Laplace Redux – Effortless Bayesian Deep Learning

Anonymous Author(s)
Affiliation
Address
email

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.

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 criticism includes that BNNs are difficult to implement, finicky to tune, expensive to train, and hard to scale to modern models and datasets. For instance, the popular variational Bayesian method [10–12, etc.] requires considerable changes to the training procedure and model architecture. Not to mention its optimization process is slower and typically more unstable unless carefully tuned [13]. Other methods, such as deep ensembles [14] and Monte Carlo dropout [6] promise to bring uncertainty quantification to standard NNs in simple manners. But they either require a significant cost increase compared to single-network training, 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 [15], the LA dates back to the 18th century [16]. It locally approximates the posterior with a Gaussian distribution centered at a local maximum, with covariance matrix corresponding to the

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

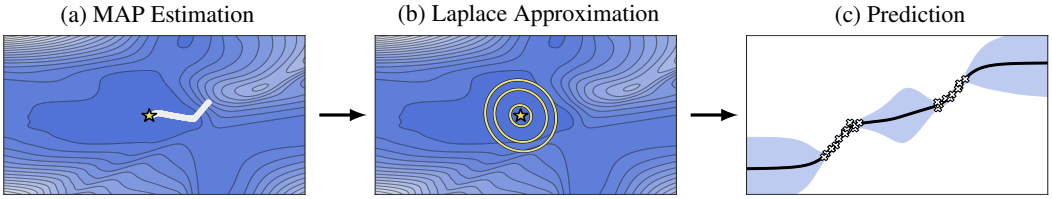


Figure 1: Background contour **(a-b)**: log-posterior landscape on the two-dimensional PCA subspace of the SGD trajectories [30]. The Laplace approximation is a local approximation to the posterior landscape—it fits a Gaussian centered at the MAP estimate (yellow star, found via e.g. SGD **(a)**) with precision equal to the Hessian at that point **(b)**. This approximate posterior then induces predictive uncertainty estimates **(c)**—black curve is the predictive mean, shade is the 95% confidence interval.

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 [17–19] and easy-to-use software libraries [20]. 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-estimated 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, 21]. Figure 1 provides an intuition of the LA.

Yet, despite recent progress in scaling and improving the LA for deep learning [22–28], it is far less widespread than other methods such as variational Bayes, deep ensembles, or SWAG [29]. This is likely due to misconceptions, like that the LA is supposedly 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. We aim to show that these are indeed misconceptions.

To this end, we first survey recent advances and present the components of scalable and practical Laplace approximations in deep learning (Section 2). We then introduce `laplace`, an easy-to-use PyTorch-based library for "turning a NN into a BNN" via the LA (Section 3). Lastly, using `laplace`, we show in an extensive empirical study that the LA is competitive to alternative Bayesian methods, especially considering how simple and cheap it is (Section 4). It is our hope that the LA can see the wider adoption it deserves in both practical and research-oriented deep learning.

2 The Laplace Approximation in Deep Learning

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 neural network (NN) $f_\theta : \mathbb{R}^M \to \mathbb{R}^C$ are trained to minimize the (regularized) empirical risk,

$$\theta_{\text{MAP}} = \arg\min_{\theta \in \mathbb{R}^D} \mathcal{L}(\mathcal{D}; \theta). \tag{1}$$

Typically – though this is not strictly necessary for what follows – the loss decomposes into a sum over empirical terms $\ell(x_n, y_n; \theta)$ and a regularizer $r(\theta)$:

$$\mathcal{L}(\mathcal{D}; \theta) = r(\theta) + \textstyle\sum_{n=1}^N \ell(x_n, y_n; \theta). \tag{2}$$

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:

$$\ell(x_n, y_n; \theta) = -\log p(y_n \mid f_\theta(x_n)) \qquad \text{and} \qquad r(\theta) = -\log p(\theta) \tag{3}$$

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

$$p(\theta \mid \mathcal{D}) = \tfrac{1}{Z}\, p(\mathcal{D} \mid \theta)\, p(\theta) = \tfrac{1}{Z} \exp(-\mathcal{L}(\mathcal{D}; \theta)), \qquad Z := \textstyle\int p(\mathcal{D} \mid \theta)\, p(\theta)\, d\theta \tag{4}$$

with an intractable ***normalizing constant*** Z. ***Laplace approximations*** [16] 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:

$$\mathcal{L}(\mathcal{D}; \theta) \approx \mathcal{L}(\mathcal{D}; \theta_{\text{MAP}}) + \tfrac{1}{2}(\theta - \theta_{\text{MAP}})^\intercal \left(\nabla^2_\theta \mathcal{L}(\mathcal{D}; \theta)|_{\theta_{\text{MAP}}}\right)(\theta - \theta_{\text{MAP}}), \tag{5}$$

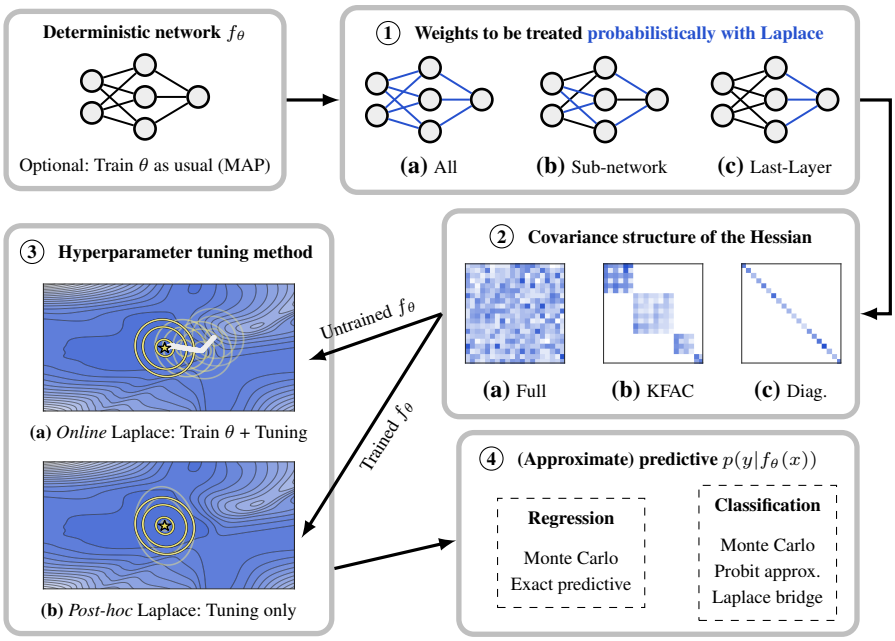


Figure 2: Four key components to scale and apply the LA to a NN f_θ. In panel ③, shades represent the loss landscape, while contours represent LA log-posteriors—faded contours represent intermediate iterates during hyperparameter tuning to obtain the final log-posterior (thick yellow contours).

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

$$p(\theta \mid \mathcal{D}) \approx \mathcal{N}(\theta; \theta_{\text{MAP}}, \Sigma) \qquad \text{with} \qquad \Sigma := -\left(\nabla^2_\theta \mathcal{L}(\mathcal{D}; \theta)|_{\theta_{\text{MAP}}}\right)^{-1}. \tag{6}$$

The normalizing constant Z is useful for model selection and can also be readily approximated as

$$Z \approx \exp(-\mathcal{L}(\mathcal{D}; \theta_{\text{MAP}}))\,(2\pi)^{D/2}\,(\det \Sigma)^{1/2}. \tag{7}$$

See Appendix A for a detailed derivation. Thus, to obtain the approximate posterior, we first need to find the maximum of the log-posterior function, i.e. do "standard" deep learning with regularized empirical risk minimization. The only *additional* step is to compute the Hessian matrix at that point (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 [20]. 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 [22]. The latter is the focus of this work.

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.[1] The Hessian $\nabla^2_\theta \mathcal{L}(\mathcal{D}; \theta)|_{\theta_{\text{MAP}}}$ then depends both on the (simple) log-prior / regularizer and the (complicated) log-likelihood / empirical risk:

$$\nabla^2_\theta \mathcal{L}(\mathcal{D}; \theta)|_{\theta_{\text{MAP}}} = -\gamma^{-2} I - \textstyle\sum_{n=1}^{N} \nabla^2_\theta \log p(y_n \mid f_\theta(x_n))|_{\theta_{\text{MAP}}}. \tag{8}$$

A naive implementation of the Hessian is infeasible because the second term in Eq. (8) 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 a schematic and Appendix A for a more detailed version of the review and discussion.

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

2.1 Four Key Components of Laplace Approximations for Deep Neural Networks

① Inference on all Weights or Subsets of Weights

A simple way to scale the LA to large networks is 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, 27]. This ***last-layer LA*** is cost-effective yet compelling both theoretically and in practice [27]. Alternatively, one can also consider a *general subset* of θ to yield a ***subnetwork LA*** [26], which is motivated by recent findings that neural nets can be heavily pruned without sacrificing test accuracy [38]. Lastly, in most cases it is also possible to use additional compute and treat *all* weights probabilistically when using appropriate approximations of the Hessian, as we discuss next.

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 Hessian of the (ill-conditioned) log-likelihoods of NNs in the second term of Eq. (8) [39]. The ***Fisher information matrix*** [40], abbreviated as *the Fisher* and defined by

$$F := \sum_{n=1}^{N} \mathbb{E}_{\widehat{y} \sim p(y \mid f_\theta(x_n))} \left[(\nabla_\theta \log p(\widehat{y} \mid f_\theta(x_n))|_{\theta_{\text{MAP}}})(\nabla_\theta \log p(\widehat{y} \mid f_\theta(x_n))|_{\theta_{\text{MAP}}})^\intercal \right], \tag{9}$$

is one such choice.[2] One can also use the ***generalized Gauss-Newton matrix (GGN)*** matrix [42]

$$G := \sum_{n=1}^{N} J(x_n) \left(\nabla_f^2 \log p(y_n \mid f)|_{f=f_{\theta_{\text{MAP}}}(x_n)} \right) J(x_n)^\intercal, \tag{10}$$

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 in the common case of exponential family log-likelihoods [39], we will henceforth refer to them interchangeably. In deep LAs, they have emerged as the default choice [22, 23, 27, 28, 26, 25, etc.].

As F and G are still quadratically large, further factorization assumptions are typically made. 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)*** [17–19], which factorizes each within-layer Fisher[3] as a Kronecker product of two smaller matrices. KFAC has been successfully applied to the LA [22, 23] and can be improved by low-rank approximations of the KFAC factors [28] by leveraging their eigendecompositions [45].

③ Hyperparameter Tuning

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 [22, 27, 26, 25, 21]. Commonly, this is done through ***cross-validation***, e.g. by maximizing the validation log-likelihood [22, 46] or, additionally, using out-of-distribution data [27, 47]. However, ***marginal likelihood maximization*** (a.k.a. ***empirical Bayes*** or ***the evidence framework*** [34, 48]) constitutes a more principled alternative to tune these hyperparameters. While both methods can be applied *post-hoc*, marginal likelihood maximization can even be performed in an online manner jointly with the MAP estimation [21] and does not require additional validation data. Other recent approaches include Bayesian optimization [49] or the addition of dedicated, trainable hidden units for the sole purpose of uncertainty tuning [47].

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 [46, 25]. Immer et al. [25] 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 [27]. For the last-layer LA, the Hessian coincides with the GGN and the linearized predictive is exact.

[2] If, instead of taking expectation in (14), we use the training label y_n, we call the matrix the ***empirical Fisher***, which is distinct to the Fisher [39, 41].

[3] 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

model = load_map_model()  # pre-trained model

# User-specified LA flavor
la = Laplace(model, 'classification',
             subset_of_weights='all',
             hessian_structure='diag')
la.fit(train_loader)
la.optimize_prior_precision(method='CV')

# User-specified predictive approx.
pred = la(x, pred_type='glm', link_approx='probit')
```

Listing 1: *Post-hoc* prior precision tuning of last-layer LA with cross-validation and making predictions.

```python
from laplace import Laplace

# Un- or pre-trained
model = load_map_model()

# Default, recommended LA flavor:
# Last-layer KFAC LA
la = Laplace(model, 'regression')
la.fit(train_loader)

# ML w.r.t. prior precision and observation noise
ml = la.marglik(prior_prec, obs_noise)
ml.backward()
```

Listing 2: Differentiating the ML of a KFAC LA w.r.t. hyperparameters, e.g. for *post-hoc* tuning or online empirical Bayes.

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) [24]. 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 of 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, 15] which approximates the logistic function with the probit function. In the multi-class case, we can use its generalization, the ***extended probit approximation*** [50]. Finally, first proposed for non-BNN applications [51, 52], the ***Laplace bridge*** approximates the softmax-Gaussian integral via a Dirichlet distribution [53]. The key advantage is that it yields a *distribution* of the integral solutions.

3 `laplace`: A Toolkit for Deep Laplace Approximations

As straightforward as the theory is, actually implementing the LA is non-trivial, as it requires efficient computation and storage of the Fisher. While these steps are not fundamentally difficult, there exists no complete, easy-to-use, and standardized implementation of various flavors of the LA— instead, it is common for deep learning researchers to repeatedly re-implement the LA and Fisher computation with varying efficiency.[4] An efficient implementation typically requires hundreds of lines of code, making it hard to quickly try out the LA. To address this, we introduce `laplace`: a simple, easy-to-use, extensible library for scalable LAs of deep NNs in PyTorch [54].[5] `laplace` enables *all* possible combinations of the four components discussed in Section 2.1. Code examples are in Listings 1 and 2.

The core of `laplace` consists of efficient implementations of the LA's key quantities: (i) posterior (i.e. Fisher computation and storage), (ii) marginal likelihood, and (iii) posterior predictive. For (i), to take advantage of advances in automatic differentiation, we outsource the Fisher computation to state-of-the-art, optimized second-order optimization libraries such as BackPACK [20]. 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. [21] in our implementation of the LA's marginal likelihood—it is thus both efficient and differentiable and allows the user to implement both the online ML training and *post-hoc* ML tuning, cf. Listing 2. Note that `laplace` also supports standard cross-validation for hyperparameter tuning [22, 27], as shown in Listing 1. Finally, for (iii), `laplace` supports all approximations to the posterior and its predictive distributions discussed in Section 2.1 — it thus provides the user with flexibility in making predictions, depending on the computational budget.

[4] `https://github.com/wjmaddox/swa_gaussian`, `https://github.com/wiseodd/last_layer_laplace`, `https://github.com/DLR-RM/curvature`, etc.

[5] The code is in the supplementary materials and will be open-sourced after the review process.

Default behavior To abstract away from a large number of options available (Section 2.1), 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 a Gaussian linear approximation 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 as backends, it inherits the available choices of Fisher factorizations from these libraries. For instance, the low-rank LA proposed by Lee et al. [28] can currently not be implemented via `laplace`, because BackPACK does not support eigenvalue-corrected KFAC [45]. Similarly, the first version of `laplace` does not yet support the subnetwork LA [26]—we plan to add it in the next iteration of the library.

4 Experiments

Here, 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 [55])—settings where other Bayesian methods are challenging to implement. Section 4.4 shows how the LA can be used as an easy-to-use yet strong baseline in continual learning.

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 B.1 and summarize the main arguments and take-aways in the following.

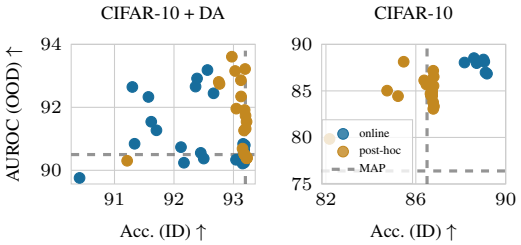
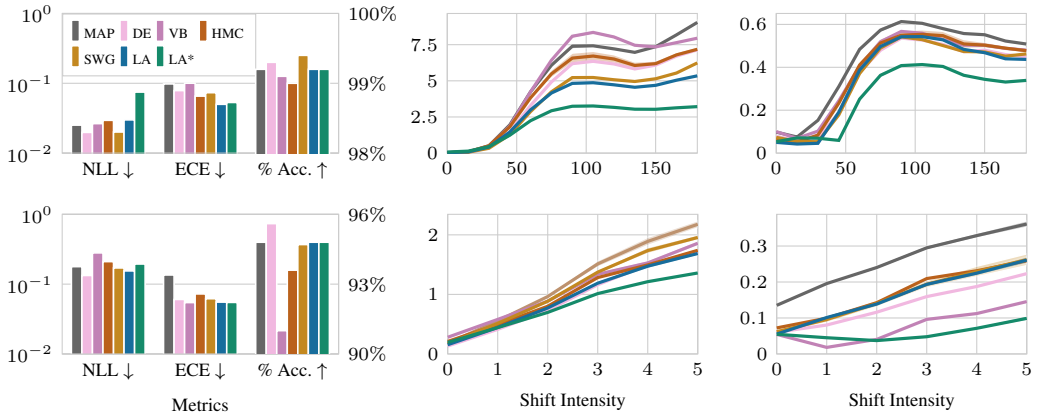


Figure 3: In- and out-of-distribution (ID and OOD, resp.) performance of different LA configurations (dots). "DA" stands for "data augmentation".

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). 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 (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



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

Figure 4: Dataset shift on the Rotated-MNIST (top) and Corrupted-CIFAR-10 datasets (bottom).

significantly worse than KFAC and are therefore not suggested. Independent of the structure, we find that the empirical Fisher (EF) approximations perform better on out-of-distribution detection tasks while GGN approximations tend to perform better on in-distribution metrics.

Predictive distribution. Considering in- and out-of-distribution (OOD) performance as well as cost, the probit provides the best approximation to the predictive for the last-layer LA. MC integration can sometimes be superior for OOD detection but at 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.

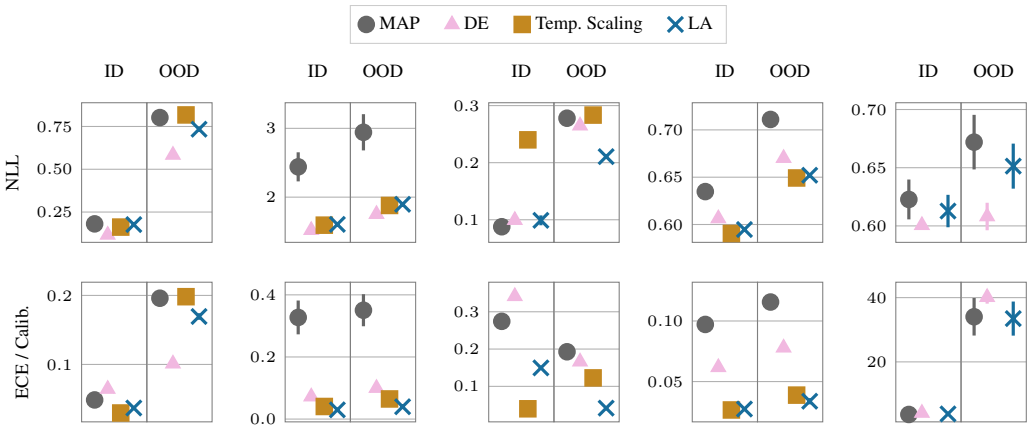
Overall recommendation. Following the experimental evidence, the default in `laplace` is a *post-hoc* KFAC last-layer LA with a GGN approximation to the Hessian. This default is applicable to all architectures that have a fully-connected last layer and can be easily applied to pre-trained networks. For problems where trained networks are unavailable or hyperparameters are unknown, the online KFAC LA with a GGN or EF 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, discovered in the preceding section (**LA***—last-layer, with a full empirical Fisher Hessian approximation, along with the probit approximation). We compare them with the MAP network (**MAP**) and various Bayesian baselines: Deep Ensemble [**DE**, 14], mean-field variational Bayes [**VB**, 11, 12] with the flipout estimator [56], cyclical stochastic-gradient Hamiltonian Monte Carlo [**HMC**, 57], and SWAG [**SWG**, 29]. For each of these baselines, we use the recommended hyperparameters as stated in the original paper—see Appendix A for details. First, we present the standard dataset-shift robustness results on the rotated MNIST and corrupted CIFAR-10 datasets in Fig. 4. The in-distribution results Fig. 4(a) show that **LA** is the best-calibrated method in terms of ECE, while also retaining the accuracy of **MAP** (unlike **VB** and **HMC**). On corrupted data Fig. 4(b-c), we observe that all Bayesian baselines improve **MAP** significantly. Even though *post-hoc*, all LAs achieve competitive results, even to **DE**. In particular, **LA*** achieves the best results over corrupted data, at the expense of slightly worse in-distribution calibration—this trade-off between in- and out-of-distribution (OOD) performance has been observed previously [58]. Finally, on OOD detection (Table 1), **LA** and **LA*** achieve better results than even **VB**, **HMC**, and **SWG**.

Table 1: OOD detection averaged over all test sets.

	Confidence ↓		AUROC ↑	
Methods	**MNIST**	**CIFAR-10**	**MNIST**	**CIFAR-10**
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



(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 [59], covering different data modalities, model architectures, and output types. `Camelyon17`: Tissue slide image tumor classification across hospitals (DenseNet-121 [60]). `FMoW`: Satellite image land use classification across regions/years (DenseNet-121). `CivilCommments`: Online comment toxicity classification across demographics (DistilBERT [61]). `Amazon`: Product review sentiment classification across users (DistilBERT). `PovertyMap`: Satellite image asset wealth regression across countries (ResNet-18 [35]). We report means $\pm$ standard errors of the NLL (top), and of the ECE (for classification) or regression calibration error [62] (bottom). The in-distribution (left panels) and OOD (right panels) dataset splits correspond to different domains (e.g. hospitals for `Camelyon17`).

LAs shines even more when we consider its cost relative to the other, more complicated 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 **HMC** 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 and updating its batch normalization statistics. Moreover, with 30 samples, as recommended by its authors [29], 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**. Together with Fig. 4 and Table 1, this justifies our default flavor in `laplace`, and importantly, validates that Bayesian deep learning does not have to be expensive.

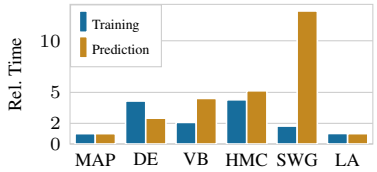


Figure 5: Wall-clock time costs on CIFAR-10, relative to MAP.

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 [55], 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` [59], 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 [59], 2) *post-hoc*

temperature scaling of the MAP models (for classification tasks) [1], and 3) deep ensembles [14].[6] More details on the experimental setup are provided in Appendix B.2. 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 [64], active learning [34, 65], and continual learning [23]. 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 [66]. 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. [23] and improve upon their result by tuning the prior precision through marginal likelihood optimization during training [21] (details in Appendix B.3). Using this scheme, the performance after 10 tasks is at around 96% accuracy, outperforming other Bayesian approaches for continual learning [7, 67, 68]. 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.

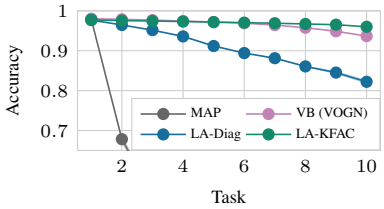


Figure 7: Permuted MNIST.

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 [29] 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 [69, 70, 57, 71, etc.] and deep ensembles [14] can explore several modes. Nevertheless, prior works—also validated in this paper's experiments in Section 4—indicate that using a single mode might not be as limiting in practice as one might think. Wilson and Izmailov [72] 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 [73, 74]. Finally, note that both the LA as well as SWAG can be extended to ensembles of modes in a *post-hoc* manner [75, 72].

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.

[6] We simply construct ensembles from the various pre-trained models provided by Koh et al. [59]. Since these models were trained in different ways, their combinations can be viewed as *hyperparameter ensembles* [63].

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

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

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

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

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

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

[21] Alexander Immer, Matthias Bauer, Vincent Fortuin, Gunnar Rätsch, and Mohammad Emtiyaz Khan. Scalable Marginal Likelihood Estimation for Model Selection in Deep Learning. *arXiv preprint arXiv:2104.04975*, 2021.

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

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

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

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

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

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

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

[29] 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.

[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. URL http://jmlr.org/papers/v21/17-678.html.

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

[47] Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. Learnable Uncertainty under Laplace Approximations. *arXiv preprint arXiv:2010.02720*, 2020.

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

[49] 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.

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

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

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

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

[54] 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.

[55] 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.

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

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

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

[59] 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. *arXiv preprint arXiv:2012.07421*, 2020.

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

[61] 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.

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

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

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

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

[66] 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.

[67] 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.

[68] 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.

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

[70] 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.

[71] Pavel Izmailov, Sharad Vikram, Matthew D Hoffman, and Andrew Gordon Wilson. What Are Bayesian Neural Network Posteriors Really Like? *arXiv preprint arXiv:2104.14421*, 2021.

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

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

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

[75] Anonymous Authors. Mixtures of Laplace Approximations for Improved *Post-Hoc* Uncertainty in Deep Learning. *Under Submission*, 2021.

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

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

[78] 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.

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

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

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Section 3
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We will release our library `laplace` as well as all code required to reproduce the main experimental results after the review process and include it in the supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] In the appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We ran all experiments in Sections 4.1, 4.2 and 4.4 over 5 random seeds, and all experiments in Section 4.3 over as many seeds as provided with the respective pre-trained models, and reported the resulting error bars.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] In the appendix.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Our library `laplace` uses BackPACK [20] in the backend as discussed in Section 3
   (b) Did you mention the license of the assets? [Yes] In the appendix.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]