GRADIENT-BASED HYPERPARAMETER OPTIMIZATION WITHOUT VALIDATION DATA FOR LEARNING FROM LIMITED LABELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Optimizing hyperparameters of machine learning algorithms, especially for limited labeled data, is important but difficult, because obtaining enough validation data in such a case is practically impossible. Bayesian model selection enables hyperparameter optimization *without validation data*, but it requires Hessian log determinants, which is computationally demanding for deep neural networks. We study methods to efficiently approximate Hessian log determinants and empirically demonstrate that approximated Bayesian model selection can effectively tune hyperparameters of algorithms of deep semi-supervised learning and learning from noisy labels.

1 INTRODUCTION

Hyperparameter optimization (HO) is essential in practical machine learning. Especially, recent deep learning algorithms have high flexibilities to be configured properly, and their needs pushes recent research of HO (Bergstra & Bengio, 2012; Bischl et al., 2021) and AutoML (Hutter et al., 2019). Especially, gradient-based HO attracts attentions (Bengio, 2000; Do et al., 2009; Domke, 2012; Maclaurin et al., 2015; Pedregosa, 2016; Franceschi et al., 2018; Liao et al., 2018; Shaban et al., 2019), which is faster than black-box methods and potent to scale to optimizing millions of hyperparameters (Lorraine et al., 2020).

These HO methods leverage an isolated dataset called a validation set, usually split from a given training dataset. It is naturally expected that the more validation data we have, the better hyperparameters we can estimate. However, to obtain more validation data, we need to reduce the number of training data, which may sacrifice the performance of the original machine learning algorithm. This dilemma is crucial, especially when the number of labeled data is limited (Oliver et al., 2018).

Indeed, we can select models, in other words, compare hyperparameter configurations without using validation data. In other words, we can optimize hyperparameters only with training data by maximizing marginal likelihood (ML, also known as *evidence*). Such an approach is called, Bayesian model selection, also known as empirical Bayes (Bernardo & Smith, 1994) or evidence approximation (MacKay, 1992). By applying Laplace's method to the marginalization, ML can be decomposed into the log posterior and the log determinant of its Hessian. Thus, maximizing ML can be interpreted as regularizing models to be as *simple* as possible, while fitting training data (Occam's Razor, MacKay (2003)). Bayesian model selection is used in Gaussian processes to select hyperparameters (Mackay, 1998; Rasmussen & Williams, 2006), *e.g.*, to learn invariance of data (van der Wilk et al., 2018; Schwöbel et al., 2021).

The application of Bayesian model selection to neural networks has more than three decades of history (Buntine & Weigend, 1991; Neal, 1994; Mackay, 1995). Especially, Mackay (1995) pointed out that this approach can 1. compare models without validation data, 2. optimize regularization hyperparameters in an online way, 3. be robust compared to cross-validation, and 4. be achieved by using gradient-based optimization. Nevertheless, its computational demand makes its application to deep neural networks difficult, as evaluating ML requires a Hessian w.r.t. neural network parameters. Therefore, its use is limited to small (Khan et al., 2019) or special (Lyle et al., 2020) neural networks. Exceptionally, Immer et al. (2021); Daxberger et al. (2021) approximate the Hessian matrix with

diagonal or block-diagonal to optimize parameters of priors of Bayesian neural networks. Ru et al. (2020) adopts cumulated training loss as a cheap alternative to ML for neural architecture search.

This paper aims to broaden the application of Bayesian model selection in deep learning research as a tool of gradient-based HO methods, especially when the number of labeled data is limited. To this end, we first compare several efficient log-determinant estimators with the exact ones using small neural networks. These estimators include structure approximations of the Hessian and stochastic estimators of log determinants that leverage matrix-vector products. Next, we show that estimated ML can be used for gradient-based hyperparameter optimization in label-scarce scenarios, namely semi-supervised learning and learning from noisy labels.

2 BACKGROUND

2.1 TRAINING NEURAL NETWORKS

We assume that we have a training dataset $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=0}^N$ consisting of data points $\boldsymbol{x}_i$ and their labels $\boldsymbol{y}_i$, a neural network $f : \mathbb{R}^{\dim \boldsymbol{x}} \to \mathbb{R}^{\dim \boldsymbol{y}}$ parameterized by $\boldsymbol{\theta} \in \mathbb{R}^P$, and hyperparameters $\boldsymbol{\phi} \in \mathbb{R}^H$. We consider an optimization of the model with the L_2 regularization:

$$\underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \ell(f(\boldsymbol{x}), \boldsymbol{y}; \boldsymbol{\theta}, \boldsymbol{\phi}) + \frac{\gamma}{2}\|\boldsymbol{\theta}\|^2, \tag{1}$$

where ℓ is a loss function, such as cross entropy, and γ is a coefficient of the L_2 regularization, which is also a member of $\boldsymbol{\phi}$. This L_2 regularization is equivalent to a weight decay, which is commonly used in neural network training. The optimization in Equation (1) leads to a maximum a posteriori (MAP) estimate, because the loss and weight decay terms are in fact a negative log-likelihood and a negative log-prior, respectively, *i.e.*,

$$\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \ell(f(\boldsymbol{x}), \boldsymbol{y}; \boldsymbol{\theta}, \boldsymbol{\phi}) = -\log p(\mathcal{D}|\boldsymbol{\theta}, \boldsymbol{\phi}), \quad \frac{\gamma}{2}\|\boldsymbol{\theta}\|^2 = -\log p(\boldsymbol{\theta}|\boldsymbol{\phi}). \tag{2}$$

Most deep learning works have relied on an external validation set $\mathcal{V}$ to find the "best" hyperparameters $\boldsymbol{\phi}$ with a certain criterion $\tilde{\ell}$ as

$$\underset{\boldsymbol{\phi}}{\operatorname{argmin}} \sum_{(\boldsymbol{x}',\boldsymbol{y}')\in\mathcal{V}} \tilde{\ell}(f(\boldsymbol{x}'), \boldsymbol{y}'; \boldsymbol{\theta}^\star, \boldsymbol{\phi}) \quad \text{s.t.} \quad \boldsymbol{\theta}^\star = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \ell(f(\boldsymbol{x}), \boldsymbol{y}; \boldsymbol{\theta}, \boldsymbol{\phi}) + \frac{\gamma}{2}\|\boldsymbol{\theta}\|^2. \tag{3}$$

In this paper, we optimize $\boldsymbol{\phi}$ *without* using validation data $\mathcal{V}$ by using a Bayesian model selection method scalable to modern neural networks.

2.2 BAYESIAN MODEL SELECTION

Bayesian model selection decides the "best" hyperparameters as $\boldsymbol{\phi}^\star = \operatorname{argmax}_{\boldsymbol{\phi}} p(\mathcal{D}|\boldsymbol{\phi})$, where $p(\mathcal{D}|\boldsymbol{\phi})$ is a marginal likelihood (ML). Strikingly, we do not need an external validation data $\mathcal{V}$ here to select $\boldsymbol{\phi}$ with this rule. By using Laplace's method[1], we can approximate log ML as

$$\log p(\mathcal{D}|\boldsymbol{\phi}) \approx \log p(\mathcal{D}, \boldsymbol{\theta}^\star|\boldsymbol{\phi}) - \frac{1}{2}\log\det\left(\frac{1}{2\pi}\frac{\partial^2}{\partial\boldsymbol{\theta}^2}\log p(D, \boldsymbol{\theta}^\star|\boldsymbol{\phi})\right). \tag{4}$$

The first term can be decomposed to $\log p(\mathcal{D}|\boldsymbol{\theta}^\star, \boldsymbol{\phi}) + \log p(\boldsymbol{\theta}^\star|\boldsymbol{\phi})$, *i.e.*, negative loss and negative L_2 regularization. A difficulty arises in the second term, Hessian log determinant, as the number of model parameters P increases, because Hessian has quadratically large space complexity $O(P^2)$. This computational cost is infeasible for modern neural networks.

[1] See *e.g.* Bishop (2006) for the detailed derivation.

2.3 RELATIONSHIP TO INFORMATION CRITERIA

Bayesian model selection is closely related to information criteria (Konishi & Kitagawa, 2007), such as Akaike's Information Criterion (AIC, Akaike (1973)), Bayesian Information Criterion (BIC, Schwarz (1978)). Especially, BIC, defined as $-\log p(\mathcal{D}|\hat{\boldsymbol{\theta}}) + \frac{\dim \hat{\boldsymbol{\theta}}}{2} \log(\#\mathcal{D})$ for a maximum likelihood estimator $\hat{\boldsymbol{\theta}}$, is an approximation of the negative of Eq. (4) by ignoring the terms independent of the number of data $\#\mathcal{D}$[2]. Watanabe (2013) extended these criteria to WAIC and WBIC, which are applicable to non-singular models including neural networks. Thomas et al. (2020) showed that Takeuchi's Information Criterion (TIC, Takeuchi (1976)) can capture the generalization gap of neural networks, though TIC is expensive to compute because it needs a Hessian inverse of the true data distribution.

3 ESTIMATION OF HESSIAN LOG DETERMINANT

To obtain the log ML in Equation (3), we only need a scalar value of the log determinant, rather than the Hessian itself. Thus, our aim is to estimate the log determinant without computing the possibly infeasible Hessian. In this section, we first introduce well-behaved approximations of a Hessian matrix in Section 3.1. Next, we describe estimates of log determinants that leverage the structure of matrices (Section 3.2) or the stochastic property of log determinants (Section 3.3). Finally, we empirically compare these estimates in Section 3.4.

3.1 APPROXIMATION OF HESSIAN MATRIX

Because Hessian is not always positive semi-definite, its log determinant is also not always defined. To ease the problem, a Hessian of the log posterior is usually approximated by a Generalized Gauss-Newton matrix (GGN),

$$\boldsymbol{G}_{\boldsymbol{\theta}} = \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\theta}}^\top \frac{\partial^2 \ell(\boldsymbol{z},\boldsymbol{y})}{\partial \boldsymbol{z}^2} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\theta}}, \quad \text{where} \quad \boldsymbol{z} = f(\boldsymbol{x};\boldsymbol{\theta}), \tag{5}$$

or a Fisher information matrix (FIM),

$$\boldsymbol{F}_{\boldsymbol{\theta}} = \sum_{(\boldsymbol{x},\cdot)\in\mathcal{D}} \mathbb{E}_{\boldsymbol{y}'\sim p(\boldsymbol{y}'|\boldsymbol{x},\boldsymbol{\theta})}\Big[\frac{\partial \ell(\boldsymbol{y}'|\boldsymbol{x})}{\partial \boldsymbol{\theta}} \frac{\partial \ell(\boldsymbol{y}'|\boldsymbol{x})}{\partial \boldsymbol{\theta}}^\top\Big]. \tag{6}$$

Indeed, these two matrices are equivalent to each other when we use cross entropy or squared loss (Martens, 2020). By altering a Hessian in Eq. (4) with a GGN and a FIM, the matrix always becomes strongly positive semi-definite if $\gamma > 0$, and thus, its log determinant always exist. These matrices are also used instead of a Hessian in optimization (Amari, 1998; Schraudolph, 2002; Botev et al., 2017).

$$\bar{\boldsymbol{F}}_{\boldsymbol{\theta}} = \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \Big(\frac{\partial \ell(\boldsymbol{y}|\boldsymbol{x})}{\partial \boldsymbol{\theta}} \frac{\partial \ell(\boldsymbol{y}|\boldsymbol{x})}{\partial \boldsymbol{\theta}}^\top\Big) \tag{7}$$

is a computationally efficient alternative of a FIM and called as an "Empirical Fisher" (EF) (Schraudolph, 2002; Roux et al., 2008). Immer et al. (2021); Daxberger et al. (2021) used an Empirical Fisher instead of an FIM to estimate the log determinant of a Hessian.

3.2 DIAGONAL AND BLOCK-DIAGONAL APPROXIMATION

A Hessian of a neural network is a $P \times P$ matrix, which is sometimes infeasibly large to store and compute its log determinant as its $O(P^3)$ time complexity. We can approximate a Hessian with

[2]See *e.g.*, Sugiyama (2015) for the detailed derivation.

a representation that is compact and easy to compute its log determinant by using a diagonal or a block-diagonal approximation (Ritter et al., 2018; Immer et al., 2021; Daxberger et al., 2021). As a diagonal approximation, the following approximation as in (Duchi et al., 2011; Kingma & Ba, 2015) is computationally efficient:

$$\boldsymbol{F}_{\boldsymbol{\theta}} \approx \mathbb{E}[\frac{\partial \ell}{\partial \boldsymbol{\theta}}]\mathbb{E}[\frac{\partial \ell}{\partial \boldsymbol{\theta}}]^{\top}. \tag{8}$$

Its ith diagonal element can be obtained by $\mathbb{E}[\frac{\partial \ell}{\partial \boldsymbol{\theta}}]_i^2$.

As a block-diagonal approximation, K-FAC (Martens & Grosse, 2015; Grosse & Martens, 2016) is popular, where the FIM or EF's block corresponding to the lth layer of a neural network is approximated as

$$\boldsymbol{F}_{\boldsymbol{\theta}}^{(l)} \approx \mathbb{E}[\boldsymbol{a}_{l-1}\boldsymbol{a}_{l-1}^{\top}] \otimes \mathbb{E}[\boldsymbol{g}_l \boldsymbol{g}_l^{\top}], \tag{9}$$

with $\boldsymbol{a}_{l-1}$, the $l-1$th layer's activation, and $\boldsymbol{g}_l$, the loss derivative w.r.t. $\boldsymbol{a}_l$. $\otimes$ denotes Kronecker product. Once these matrices ignoring off-(block) diagonal elements are obtained, log determinants can be computed cheaply as a sum of block-wise log determinants.

3.3 Stochastic Approximation

Another approach is stochastic estimations of log determinants. For a positive semi-definite matrix $\boldsymbol{A} \in \mathbb{R}^{M \times M}$, we can estimate its log determinant with a random probe vector $\boldsymbol{v} \in \mathbb{R}^M$ sampled from $\mathcal{P}$, a Rademacher distribution or an isotropic Gaussian distribution[3], using Hutchinson's estimator (Hutchinson, 1990; Avron & Toledo, 2011) as

$$\log\det \boldsymbol{A} = \operatorname{tr} \log \boldsymbol{A} = \mathbb{E}_{\boldsymbol{v}\sim\mathcal{P}}[\boldsymbol{v}^{\top}(\log \boldsymbol{A})\boldsymbol{v}]. \tag{10}$$

Although directly evaluating a matrix-logarithm $\log \boldsymbol{A}$ may be infeasible in our case, $\boldsymbol{v}^{\top}(\log \boldsymbol{A})\boldsymbol{v}$ can be efficiently estimated by using matrix-vector product. Namely, $\boldsymbol{v}^{\top}(\log \boldsymbol{A})\boldsymbol{v}$ can be approximated by using a degree-m polynomial function $\boldsymbol{p}^{(m)}(z) = \sum_{i=0}^{m} p_i^{(m)} z^i$, such as a Taylor polynomial (Boutsidis et al., 2017) or a Chebyshev polynomial of the first kind (Han et al., 2015; 2017), as

$$\boldsymbol{v}^{\top}(\log \boldsymbol{A})\boldsymbol{v} \approx \sum_{i=0}^{m} p_i^{(m)} \boldsymbol{v}^{\top}\boldsymbol{A}^i\boldsymbol{v}. \tag{11}$$

In the following, we only consider Chebyshev polynomials, because it converges faster than Taylor polynomials (Phillips, 2003). Alternatively, $\boldsymbol{v}^{\top}(\log \boldsymbol{A})\boldsymbol{v}$ can also be estimated by the stochastic Lanczos quadrature method (Ubaru et al., 2017; Chen et al., 2021) as

$$\boldsymbol{v}^{\top}(\log \boldsymbol{A})\boldsymbol{v} \approx \sum_{i=1}^{m} e_i^2 \log(d_i), \tag{12}$$

where e_i is the first element of the ith eigenvector, and d_i is the ith eigenvalue of a tridiagonal matrix generated by m iterations of the Lanczos algorithm. Significantly, these approaches do not require to hold $\boldsymbol{A}$ explicitly, if a matrix-vector product $\boldsymbol{A}\boldsymbol{v}$ is available. Fortunately, in our case, $\boldsymbol{G}_{\boldsymbol{\theta}}$ is positive semi-definite, and $\boldsymbol{G}_{\boldsymbol{\theta}}\boldsymbol{v}$ can be computed by using combination of vector-Jacobian products and a vector-Hessian product that are efficiently computed with reverse-mode automatic differentiation tools (Schraudolph, 2002; Baydin et al., 2018), such as PyTorch (Paszke et al., 2019) and JAX (Bradbury et al., 2018).

[3]In the main experiment, we use an isotropic Gaussian distribution, which shows no significant difference from a Rademacher distribution (Appendix A.1).

These methods have been used to estimate Hessian spectra of neural networks (Ghorbani et al., 2019) or log determinants of Gaussian processes (Dong et al., 2017), but not for Bayesian model selection of deep neural networks.

3.4 COMPARISON OF APPROXIMATED LOG-DET WITH EXACT LOG-DET

Although computing exact Hessian and GGN matrices is infeasible for modern neural networks with millions of parameters, computing them for small networks with thousands of parameters is feasible. To benchmark the ability of approximation methods, we compute exact Hessian and GGN matrices to obtain their log determinants. As for approximated log determinants, we used the following methods:

Diag Diagonal approximation of $\bar{\boldsymbol{F}}_{\boldsymbol{\theta}}$ as in Eq. (8),

KFAC K-FAC approximation $\bar{\boldsymbol{F}}_{\boldsymbol{\theta}}$ as in Eq. (9),

Chebyshev (m) Polynomial approximation of $\boldsymbol{G}_{\boldsymbol{\theta}}$ using Chebyshev polynomials of the first kind with degree of m as in Eq. (11),

SLQ (m) Stochastic Lanczos quadrature approximation of $\boldsymbol{G}_{\boldsymbol{\theta}}$ with m iterations of the Lanczos algorithm as in Eq. (12).

For this comparison, we prepared a three-layer MLP model and a variant of LeNet, consisting of two convolutional layers succeeded by a two-layer MLP, which have 26,500 and 19,700 parameters, respectively. We trained these models on a subset of MNIST dataset (LeCun et al., 2010), consisting of hand-written digit images and their labels, for 100 epochs (Fig. 1 Left).

Figure 1 (Right) presents comparisons of log determinants of exact matrices as well as those of approximated ones. For stochastic estimators, *i.e.*, Chebyshev and SLQ, we reported the averaged value of 100 trials with different probe vectors. As can be seen, Diag and SLQ ($m = 8$) well approximate the actual value, the log determinant of the GGN matrix, on both networks. On the other hand, the results suggest that K-FAC and Chebyshev approximations may not be appropriate methods for estimating the GGN matrix's log determinant. Estimated values by Chebyshev and SLQ reflect warps in test loss curves, while those by Diag and K-FAC appear almost no reflection of the warps.

Unlike Diag and K-FAC, Chebyshev(m) and SLQ(m) are stochastic estimators and have a freedom in the choice of m, which corresponds to the accuracy and computational cost of the approximation. Figure 2 compares log determinants estimated by Chebyshev(m) and SLQ(m) with $m = 4, 8, 12$, and each plot shows 100 trials. We observe that SLQ can approximate the log determinant with less computational cost and higher accuracy than Chebyshev polynomial, which aligns with the observations by Dong et al. (2017). This difference may be attributed to the fact that most of the eigenvalues of (approximated) Hessian matrices of neural networks are quite small (Ghorbani et al., 2019; Karakida et al., 2019), and Chebyshev polynomial fails to approximate the logarithm function $\log x$ that exponentially goes to the negative infinity as $x \to 0$. See Appendix A.2 for the results using LeNet.

In the above experiments, we used the entire training data to evaluate log determinants. In practice, evaluating log determinants on full data is sometimes infeasible, and minibathces are used instead. Figure 3 compares estimated values of Diag and SLQ (m=8) with different minibatch sizes. Each plot shows 100 trials with different minibatches. In the case of SLQ, we used different probe vectors for each minibatch. While the minibatch Diag seems to converge to its full version (*c.f.* Fig. 1), the mean of minibatch SLQ appears to approximate the exact log determinant accurately. Its LeNet counterpart is presented in Appendix A.2.

Based on these observations, we used Diag and SLQ as approximations of the log determinant for the experiments in Section 5. Finally, we summarize the comparison of these methods in Table 1.

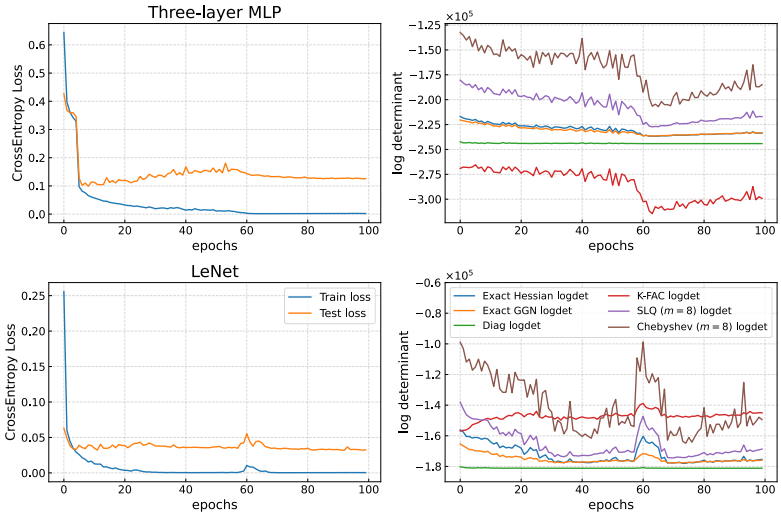

Figure 1: *Diagonal and Stochastic Lanczos Quadrature (SLQ) approximate log determinant of GGN matrix accurately.* (Left) We trained a three-layer MLP and a variant of LeNet until convergence for 100 epochs. (Right) We computed exact Hessian and GGN matrices of these networks and compare their log determinants with approximated ones.

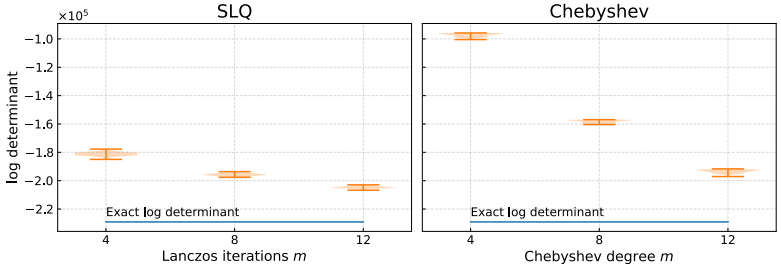

Figure 2: *SLQ (m) can approximate the exact log determinant more efficiently and accurately than Chebyshev (m).* Log determinant of the exact GGN matrix, SLQ, and Chebyshev of a trained three-layer MLP are presented.

4 Gradient-based Hyperparameter Optimization using Marginal Likelihood

Hyperparameters $\boldsymbol{\phi}$ consist of two groups, $\boldsymbol{\phi}_d$ and $\boldsymbol{\phi}_n$: the posterior $p(\mathcal{D}, \boldsymbol{\theta}|\boldsymbol{\phi})$ is a differentiable function for $\boldsymbol{\phi}_d$ but not for $\boldsymbol{\phi}_n$. $\boldsymbol{\phi}_d$ includes coefficients of multiple loss terms, such as L_2 regularization factor. On the other hand, the number of training epochs and the momentum rate of optimization can be regarded as $\boldsymbol{\phi}_n$.

Notably, the above-mentioned Hessian log-determinant estimators are differentiable w.r.t. $\boldsymbol{\phi}_d$, and $\boldsymbol{\phi}_d$ can be optimized by using gradient-based optimization of a step size of α as

$$\boldsymbol{\phi}_d \leftarrow \boldsymbol{\phi}_d + \alpha \frac{\partial \log p(\mathcal{D}|\boldsymbol{\phi})}{\partial \boldsymbol{\phi}_d}, \tag{13}$$

together with optimization of $\boldsymbol{\theta}$ in an online fashion (Immer et al., 2021), as an Expectation-Maximization algorithm (Bishop, 2006). Though the parameters $\boldsymbol{\theta}$ may not be an MAP estimate $\boldsymbol{\theta}^\star$ as required in Eq. (4) during training, Immer et al. (2021) found that we can ignore this difference during online optimization.

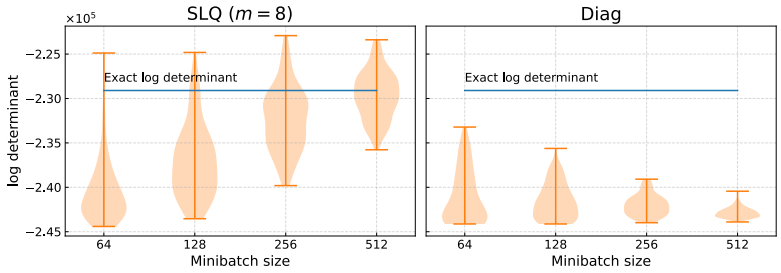

Figure 3: *SLQ with minibatches can estimate the exact logdeterminant accurately.* Log determinants of the exact GGN matrix, SLQ, and Diag of a trained three-layer MLP are presented.

	Diag	K-FAC	Chebyshev	SLQ
Time Complexity	$O(P)$	$O(\sum_l P_l^3)$	$O(mP)$	$O(mP)$
Space Complexity	$O(P)$	$O(\sum_l P_l^2)$	$O(mP)$	$O(mP)$
Empirical Precision	✓			✓

Table 1: Summary of comparison of log determinant estimators. P_l is the size of lth block ($\sum_l P_l = P$). See Appendix B for the details.

Evaluating ML on the entire dataset $\mathcal{D}$ is sometimes difficult in practice, and the following stochastic gradient of a minibatch $\mathcal{B} \subset \mathcal{D}$ can be used instead as

$$\frac{\partial \log p(\mathcal{D}|\boldsymbol{\phi})}{\partial \boldsymbol{\phi}_d} \propto \mathbb{E}_{\mathcal{B}\subset\mathcal{D}}\left[\frac{\partial \log p(\mathcal{B}|\boldsymbol{\phi})}{\partial \boldsymbol{\phi}_d}\right]. \tag{14}$$

When we apply a stochastic ML estimator, such as Eq. (11), we use $\mathbb{E}_{\boldsymbol{v}\sim\mathcal{P}}\mathbb{E}_{\mathcal{B}\subset\mathcal{D}}\left[\frac{\partial \log p^{(\boldsymbol{v})}(\mathcal{B}|\boldsymbol{\phi})}{\partial \boldsymbol{\phi}_d}\right]$, where $p^{(\boldsymbol{v})}$ denotes an ML estimate with a random probe vector $\boldsymbol{v}$, such as Eqs. (11) and (12). It is also possible to select $\boldsymbol{\phi}_n$: namely, for several candidates of $\boldsymbol{\phi}_n$, such as $(\boldsymbol{\phi}_n^{(1)}, \boldsymbol{\phi}_n^{(2)}, \ldots, \boldsymbol{\phi}_n^{(M)})$, we can select $\boldsymbol{\phi}_n^\star = \operatorname*{argmax}_m p(\mathcal{D}|\boldsymbol{\phi}_d, \boldsymbol{\phi}_n^{(m)})$. This selection can be applied to, for example, neural architecture search (Ru et al., 2020; Immer et al., 2021).

5 Experiments

We demonstrate that gradient-based HO using ML is applicable to limited-labeled data problems.

We maximized ML w.r.t. hyperparameters using stochastic gradient in Eq. (14) after each epoch with 30 iteration of updates. This HO starts after the first 10% of total training iterations for the model parameter optimization ends. We adopted Adam optimizer (Kingma & Ba, 2015) with learning rate of 1.0×10^{-4} and gradient norm clipping of 1. In addition to log ML estimations using the whole model parameters, we also used estimations only using the last-layer parameters as Immer et al. (2021). We denote results with these estimators as LLO. LLO further reduces the computational burden of computation of Hessian log determinants, but Immer et al. (2021) reported it may sacrifice performance in some cases. We found that full SLQ sometimes suffers from numerical instability.

We describe further experimental details in Section 7.

5.1 Semi-supervised Learning

Semi-supervised learning (SSL) algorithms learn to classify data by leveraging unlabeled data in addition to limited labeled data (Chapelle et al., 2006). Because labeled data are scarce in this scenario, HO without validation data is appealing.

We used an SSL algorithm of FixMatch (Sohn et al., 2020) with an objective function of

	CIFAR-10 (1,000)	CIFAR-10 (250)	CIFAR-100 (1,000)
Baseline	6.37	9.74	47.5
Diag	6.44	12.7	50.7
Diag (LLO)	6.59	12.8	53.2
SLQ	5.91	8.21	48.2
SLQ (LLO)	5.78	7.20	46.8

Table 2: *SLQ can effectively optimize hyperparameters of FixMatch without validation data.* Test error rates on CIFAR-10 and CIFAR-100 are presented. The figures in parentheses correspond to the number of labeled data.

$$\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}_L}\ell_C(f(\boldsymbol{x}),\boldsymbol{y}) + \mathbb{E}_\epsilon\mathbb{E}_{\substack{\boldsymbol{u} \text{ s.t.} \\ \boldsymbol{u}\sim\mathcal{D}_U, \max f(s_\epsilon(\boldsymbol{u}))>\eta}} \ell_C\left(f(s_\epsilon(\boldsymbol{u})), \varsigma(f(\boldsymbol{u})/\tau)\right), \tag{15}$$

where ℓ_C is cross entropy loss, $\mathcal{D}_L, \mathcal{D}_U$ are labeled and unlabeled data, s is a strong image transformation policy with randomness ϵ, ς is the softmax function, η is a confidence threshold, and τ is a temperature parameter. We optimized η and τ by maximizing ML derived from Eq. (15). We simulated a semi-supervised dataset by hiding some labels of CIFAR-10 and CIFAR-100 datasets (Krizhevsky, 2009), which is a common procedure in SSL research (Oliver et al., 2018). The ratio of labeled to unlabeled data in minibatch was set to 4 and trained for 1.3×10^5 iterations. We used WideResNet-28-2 (Zagoruyko & Komodakis, 2016) as an image classifier.

Table 2 shows test error rates. The baseline uses the default values $\eta = 1$ and $\tau = 0.95$. After optimizing hyperparameters initialized with these values, SLQ (LLO) achieved performance improvement of 2.5% on CIFAR-10 with 250 labeled data. We observed that SLQ decreases both the threshold and temperature as training proceeds, which yields more solid pseudo labels and more unlabeled data to be used. Diag, on the other hand, enlarges the temperature, which results in poor performance.

5.2 Learning from Noisy Labels

Deep neural networks can memorize randomly assigned labels of image datasets (Arpit et al., 2017). Because such label noise problem is inevitable, robust learning algorithms are required. For HO, a cleanly labeled validation set is needed in this setting, but they are hard to obtain in practice.

We adopted a generalized cross entropy loss (Zhang & Sabuncu, 2018), which is a loss function of

$$\ell_q(f(\boldsymbol{x}),\boldsymbol{y}) = \frac{1-\varsigma(f(\boldsymbol{x}))^q}{q}. \tag{16}$$

When $q \to 0$, ℓ_q behaves as the cross entropy loss, which converges faster but is sensitive to noise. On the other hand, when $q \to 1$, ℓ_q acts as the mean absolute error, which is robust but converges slower, especially when the number of categories is large. Therefore, this hyperparameter q needs to be carefully chosen, and we optimize q by maximizing ML. We simulated noisy labels by replacing randomly chosen labels of CIFAR-10 and CIFAR-100 with others. We adopted WideResNet-28-2 (Zagoruyko & Komodakis, 2016).

Table 3 presents test error rates when 40% of labels are corrupted. The baselines are generalized cross entropy of $q = 0$ (cross entropy), $q = 1$ (mean absolute error), and $q = 0.7$ as Zhang & Sabuncu (2018). For HO, we initialized $q = 0$. Again, Diag and SLQ consistently yielded performance increase without relying on external validation data.

6 Conclusion

In this paper, we studied efficient approximations of Bayesian model selection for deep neural networks and demonstrated that this approach is effectively applicable to gradient-based hyperparameter optimization in the limited-labeled data scenarios. Specifically, we first compared methods to efficiently approximate Hessian log determinants and found that diagonal approximation and

	CIFAR-10 (40%)	CIFAR-100 (40%)
Baseline ($q = 0$)	15.7	40.8
Baseline ($q = 0.7$)	11.4	37.8
Baseline ($q = 1$)	11.2	97.3
Diag	9.91	33.5
Diag (LLO)	9.94	33.3
SLQ	9.78	34.7
SLQ (LLO)	9.97	33.4

Table 3: *Diag can effectively optimize hyperparameters of Generalized Cross Entropy without validation data.* Test error rates on CIFAR-10 and CIFAR-100 with 40% of label noise are presented.

stochastic Lanczos quadrature are effective for the approximation. Then, we empirically showed that hyperparameters could be optimized by maximizing the estimated marginal likelihoods in a gradient-based manner.

This approach is only applicable to hyperparameters that are directly dependent on loss values. Additionally, some hyperparameters of optimizing algorithms may be converted to optimizable ones. For example, as we discussed in Section 2.1, a weight decay factor can be regarded as an L2 regularization factor, and a learning rate can be viewed as a multiplier of a loss value. Neural network architectures can also be treated in this way (Liu et al., 2018). Thus, we believe that this HO approach is widely applicable, without requiring validation data.

7 Reproducibility

7.1 Data Processing

For training images of MNIST used in Section 3.4, we standardized them using their training data statistics. For training images of CIFAR-10 and CIFAR-100 used in Sections 5.1 and 5.2, we applied random horizontal flipping and random cropping into 32×32 pixels after padding 4 pixels to each border as standard data augmentation. Then, both training and testing images are standardized by statistics of training data.

7.2 Implementation Details

We implemented the methods using `PyTorch` v1.9.0 (Paszke et al., 2019) using CUDA 11.1 and conducted experiments on NVIDIA A100 GPUs. We set a random seed for each experiment and reported averaged values of three runs with different seeds. During computation of ML, Batch Normalization (Ioffe & Szegedy, 2015) in neural networks is computed using running statistics.

For stochastic log determinant estimators, we used isotropic Gaussian vectors as probes. See Appendix A.1 for comparison with ones using Rademacher probe vectors. To compute K-FAC, we used `backpack`[4].

The FixMatch algorithm in Section 5.1 follows hyperparameter configurations of a PyTorch implementation[5] except for the ratio of labeled to unlabeled data in minibatch (set to 4) and the number of training iterations (set to 1.3×10^5). We updated log ML after each epoch, where an epoch is 440 iterations. The activation function of WideResNet is replaced with Leaky ReLU with a slope of 0.1 following (Sohn et al., 2020).

Training with Generalized Cross Entropy in Section 5.2 is for 200 epochs with a minibatch size of 128. After each epoch, the log ML is updated with a minibatch size of 512. The model is trained with SGD with a momentum of 0.9 and a weight decay of 5.0×10^{-4}. The initial learning rate was set to 0.1, which decays according to the cosine annealing rule.

[4] `https://backpack.pt/`

[5] `https://github.com/kekmodel/FixMatch-pytorch/`

7.3 SOURCE CODE

We will make the source code to reproduce the experiments publicly available after publication.

REFERENCES

Hirotsugu Akaike. Information theory and an extension of the maximum likelihood principle. *International Symposium on Information Theory*, 1973.

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

Devansh Arpit, Stanisław Jastrzundefinedbski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S. Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, and Simon Lacoste-Julien. A closer look at memorization in deep networks. In *ICML*, 2017.

Haim Avron and Sivan Toledo. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *Journal of the ACM*, 58(2), April 2011.

Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *JMLR*, 18(153):1–43, 2018.

Yoshua Bengio. Gradient-based optimization of hyperparameters. *Neural Computation*, 12(8): 1889–1900, 2000.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *JMLR*, 13: 281–305, 2012.

José M Bernardo and Adrian FM Smith. *Bayesian theory*. John Wiley & Sons, 1994.

Bernd Bischl, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas, Theresa Ullmann, Marc Becker, Anne-Laure Boulesteix, Difan Deng, and Marius Lindauer. Hyperparameter optimization: Foundations, algorithms, best practices and open challenges, 2021.

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

Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical gauss-newton optimisation for deep learning. In *ICML*, 2017.

Christos Boutsidis, Petros Drineas, Prabhanjan Kambadur, Eugenia-Maria Kontopoulou, and Anastasios Zouzias. A randomized algorithm for approximating the log determinant of a symmetric positive definite matrix. *Linear Algebra and its Applications*, 533:95–117, 2017.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

Wray L. Buntine and Andreas S. Weigend. Bayesian back-propagation. *Complex Systems*, 5(6): 603–643, 1991.

Olivier Chapelle, Bernhard Scholköpf, and Alexander Zien. *Semi-supervised learning*. MIT Press, 2006.

Tyler Chen, Thomas Trogdon, and Shashanka Ubaru. Analysis of stochastic lanczos quadrature for spectrum approximation. In *ICML*, 2021.

Erik Daxberger, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig. Laplace redux–effortless bayesian deep learning, 2021.

Chuong B. Do, Chuan Sheng Foo, and Andrew Y. Ng. Efficient multiple hyperparameter learning for log-linear models. In *NIPS*, 2009.

Justin Domke. Generic methods for optimization-based modeling. *JMLR*, 22:318–326, 2012.

Kun Dong, David Eriksson, Hannes Nickisch, David Bindel, and Andrew G Wilson. Scalable log determinants for gaussian process kernel learning. In *NeurIPS*, 2017.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 12(61):2121–2159, 2011.

J. K. Fitzsimons, M. A. Osborne, S. J. Roberts, and J. F. Fitzsimons. Improved stochastic trace estimation using mutually unbiased bases, 2016.

Luca Franceschi, Paolo Frasconi, Saverio Salzo, and Massimilano Pontil. Bilevel Programming for Hyperparameter Optimization and Meta-Learning. In *ICML*, 2018.

Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An investigation into neural net optimization via hessian eigenvalue density. In *ICML*, 2019.

Roger Grosse and James Martens. A kronecker-factored approximate fisher matrix for convolution layers. In *ICML*, 2016.

Insu Han, Dmitry Malioutov, and Jinwoo Shin. Large-scale log-determinant computation through stochastic chebyshev expansions. In *ICML*, 2015.

Insu Han, Dmitry Malioutov, Haim Avron, and Jinwoo Shin. Approximating spectral sums of large-scale matrices using stochastic chebyshev approximations. *SIAM Journal on Scientific Computing*, 39(4):A1558–A1585, 2017.

M.F. Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics - Simulation and Computation*, 19(2):433–450, 1990.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren (eds.). *Automatic Machine Learning: Methods, Systems, Challenges*. Springer, 2019.

Alexander Immer, Matthias Bauer, Vincent Fortuin, Gunnar Rtsch, and Mohammad Emtiyaz Khan. Scalable marginal likelihood estimation for model selection in deep learning. In *ICML*, 2021.

Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *ICML*, 2015.

Ryo Karakida, Shotaro Akaho, and Shun-ichi Amari. Universal statistics of fisher information in deep neural networks: Mean field approach. In *AISTATS*, 2019.

Mohammad Emtiyaz E Khan, Alexander Immer, Ehsan Abedi, and Maciej Korzepa. Approximate inference turns deep networks into gaussian processes. In *NeurIPS*, 2019.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Sadanori Konishi and Genshiro Kitagawa. *Information Criteria and Statistical Modeling*. Springer Publishing Company, Incorporated, 1st edition, 2007.

Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images, 2009.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Renjie Liao, Yuwen Xiong, Ethan Fetaya, Lisa Zhang, Ki Jung Yoon, Xaq Pitkow, Raquel Urtasun, and Richard Zemel. Reviving and improving recurrent back-propagation. In *ICML*, 2018.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable Architecture Search. In *ICLR*, 2018.

Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *AISTATS*, 2020.

Clare Lyle, Lisa Schut, Robin Ru, Yarin Gal, and Mark van der Wilk. A bayesian perspective on training speed and model selection. In *NeurIPS*, 2020.

David MacKay. *Bayesian Interpolation*, pp. 39–66. Springer Netherlands, 1992.

David Mackay. Probable networks and plausible predictions – a review of practical bayesian methods for supervised neural networks. *Network: Computation In Neural Systems*, 6:469–505, 1995.

David Mackay. Introduction to gaussian processes. 1998.

David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge University Press, 2003.

Dougal Maclaurin, David Duvenaud, and Ryan P. Adams. Gradient-based Hyperparameter Optimization through Reversible Learning. In *ICML*, 2015.

James Martens. New insights and perspectives on the natural gradient method. *JMLR*, 21(146): 1–76, 2020.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *ICML*, 2015.

Radford M Neal. *BAYESIAN LEARNING FOR NEURAL NETWORKS*. PhD thesis, University of Toronto, 1994.

Avital Oliver, Augustus Odena, Colin A Raffel, Ekin Dogus Cubuk, and Ian Goodfellow. Realistic evaluation of deep semi-supervised learning algorithms. In *NeurIPS*, 2018.

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.

Fabian Pedregosa. Hyperparameter optimization with approximate gradient. In *ICML*, 2016.

George M Phillips. *Interpolation and approximation by polynomials*, volume 14. Springer Science & Business Media, 2003.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

Hippolyt Ritter, Aleksandar Botev, and David Barber. A scalable laplace approximation for neural networks. In *ICLR*, 2018.

Nicolas Roux, Pierre-antoine Manzagol, and Yoshua Bengio. Topmoumoute online natural gradient algorithm. In *NIPS*, 2008.

Binxin Ru, Clare Lyle, Lisa Schut, Mark van der Wilk, and Yarin Gal. Revisiting the train loss: an efficient performance estimator for neural architecture search, 2020.

Nicol N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738, 2002.

Gideon Schwarz. Estimating the Dimension of a Model. *The Annals of Statistics*, 6(2):461 – 464, 1978.

Pola Schwöbel, Martin Jørgensen, Sebastian W. Ober, and Mark van der Wilk. Last layer marginal likelihood for invariance learning, 2021.

Amirreza Shaban, Ching-An Cheng, Nathan Hatch, and Byron Boots. Truncated Back-propagation for Bilevel Optimization. In *AISTATS*, 2019.

Kihyuk Sohn, David Berthelot, Nicholas Carlini, Zizhao Zhang, Han Zhang, Colin A Raffel, Ekin Dogus Cubuk, Alexey Kurakin, and Chun-Liang Li. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. In *NeurIPS*, 2020.

Masashi Sugiyama. *Introduction to Statistical Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2015.

Kei Takeuchi. Distribution of information statistics and validity criteria of models. *Mathematical Science*, 153:12–18, 1976.

Valentin Thomas, Fabian Pedregosa, Bart van Merriënboer, Pierre-Antoine Manzagol, Yoshua Bengio, and Nicolas Le Roux. On the interplay between noise and curvature and its effect on optimization and generalization. In *AISTATS*, 2020.

Shashanka Ubaru, J. Chen, and Y. Saad. Fast estimation of tr(f(a)) via stochastic lanczos quadrature. *SIAM J. Matrix Anal. Appl.*, 38:1075–1099, 2017.

Mark van der Wilk, Matthias Bauer, ST John, and James Hensman. Learning invariances using the marginal likelihood. In *NeurIPS*, 2018.

Sumio Watanabe. A widely applicable bayesian information criterion. *JMLR*, 14(1):867897, Mar 2013. ISSN 1532-4435.

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

Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. In *Advances in Neural Information Processing Systems*, 2018.

A Additional Experiments

A.1 Stochastic log-det Estimators with Rademacher distribution

In the main part, we adopted random probe vectors sampled from the standard normal distribution. Hutchinson (1990) used the Rademacher distribution, and (Fitzsimons et al., 2016) used the mutually unbiased bases (MUB) in a complex space, which may be difficult to apply to our case. The use of these distributions reduces the variance of trace estimators. Figure 4 shows estimated log determinants when using Rademacher random vectors. Compared with Gaussian vectors in Fig. 2, Rademacher random vectors show only limited difference.

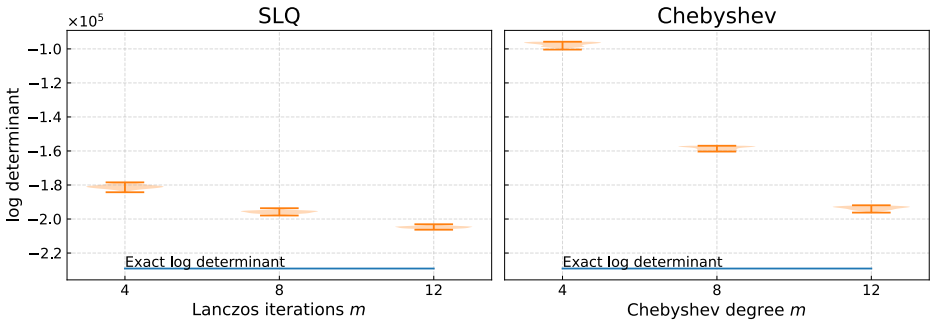

Figure 4: Estimated log determinants of Rademacher-based SLQ and a Chebyshev polynomial. We found no significant difference from Gaussian-based ones presented in Fig. 2.

A.2 LeNet counterparts

We present LeNet counterparts of Figs. 2, 3 and 4 in Figs. 5 to 7. Importantly, the choice of networks does not alter our claims.

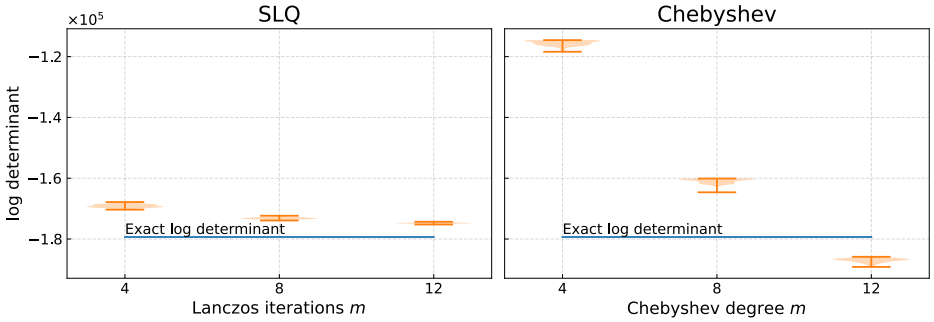

Figure 5: LeNet counterpart of Fig. 2.

B On Computational Complexities

We shortly explain computational complexities in Table 1. Block-wise cubic time complexity of K-FAC ($O(\sum_l P_l^3)$) stems from the need for the exact log determinant computation for each block. The computation of Chebyshev and SLQ is dominated by m times of matrix-vector products of GGN matrices and probe vectors, each of which costs $O(P)$.

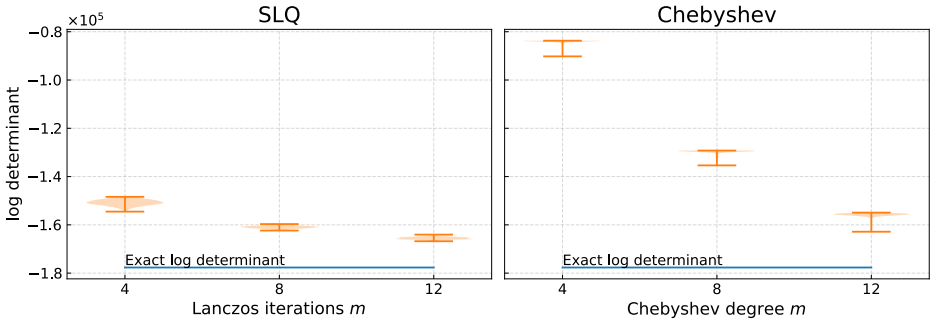

Figure 6: LeNet counterpart of Fig. 4.

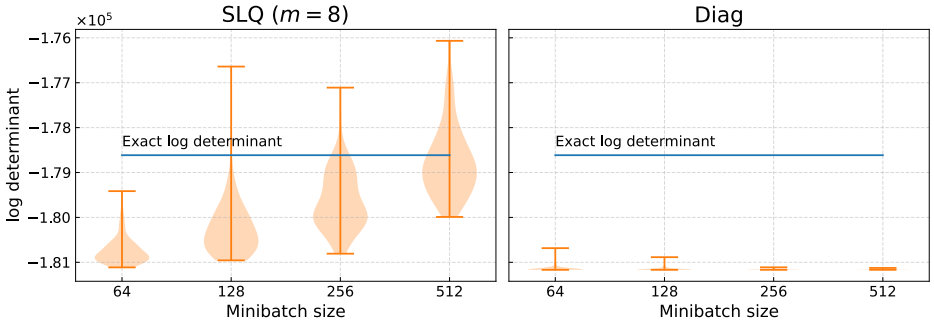

Figure 7: LeNet counterpart of Fig. 3.