Neural Networks 145 (2022) 164–176

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Epistemic uncertainty quantification in deep learning classification by the Delta method

Geir K. Nilsen *, Antonella Z. Munthe-Kaas, Hans J. Skaug, Morten Brun

Department of Mathematics, University of Bergen, Norway

ARTICLE INFO

Article history:
Received 28 February 2021
Received in revised form 18 August 2021
Accepted 18 October 2021
Available online 23 October 2021

Keywords:
Uncertainty quantification
Predictive epistemic uncertainty
Neural networks
Deep learning
Hessian
Fisher information

ABSTRACT

The Delta method is a classical procedure for quantifying epistemic uncertainty in statistical models, but its direct application to deep neural networks is prevented by the large number of parameters P. We propose a low cost approximation of the Delta method applicable to L_2-regularized deep neural networks based on the top K eigenpairs of the Fisher information matrix. We address efficient computation of full-rank approximate eigendecompositions in terms of the exact inverse Hessian, the inverse outer-products of gradients approximation and the so-called Sandwich estimator. Moreover, we provide bounds on the approximation error for the uncertainty of the predictive class probabilities. We show that when the smallest computed eigenvalue of the Fisher information matrix is near the L_2-regularization rate, the approximation error will be close to zero even when $K \ll P$. A demonstration of the methodology is presented using a TensorFlow implementation, and we show that meaningful rankings of images based on predictive uncertainty can be obtained for two LeNet and ResNet-based neural networks using the MNIST and CIFAR-10 datasets. Further, we observe that false positives have on average a higher predictive epistemic uncertainty than true positives. This suggests that there is supplementing information in the uncertainty measure not captured by the classification alone.

1. Introduction

The predictive probabilities at the output layer of neural network classifiers are often misinterpreted as model (epistemic) uncertainty (Gal & Ghahramani, 2016). Bayesian statistics provides a coherent framework for representing uncertainty in neural networks (Goodfellow, Bengio, & Courville, 2016; MacKay, 1992), but has not so far gained widespread use in deep learning — presumably due to the high computational cost that traditionally comes with second-order methods. Recently, Gal and Ghahramani (2016) developed a theoretical framework which casts dropout at test time in deep neural networks as approximate Bayesian inference. Due to its mathematical elegance and negligible computational cost, this work has caught great interest in a variety of different fields (Litjens et al., 2017; Loquercio, Segu, & Scaramuzza, 2020; Yan, Gong, Wei, & Gao, 2020; Zhu & Laptev, 2017), but has also generated questions as to what types of uncertainty these approximations actually lead (Osband, 2016; Osband, Blundell, Pritzel, & Roy, 2016) and what types are relevant (Kendall & Gal, 2017). For a general treatment of uncertainty in machine learning, we refer to Hüllermeier and Waegeman (2020).

Epistemic uncertainty is commonly understood as the reducible component of uncertainty — the uncertainty of the model itself, or its parameters. In our context this amounts to the uncertainty in the estimated class probabilities due to limited amount of training data. While the epistemic uncertainty can be reduced by increasing the amount of training data, the other component of uncertainty known as aleatoric uncertainty, is irreducible and stems from the uncertainty in the label assignment process (Song, Kim, Park, & Lee, 2020). However, in this paper we only address the epistemic part, and treat the labels as constant when estimating uncertainty.

Our approach goes back to the work of MacKay (1992), and we show that the above reasoning leads to the method known as the Delta method[1] (Hoef, 2012; Khosravi & Creighton, 2011; Newey & McFadden, 1994) in statistics. However, as the Delta method depends on the empirical Fisher information matrix which grows quadratically with the number of neural network parameters P – its direct application in modern deep learning is prohibitively expensive. We therefore propose a low cost variant of the Delta method applicable to L_2-regularized deep neural networks based on the top K eigenpairs of the Fisher information matrix. We

* Corresponding author.
E-mail addresses: geir.kjetil.nilsen@gmail.com, geir.nilsen@uib.no (G.K. Nilsen), antonella.zanna@uib.no (A.Z. Munthe-Kaas), hans.skaug@uib.no (H.J. Skaug), morten.brun@uib.no (M. Brun).

[1] Also known as the Laplace approximation.

https://doi.org/10.1016/j.neunet.2021.10.014

address efficient computation of full-rank approximate eigendecompositions in terms of either the exact inverse Hessian, the inverse outer-products of gradients (OPG) approximation or the so-called Sandwich estimator. Further, we exhibit the fact that deep learning classifiers tend to be heavily over-parameterized. This leads to flat Fisher information eigenvalue spectra which we show can be exploited in terms of a simple linearization.

Another classical epistemic uncertainty quantification procedure is the Bootstrap (Efron, 1979; Khosravi & Creighton, 2011). A comparison of the Delta methodology presented in this paper and the classical Bootstrap procedure applied to deep learning classification can be found in Nilsen, Munthe-Kaas, Skaug, and Brun (2021).

The theoretical Fisher information matrix is always positive (semi)-definite, and we constrain our empirical counterpart to be the same. Recent research (Alain, Roux, & Manzagol, 2019; Ghorbani, Krishnan, & Xiao, 2019; Sagun, Bottou, & LeCun, 2017; Sagun, Evci, Guney, Dauphin, & Bottou, 2018), consistent with our own observations, show that the exact Hessian after training is rarely positive definite in deep learning. To mitigate this, we propose a simple correction of the right tail of the Hessian eigenvalue spectrum to achieve positive definiteness. We corroborate our choice with two observations: a) negative eigenvalues of the Hessian matrix are highly stochastic across different weight initialization values, and b) correcting the eigenvalue spectrum to achieve positive definiteness yields stable predictive epistemic uncertainty estimates which are perfectly correlated with the estimates based on the OPG approximation — which by construction is always positive (semi)-definite (Martens, 2020).

As the computational cost of the exact inverse Hessian matrix or its full eigendecomposition is prohibitively expensive in deep learning, we propose to use the Lanczos iteration (Trefethen & III, 1997) in combination with Pearlmutter's technique (Pearlmutter, 1994) to compute the needed eigenpairs. Consequently, the matrix inversion will be straightforward, and the net computational complexity will be $O(SPN)$ time and $O(KP)$ space, where N is the number of training examples and S is the number of Lanczos–Pearlmutter steps required to compute K eigenpairs.

Also the inverse OPG approximation or its full eigendecomposition is prohibitively costly in deep learning. Even if we disregard the cubic time inversion and the quadratic space complexity, one is first left to compute and store the $N \times P$-dimensional Jacobian matrix. In deep learning software provisions based on backward-mode automatic differentiation, only the sum of mini-batch gradients can be computed efficiently. We therefore propose to compute mini-batches of the Jacobian using efficient per-example gradients (Nilsen, Munthe-Kaas, Skaug, & Brun, 2019) in combination with incremental singular value decompositions (Levy & Lindenbaum, 2000). Since the OPG approximation can be written as a Jacobian matrix product, its eigenvectors will be the right singular vectors of the Jacobian, and its eigenvalues the squared singular values. This leads to a computational complexity of $O(KPN)$ time and $O(KP)$ space, also accounting for the inversion. The Sandwich estimator requires both the inverse Hessian and the OPG approximation, and is thus $O(max\{K, S\}PN)$ time and $O(KP)$ space.

This work is a continuation of Nilsen et al. (2019), and we here introduce the fully deterministic (Nagarajan & Warnell, 2019) open sourced TensorFlow module `pydeepdelta` (pyDeepDelta, 2018-2021), and illustrate the methodology on two LeNet and ResNet-based convolutional neural network classifiers using the MNIST and CIFAR-10 datasets. The main contributions of the paper can be summarized as follows:

- We recognize the Delta method as a measure of epistemic as opposed to aleatoric uncertainty and break it into two components: the eigenvalue spectrum of the Fisher information (i.e. Hessian) of the cost function and the per-example sensitivities (i.e. gradients) of the model function.
- We show how to approximate the naïve Delta method and thereby reducing the computational complexity in P from quadratic in space and cubic in time, to linear in both space and time. Bounds of the approximation error are provided.
- We provide an accompanying TensorFlow implementation, and demonstrate how it can be applied on a few well known architectures using the MNIST and CIFAR-10 datasets.

The paper is organized as follows: In Section 2 we give definitions which will be used throughout the paper. In Section 3 we review the Delta method in a deep learning classification context, and in Section 4 we outline the details of the proposed methodology. In Sections 5 and 6 we demonstrate the method, and finally, in Section 7 we summarize the paper and give some concluding remarks and ideas of future work.

2. Deep neural networks

We use a feed-forward neural network architecture with dense layers to introduce terminology and symbols, but emphasize that the theory presented in the paper is directly applicable to any L_2-regularized architecture.

2.1. Architectural

A feed-forward neural network is shown in Fig. 1. There are L layers $l = 1, 2, \ldots, L$ with T_l neurons in each layer. The input layer $l = 1$, is represented by the input vector $x_n = \begin{pmatrix} x_{n,1} & x_{n,2} & \ldots & x_{n,T_1} \end{pmatrix}^T$ where $n = 1, 2, \ldots, N$ is the input index. Furthermore, there are $L - 2$ dense hidden layers, $l = 2, 3, \ldots, L-1$, and a dense output layer $l = L$, each represented by weight matrices $W^{(l-1)} \in \mathbb{R}^{T_l \times T_{l-1}}$, bias vectors $b^{(l)} \in \mathbb{R}^{T_l}$ and vectorized activation functions $a^{(l)}$.

2.2. Parameter vectors

The total number of parameters in the model shown in Fig. 1 can be written,

$$P = \sum_{l=2}^{L} P^{(l)} = \sum_{l=2}^{L} T_{l-1} T_l + T_l, \tag{1}$$

where $P^{(l)}$ denotes the number of parameters in layer l. By definition, $P^{(1)} = 0$ since the input layer contains no weights or biases. Furthermore, we define parameter vectors representing the layer-wise weights and biases as follows,

$$\omega^{(l)} = \begin{bmatrix} \mathrm{vec}(W^{(l)}) \\ b^{(l)} \end{bmatrix} \in \mathbb{R}^{P^{(l)}}, \tag{2}$$

for $l = 2, 3, \ldots, L$, with components $\omega_i^{(l)}$, $i = P^{(l-1)} + 1, P^{(l-1)} + 2, \ldots, P^{(l)}$. The notation $\mathrm{vec}(W)$ denotes a row-wise vectorization[2] of the matrix $W^{A \times B}$ into a column vector of dimension $\mathbb{R}^{AB}$. In the rest of the paper, we consider the full model and define the parameter vector,

$$\omega = \begin{bmatrix} \omega^{(2)} \\ \omega^{(3)} \\ \vdots \\ \omega^{(L)} \end{bmatrix} \in \mathbb{R}^P. \tag{3}$$

[2] Standard method in TensorFlow: tf.reshape(W, [−1]).

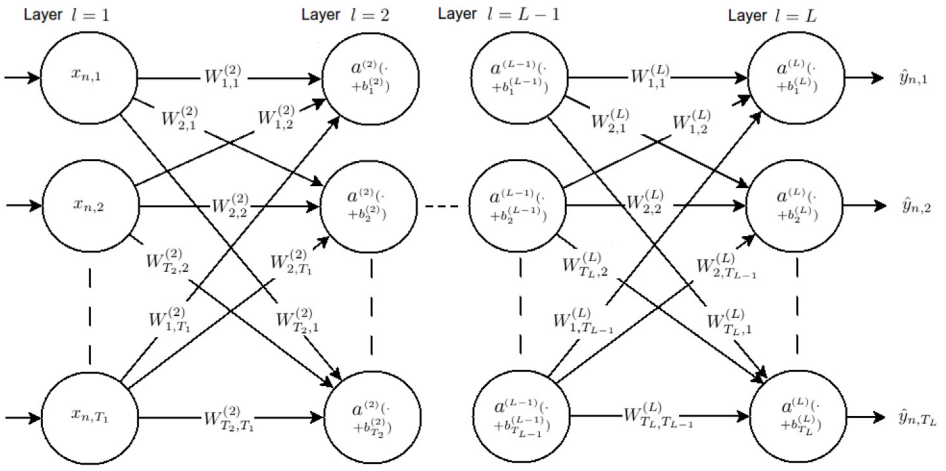


Fig. 1. A feed-forward neural network with dense layers.

2.3. Training, model and cost function

The *model function* $f : \mathbb{R}^{T_1 \times P} \to \mathbb{R}^{T_L}$ associated to the architecture shown in Fig. 1 is defined as

$$f(x_n, \omega) = a^{(L)}[W^{(L)} a^{(L-1)}(\cdots a^{(2)}\{W^{(2)} x_n + b^{(2)}\} + \cdots) + b^{(L)}]. \quad (4)$$

We use a softmax cross-entropy *cost function* $C : \mathbb{R}^P \to \mathbb{R}$ and require L_2-regularization with a rate factor $\lambda > 0$,

$$\begin{aligned} C(\omega) &= \frac{1}{N}\sum_{n=1}^{N} C_n(y_n, \hat{y}_n) + \frac{\lambda}{2}\sum_{p=1}^{P}\omega_p^2 \\ &= \frac{1}{N}\sum_{n=1}^{N}\left(-\sum_{m=1}^{T_L} y_{n,m}\log \hat{y}_{n,m}\right) + \frac{\lambda}{2}\sum_{p=1}^{P}\omega_p^2, \end{aligned} \quad (5)$$

where y_n represents the target vector for the nth example (N examples), and where $\hat{y}_n = f(x_n, \omega)$ represents the corresponding prediction vector obtained by evaluating the *model function* (4) using the input vector x_n and the parameter vector (3). The activation function $a^{(L)} : \mathbb{R}^{T_L} \to \mathbb{R}^{T_L}$ in the output layer is the vectorized softmax function defined as

$$\begin{aligned} a^{(L)}(z) &= \text{softmax}(z) \\ &= \frac{\exp(z)}{\sum_{m=1}^{T_L}\exp(z_m)}, \end{aligned} \quad (6)$$

where $\exp(\cdot)$ denotes the vectorized exponential function. Training of the neural network can be defined as finding an 'optimal' parameter vector $\hat{\omega}$ by minimizing the cost function (5),

$$\hat{\omega} = \arg\min_{\omega \in \mathbb{R}^P} C(\omega). \quad (7)$$

3. The delta method

The Delta method (Hoef, 2012) views a modern deep neural network as a (huge) non-linear regression. In our classification setting, we regard the labels as constant, and thus the epistemic component of the uncertainty associated with predictions of an arbitrary input example x_0 reduces to the evaluation of the covariance matrix of the network outputs (Khosravi & Creighton, 2011). By a first-order Taylor expansion (Grosse, 2020), it can be shown that the covariance matrix of the network outputs $\hat{y}_0$, i.e. the model function (4), can be approximated by

$$\text{Cov}(\hat{y}_0) \approx F \Sigma F^T \in \mathbb{R}^{T_L \times T_L}, \quad (8)$$

where

$$F = \left[F_{ij}\right] \in \mathbb{R}^{T_L \times P}, \; F_{ij} = \frac{\partial}{\partial \omega_j} f_i(x_0, \omega)\Big|_{\omega=\hat{\omega}} \quad (9)$$

is the Jacobian matrix of the model function, and where Σ is the covariance matrix of the model parameter estimate $\hat{\omega}$. For a given x_0, an approximate standard deviation of $\hat{y}_0$ is provided by the formula

$$\sigma(x_0) \approx \sqrt{\text{diag}\left(F \Sigma F^T\right)} \in \mathbb{R}^{T_L}. \quad (10)$$

Eq. (10) means that when the neural network predicts for an input x_0, the associated epistemic uncertainty per class output is determined by a linear combination of parameter sensitivity (i.e. F) and parameter uncertainty (i.e. Σ). Parameter sensitivity (F) prescribes the amount of change in the neural network output for an infinitesimal change in the parameter estimates, whereas the parameter uncertainty (Σ) prescribes the amount of uncertainty in the parameter estimates themselves.

We apply and compare three different approximations to Σ. The first one is called the **Hessian estimator**, and is defined by

$$\Sigma^{\text{H}} = \frac{1}{N}H^{-1} = \frac{1}{N}\left[\frac{1}{N}\sum_{n=1}^{N}\frac{\partial^2 C_n}{\partial\omega\partial\omega^T}\Big|_{\omega=\hat{\omega}} + \lambda I\right]^{-1} \in \mathbb{R}^{P\times P}, \quad (11)$$

where H is the empirical Hessian matrix of the cost function evaluated at $\hat{\omega}$.

The second estimator is called the **Outer-Products of Gradients (OPG) estimator** and is defined by

$$\Sigma^{\text{G}} = \frac{1}{N}G^{-1} = \frac{1}{N}\left[\frac{1}{N}\sum_{n=1}^{N}\frac{\partial C_n}{\partial\omega}\frac{\partial C_n}{\partial\omega}^T\Big|_{\omega=\hat{\omega}} + \lambda I\right]^{-1} \in \mathbb{R}^{P\times P}, \quad (12)$$

where the summation part of G corresponds to the empirical covariance of the gradients of the cost function evaluated at $\hat{\omega}$. Finally, the third estimator is known as the **Sandwich estimator** (Freedman, 2006; Schulam & Saria, 2019) and is defined by

$$\Sigma^{\text{S}} = \frac{1}{N}H^{-1}GH^{-1} \in \mathbb{R}^{P\times P}. \quad (13)$$

Across various fields and contexts, the two famous Eqs. (11) and (12) are often presented and interpreted differently, and the inconsistency in the vast literature is nothing but intriguing. We therefore feel that their appearance in this paper requires some elaboration. Firstly, for the Hessian estimator (11), we note that the differentials act only on the data dependent part of the cost function (5), C_n, so the second term, λI, here comes from the

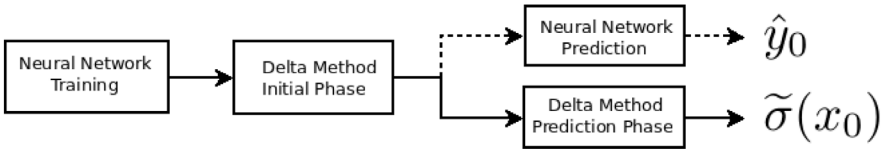


Fig. 2. The Delta method for quantifying the predictive epistemic uncertainty $\widetilde{\sigma}(x_0)$ of $\hat{y}_0 = f(x_0, \hat{w})$ in deep learning (solid line).

second-order derivatives of the L_2-regularization term. Secondly, for the OPG estimator (12), also here the differentials act on the data dependent part of the cost function, but the crucial detail often confused or let out in the literature comes with the second term, λI: under L_2-regularization it must be added explicitly in order for G to be asymptotically equal to H (See Appendix for a proof) – as is the primary motivation of the OPG estimator as a plug-in replacement of the Hessian estimator in the first place. If let out, G will almost always be singular (Murfet et al., 2020; Watanabe, 2007), and thus cannot be used in (12).

At this point, we can see that two fundamental difficulties arise when applying the Delta method in deep learning: (a) the sheer size of the covariance matrix grows quadratically with P, and (2) the covariance matrix must be positive definite. In other words, we are virtually forced to compute and store the full covariance matrix, and are in terms of the Hessian estimator dependent on that the optimizer can find a true local (or global) minimum of the cost function. Nevertheless, with the OPG and the Sandwich estimators, the second obstacle is virtually inapplicable since they by definition always will be positive definite when $\lambda > 0$.

In the next section we present methodology that addresses both these aspects. We present an indirect correction leaving the Hessian estimator positive definite, and introduce methodology with computational time and space complexity which is linear in P.

4. The delta method in deep learning

We present our approach to the Delta method in deep learning as a procedure carried out in two phases after the neural network has been trained. See Fig. 2.

The first phase – the 'initial phase' – is carried out only once, with the scope of indirectly computing full-rank, positive definite approximations of the covariance matrices (11), (12) or (13) based on approximate eigendecompositions of H and G. The second phase – the 'prediction phase' – is carried out hand in hand with the regular neural network prediction process (4), and is used to approximate the epistemic component of the predictive uncertainty governed by (10) using the indirect covariance matrix approximation found in the 'initial phase'.

In the next sections, we address the following aspects of the proposed methods: (a) how to efficiently compute eigenvalues and eigenvectors of the Hessian estimator via the Lanczos iteration and exact Hessian vector products, (b) how to efficiently compute eigenvalues and eigenvectors of the OPG estimator via incremental singular value decompositions, (c) how to combine the former two to obtain an approximation of the Sandwich estimator, and (d) how to apply these estimators to efficiently compute an approximation of (10).

4.1. Computing eigenvalues and eigenvectors of the covariance matrix

The full eigendecomposition of the covariance matrix in (10) is defined by

$$\Sigma = Q \Lambda^{-1} Q^T \in \mathbb{R}^{P \times P}, \tag{14}$$

Table 1
The computational complexity of the outlined methodology is linear in P across both phases.

	Initial phase		Prediction phase (Per-Example)	
	Time	Space	Time	Space
Hessian	$O(SPN)$			
OPG	$O(KPN)$	$O(KP)$	$O(T_L PK + T_L^2 K + K^2 T_L)$	$O(max\{K, T_L\}P)$
Sandwich	$O(max\{K, S\}PN)$			

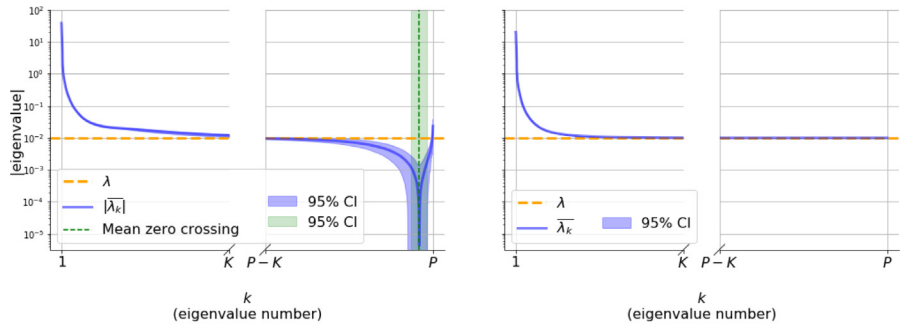
where $Q \in \mathbb{R}^{P \times P}$ is the matrix whose kth column is the eigenvector q_k of Σ, and $\Lambda \in \mathbb{R}^{P \times P}$ is the diagonal matrix whose elements are the corresponding eigenvalues, $\Lambda_{kk} = \lambda_k$. We assume that the eigenvalues are algebraically sorted so that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_P$. Note that in terms of the Hessian estimator, the eigenvalues are precisely the second derivatives of the cost function along the principal axes of the ellipsoids of equal cost, and that Q is a rotation matrix which defines the directions of these principal axes (LeCun, Simard, & Pearlmutter, 1993).

For the Hessian estimator (11), the Lanczos iteration (Trefethen & III, 1997) can be applied to find $K < P$ eigenvalues (and corresponding eigenvectors) in $O(SNP)$ time and $O(KP)$ space when Pearlmutter's technique (Pearlmutter, 1994) is applied inside the iteration (Nilsen et al., 2019). Pearlmutter's technique can simply be described as a procedure based on two-pass back-propagations of complexity $O(NP)$ time and $O(P)$ space to obtain exact Hessian vector products without requiring to keep the full Hessian matrix in memory. The number S denotes the number of Lanczos iterations to reach convergence. We observe that the convergence of the Lanczos algorithm is quite fast in our experiments, and we find that S is practically orders of magnitude less than P.

For the OPG estimator (12), a slightly different approach can be applied. Since the OPG estimator can be written as a Jacobian matrix product (Nilsen et al., 2019), we get by the singular value decomposition that its eigenvectors will be the right singular vectors of the Jacobian, and its eigenvalues the squared singular values. Mini-batches of the Jacobian matrix can easily be obtained by standard back-propagation, and so an incremental singular value decomposition (Cardot & Degras, 2015; Levy & Lindenbaum, 2000) can be applied to each mini-batch. The computational cost is thus $O(KNP)$ time and $O(KP)$ space. The Sandwich estimator combines the Hessian and the OPG approximation via the product (13), and thus has a computational complexity of $O(max\{K, S\}NP)$ time and $O(KP)$ space. The computational complexity of the outlined methodology is summarized in Table 1.[3]

Our TensorFlow module `pydeepdelta` (pyDeepDelta, 2018-2021) utilizes the Lanczos implementation available in the SciPy distribution (SciPy), as well as the incremental singular value decomposition available in the scikit-learn distribution (scikit-learn).

[3] Assuming naive matrix multiplication.



(a) H (b) G

Fig. 3. Log scale eigenvalue magnitude spectra of H and G showing the $K = 1500$ largest (left tail subspace) and the $K = 1500$ smallest (right tail subspace) eigenvalues and their variation across sixteen trained instances of the MNIST network distinguished only by a different random weight initialization. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4.2. The eigenvalue spectra of H and G

To better understand the proposed covariance approximations, we first need to explore the prototypical deep learning eigenvalue spectrum of the empirical Hessian matrix H (11) and the empirical covariance of the gradients G (12). To this end, we introduce two LeNet-based convolutional neural network classifiers using the MNIST and CIFAR-10 datasets, and draw parallels to the findings in the literature.

4.2.1. Classifier architectures, parameters and training

The MNIST classifier has $L = 6$ layers, layer $l = 1$ is the input layer represented by the input vector. Layer $l = 2$ is a $3 \times 3 \times 1 \times 32$ convolutional layer followed by max pooling with stride equal to 2 and with a ReLU activation function. Layer $l = 3$ is a $3 \times 3 \times 32 \times 64$ convolutional layer followed by max pooling with a stride equal to 2, and with ReLU activation function. Layer $l = 4$ is a $3 \times 3 \times 64 \times 64$ convolutional layer with ReLU activation function. Layer $l = 5$ is a 576×64 dense layer with ReLU activation function, and the output layer $l = 6$ is a $64 \times T_L$ dense layer with softmax activation function, where the number of classes (outputs) is $T_L = 10$. The total number of parameters is $P = 93{,}322$.

The CIFAR-10 classifier has $L = 6$ layers, layer $l = 1$ is the input layer represented by the input vector. Layer $l = 2$ is a $3 \times 3 \times 3 \times 32$ convolutional layer followed by max pooling with stride equal to 2 and with a ReLU activation function. Layer $l = 3$ is a $3 \times 3 \times 32 \times 64$ convolutional layer followed by max pooling with a stride equal to 2, and with ReLU activation function. Layer $l = 4$ is a $3 \times 3 \times 64 \times 64$ convolutional layer with ReLU activation function. Layer $l = 5$ is a 1024×64 dense layer with ReLU activation function, and the output layer $l = 6$ is a 64×10 dense layer with softmax activation function, where the number of classes (outputs) is $T_L = 10$. The total number of parameters is $P = 122{,}570$.

We apply random normal weight initialization and zero bias initialization. We use (5) as the cost function with a L_2-regularization rate $\lambda = 0.01$. We utilize the Adam optimizer (Bottou, Curtis, & Nocedal, 2018; Kingma & Ba, 2014) with a batch size of 100, and apply no form of randomized data shuffling. To ensure convergence (i.e. $\|\nabla C(\hat{\omega})\|_2 \approx 0$) we apply the following learning rate schedules given by the following (step, rate) pairs: MNIST $= \{(0, 10^{-3}), (60\text{k}, 10^{-4}), (70\text{k}, 10^{-5}), (80\text{k}, 10^{-6})\}$ and CIFAR-10 $= \{(0, 10^{-3}), (55\text{k}, 10^{-4}), (85\text{k}, 10^{-5}), (95\text{k}, 10^{-6}), (105\text{k}, 10^{-7})\}$. For MNIST, we stop the training after 90,000 steps – corresponding to a training accuracy of 0.979, test accuracy 0.981, training cost $C(\hat{\omega}) = 0.257$ and a gradient norm $\|\nabla C(\hat{\omega})\|_2 = 0.016$. For CIFAR-10, we stop the training after 115,000 steps – corresponding to a training accuracy of 0.701, test accuracy 0.687, training cost $C(\hat{\omega}) = 1.284$ and a gradient norm $\|\nabla C(\hat{\omega})\|_2 = 0.030$.

4.2.2. The eigenvalue spectrum approximation

The general assumption in deep learning is that H after training is not positive definite and mostly contain eigenvalues close to zero (Alain et al., 2019; Ghorbani et al., 2019; Granziol et al., 2019; Sagun et al., 2017, 2018; Watanabe, 2007). The same holds true for G although it by definition must at least be positive semi-definite (Martens, 2020). However, given the discussion in Section 3, we know that L_2-regularization with rate $\lambda/2$ has the effect of shifting the eigenvalues of H and G upwards by λ.

To test this hypothesis, we study the $K = 1500$ algebraically largest and the $K = 1500$ algebraically smallest eigenvalues of H and G for 16 trained instances of the MNIST network defined in Section 4.2.1. These sixteen networks are thus only distinguished from each other by a different random weight initialization prior to training. The two corresponding log-scale eigenvalue magnitude spectra are shown in Fig. 3.

Firstly, we note that in the midpoint gaps of the spectra, there are $P - 2K = 90{,}195$ 'missing' central eigenvalues which we have not computed. Since the eigenvalues are sorted in decreasing order, all the central eigenvalues must be close to the L_2-regularization rate λ. We refer to this part of the eigenvalue spectrum as the gap. Secondly, we note that the confidence intervals in the plots are taken across instance space, thus telling how the eigenvalue spectrum change based on the 16 random weight initializations. In both plots, the blue confidence interval tells that the largest eigenvalues of H and G (called left tail) are stable across the 16 trained networks, but the smallest eigenvalues of H are changing dramatically (called right tail, left plot). On the contrary, all the eigenvalues of G are stable. Thirdly, as shown by the green vertical dotted line in the upper plot representing the mean zero-crossing, H is clearly not positive definite — even with L_2-regularization. The green confidence interval around the zero-crossing shows that the number of negative eigenvalues also change across the networks.

In Granziol et al. (2019) it was hypothesized that negative Hessian eigenvalues are caused by a discrepancy between the empirical Hessian (i.e. H) and its theoretical counterpart (expected Hessian) in which the summation of (11) is replaced with an expectation so that effectively $N \to \infty$. They showed that as N grows (holding $\hat{\omega}$ fixed), the empirical right tail grows toward λ whereas the rest of the spectrum is stable. Supported by the fact that H and G will be equal in expectation (Appendix), the expected Hessian eigenvalue spectrum might be more similar to that of G where all the eigenvalues are greater than equal to λ.

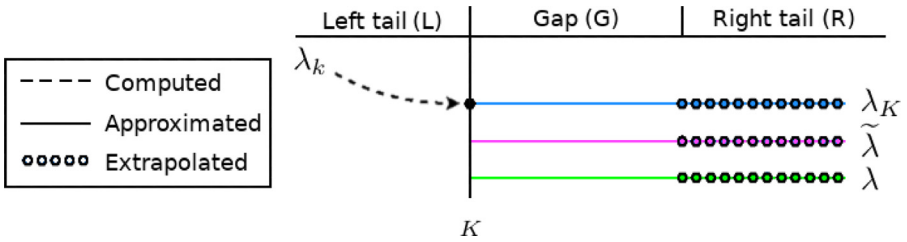


Fig. 4. In terms of its eigenvalue spectrum, the covariance matrix can be partitioned as given by Eq. (15): the left tail subspace (eigenpairs computed), the gap subspace (eigenvalues approximated, eigenvectors implicitly found by orthonormality) and the right tail subspace (eigenvalues extrapolated, eigenvectors implicitly found using orthonormality). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

In line with these ideas and the empirical evidence presented in Fig. 3, we assume that all the smallest eigenvalues of H in the right tail are inherently noisy, and should not be used by the Hessian estimator. Therefore, with reference to Fig. 4, for the Hessian estimator, we (a) calculate all the eigenpairs in the left tail, (b) approximate all the eigenvalues in the gap and (c) extrapolate the eigenvalues from the gap into the right tail. The eigenvectors corresponding to the gap and right tail can implicitly be accounted for by orthonormality as discussed in the next section.

For the OPG estimator, the same principle applies apart from that the extrapolation inherently becomes a part of the gap subspace approximation because we know that G always is positive definite when $\lambda > 0$. Finally, for the Sandwich estimator, we simply apply the aforementioned procedures and estimate the product (13).

4.3. Closing the gap

Based on the observations in the previous section, we now propose a partitioning of the eigendecomposition which reveals that full-rank, positive definite approximations of the Hessian and OPG estimators can be obtained by computing only the eigenpairs corresponding to the K algebraically largest eigenvalues of H and G respectively. Finally, we show how to use these approximations to construct an approximation of the Sandwich estimator.

4.3.1. The Hessian and OPG estimators

In terms of the Hessian and OPG estimators, the full eigendecomposition of the covariance matrix can be partitioned into three subspaces as shown in Fig. 4

$$\Sigma = \Sigma_{\mathrm{L}} + \Sigma_{\mathrm{G}} + \Sigma_{\mathrm{R}} = Q_{\mathrm{L}} \Lambda_{\mathrm{L}}^{-1} Q_{\mathrm{L}}^T + Q_{\mathrm{G}} \Lambda_{\mathrm{G}}^{-1} Q_{\mathrm{G}}^T + Q_{\mathrm{R}} \Lambda_{\mathrm{R}}^{-1} Q_{\mathrm{R}}^T. \quad (15)$$

This decomposition applies to both Σ^{H} (11) and Σ^{G} (12), and thus we have omitted the superscripts in our notation. In practice, the two merely differs by which of the two matrices H and G the calculated eigenpairs come from. The subscript 'G' denotes the gap subspace which is based on eigenvectors with eigenvalues λ_{K+1} to λ_{P-K-1}. Subscript 'L' denotes the left tail subspace and is based on eigenvectors with eigenvalues λ_1 to λ_K. Finally, the subscript 'R' denotes the right tail subspace which is based on eigenvectors with eigenvalues λ_{P-K} to λ_P. Accordingly, we have that $Q_{\mathrm{L}} \in \mathbb{R}^{P\times K}$, $\Lambda_{\mathrm{L}} \in \mathbb{R}^{K\times K}$, $Q_{\mathrm{G}} \in \mathbb{R}^{P\times(P-2K)}$, $\Lambda_{\mathrm{G}} \in \mathbb{R}^{(P-2K)\times(P-2K)}$, $Q_{\mathrm{R}} \in \mathbb{R}^{P\times K}$ and $\Lambda_{\mathrm{R}} \in \mathbb{R}^{P\times K}$.

If $\lambda_K \approx \lambda$ we can safely assume that all the eigenvalues in the gap subspace must be close to λ. In line with (Granziol et al., 2019) and the empirical evidence presented in Fig. 3, we assume that all the eigenvalues in the right subspace are inherently noisy, and should not be used by the Hessian estimator. Consequently, we assume that also the eigenvalues in the right subspace are approximately equal to λ. Since the OPG estimator is always positive definite when $\lambda > 0$, the same assumption also holds true.

With reference to Fig. 4, there are now two possible extreme conditions: (a) when all the eigenvalues in the gap and right subspaces are set to λ_K (blue), or (b) when all the eigenvalues in the gap and right subspaces are set to λ (green). By defining $\widetilde{\lambda}$ (purple) as the harmonic mean of λ and λ_K, and ϵ_λ as the midpoint of their reciprocals,

$$\widetilde{\lambda} = \left(\frac{\lambda^{-1} + \lambda_K^{-1}}{2}\right)^{-1} \quad \text{and} \quad \epsilon_\lambda = \frac{\lambda^{-1} - \lambda_K^{-1}}{2}, \quad (16)$$

it follows that $\widetilde{\lambda}^{-1} \pm \epsilon_\lambda$ will enclose the interval $[\lambda_K^{-1}, \lambda^{-1}]$. The covariance matrix can now be approximated by

$$\widetilde{\Sigma} = \frac{1}{N}\left[Q_{\mathrm{L}} \Lambda_{\mathrm{L}}^{-1} Q_{\mathrm{L}}^T + \widetilde{\lambda}^{-1}(Q_{\mathrm{G}} Q_{\mathrm{G}}^T + Q_{\mathrm{R}} Q_{\mathrm{R}}^T)\right], \quad (17)$$

with a worst-case approximation error Δ given by

$$\Delta = \frac{\epsilon_\lambda}{N}\left[Q_{\mathrm{G}} Q_{\mathrm{G}}^T + Q_{\mathrm{R}} Q_{\mathrm{R}}^T\right], \quad (18)$$

such that Σ is bounded by $\widetilde{\Sigma} \pm \Delta$. Since Q is an orthonormal basis, we see that it is possible to express (17) and (18) without an explicit need to compute any of the eigenvectors relative to the gap nor right tail subspaces because

$$Q_{\mathrm{G}} Q_{\mathrm{G}}^T + Q_{\mathrm{R}} Q_{\mathrm{R}}^T = I - Q_{\mathrm{L}} Q_{\mathrm{L}}^T. \quad (19)$$

Inserting (17) into (10) with use of (19), yields the final form of the approximation to the uncertainty associated with prediction of x_0

$$\widetilde{\sigma}^2(x_0) = \frac{1}{N}\mathrm{diag}\left\{F\left[Q_{\mathrm{L}} \Lambda_{\mathrm{L}}^{-1} Q_{\mathrm{L}}^T + \widetilde{\lambda}^{-1}(I - Q_{\mathrm{L}} Q_{\mathrm{L}}^T)\right] F^T\right\} \in \mathbb{R}^{T_L}, \quad (20)$$

with a worst-case approximation error δ given by

$$\delta = \frac{\epsilon_\lambda}{N}\mathrm{diag}\left\{F\left(I - Q_{\mathrm{L}} Q_{\mathrm{L}}^T\right) F^T\right\} \in \mathbb{R}^{T_L}, \quad (21)$$

such that $\sigma^2(x_0)$ is bounded by $\widetilde{\sigma}^2(x_0) \pm \delta$.

In terms of standard deviations, the worst-case approximation error ϵ of $\widetilde{\sigma}(x_0)$ is given by

$$\epsilon = \frac{1}{2}\left(\sqrt{\widetilde{\sigma}^2(x_0) + \delta} - \sqrt{\widetilde{\sigma}^2(x_0) - \delta}\right) \in \mathbb{R}^{T_L}, \quad (22)$$

such that $\sigma(x_0)$ is bounded by $\widetilde{\sigma}(x_0) \pm \epsilon$. Lastly, we define an 'uncertainty score' (which we will use later to rank images) by summing the variances per class output (class variance), and then take the square root to get the total uncertainty in standard deviations

$$\widetilde{\sigma}_{\mathrm{score}}(x_0) = \sqrt{\sum_{m=1}^{T_L} \widetilde{\sigma}_m^2(x_0)} \in \mathbb{R}, \quad (23)$$

with the corresponding worst-case approximation error $\epsilon_{\mathrm{score}}$ given by,

$$\epsilon_{\mathrm{score}} = \frac{1}{2}\left(\sqrt{\sum_{m=1}^{T_L} \widetilde{\sigma}_m^2(x_0) + \delta_m} - \sqrt{\sum_{m=1}^{T_L} \widetilde{\sigma}_m^2(x_0) - \delta_m}\right) \in \mathbb{R}, \quad (24)$$

such that the true quantity is bounded by $\widetilde{\sigma}_{\text{score}}(x_0) \pm \epsilon_{\text{score}}$. We note that the worst-case approximation errors (21), (22) and (24) are functions of x_0 but we have notationally dropped this from the equations to avoid cluttering. The approximation errors should be thought of as an uncertainty of the predictive uncertainty which accounts for the worst-case loss of not computing the gap subspace explicitly. Since the right tail subspace can be extrapolated when H is not positive definite, the concept of an approximation error for the Hessian estimator must be used carefully.

At this point we make a few comments regarding Eq. (20). The first term on the right hand side, $Q_{\text{L}}\Lambda_{\text{L}}^{-1}Q_{\text{L}}^T$, corresponds to a low-rank approximation of the covariance matrix based on K explicitly computed principal eigenpairs. However, when the second term, $\widetilde{\lambda}^{-1}(I - Q_{\text{L}}Q_{\text{L}}^T)$, is added — the approximation becomes full-rank. When accounting for the left and right multiplication of the sensitivity matrix F, the per-class predictive uncertainties of x_0 can be interpreted as weighted sums of the squared sensitivities in the directions expressed by the eigenbasis Q using the inverse eigenvalues as weights. Hence, for the low-rank approximation – regardless of the sensitivity – the contribution to the predictive uncertainty will be zero in directions $k > K$, whereas for the full-rank approximation — the contribution can still be high. We will come back to this when we discuss out-of-distribution examples in Section 5.

4.3.2. The sandwich estimator

The approximation of the Sandwich estimator is defined by

$$\widetilde{\Sigma} = \frac{1}{N}\widetilde{H}^{-1}\widetilde{G}\widetilde{H}^{-1}. \tag{25}$$

We introduce two separate linearization constants for the approximation of the gap (and right tail) subspace of G and H^{-1} using the harmonic means

$$\widetilde{\lambda}^{\text{H}} = \left(\frac{\lambda^{-1} + \lambda_K^{\text{H}^{-1}}}{2}\right)^{-1}, \tag{26}$$

$$\widetilde{\lambda}^{\text{G}} = \left(\frac{\lambda^{-1} + \lambda_K^{\text{G}^{-1}}}{2}\right)^{-1}. \tag{27}$$

The approximation of H^{-1} is thus given by

$$\widetilde{H}^{-1} = Q_{\text{L}}^{\text{H}}\Lambda_{\text{L}}^{\text{H}^{-1}}Q_{\text{L}}^{\text{H}^T} + \widetilde{\lambda}^{\text{H}^{-1}}(I - Q_{\text{L}}^{\text{H}}Q_{\text{L}}^{\text{H}^T}), \tag{28}$$

and the approximation of G given by

$$\widetilde{G} = Q_{\text{L}}^{\text{G}}\Lambda_{\text{L}}^{\text{G}}Q_{\text{L}}^{\text{G}^T} + \widetilde{\lambda}^{\text{G}}(I - Q_{\text{L}}^{\text{G}}Q_{\text{L}}^{\text{G}^T}). \tag{29}$$

The superscripts 'H' and 'G' are used to distinguish the eigenvectors and eigenvalues of H and G respectively. By inserting (28) and (29) into (25) and working out the product, we define the following eight matrices

$$\mathcal{S} = Q_{\text{L}}^{\text{H}}\Lambda_{\text{L}}^{\text{H}^{-1}}Q_{\text{L}}^{\text{H}^T}Q_{\text{L}}^{\text{G}}\Lambda_{\text{L}}^{\text{G}}Q_{\text{L}}^{\text{G}^T}Q_{\text{L}}^{\text{H}}\Lambda_{\text{L}}^{\text{H}^{-1}}Q_{\text{L}}^{\text{H}^T} \tag{30}$$

$$\mathcal{A} = Q_{\text{L}}^{\text{H}}\Lambda_{\text{L}}^{\text{H}^{-1}}Q_{\text{L}}^{\text{H}^T}(I - Q_{\text{L}}^{\text{G}}Q_{\text{L}}^{\text{G}^T})Q_{\text{L}}^{\text{H}}\Lambda_{\text{L}}^{\text{H}^{-1}}Q_{\text{L}}^{\text{H}^T} \tag{31}$$

$$\mathcal{N} = (I - Q_{\text{L}}^{\text{H}}Q_{\text{L}}^{\text{H}^T})Q_{\text{L}}^{\text{G}}\Lambda_{\text{L}}^{\text{G}}Q_{\text{L}}^{\text{G}^T}Q_{\text{L}}^{\text{H}}\Lambda_{\text{L}}^{\text{H}^{-1}}Q_{\text{L}}^{\text{H}^T} \tag{32}$$

$$\mathcal{D} = (I - Q_{\text{L}}^{\text{H}}Q_{\text{L}}^{\text{H}^T})(I - Q_{\text{L}}^{\text{G}}Q_{\text{L}}^{\text{G}^T})Q_{\text{L}}^{\text{H}}\Lambda_{\text{L}}^{\text{H}^{-1}}Q_{\text{L}}^{\text{H}^T} \tag{33}$$

$$\mathcal{W} = Q_{\text{L}}^{\text{H}}\Lambda_{\text{L}}^{\text{H}^{-1}}Q_{\text{L}}^{\text{H}^T}Q_{\text{L}}^{\text{G}}\Lambda_{\text{L}}^{\text{G}}Q_{\text{L}}^{\text{G}^T}(I - Q_{\text{L}}^{\text{H}}Q_{\text{L}}^{\text{H}^T}) = \mathcal{N}^T \tag{34}$$

$$\mathcal{I} = Q_{\text{L}}^{\text{H}}\Lambda_{\text{L}}^{\text{H}^{-1}}Q_{\text{L}}^{\text{H}^T}(I - Q_{\text{L}}^{\text{H}}Q_{\text{L}}^{\text{H}^T})(I - Q_{\text{L}}^{\text{G}}Q_{\text{L}}^{\text{G}^T}) = \mathcal{D}^T \tag{35}$$

$$\mathcal{C} = (I - Q_{\text{L}}^{\text{H}}Q_{\text{L}}^{\text{H}^T})Q_{\text{L}}^{\text{G}}\Lambda_{\text{L}}^{\text{G}}Q_{\text{L}}^{\text{G}^T}(I - Q_{\text{L}}^{\text{H}}Q_{\text{L}}^{\text{H}^T}) \tag{36}$$

$$\mathcal{H} = (I - Q_{\text{L}}^{\text{H}}Q_{\text{L}}^{\text{H}^T})(I - Q_{\text{L}}^{\text{G}}Q_{\text{L}}^{\text{G}^T})(I - Q_{\text{L}}^{\text{H}}Q_{\text{L}}^{\text{H}^T}). \tag{37}$$

The uncertainty associated with prediction of x_0 can now be written

$$\begin{aligned}\widetilde{\sigma}^2(x_0) = \frac{1}{N}\text{diag}\Big\{F\Big[&\mathcal{S} + \widetilde{\lambda}^{\text{G}}\mathcal{A} \\ &+ \widetilde{\lambda}^{\text{H}^{-1}}(\mathcal{N} + \mathcal{N}^T) \\ &+ \widetilde{\lambda}^{\text{G}}\widetilde{\lambda}^{\text{H}^{-1}}(\mathcal{D} + \mathcal{D}^T) \\ &+ \widetilde{\lambda}^{\text{H}^{-2}}\mathcal{C} \\ &+ \widetilde{\lambda}^{\text{G}}\widetilde{\lambda}^{\text{H}^{-2}}\mathcal{H}\Big]F^T\Big\} \in \mathbb{R}^{T_L},\end{aligned} \tag{38}$$

with the worst-case approximation error given by

$$\begin{aligned}\delta = \frac{1}{2N}\text{diag}\Big\{F\Big[&(\lambda_K^{\text{G}} - \lambda)\mathcal{A} \\ &+ (\lambda^{-1} - \lambda_K^{\text{H}^{-1}})(\mathcal{N} + \mathcal{N}^T) \\ &+ (\lambda_K^{\text{G}}\lambda^{-1} - \lambda_K^{\text{H}^{-1}}\lambda)(\mathcal{D} + \mathcal{D}^T) \\ &+ (\lambda^{-2} - \lambda_K^{\text{H}^{-2}})\mathcal{C} \\ &+ (\lambda^{-2}\lambda_K^{\text{G}} - \lambda_K^{\text{H}^{-2}}\lambda)\mathcal{H}\Big]F^T\Big\} \in \mathbb{R}^{T_L},\end{aligned} \tag{39}$$

such that $\sigma^2(x_0)$ is bounded by $\widetilde{\sigma}^2(x_0) \pm \delta$. In terms of standard deviations, the approximation error is readily found by inserting (38) and (39) into (22).

4.4. On the relation between the effective number of parameters and K

In MacKay (1992), the so-called effective number of parameters is defined in terms of the eigenvalues of the Hessian matrix. It is noted that directions in parameter space for which the eigenvalues are close to λ do not contribute to the number of good parameter measurements. Therefore, the effective number of parameters is a measure of the number of parameters which are well determined by the training data. In other words, when we select K so that $\lambda_K \approx \lambda$, we loosely cover the data dependent part of the Hessian matrix (first term of right hand side of (5)) and can therefore expect that K will be a crude estimate of the number of effective parameters.

As seen by Eqs. (21) and (39), the approximation error will be zero when the smallest eigenvalue λ_K in the left tail subspace (of H and G) is exactly equal to the L_2-regularization rate λ.

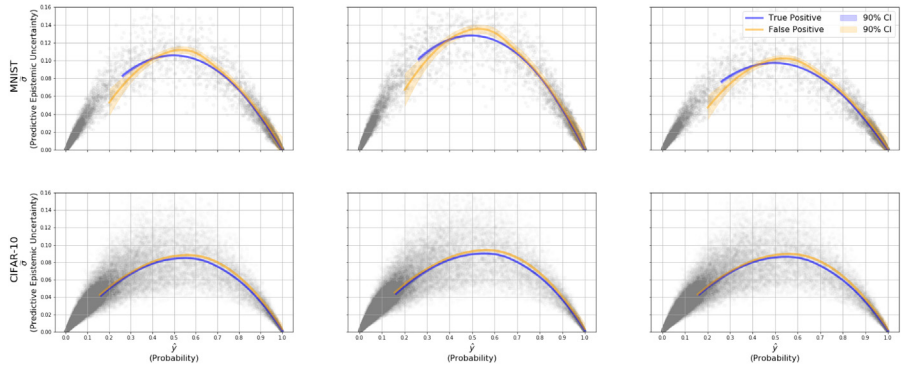
5. Demonstration and proof of concept

In the following Section we explore and demonstrate the approximate predictive epistemic uncertainty estimate governed by (10) for the two LeNet-based neural network classifiers that were introduced in Section 4.2.1. We establish by the use of regressions that the three estimators (11)–(13) yield close to perfectly correlated predictive epistemic uncertainty estimates for both of the classifiers.

5.1. The distribution of approximate predictive epistemic uncertainty

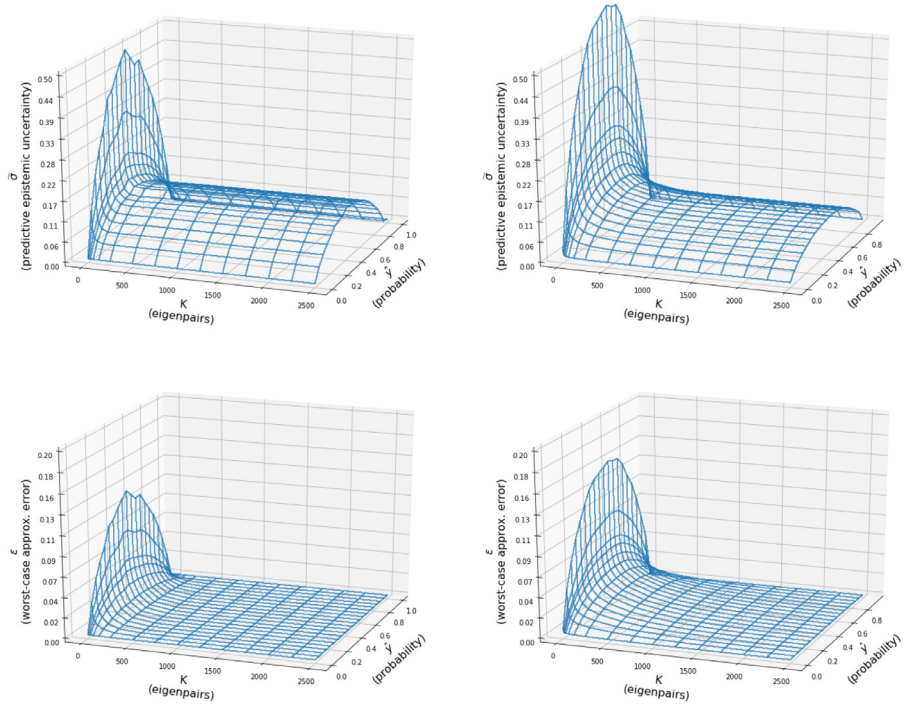
Fig. 5 shows nonparametrically smoothed versions of the predictive epistemic uncertainty for the three proposed estimators against class probability for all the images in the MNIST and CIFAR-10 test sets. Clearly, the three estimators yield close to identical results. Further, we observe that the average predictive epistemic uncertainty associated with false positives (yellow line) is higher than for true positives (blue line). The banana-shaped appearance of these plots suggests that there is a negative quadratic relationship between probability and uncertainty. The explanation for this is attributed to the softmax activation function whose gradient (i.e. sensitivity F) will always be weighted by a quantity which is negative quadratic in probability (i.e. $\hat{y}(1-\hat{y})$).

The evolution of the nonparametrically smoothed uncertainty levels and approximation errors for the OPG estimator as functions of the number of computed eigenpairs K and class probability is displayed in Fig. 6. As expected, for a growing K, the



(a) Hessian (b) OPG (c) Sandwich

Fig. 5. Nonparametrically smoothed versions of the predictive epistemic uncertainty (10) for the true positives (blue) and false positives (orange) in the MNIST (upper row, $K = 1500$) and CIFAR-10 (lower row, $K = 2500$) test sets as functions of class probability for each of the three estimators. The shaded gray bullets ($N \times T_L$ such bullets) represent the raw predictive uncertainty for all T_L classes against probability. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



(a) MNIST (b) CIFAR-10

Fig. 6. Nonparametrically smoothed versions of the predictive epistemic uncertainty (upper row) and the approximation error (lower row) in the MNIST and CIFAR-10 test sets as functions of the number of computed eigenpairs K and class probability using the OPG estimator.

approximation errors diminish and the uncertainty stabilizes. Although we do not display similar plots for the other two estimators, we note that for MNIST, the approximation errors are smallest for the OPG estimator, followed by the Hessian estimator and the Sandwich estimator. The larger the difference between λ and the smallest eigenvalue λ_K, the higher the average approximation error. As seen by the eigenvalue spectra in Fig. 3, the drop-off rate toward λ is faster for G, thus explaining why the OPG estimator leads to the lowest approximation errors on MNIST. We note that since the Sandwich estimator is dependent on both the approximation of H and G, its approximation errors are not unexpectedly the highest. Furthermore, the fall-off rate toward λ in the eigenvalue spectrum for CIFAR-10 is slightly lower than for MNIST. This means that the CIFAR-10 classifier has a greater number of effective parameters — and thus requires a higher K to achieve acceptable approximation error levels. This fact is evident by Fig. 6, where we see that the OPG approximation errors for CIFAR-10 are dropping off to zero slower than for MNIST.

For all three estimators, it is evident by Fig. 6 that most of the contribution to the predictive epistemic uncertainty comes from the left subspace corresponding to the largest eigenvalues of H and G. This observation can be counter-intuitive since it is the directions with the smallest eigenvalues that will be the largest contributors to the variance when accounting for the inversions in (11), (12) or (13).

The explanation for this phenomenon is attributed to the sensitivity F (9). We observe that the training and test set sensitivity

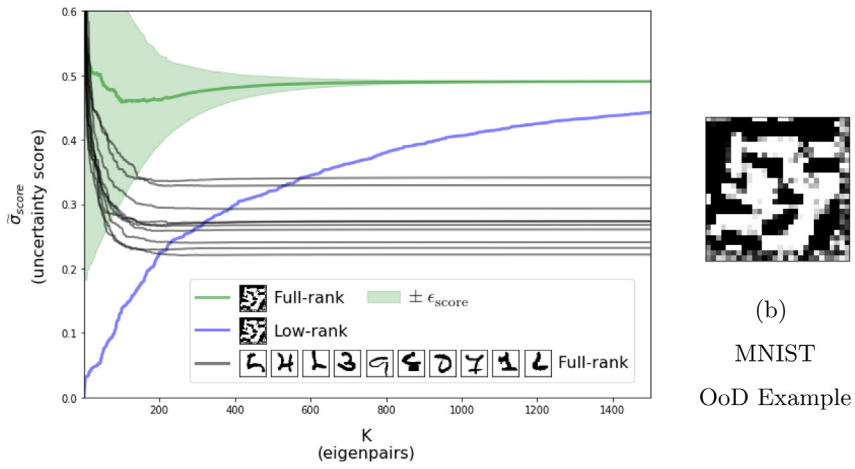


Fig. 7. The uncertainty score (a) as a function of K for the MNIST OoD example in (b) using the full-rank OPG approximation (green curve) vs. its low-rank counterpart (blue curve) from Eqs. (20) and (23). The green interval corresponds to the approximation error. The reference images (black curves) are computing using the full-rank approximation, and corresponds to the ten images in the training set with the highest uncertainty scores sorted in descending order. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 2
Regression comparison of $\widetilde{\sigma}^{\mathrm{H}}$, $\widetilde{\sigma}^{\mathrm{G}}$ and $\widetilde{\sigma}^{\mathrm{S}}$ across all the images in the MNIST and CIFAR-10 training and test sets. The respective superscripts H, G and S denote Hessian, OPG and Sandwich. The regression intercept, slope and squared correlation coefficient are denoted by α, β and R^2, respectively.

		Hessian vs. OPG $\widetilde{\sigma}^{\mathrm{G}}(x_n) = \alpha + \beta\widetilde{\sigma}^{\mathrm{H}}(x_n)$			Hessian vs. Sandwich $\widetilde{\sigma}^{\mathrm{S}}(x_n) = \alpha + \beta\widetilde{\sigma}^{\mathrm{H}}(x_n)$			OPG vs. Sandwich $\widetilde{\sigma}^{\mathrm{S}}(x_n) = \alpha + \beta\widetilde{\sigma}^{\mathrm{G}}(x_n)$		
		R^2	α	β	R^2	α	β	R^2	α	β
MNIST	Training set	0.997	0.000	1.206	0.998	0.000	0.923	0.990	0.000	0.761
	Test set	0.998	0.000	1.219	0.999	0.000	0.915	0.995	0.000	0.748
CIFAR-10	Training set	0.999	0.000	1.062	0.999	0.000	1.017	0.997	0.000	0.956
	Test set	1.000	0.000	1.066	1.000	0.000	1.014	0.998	0.000	0.950

drops to zero in directions k for which $\lambda_k \approx \lambda$ and is thus canceling with the reciprocals of the smallest eigenvalues in the linear combinations formed by (20) or (38). Nevertheless, as the sensitivity for data not belonging to the same distribution as the training can still be high in these directions, the corresponding predictive epistemic uncertainty can still receive significant contributions from directions $k > K$. This emphasizes the importance of making the estimators full-rank using the orthonormal basis technique presented in Section 4.3. We add that due to the full-rank property, the number K should be thought of as the number of explicitly computed eigenpairs rather than the number of utilized eigenpairs — as the latter will effectively be equal to P.

To illustrate the concept of a low vs. full-rank approximation, Fig. 7a displays the uncertainty scores as functions of K for the low and full-rank version of the OPG estimator applied to the out-of-distribution (OoD) example shown in Fig. 7b. For reference, we also plot the uncertainty scores for the ten images in the training set with the highest uncertainty scores sorted in descending order. Comparing the green curve with the blue curve shows that the OoD example has a sensitivity spectrum stretching out far beyond $K = 1500$ because the low-rank version (blue) has not yet reached the stable level achieved by the full-rank approximation (green) at this K. That the full-rank approximation quickly stabilizes already at around $K = 600$, can be explained by that it receives contribution from the full spectrum even though only K principal eigenpairs are computed explicitly at each stage. The reference images (black curves) are computing using the full-rank approximation, and are all lower ranked than the OoD example.

A detailed comparison of the three estimators is shown in Table 2. By regressing their outcomes against each other, we clearly see that the relative estimated uncertainty levels are near to perfectly correlated since the squared correlations coefficients are close to 1. As seen by the slopes β, only the absolute levels of the estimated uncertainty differ, and since the intercepts α are zero, there are no offsets.

5.2. Ranking images based on the 'uncertainty score'

We propose to validate our results by studying the MNIST and CIFAR-10 images associated with the maximum and minimum amount of total predictive epistemic uncertainty as defined in (23) using the Hessian estimator. Unsurprisingly, since the squared correlation coefficients in Table 2 are close to 1, the OPG and Sandwich estimators yield almost identical results and are not shown.

The idea is based on the following reasoning: if a neural network classifies an image with low predictive epistemic 'uncertainty score', the image should be easy to classify also for a human. Conversely, if the neural network classifies an image with a high predictive epistemic 'uncertainty score', the image should be hard to classify for a human. Effectively, the predictive epistemic 'uncertainty score' ranks images according to the degree of 'doubt' expressed by the neural network — and by the figures we find striking evidence that this corresponds well with human judgment.

6. Computational considerations and larger architectures

Despite the fact that we have reduced the naïve Delta method's computational complexity to be linear in P, the presented methodology still requires considerable amount of computing power when P grows very large. For reference, the Hessian estimator's initial phase on the MNIST LeNet with P in the order of

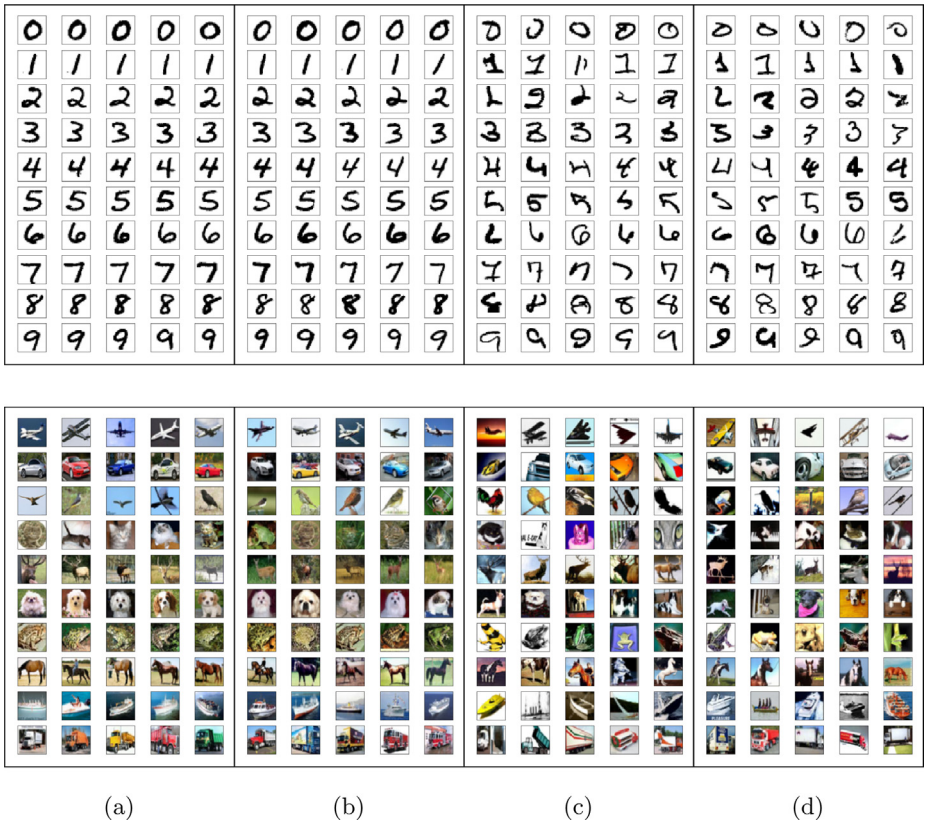


Fig. 8. The MNIST and CIFAR-10 images ranked by the predictive epistemic 'uncertainty score' per class: (a) lowest 5 in the training set, (b) lowest 5 in the test set, (c) highest 5 in the training set and (d) highest 5 in the test set.

10^5 using $K = 600$ (as seen by Fig. 6, $K = 600$ leads to acceptable approximations errors), amounts to an order of Tflops as $N = 60,000$ and as the final number of Lanczos steps turned out to be $S = 2330$ in this case. This corresponds to a computational time of about one hour using an AMD Ryzen 5 2600 CPU @ 3.4 GHz with eight cores and 32 GB memory along with an NVIDIA RTX 2080 Ti based GPU with 11 GB memory. The Hessian estimator's memory requirement amounts to about 500 MBs assuming single precision. For comparison, the naïve Delta method would clock in at the order of Pflops with a theoretical memory requirement of 35 GBs. Since in practice one would need to store both H and its inverse, as well as temporary variables depending on the type of inversion algorithm, the effective memory consumption can be as much as 320 GBs.[4] On top of this, handling the possibility of an indefinite H would require an additional explicit eigenvalue decomposition and several large matrix multiplications. In this regime, the use of direct linear algebra methods is infeasible.

With larger architectures such as ResNets (He, Zhang, Ren, & Sun, 2016), P is several orders of magnitude larger than for the LeNets discussed so far. In particular, ResNet-18 has a P in the order of 10^7. To further investigate the practicality of our methodology in this context, we present supplementing experiments for the Hessian estimator with MNIST and CIFAR-10 using the pre-activation ResNet-18 architecture.

6.1. ResNet-18

Adapting pre-activation ResNet-18 (He et al., 2016) to MNIST and CIFAR-10 leads to a total number of parameters $P = 11,175,818$ and $P = 11,176,970$, respectively. A vital building block of the ResNet architecture family is the batch-normalization (BN) layer (Ioffe & Szegedy, 2015). The β and γ parameters of the BN layers are in the following experiments treated as trainable parameters, and are thus included in both P and all relevant computations (e.g. Hessians). Furthermore, since the operation of BN layers depends on the mode to which they are configured (i.e. training mode or inference mode), we use the following rule: all quantities involving the training set as input data are computed in training mode (e.g. training cost, training accuracy, gradients, Hessians), while quantities involving the test set as input data are computed in inference mode (e.g. test predictions/test accuracy and sensitivity matrices (9)).

The training details are as follows: we apply uniform He (He, Zhang, Ren, & Sun, 2015) weight initialization and zero bias initialization. We use (5) as the cost function with a L_2-regularization rate $\lambda = 0.01$. We utilize the Adam optimizer (Bottou et al., 2018; Kingma & Ba, 2014) with a batch size of 100, and apply no form of randomized data shuffling. To ensure convergence (i.e. $\|\nabla C(\hat{\omega})\|_2 \approx 0$) we apply the following learning rate schedules given by the following (step, rate) pairs: MNIST $= \{(0, 10^{-3}), (60\text{k}, 10^{-4}), (70\text{k}, 10^{-5}), (80\text{k}, 10^{-6})\}$ and CIFAR-10 $= \{(0, 10^{-3}), (55\text{k}, 10^{-4}), (85\text{k}, 10^{-5}), (125\text{k}, 10^{-6}), (155\text{k}, 10^{-7}), (205\text{k}, 10^{-8}), (255\text{k}, 10^{-9})\}$. For MNIST, we stop the training after 90,000 steps — corresponding to a training accuracy of 0.999, test accuracy 0.995, training cost $C(\hat{\omega}) = 0.281$ and a gradient norm $\|\nabla C(\hat{\omega})\|_2 = 0.018$. For CIFAR-10, we stop the training after 285,000 steps – corresponding to a training accuracy of 0.994, test accuracy 0.773, training cost $C(\hat{\omega}) = 0.520$ and a gradient norm $\|\nabla C(\hat{\omega})\|_2 = 0.076$.

To stay within the 32 GB memory bound of the aforementioned computer specification, we managed to compute up to $K = 200$ eigenpairs for the Hessian estimator. For both MNIST and CIFAR-10, the Lanczos algorithm converged at exactly

[4] Try running numpy.linalg.inv(np.diag(10**5).astype('float32')) and watch the memory consumption throughout the whole process.

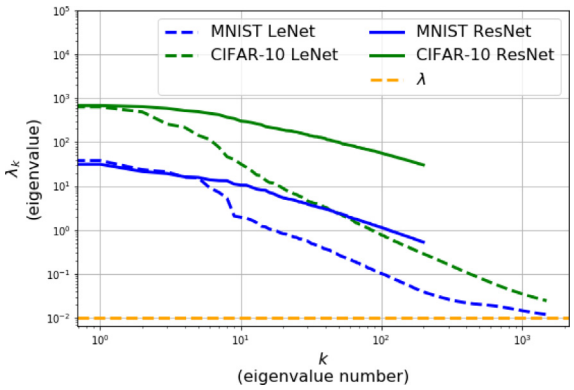


Fig. 9. Comparison of the Hessian eigenvalue spectra of LeNet and ResNet-18 for MNIST and CIFAR-10 data.

$S = 502$ iterations with total computation times of about 14 and 15 h, respectively.

In agreement with the findings in Yao, Gholami, Keutzer, and Mahoney (2020), Fig. 9 shows that the curvature (i.e. Hessian eigenvalue spectrum) of the ResNets has a slower decay to λ. In other words, for a given K, we can expect that a (large) ResNet will yield larger approximation errors (18) than for a (small) LeNet.

Fig. 10 shows the nonparametrically smoothed versions of the predictive epistemic uncertainty for the ResNets against the class probability for all the images in the MNIST and CIFAR-10 test sets. For reference, the corresponding plots for the LeNets were shown in Fig. 5. The absolute level of the predictive uncertainty for the ResNets is larger than for the LeNets, and exceeds the theoretical maximum standard deviation of 0.5 for softmax-based neural networks. A simple inspection of the computed approximation errors (22) for the ResNets rules out a too low $K = 200$ as the only culprit, because the MNIST test set image with the largest approximation error corresponds to $\widetilde{\sigma} \pm \epsilon \approx 19 \pm 4$ and the CIFAR-10 equivalent to $\widetilde{\sigma} \pm \epsilon \approx 50 \pm 13$ (i.e. lower bounds still greater than the theoretical bound 0.5). We leave to investigate the root cause of this anomaly, but speculate that the number of training examples N might simply be too small now that $N \ll P$. Nevertheless, the relative uncertainty levels are still reasonable in terms of the raised level for false positives (as seen by Fig. 10), and in terms of that meaningful rankings similar to those shown in Fig. 8 for the LeNets still can be obtained for the ResNets.

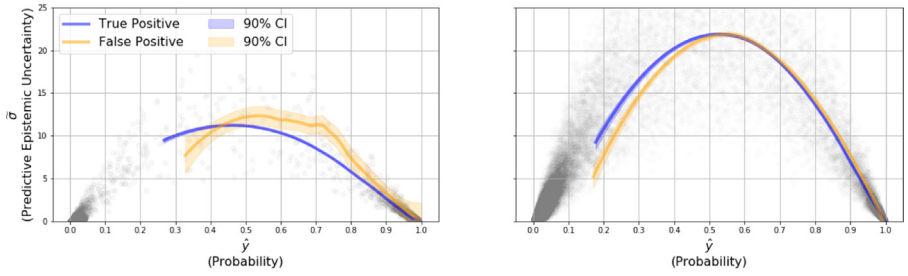
7. Summary, concluding remarks and further work

We have presented a computationally tractable framework for traditional Fisher information based (epistemic) uncertainty quantification in deep learning classification. To this end, we have introduced full-rank, positive definite covariance estimators using approximate eigendecompositions in terms of the Hessian, the OPG approximation and the so-called Sandwich estimator. We have recognized the Delta method as a measure of epistemic as opposed to aleatoric uncertainty and break it into two components: the eigenvalue spectrum of the Fisher information (i.e. Hessian) of the cost function and the per-example sensitivities (i.e. gradients) of the model function. Further, we have proposed to utilize the Lanczos algorithm in combination with Pearlmutter's technique to compute the needed eigenpairs of the Hessian, and to compute mini-batches of the Jacobian matrix using efficient per-example gradients in combination with incremental singular value decompositions for the OPG approximation. As the computational complexity of these methods scale linearly with the number of model parameters, they are therefore suited for deep learning. However, since the computational complexity also scales (linearly) with the number of eigenpairs K, it seems that with today's computing power, the bottleneck of our methodology is reached when the number of parameters is in the order of 10^7.

We have shown that the three estimators yield almost identical prediction uncertainty estimates when applied on two different LeNet-based neural network classifiers. We have seen that only the top $K \ll P$ Fisher information matrix eigenpairs contribute significantly to the predictive uncertainty for data in the same distribution as the training set. As this does not necessarily hold true for OoD examples, we have shown that thanks to the full-rank property of the proposed estimators, these too will converge quickly under the same framework.

We have also seen that when images are ranked according to their relative level of predictive epistemic uncertainty, the ordering corresponds well with human judgment: ambiguous images tend to be highly ranked, and we clearly see why data augmentation is beneficial — since the top ranked images often are prone to unusual perspectives and/or rare colors. Generally, we conjecture that deep learning classifiers can benefit from incorporating also the uncertainty measure in the classification rule. As a corroborative example we have empirically shown that false positives appears to have an average higher prediction uncertainty than true positives.

Looking forward, we point at several specific areas of research which could be investigated. The first candidate is to establish



(a) MNIST (b) CIFAR-10

Fig. 10. Nonparametrically smoothed versions of the predictive epistemic uncertainty (10) based on the Hessian estimator for the ResNets using $K = 200$. The true positives (blue) and false positives (orange) are shown for MNIST in (a) and for CIFAR-10 in (b) using the test sets as input data and are plotted as functions of the class probability. The shaded gray bullets ($N \times T_L$ such bullets) represent the raw predictive uncertainty for all T_L classes against probability. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

how the Fisher information eigenspectrum of very large networks and datasets behave. If the contraction of the eigenspectrum toward λ continues to be fast with growing network and dataset sizes, the methodology presented can be tractable even for the most complex models. However, if the largest affordable K yields a λ_K far from λ, it can render the methodology intractable as the approximation errors can be too large. This points to understanding what causes the contraction phase in the first place, and hence uncovering the factors that drive it. Promising research in this direction is the Stochastic Lanczos algorithm (Lin, Saad, & Yang, 2016; Yao et al., 2020) as well as the investigation of pathological spectra of the Fisher information (Karakida, Akaho, & Amari, 2019). However, inconveniently for our methodology, we find evidence in agreement with recent literature (Yao et al., 2020) showing that ResNets have a higher number of effective parameters than LeNets. In other words, for a given K, we can expect that a (large) ResNet will yield larger approximation errors than for a (small) LeNet. Secondly, we leave the discussion regarding which of the three estimators (or other combinations) one should use – and when – as an opportunity for future research. Thirdly, as this work has been focused on the classification task, a natural extension is to see how the framework behaves under deep learning regression (Khosravi & Creighton, 2011). Fourthly, we point at a fundamental issue with the Delta method itself. The Delta method is inevitably based on the local curvature around the parameter estimate $\hat{\omega}$, hence incorporating no means about the uncertainty of the parameter estimate outside this local region. What is lost, and how much, by disregarding the broader perspective of the solution space – a space potentially within reach for sampling methods.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Parts of this work have been done in the context of CEDAS (Center for Data Science, University of Bergen, Norway). The lead author would also like to thank Dr. Berent Å. S. Lunde, Dr. Kristian Gundersen, Øyvind Lunde Rørtveit and Egil Martinsen Aspevik for their helpful advice on various technical issues examined in this paper.

Appendix

The cost function $C(\omega)$ can be interpreted as the negative log posterior,

$$C(\omega) = -\log p(\mathcal{D}|\omega)p(\omega), \tag{40}$$

for the parameter ω and some training data $\mathcal{D}$, where $p(\mathcal{D}|\omega)$ is the likelihood and $p(\omega)$ the prior. Under L_2-regularization with rate $\lambda/2$, the prior takes the form of a multivariate normal distribution with zero mean and covariance $(\lambda/2)^{-1}I$

$$\omega \sim \mathcal{N}\big(0, (\lambda/2)^{-1}I\big). \tag{41}$$

It follows that

$$H_{C(\omega)} = -H_{\log p(\mathcal{D}|\omega)p(\omega)} = -H_{\log p(\mathcal{D}|\omega)} + \lambda I, \tag{42}$$

where we have used that $H_{\log p(\omega)} = -\lambda I$. Taking expectation with respect to $p(\mathcal{D}|\omega)$, and drawing on the well known result for the expected Fisher information matrix (Lehmann & Casella, 1998):

$$\mathop{\mathbb{E}}_{p(\mathcal{D}|\omega)}\left[H_{\log p(\mathcal{D}|\omega)}\right] = -\mathop{\mathbb{E}}_{p(\mathcal{D}|\omega)}\left[\nabla \log p(\mathcal{D}|\omega)\nabla \log p(\mathcal{D}|\omega)^T\right], \tag{43}$$

it follows that

$$\mathop{\mathbb{E}}_{p(\mathcal{D}|\omega)}\left[H_{C(\omega)}\right] = \mathop{\mathbb{E}}_{p(\mathcal{D}|\omega)}[G] + \lambda I \quad \square \tag{44}$$

References

Alain, G., Roux, N. L., & Manzagol, P.-A. (2019). Negative eigenvalues of the hessian in deep neural networks. https://arxiv.org/abs/1902.02366, arXiv:1902.02366v1 [cs.LG].

Bottou, L., Curtis, F. E., & Nocedal, J. (2018). Optimization methods for large-scale machine learning. *SIAM Review*, *60*(2), 223–311.

Cardot, H., & Degras, D. (2015). Online principal component analysis in high dimension: Which algorithm to choose?. https://arxiv.org/abs/1511.03688 arXiv:1511.03688 [stat.ML].

Efron, B. (1979). Bootstrap methods: Another look at the jackknife. *Annals of Statistics*, *7*(1), 1–26.

Freedman, D. A. (2006). On the so-called "Huber Sandwich Estimator" and 'Robust Standard Errors". *The American Statistician*, *60*(4), 299–302, https://www.jstor.org/stable/27643806.

Gal, Y., & Ghahramani, Z. (2016). Dropout as a approximation: Representing model uncertainty in deep learning. https://arxiv.org/pdf/1506.02142, arXiv:1506.02142v6 [stat.ML].

Ghorbani, B., Krishnan, S., & Xiao, Y. (2019). An investigation into neural net optimization via hessian eigenvalue density. https://arxiv.org/abs/1901.10159, arXiv:1901.10159v1 [cs.LG].

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press, http://www.deeplearningbook.org.

Granziol, D., Garipov, T., Zohren, S., Vetrov, D., Roberts, S., & Wilson, A. G. (2019). The Deep Learning Limit: are negative neural network eigenvalues just noise? In *Presented at the ICML 2019 workshop on theoretical physics for deep learning*.

Grosse, R. (2020). Lecture 2: Taylor approximations. https://www.cs.toronto.edu/~rgrosse/courses/csc2541_2021/readings/L02_Taylor_approximations.pdf.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision* (pp. 1026–1034).

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770–778).

Hoef, J. M. V. (2012). Who invented the delta method? *The American Statistician*, *66*(2), 124–127, https://www.researchgate.net/publication/254329376_Who_Invented_the_Delta_Method.

Hüllermeier, E., & Waegeman, W. (2020). Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. https://arxiv.org/abs/1910.09457, arXiv:1910.09457v2 [cs.LG].

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (pp. 448–456). PMLR.

Karakida, R., Akaho, S., & Amari, S.-i. (2019). Pathological spectra of the Fisher information metric and its variants in deep neural networks. arXiv preprint arXiv:1910.05992.

Kendall, A., & Gal, Y. (2017). What uncertainties do we need in Bayesian deep learning for computer vision? https://arxiv.org/pdf/1703.04977, arXiv:1703.04977v2 [cs.CV].

Khosravi, A., & Creighton, D. (2011). A comprehensive review of neural network-based prediction intervals and new advances. *IEEE Transactions on Neural Networks*, *22*(9), https://www.researchgate.net/publication/51534965_Comprehensive_Review_of_Neural_Network-Based_Prediction_Intervals_and_New_Advances.

Kingma, D. P., & Ba, J. L. (2014). Adam: A method for stochastic optimization. In *Proc. 3rd int. conf. learn. representations*.

LeCun, Y., Simard, P. Y., & Pearlmutter, B. (1993). Automatic learning rate maximization by on-line estimation of the Hessian's eigenvectors. In *Advances in neural information processing systems (NIPS 1992)*. http://yann.lecun.com/exdb/publis/pdf/lecun-simard-pearlmutter-93.pdf.

Lehmann, E. L., & Casella, G. (1998). *Theory of point estimation* (2nd ed.). (p. 125). Springer Science & Business Media.

Levy, A., & Lindenbaum, M. (2000). Sequential karhunen–loeve basis extraction and its application to images. *IEEE Transactions on Image Processing*, *9*(8), http://www.cs.technion.ac.il/~mic/doc/skl-ip.pdf.

Lin, L., Saad, Y., & Yang, C. (2016). Approximating spectral densities of large matrices. *SIAM Review*, *58*(1), 34–65.

Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., et al. (2017). A survey on deep learning in medical image analysis. *Medical Image Analysis*, *42*, 60–88, http://www.sciencedirect.com/science/article/pii/S1361841517301135.

Loquercio, A., Segu, M., & Scaramuzza, D. (2020). A general framework for uncertainty estimation in deep learning. https://arxiv.org/pdf/1907.06890, arXiv:1907.06890v4 [cs.CV].

MacKay, D. (1992). A practical Bayesian framework for backpropagation networks. *Neural Computation*, *4*(3), 448–472, http://www.inference.org.uk/mackay/PhD.html#PhD.

Martens, J. (2020). New insights and perspectives on the natural gradient method. https://arxiv.org/abs/1412.1193, arXiv:1412.1193 [cs.LG].

Murfet, D., Wei, S., Gong, M., Li, H., Gell-Redman, J., & Quella, T. (2020). Deep learning is singular, and that's good. https://arxiv.org/abs/2010.11560, arXiv:2010.11560 [cs.LG].

Nagarajan, P., & Warnell, G. (2019). Deterministic implementations for reproducibility in deep reinforcement learning. https://arxiv.org/abs/1809.05676, arXiv:1809.05676 [cs.AI].

Newey, W. K., & McFadden, D. (1994). *Handbook of econometrics*. https://www.sciencedirect.com/science/article/pii/S1573441205800054.

Nilsen, G. K., Munthe-Kaas, A. Z., Skaug, H. J., & Brun, M. (2019). Efficient computation of Hessian matrices in TensorFlow. https://arxiv.org/abs/1905.05559, arXiv:1905.05559v1 [cs.LG].

Nilsen, G. K., Munthe-Kaas, A. Z., Skaug, H. J., & Brun, M. (2021). A comparison of the delta method and the bootstrap in deep learning classification. http://arxiv.org/abs/2107.01606, arXiv:2107.01606 [cs.LG].

Osband, I. (2016). Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout. In *NIPS workshop on Bayesian deep learning*. http://bayesiandeeplearning.org/2016/papers/BDL_4.pdf.

Osband, I., Blundell, C., Pritzel, A., & Roy, B. V. (2016). Deep exploration via bootstrapped DQN. In *Conference on neural information processing systems (NIPS)*. https://papers.nips.cc/paper/6501-deep-exploration-via-bootstrapped-dqn.pdf.

Pearlmutter, B. A. (1994). Fast exact multiplication by the hessian. *Neural Computation*, *6*(1), 147–160, http://www.bcl.hamilton.ie/~barak/papers/nc-hessian.pdf.

pyDeepDelta: A TensorFlow Module Implementing the Delta Method in Deep Learning Classification, https://github.com/gknilsen/pydeepdelta.git.

Sagun, L., Bottou, L., & LeCun, Y. (2017). Eigenvalues of the hessian in deep learning: Singularity and beyond. https://arxiv.org/abs/1611.07476, arXiv:1611.07476v2 [cs.LG].

Sagun, L., Evci, U., Guney, V. U., Dauphin, Y., & Bottou, L. (2018). Empirical analysis of the hessian of over-parametrized neural networks. https://arxiv.org/abs/1706.04454, arXiv:1706.04454v3 [cs.LG].

Schulam, P., & Saria, S. (2019). Can you trust this prediction? Auditing pointwise reliability after learning. https://arxiv.org/abs/1901.00403, arXiv:1901.00403 [stat.ML].

scikit-learn, https://scikit-learn.org/.

SciPy, http://www.scipy.org.

Song, H., Kim, M., Park, D., & Lee, J.-G. (2020). Learning from noisy labels with deep neural networks: A survey. https://arxiv.org/pdf/2007.08199, arXiv:2007.08199v2 [cs.LG].

Trefethen, L. N., & III, D. B. (1997). *Numerical linear algebra* (pp. 243–284). Siam.

Watanabe, S. (2007). Almost all learning machines are singular. In *Proceedings of the 2007 IEEE symposium on foundations of computational intelligence*. http://watanabe-www.math.dis.titech.ac.jp/users/swatanab/foci2007.pdf.

Yan, C., Gong, B., Wei, Y., & Gao, Y. (2020). Deep multi-view enhancement hashing for image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *43*(4), 1445–1451.

Yao, Z., Gholami, A., Keutzer, K., & Mahoney, M. W. (2020). Pyhessian: Neural networks through the lens of the hessian. In *2020 IEEE international conference on big data (Big data)* (pp. 581–590). IEEE.

Zhu, L., & Laptev, N. (2017). Deep and confident prediction for time series at uber. In *2017 IEEE international conference on data mining workshops (ICDMW)*.