12MB**
VB	$2M$	22MB
DE	NM	55MB
CSGHMC	SM	132MB
SWAG	RM	440MB