The Memory Perturbation Equation: Understanding Model's Sensitivity to Data

Peter Nickl[†] peter.nickl@riken.jp

Lu Xu[*†] lu.xu.sw@riken.jp

Dharmesh Tailor[*‡] d.v.tailor@uva.nl

Thomas Möllenhoff[†] thomas.moellenhoff@riken.jp

Mohammad Emtiyaz Khan[†§] emtiyaz.khan@riken.jp

Abstract

Understanding model's sensitivity to its training data is crucial not only for safe and robust operation but also for future adaptations. We present the memory-perturbation equation (MPE) which relates model's sensitivity to perturbation in its training data. Derived using Bayesian principles, the MPE unifies existing influence measures, generalizes them to a wide-variety of models and algorithms, and unravels useful properties regarding sensitivity. Our empirical results show that sensitivity estimates obtained during training can faithfully predict generalization on unseen test data and avoid the need for expensive retraining. The equation is useful for future research on robust and adaptive learning.

1 Introduction

Understanding model's sensitivity to training data is important to handle issues related to quality, privacy, and security. For example, we can use it to understand (i) the effect of errors and biases in the data; (ii) model's dependence on private information to avoid data leakage; (iii) model's weakness to malicious manipulations. However, despite their importance, sensitivity properties of machine learning (ML) models in general are not well understood. Sensitivity is often studied through empirical investigations, but conclusions drawn this way do not always generalize across models or algorithms. Such studies are also costly, sometimes requiring thousands of GPUs [30], which can quickly become infeasible if we need to repeat them every time the model is updated.

A cheaper solution is to use local perturbation methods [17], for instance, influence measures that study sensitivity of the model to data removal (Fig. 1(a)) [7, 6]. Such methods too fall short of providing a clear understanding of general causes of influence for generic models and algorithms. Influence measures are useful for studying trained models but it remains challenging to generalize them to analyze training trajectories [12, 39]. Another challenge is to handle non-differentiable loss or discrete parameters where a natural choice of perturbation mechanisms is often unclear [26]. In general, sensitivity analysis of generic ML models and algorithms is a difficult problem.

In this paper, we simplify such issues by proposing a new method to unify, generalize, and understand perturbation methods for sensitivity analysis. We present the memory-perturbation equation (MPE) as a unifying equation to understand sensitivity of generic ML algorithms; see Eq. 4. The equation builds upon the Bayesian learning rule (BLR) [22] which unifies many popular algorithms

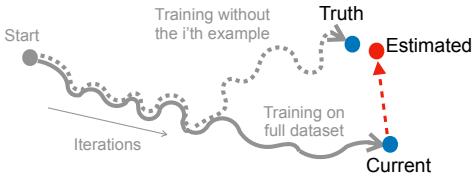
[*]Equal contribution. Part of this work was carried out when Dharmesh Tailor was at RIKEN AIP.

[†]RIKEN Center for AI Project, Tokyo, Japan.

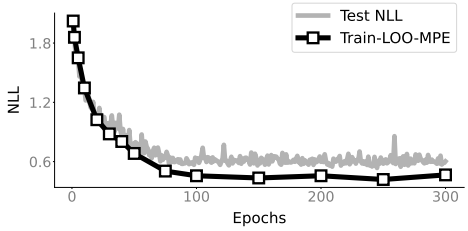
[‡]University of Amsterdam, Amsterdam, Netherlands.

[§]Corresponding author.

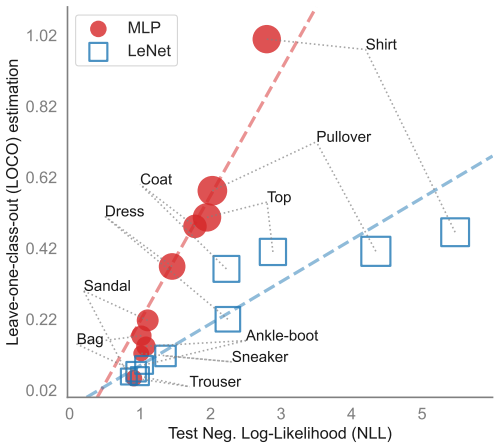
Preprint. Under review.



(a) Effect of removing an example.



(b) Predicting test loss during training, CIFAR10



(c) Predicting test loss for leave-one-class-out, FMNIST

Figure 1: Panel (a) illustrates the local perturbation method where sensitivity of a training trajectory (solid gray line) is studied by removing (or perturbing) an example and observing the deviation to the trajectory (dashed line). The goal is to estimate the deviation without rerunning the algorithm. Panel (b) shows that the Leave-One-Out (LOO) estimates (black line) computed solely on training data can predict the test NLL trends (gray line) during training of ResNet–20 on CIFAR10. Panel (c) shows the Leave-One-Class-Out (LOCO) estimates (y-axis) that correlate well the test NLL (x-axis) on FMNIST using MLP (red) and LeNet (blue). The left-out class is marked for each case.

from various fields as specific instances of a *natural-gradient* descent over the Bayes objective. The MPE suggests that natural-gradients can also be used to understand sensitivity of all such algorithms. We use the MPE to show several new results regarding sensitivity of generic ML algorithms:

1. We show that sensitivities to a group of examples can be estimated by using natural-gradients of those examples alone. Examples with larger natural-gradients contribute more to the sensitivity and, due to this, most of the sensitivity is explained by just a few examples. These highly-sensitive examples characterize the model's memory because perturbing such examples can make the model forget its essential knowledge. The MPE relates the model's sensitivity to perturbation in the memory of the model.
2. We derive Influence Function [7, 25] as a special case of the MPE where natural-gradients with respect to Gaussian posterior are used. Not only this, the MPE gives rise to new measures that, unlike influence functions, can be applied *during* training for all algorithms covered under the BLR (such as those used in deep learning and optimization).
3. Measures derived using Gaussian posteriors share a common property: sensitivity to an example is simply obtained by multiplying its prediction error and variance. That is, the model is expected to be most sensitive to examples it finds difficult to predict confidently.
4. Sensitivity can be estimated cheaply (even during training) whenever natural-gradients are cheap to compute. Already computed quantities (e.g., error and variance) can be reused and estimation does not add any additional computational overhead.
5. Finally, our empirical results show that, by using sensitivity of the training data alone, we can accurately predict model generalization, even during training (Fig. 1(b)). Effect of class-removal can also be faithfully estimated (Figs. 1(b) and 1(c)) without any retraining. These agree with similar studies showing effectiveness of sensitivity [18, 10, 16, 4].

2 Understanding Model's Sensitivity to Training Data

Understanding model's sensitivity to the training data is important but is often done by a costly process of retraining the model multiple times. For example, consider a model with parameter vector $\boldsymbol{\theta} \in \mathbb{R}^P$ and data $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_N\}$ trained by using an algorithm $\mathcal{A}_t$ that generates a

sequence $\{\boldsymbol{\theta}_t\}$, at each iteration t, converging to a minimizer $\boldsymbol{\theta}_*$, as shown below:

$$\boldsymbol{\theta}_t \leftarrow \mathcal{A}_t\left(\boldsymbol{\theta}_{t-1}, \mathcal{L}(\boldsymbol{\theta})\right) \quad \text{where } \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \ell_i(\boldsymbol{\theta}) + \mathcal{R}(\boldsymbol{\theta}), \tag{1}$$

Here, we use a loss $\ell_i(\boldsymbol{\theta})$ for the i'th example $\mathcal{D}_i$ and a regularizer $\mathcal{R}(\boldsymbol{\theta})$. Now, because $\boldsymbol{\theta}_t$ are all functions of $\mathcal{D}$ or its subsets, we can analyze their sensitivity by simply 'perturbing' the data. For example, we can remove a data example, say the i'th one, to get a perturbed dataset $\mathcal{D}^{\setminus i} = \{\mathcal{D}_1, \ldots, \mathcal{D}_{i-1}, \mathcal{D}_{i+1}, \ldots, \mathcal{D}_N\}$ and then retrain the model to get a new minimizer, denoted by $\boldsymbol{\theta}_*^{\setminus i}$. If the deviation $\boldsymbol{\theta}_*^{\setminus i} - \boldsymbol{\theta}_*$ is large, then we may deem the model to be highly sensitive to the i'th example. We may also choose to analyze deviations in the function outputs $f_i(\boldsymbol{\theta}_*)$ for inputs $\mathbf{x}_i \in \mathcal{D}_i$ or even the loss function $\mathcal{L}(\boldsymbol{\theta})$. The same procedure can be repeated for $\boldsymbol{\theta}_t$ obtained during training; see Fig. 1(a), but such brute-force retraining is too costly and even infeasible when training trajectories are long or the model and data are large. More importantly, conclusions drawn from a purely empirical study may not always generalize across models or algorithms.

A cheaper alternative is to use local perturbation methods [17], for instance, influence measures that *estimate* sensitivity of the model without retraining (illustrated in Fig. 1(a) by the dashed red arrow). The simplest result of this kind is for linear regression which dates back to 1977 [6]. Consider input-output pairs $(\mathbf{x}_i, y_i)$ modeled by the loss $\ell_i(\boldsymbol{\theta}) = \frac{1}{2}(y_i - \mathbf{x}_i^\top \boldsymbol{\theta})^2$ and a regularizer $\mathcal{R}(\boldsymbol{\theta}) = \frac{1}{2}\delta\|\boldsymbol{\theta}\|^2$. As shown in App. A, for this case, we have closed-form expressions for the deviations:

$$\boldsymbol{\theta}_*^{\setminus i} - \boldsymbol{\theta}_* = \boldsymbol{\Sigma}_* \mathbf{x}_i e_i^{\setminus i}, \qquad f_i(\boldsymbol{\theta}_*^{\setminus i}) - f_i(\boldsymbol{\theta}_*) = v_i e_i^{\setminus i}, \tag{2}$$

where $e_i^{\setminus i} = \mathbf{x}_i^\top \boldsymbol{\theta}_*^{\setminus i} - y_i$ is the prediction error of $\boldsymbol{\theta}_*^{\setminus i}$ and $v_i = \mathbf{x}_i^\top \boldsymbol{\Sigma}_* \mathbf{x}_i$ is the prediction variance with $\boldsymbol{\Sigma}_* = (\nabla^2 \mathcal{L}(\boldsymbol{\theta}_*))^{-1}$ as the covariance matrix [5, Chapter 3, Eq. 3.59]. The expressions show that the influence is bi-linearly related to both prediction error and variance, that is, examples with high values of either of these lead to a large change in the model if removed. In other words, the model is most sensitive to examples it finds difficult to predict confidently.

These ideas are generalized using *infinitesimal perturbation* [17] where we use a perturbation model and then take partial derivatives to measure influence, leading up to the idea of *influence functions* [7, 25] where we use a perturbation model $\boldsymbol{\theta}_*^{\epsilon_i} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) - \epsilon_i \ell_i(\boldsymbol{\theta})$ with a scalar perturbation $\epsilon_i \in \mathbb{R}$, then Eq. 2 can be obtained by simply taking derivatives of $\boldsymbol{\theta}_*^{\epsilon_i}$ at $\epsilon_i = 1$; see App. A. We are free to choose other perturbation model, but often we do see a bi-linear relationship with respect to prediction error and variance; see Eq. 27 for an example. The infinitesimal-perturbation method can be generalized to other models such as neural networks [26].

Despite their generality, influence measures fall short of providing a clear understanding of general causes of influence for generic models and algorithms. This is due to several reasons:

1. Influence measures are valid only at a stationary point $\boldsymbol{\theta}_*$ where the gradient is assumed to be 0, and extending them to iterates $\boldsymbol{\theta}_t$ generated by a specific algorithm is non-trivial [12]. This is even more important for deep learning where we may never reach such stationary point, for example, due to stochastic training or early stopping.
2. Applying such measures to non-differentiable loss functions or discrete parameter spaces is also difficult. This is because the choice of perturbation model is not always obvious [26].
3. Finally, despite their generality, the measures do not directly reveal the causes of high influence. Does the bi-linear relationship in Eq. 2 hold more generally? If yes, under what conditions? Answers to such questions are currently unknown.

Studies to fix these issues are rare in ML, rather it is more common to simply use heuristics and draw conclusions based on extensive empirical investigations. Many such heuristic measures have been proposed in the recent years, for example, those using gradients [19, 2, 30], variations of Cook's distance [14], prediction error [3, 38, 33, 32], backtracking training trajectories [13], or simply by retraining [11]. These works, although useful, do not directly address the issues. They are also missing connections regarding any bi-linear relationship similar to Eq. 2. Our goal here is fix this issue by unifying and generalizing perturbation methods of sensitivity analysis, and to understand and reveal general causes of influence for generic models and algorithms.

3 The Memory-Perturbation Equation

We propose the memory-perturbation equation (MPE) to unify, generalize, and understand the sensitivity of generic ML algorithms. The equation is derived using the Bayesian learning rule (BLR) [22] which unifies many popular algorithms from various fields. We will first describe the BLR, then present the MPE, and show its application to derive new results regarding sensitivity of algorithms.

The BLR optimizes a Bayesian formulation of Eq. 1 to find an approximation $q(\boldsymbol{\theta})$ to the posterior $p(\boldsymbol{\theta}|\mathcal{D}) \propto e^{-\mathcal{L}(\boldsymbol{\theta})}$. Denoting by $q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})$ an exponential-family distribution with natural parameter $\boldsymbol{\lambda}$, the BLR update can be written as the following,

$$\boldsymbol{\lambda}_t = (1-\rho)\boldsymbol{\lambda}_{t-1} + \rho \sum_{j=0}^{N} \widetilde{\nabla} \mathbb{E}_{q_{\boldsymbol{\lambda}_{t-1}}}[-\ell_j(\boldsymbol{\theta})] \tag{3}$$

where $\boldsymbol{\lambda}_t$ is the natural parameter at the t'th iteration, $\widetilde{\nabla} = \mathbf{F}(\boldsymbol{\lambda})^{-1}\nabla_{\boldsymbol{\lambda}}$ is the natural gradient with respect to $\boldsymbol{\lambda}$ and defined using the Fisher $\mathbf{F}(\boldsymbol{\lambda})$ of $q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})$, and $\rho > 0$ is the learning rate. For brevity, we denote $\ell_0(\boldsymbol{\theta}) = \mathcal{R}(\boldsymbol{\theta})$. The rule can obtain many widely-used algorithms as special cases by choosing an appropriate exponential-family form for $q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})$ and employing further approximations to natural gradients. These include popular algorithms in deep learning (e.g., RMSprop, Adam, and SAM), optimization (e.g., gradient descent and Newton's method), and approximate inference (e.g., ridge regression, Kalman filters, EM, Laplace's method). We will now propose a method to analyze sensitivity of the BLR which then enables us to do the same for all the algorithms covered under it.

Our key idea is the following: Eq. 3 shows that each $\boldsymbol{\lambda}_t$ is an accumulation of all the past natural gradients computed at iterations before t, and so it lies in their span. Therefore, to estimate the effect of removal of a group of examples in a set $\mathcal{M} \subset \mathcal{D}$, we propose to simply subtract the natural gradients of that example. Specifically, we take a BLR step in the opposite direction:

$$\text{MPE:} \quad \hat{\boldsymbol{\lambda}}_t^{\setminus \mathcal{M}} - \boldsymbol{\lambda}_t = \sum_{j \in \mathcal{M}} \widetilde{\nabla} \mathbb{E}_{q_{\boldsymbol{\lambda}_t}}[\ell_j(\boldsymbol{\theta})], \tag{4}$$

where $\hat{\boldsymbol{\lambda}}_t^{\setminus \mathcal{M}}$ is an estimate of the true $\boldsymbol{\lambda}_t^{\setminus \mathcal{M}}$ obtained by retraining with the BLR but without using the examples in $\mathcal{M}$. We call this the memory-perturbation equation (MPE). To estimates the effects of general additive perturbations ϵ_j to the BLR iterates, we can replace ℓ_j in Eq. 4 by ϵ_j.

The MPE has two additional unique features: first, the deviations caused by perturbations in $\mathcal{M}$ are estimated entirely by using natural-gradients of examples in $\mathcal{M}$ alone; second, the deviation is additive over all $j \in \mathcal{M}$, therefore examples with larger natural-gradients contribute more to the sensitivity estimate. This is similar to the representation theorem [23, 36], for example, in support vector machines [8], and in a similar spirit we expect most of the sensitivity to be explained by a small fraction of the data examples. These highly-sensitive examples characterize the model's memory because perturbing such examples can make the model forget its essential knowledge. The MPE gives a way to relate the model's sensitivity to perturbation in its memory.

The equation is analogous to Eq. 2 for linear models which can be seen as a Newton-step in the opposite direction [26], but here we estimate the deviations in the natural-parameter space and we will now show that this choice allows us to derive influence functions as special cases of the MPE and derive new scores that apply during training.

3.1 Linear-model influence function as a special case of the MPE

Our first result derives Eq. 2 for linear regression by using the MPE.

Theorem 1 *For a full-covariance Gaussian $q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})$, the MPE for linear regression leads to Eq.* 2.

Proof: For Gaussian distributions $q_{\boldsymbol{\lambda}}(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{m}, \boldsymbol{\Sigma})$ with mean $\mathbf{m}$ and covariance $\boldsymbol{\Sigma}$ (and precision $\mathbf{S} = \boldsymbol{\Sigma}^{-1}$), we have $\boldsymbol{\lambda} = (\mathbf{S}\mathbf{m},\ -\frac{1}{2}\mathbf{S})$ and $\boldsymbol{\mu} = \mathbb{E}_{q_{\boldsymbol{\lambda}}}(\boldsymbol{\theta},\ \boldsymbol{\theta}\boldsymbol{\theta}^\top)$. For linear regression the (exact) posterior is a Gaussian with $\mathbf{m}_* = \boldsymbol{\theta}_*$ and $\mathbf{S}_* = \nabla^2\mathcal{L}(\boldsymbol{\theta}_*)$. To compute natural gradients, we rewrite them in terms of the gradient and Hessian of the loss [22, Eq. 10-11],

$$\widetilde{\nabla}\mathbb{E}_{q_{\boldsymbol{\lambda}}}[\ell_i] = \mathbb{E}_{q_{\boldsymbol{\lambda}}}\left(\nabla\ell_i(\boldsymbol{\theta}) - \nabla^2\ell_i(\boldsymbol{\theta})\mathbf{m},\ \ \tfrac{1}{2}\nabla^2\ell_i(\boldsymbol{\theta})\right), \tag{5}$$

For $\ell_i(\boldsymbol{\theta}) = \frac{1}{2}(y_i - \mathbf{x}_i^\top \boldsymbol{\theta})^2$, we get

$$\widetilde{\nabla}\mathbb{E}_{q_{\boldsymbol{\lambda}}}[\ell_i(\boldsymbol{\theta})] = \mathbb{E}_{q_{\boldsymbol{\lambda}}}\left(\mathbf{x}_i(\mathbf{x}_i^\top \boldsymbol{\theta} - y_i) - \mathbf{x}_i\mathbf{x}_i^\top \mathbf{m},\ \tfrac{1}{2}\mathbf{x}_i\mathbf{x}_i^\top\right) = \left(-\mathbf{x}_i y_i,\ \tfrac{1}{2}\mathbf{x}_i\mathbf{x}_i^\top\right). \tag{6}$$

For removal of i'th example, we denote $\boldsymbol{\lambda}_* = (\mathbf{S}_*\boldsymbol{\theta}_*,\ -\frac{1}{2}\mathbf{S}_*)$ and $\hat{\boldsymbol{\lambda}}_*^{\setminus i} = (\hat{\mathbf{S}}_*^{\setminus i}\hat{\boldsymbol{\theta}}_*^{\setminus i},\ -\frac{1}{2}\hat{\mathbf{S}}_*^{\setminus i})$. Then, plugging Eq. 6 into Eq. 4, we have the MPE for linear regression,

$$\hat{\mathbf{S}}_*^{\setminus i}\hat{\boldsymbol{\theta}}_*^{\setminus i} - \mathbf{S}_*\boldsymbol{\theta}_* = -\mathbf{x}_i y_i, \qquad -\tfrac{1}{2}\hat{\mathbf{S}}_*^{\setminus i} + \tfrac{1}{2}\mathbf{S}_* = \tfrac{1}{2}\mathbf{x}_i\mathbf{x}_i^\top. \tag{7}$$

We can rearrange this to get the deviation $\hat{\boldsymbol{\theta}}_*^{\setminus i} - \boldsymbol{\theta}_*$ over the mean to recover Eq. 2. ,

$$(\mathbf{S}_* - \mathbf{x}_i\mathbf{x}_i^\top)\hat{\boldsymbol{\theta}}_*^{\setminus i} - \mathbf{S}_*\boldsymbol{\theta}_* = -\mathbf{x}_i y_i \qquad \Longrightarrow\ \hat{\boldsymbol{\theta}}_*^{\setminus i} - \boldsymbol{\theta}_* = \boldsymbol{\Sigma}_*\mathbf{x}_i(\mathbf{x}_i^\top\hat{\boldsymbol{\theta}}_*^{\setminus i} - y_i) = \boldsymbol{\Sigma}_*\mathbf{x}_i\hat{e}_i^{\setminus i} \tag{8}$$

This completes the proof. ■

The right hand side requires $\hat{\boldsymbol{\theta}}_*^{\setminus i}$ which is not always available, but we can assume $\hat{e}_i^{\setminus i} \approx e_i$ which leads us to a score at a slightly different perturbation given in Eq. 27. In general, we can make such assumptions to simplify the computation.

For linear regression, the posterior approximation is exact and the MPE then recovers the exact deviations. This property holds in general for *all* exponential-family conjugate models, including mixture models, linear state-space models, probabilistic PCA, etc. A formal statement is given below; a proof is in App. B and an example on Beta-Bernoulli model is in App. C.

Theorem 2 *When $q_{\boldsymbol{\lambda}} = p(\boldsymbol{\theta}|\mathcal{D})$, the MPE estimate is equal to the exact deviations.*

3.2 Neural-network influence function as a special case of the MPE

We will now show that influence functions for any optimizer that satisfies second-order optimality condition at convergence can be derived as a special case of the MPE with Gaussian posterior. For example, in [26], the following influence function is derived for neural network loss function,

$$\hat{\boldsymbol{\theta}}_*^{\setminus i} - \boldsymbol{\theta}_* \approx \left(\nabla^2\mathcal{L}(\boldsymbol{\theta}_*)\right)^{-1}\nabla\ell_i(\boldsymbol{\theta}_*). \tag{9}$$

We will derive this as a special case of the MPE, and also propose a generalization which can be applied *during* training. Throughout, we will use a differentiable loss $\ell_i(y_i, f_i(\boldsymbol{\theta})) = -\log p(y_i|\sigma(f_i(\boldsymbol{\theta})))$ defined using an exponential-family $p(y|\sigma(f))$ for a scalar y and link function $\sigma(f)$ (for example, softmax). We denote by $f_i(\boldsymbol{\theta})$ the function output for the i'th input $\mathbf{x}_i$.

The BLR update for Gaussian posteriors takes a Newton-like form (see [22, Eq. 12] or [20]),

$$\mathbf{m}_t = \mathbf{m}_{t-1} - \rho\mathbf{S}_t^{-1}\,\mathbb{E}_{q_{\boldsymbol{\lambda}_{t-1}}}[\nabla\mathcal{L}(\boldsymbol{\theta})], \qquad \mathbf{S}_t = (1-\rho)\mathbf{S}_{t-1} + \rho\,\mathbb{E}_{q_{\boldsymbol{\lambda}_{t-1}}}[\nabla^2\mathcal{L}(\boldsymbol{\theta})] \tag{10}$$

Deep-learning algorithms such as RMSprop and Adam can be derived from this algorithm [22, Sec. 4.2] by using the Delta method. Essentially, we approximate the expectation of a function by the value at the mean, for instance, $\mathbb{E}_{q_{\boldsymbol{\lambda}}}[g(\boldsymbol{\theta})] \approx g(\mathbf{m})$ for an arbitrary function g; see [22, Eq. 13]. With this approximation, the BLR recovers a minimizer $\boldsymbol{\theta}_*$ of Eq. 1, that is, we get $\mathbf{m}_* = \boldsymbol{\theta}_*$ and $\mathbf{S}_* = \nabla^2\mathcal{L}(\boldsymbol{\theta}_*)$, similarly to the linear regression case shown in Theorem 1. We will use the same technique to derive influence functions of deep-learning optimizers from the MPE.

Proceeding similarly to Eq. 5 we get (suppressing the dependence on $\boldsymbol{\theta}$ for notational ease),

$$\widetilde{\nabla}\mathbb{E}_{q_{\boldsymbol{\lambda}}}[\ell_i] = \mathbb{E}_{q_{\boldsymbol{\lambda}}}\left(\nabla\ell_i - \nabla^2\ell_i\mathbf{m},\ \tfrac{1}{2}\nabla^2\ell_i\right) = \mathbb{E}_{q_{\boldsymbol{\lambda}}}\left(\nabla f_i e_i - \mathbf{H}_i\mathbf{m},\ \tfrac{1}{2}\mathbf{H}_i\right), \tag{11}$$

where $e_i(\boldsymbol{\theta}) = \sigma(f_i(\boldsymbol{\theta})) - y_i$ is the error and $\mathbf{H}_i(\boldsymbol{\theta}) = \nabla^2\ell_i(\boldsymbol{\theta})$ is the Hessian. Note that Eq. 6 is a special case of the general result in Eq. 11. Plugging in Eq. 4,

$$\hat{\mathbf{S}}_t^{\setminus i}\hat{\mathbf{m}}_t^{\setminus i} - \mathbf{S}_t\mathbf{m}_t = \mathbb{E}_{q_{\boldsymbol{\lambda}_t}}\left[\nabla f_i e_i - \mathbf{H}_i\mathbf{m}_t\right], \qquad -\tfrac{1}{2}\hat{\mathbf{S}}_t^{\setminus i} + \tfrac{1}{2}\mathbf{S}_t = \tfrac{1}{2}\mathbb{E}_{q_{\boldsymbol{\lambda}_t}}[\mathbf{H}_i], \tag{12}$$

and then we rearrange it to get the deviation over the mean, that is, $\hat{\mathbf{m}}_t^{\setminus i} - \mathbf{m}_t$,

$$\begin{aligned}
&\left(\mathbf{S}_t - \mathbb{E}_{q_{\boldsymbol{\lambda}_t}}[\mathbf{H}_i]\right)\hat{\mathbf{m}}_t^{\setminus i} - \mathbf{S}_t\mathbf{m}_t = \mathbb{E}_{q_{\boldsymbol{\lambda}_t}}\left[\nabla f_i e_i + \mathbf{H}_i\mathbf{m}_t\right] \\
\Longrightarrow\ &\left(\mathbf{S}_t - \mathbb{E}_{q_{\boldsymbol{\lambda}_t}}[\mathbf{H}_i]\right)\left(\hat{\mathbf{m}}_t^{\setminus i} - \mathbf{m}_t\right) = \mathbb{E}_{q_{\boldsymbol{\lambda}_t}}\left[\nabla f_i e_i\right], \\
\Longrightarrow\ &\hat{\mathbf{m}}_t^{\setminus i} - \mathbf{m}_t = \hat{\boldsymbol{\Sigma}}_t^{\setminus i}\mathbb{E}_{q_{\boldsymbol{\lambda}_t}}\left[\nabla f_i(\boldsymbol{\theta})e_i(\boldsymbol{\theta})\right]
\end{aligned} \tag{13}$$

If we use the Delta method and additionally assume that $\hat{\boldsymbol{\Sigma}}_t^{\setminus i} \approx \boldsymbol{\Sigma}_t$, we get the expression in the left hand side, which at convergence can be written in the form shown in the right hand side,

$$\hat{\boldsymbol{\theta}}_t^{\setminus i} - \boldsymbol{\theta}_t \approx \boldsymbol{\Sigma}_t \nabla f_i(\boldsymbol{\theta}_t) e_i(\boldsymbol{\theta}_t), \qquad \hat{\boldsymbol{\theta}}_*^{\setminus i} - \boldsymbol{\theta}_* \approx \left(\nabla^2 \mathcal{L}(\boldsymbol{\theta}_*)\right)^{-1} \nabla \ell_i(\boldsymbol{\theta}_*). \tag{14}$$

The second result is obtained using $\boldsymbol{\Sigma}_* = (\nabla^2 \mathcal{L}(\boldsymbol{\theta}_*))^{-1}$. The second expression in Eq. 14 is exactly the influence function derived in [26]. They use linearization to obtain the expression, but the MPE obtains similar results. The first expression can be seen as a generalization of [13] to Newton-like steps where their 'back-tracking' factor is replaced by its linear approximation $\boldsymbol{\Sigma}_t$. This simplifies the computation of their method.

Why do we recover existing influence functions from the MPE? This is because of the property of the BLR where natural-gradients are assigned to appropriate natural parameters [22, Sec. 1.2]. The two natural gradients are related to the gradient and Hessian respectively which are then connected to the sensitivity in the mean and covariance of the Gaussian. Essentially, the expressions in Eq. 13 and Eq. 14 hold whenever a Gaussian posterior is obtained using the BLR.

More importantly, the sensitivity measure is valid for iterations of the BLR, and can be directly applied to analyze sensitivity of any algorithm derived from it. This includes optimization algorithms such as gradient descent and Newton's method but also deep-learning optimizers such as RMSprop and Adam [24]. The proposed sensitivity measure is also valid for Bayesian deep-learning methods which build upon the BLR [20, 31, 27]. Note that these algorithms are related to various kinds of Gaussian approximations, some of which are more accurate than others. The posterior approximation directly affects the accuracy of the sensitivity measures but also the computation cost. In general, we expect there to be a trade-off between the accuracy and computation and a suitable method can be chosen accordingly.

3.3 Causes of influence for the Gaussian case

We now show that the bi-linear relationship in the influence function, seen in the case of linear regression (Eq. 2), holds in general for all influence functions derived using the Gaussian perturbation model. This is easy to show by using Taylor's approximation and Eq. 14,

$$f_{it}^{\setminus i} - f_{it} \approx \nabla f_i(\boldsymbol{\theta}_t)^\top (\boldsymbol{\theta}_t^{\setminus i} - \boldsymbol{\theta}_t) \approx \nabla f_i(\boldsymbol{\theta}_t)^\top \boldsymbol{\Sigma}_t \nabla f_i(\boldsymbol{\theta}_t) e_{it} = v_{it} e_{it}, \tag{15}$$

where we define $e_{it} = e_i(\boldsymbol{\theta}_t)$ and $f_{it} = f_i(\boldsymbol{\theta}_t)$, $f_{it}^{\setminus i} = f_i(\boldsymbol{\theta}_t^{\setminus i})$ to be the function outputs at $\boldsymbol{\theta}_t$ and $\boldsymbol{\theta}_t^{\setminus i}$ respectively. $v_{it} = \nabla f_i(\boldsymbol{\theta}_t)^\top \boldsymbol{\Sigma}_t \nabla f_i(\boldsymbol{\theta}_t)$ is the (linearized) prediction variance of $f_i(\boldsymbol{\theta})$. The variance is equal to the prediction variance of the error for linear models, but otherwise it is an estimate. A similar expression can be obtained for the predictions,

$$\sigma(f_{it}^{\setminus i}) - \sigma(f_{it}) \approx \sigma'(f_{it}) \nabla f_i(\boldsymbol{\theta}_t)^\top (\boldsymbol{\theta}_t^{\setminus i} - \boldsymbol{\theta}_t) \approx \sigma'(f_{it}) v_{it} e_{it}. \tag{16}$$

This expression clearly shows that the examples with high prediction error, variances and $\sigma'(f)$ are most likely to cause high sensitivity. We can also write this for removal of a group of examples in $\mathcal{M}$. Denoting the vector of $f_i(\boldsymbol{\theta}_t)$ for $i \in \mathcal{M}$ by $\mathbf{f}_{\mathcal{M},t}$ we get a similar expression,

$$\sigma(\mathbf{f}_{\mathcal{M},t}^{\setminus \mathcal{M}})) - \sigma(\mathbf{f}_{\mathcal{M},t}) \approx \boldsymbol{\Lambda}(\boldsymbol{\theta}_t) \mathbf{V}_{\mathcal{M}}(\boldsymbol{\theta}_t) \mathbf{e}_{\mathcal{M}}(\boldsymbol{\theta}_t) \approx \sum_{i \in \mathcal{M}} \sigma'(f_{it}) v_{it} e_{it}, \tag{17}$$

where $\boldsymbol{\Lambda}(\boldsymbol{\theta}_t)$ is a diagonal matrix containing all $\sigma'(f_i(\boldsymbol{\theta}_t))$, $\mathbf{V}_{\mathcal{M}}(\boldsymbol{\theta}_t)$ is the prediction covariance, and $\mathbf{e}_{\mathcal{M}}(\boldsymbol{\theta}_t)$ is the vector of prediction errors. The second approximation gives a cheaper expression which avoids building the covariance and instead uses the deviations with respect to each example.

3.4 Extensions and implications on computations

The bi-linear relationship also holds for other models, such as, Bayesian logistic regression, Gaussian processes, and their sparse variants. For example, for Bayesian logistic regression we use Eq. 13 to get a similar expression by noting that $\nabla f_i(\boldsymbol{\theta}) = \mathbf{x}_i$,

$$f_i(\boldsymbol{\theta}_t^{\setminus i}) - f_i(\boldsymbol{\theta}_t) \approx \mathbf{x}_i^\top (\boldsymbol{\theta}_t^{\setminus i} - \boldsymbol{\theta}_t) \approx \mathbf{x}_i^\top \boldsymbol{\Sigma}_t \mathbf{x}_i \mathbb{E}_{q_{\boldsymbol{\lambda}_t}}[e_i(\boldsymbol{\theta})] = v_{it} e_{it}, \tag{18}$$

where $v_{it} = \mathbf{x}_i^\top \boldsymbol{\Sigma}_t \mathbf{x}_i$ is the predictive variance for the function outputs and $e_{it} = \mathbb{E}_{q_{\boldsymbol{\lambda}_t}}[e_i(\boldsymbol{\theta})]$ is the prediction error. In general, whenever a Gaussian posterior is used, we expect to see similar causes of high influence. We show the derivation for sparse variational GP in App. D.

Non-differentiable $\mathcal{L}$ are also covered where we get $\hat{\mathbf{m}}_t^{\backslash i} - \mathbf{m}_t = \hat{\boldsymbol{\Sigma}}_t^{\backslash i} \nabla_{\mathbf{m}} \mathbb{E}_{q_{\boldsymbol{\lambda}_t}}[\ell_i(\boldsymbol{\theta})]$; see App. E for a derivation. This is the extension of Eq. 13. It is a much more principled approach than [26] who used an ad-hoc smoothing of the non-differentiable loss. The smoothing here is automatically done by using the posterior distribution.

What happens when we use non-Gaussian approximations? The principle is the same. The causes are determined by the sufficient statistics $\mathbf{T}(\boldsymbol{\theta})$. For Gaussians, there are two sufficient statistics for the mean and covariance respectively. Each statistics leads to a cause of high influence, for example, in Eq. 12, the deviation in the first natural parameter ($\mathbf{Sm}$) brings in the prediction error into play, while the deviations in the second natural parameters ($\mathbf{S}$) brings in prediction variance as another cause. In general, $\mathbf{T}(\boldsymbol{\theta})$ determines the causes of high influence, and its size determines the number of causes.

Finally, due to its connection to the BLR, the MPE is can be applied to all sorts of algorithms in fields such as deep learning, optimization, graphical models, Bayesian models, kernel methods etc; see App. F for deep learning. for For all these algorithms, we expect similar causes of high influence. Many unsupervised models can also be analyzed this way. The MPE unravels useful properties regarding sensitivity of generic ML models. For all these cases, the MPE also suggest that the estimation of sensitivity to be cheap. We can use byproducts of the algorithm (such as, prediction error and variance) to cheaply estimate the sensitivity. No additional computations are required.

3.5 Prediction of generalization performance

We will show that the MPE can be used to accurately predict the generalization performance at any time during training iterations. Essentially, we use Eq. 15 to estimate leave-one-out (LOO) performance on the training set,

$$\text{LOO}(\boldsymbol{\theta}_t) = \sum_{i=1}^{N} \ell\left(y_i, f_i(\boldsymbol{\theta}_t^{\backslash i})\right) \approx \sum_{i=1}^{N} \ell\left(y_i,\ f_{it} + v_{it} e_{it}\right). \tag{19}$$

The measure is very closely related to both marginal likelihood and sharpness both of which are known to be useful in predicting generalization [18, 10, 16]. Recently, [4] proposed similar approximations of leave-one-out loss to predict generalization, however they do not apply it to iterations. We can also estimate model's performance when a group or entire class of examples are removed,

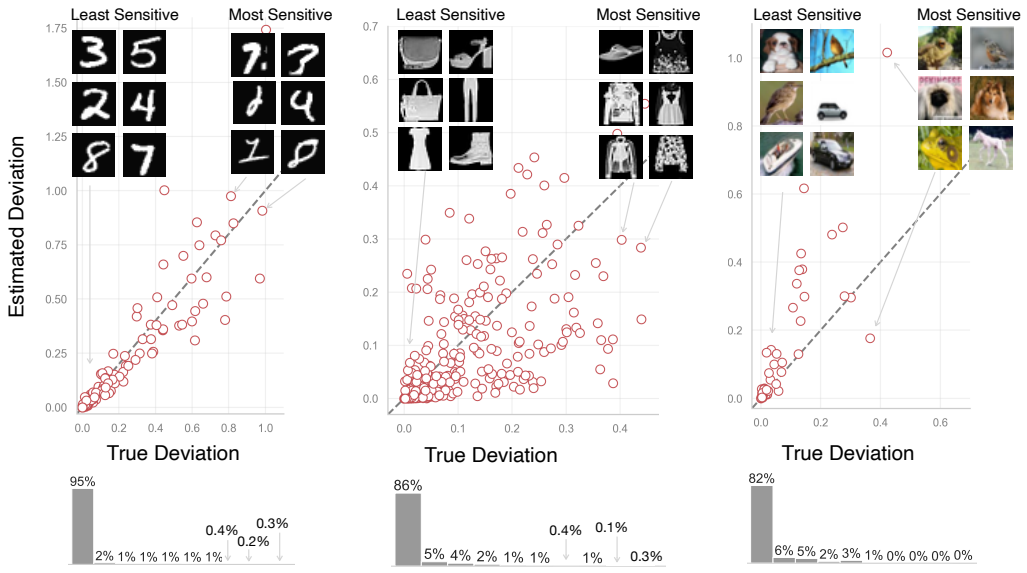
$$\text{LOCO}(\boldsymbol{\theta}_t, \mathcal{C}) = \sum_{i \in \mathcal{C}} \ell\left(y_i, f_i\left(\boldsymbol{\theta}_t^{\backslash \mathcal{C}}\right)\right) \approx \sum_{i \in \mathcal{C}} \ell\left(y_i,\ f_{it} + v_{it} e_{it}\right), \tag{20}$$

where $\mathcal{C} \subset \{1, \ldots, N\}$ is a set of training examples belonging to the same class. This quantity measures the model's sensitivity to various classes.

4 Experiments

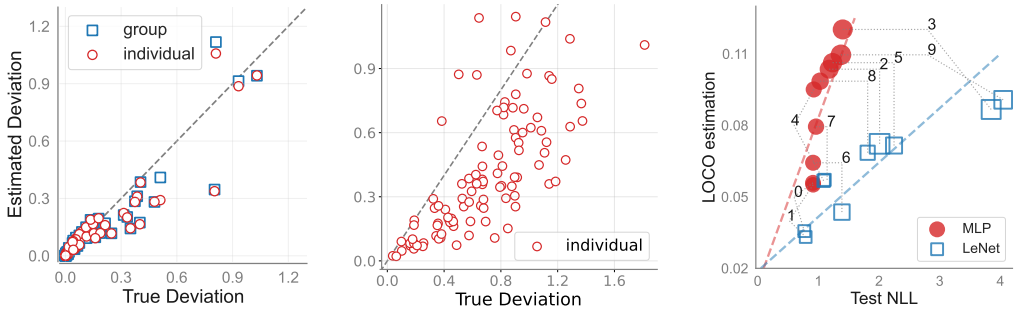
We show experimental results to demonstrate the usefulness of the MPE (4) in understanding sensitivity and generalization for deep learning. We show the following: **(i)** we verify that the estimated deviation in Eq. 16 correlates with the truth; **(ii)** we predict the effect of class removal on generalization error in deep learning; **(iii)** we estimate the cross-validation error for hyperparameter tuning; **(iv)** we study of sensitivities during training. More details of the training setup are in App. G.

Do estimated deviations correlate with the truth? We verify Eq. 16 and show that the estimated deviations correlate well with the truth. The results are summarized in Fig. 2, where we show deviations on three datasets. Each marker shows a point, where we clearly see that the ranking of the examples according to their sensitivity is maintained. We also show images of a few high and low sensitivity examples where we find that sensitive examples are often interesting cases, for example, clothes with unusual or elaborate design in FashionMNIST. In contrast, low sensitive examples are all regular looking. This is expected because highly sensitive points have either high prediction error



(a) MLP on MNIST (b) LeNet on FMNIST (c) CNN on CIFAR-10

Figure 2: We validate Eq. 16 to show that the estimated deviation correlates well with the truth. Highly sensitive examples are often interesting corner-cases, for example clothes with unusual or elaborate design in FashionMNIST as seen Fig. 2(b). On the other hand, the model is not sensitive to the examples shown in the lower-left corner, corresponding to examples that look more regular. The histograms of sensitivities (shown in the bottom) show that only a fraction of data have high sensitivities as suggested by the MPE.

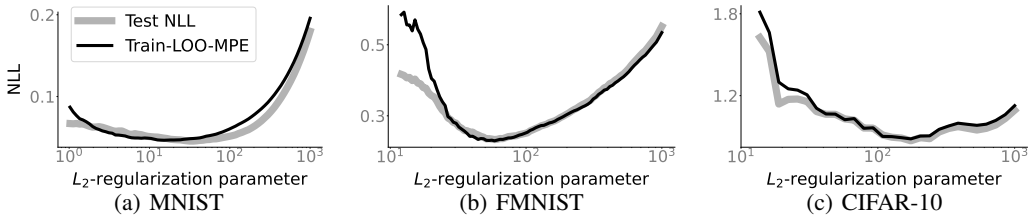


(a) MLP on USPS-3vs5, $|\mathcal{M}| = 16$ (b) MLP on MNIST, $|\mathcal{M}| = 64$ (c) Class removal on MNIST

Figure 3: Panel (a) and Panel (b) show that the estimated deviation for removal of groups of examples correlates well with the true deviations. Each point corresponds to a removed group of examples, the red points show the second approximation in Eq. 17, while the blue squares show the first one. In Panel (c) we show the estimated NLL for one-class-leave-out obtained by using Eq. 20 indicates the true class sensitivity in terms of test NLL when the class is removed. The models find some classes more sensitive than others.

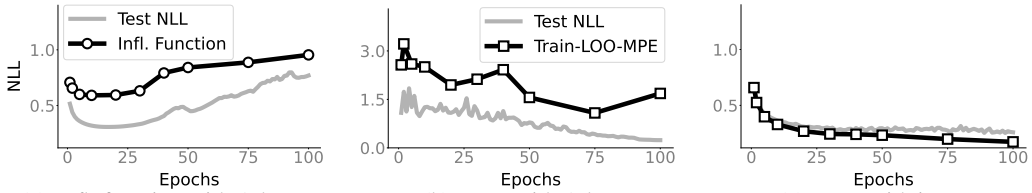
or variance. They can be examples that are perhaps mislabeled or ambiguous or may be they are not and perhaps the model views them as such.

We also study how the deviation for group removal can be estimated using Eq. 17. In Fig. 3(a) we consider the USPS dataset and show that Eq. 17 and its second approximation using a sum over the individual data points both correlate well with the truth. In Fig. 3(b) we show the group removal on MNIST where we show that the approximate version which also correlates well with the truth.



(a) MNIST (b) FMNIST (c) CIFAR-10

Figure 4: Panels (a), (b) and (c) show that our method can faithfully estimate the LOO-CV curve for predicting generalization and tuning of the L_2-regularization parameter on MNIST, FMNIST and CIFAR-10.



(a) Infl. function with AdamW (b) MPE with Adam (c) MPE with iBLR

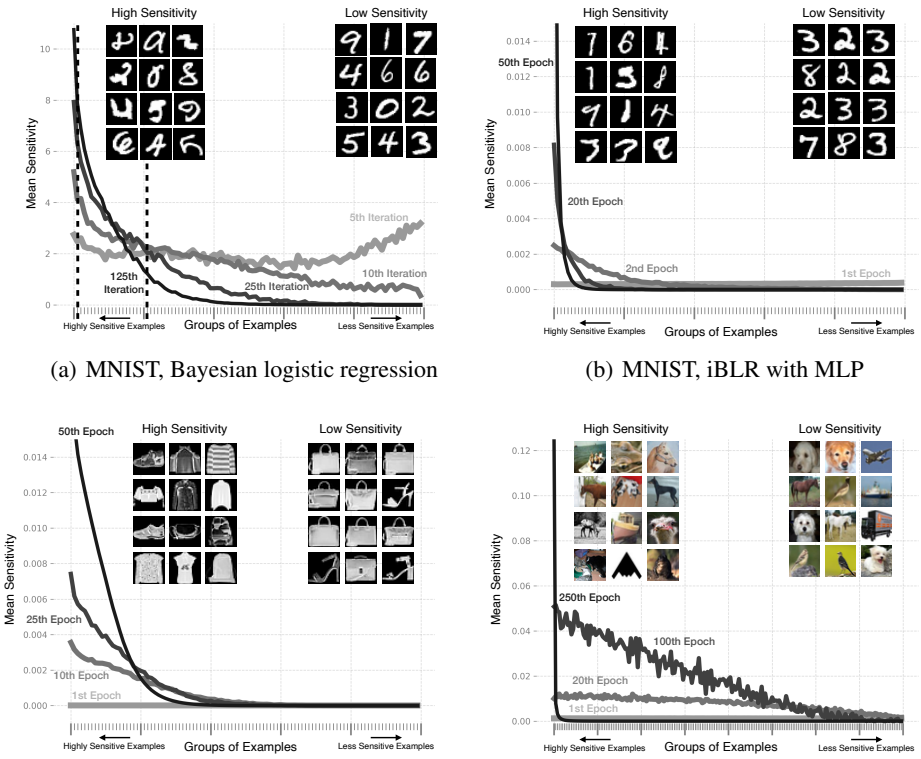
Figure 5: Panel (a) shows an estimate of the test NLL using influence function during training of a LeNet5 on FMNIST. We use a generalized Gauss-Newton approximation of the Hessian with diagonal matrix structure. Panels (b) and (c) display the Train-LOO estimate obtained from MPE. The results suggest that MPE with iBLR can faithfully estimate the test NLL and improves upon applying influence function. It performs better than the estimation with MPE that computes variances heuristically based on the Adam optimizer.

Predicting the effect of class removal on generalization: We use the MPE to predict class sensitivity: change in the test error when a whole class is removed from train data. Typically, one would have to retrain the whole model, but we show that this can be avoided by using Eq. 20 to predict the generalization error. In Fig. 1(c) and Fig. 3(c) we show two models on MNIST and FMNIST respectively. We see that the test NLL after class removal correlates well with the estimated NLL obtained with Eq. 20. We also see that the models are more sensitive to some classes than others. In MNIST (see Fig. 3(c)), *3*, *9*, and *5* are the most sensitive classes, while *0*, *1* are not sensitive. In FashionMNIST (see Fig. 1(c)) *Shirt*, *Pullover* are the most sensitive classes, while *Bag*, *Trouser* are the least sensitive ones.

Estimating the leave-one-out cross-validation curves for hyperparameter tuning: Here, we use the MPE to estimate the leave-one-out cross-validation (LOO-CV) curves for hyperparameter tuning. We consider the tuning of the regularization parameter δ of an L_2 regularizer $\mathcal{R}(\boldsymbol{\theta}) = \frac{1}{2}\delta\|\boldsymbol{\theta}\|^2$. We retrain the model for each δ setting and for each of those compute the LOO-CV estimate on training data using Eq. 19.Fig. 4 shows results for three models trained on MNIST, FMNIST and CIFAR-10 respectively. The estimated negative log-likelihood (NLL) curve matches the test NLL.

The match is almost perfect here and we do not see such good correlations in other studies [18, 10, 16] but our result supports their conclusions that sensitivity based measures can work well to predict generalization performance. The cost of computing these curves is almost negligible compared to cross-validation where we have train many more models for each setting of the hyperparameter.

How do sensitivities evolve during training: We use the MPE to analyze the evolution of sensitivities during training. We consider Bayesian logistic regression in Fig. 6(a) and neural network classification with the iBLR optimizer inFig. 6(b), Fig. 6(c) and Fig. 6(d). We run the BLR iterates of Eq. 10. For Bayesian logistic regression we estimate deviations using Eq. 16 where we use expectations for the residual as in Eq. 18 and plot them for various iterations. For iBLR we also use Eq. 16 and plot for several selected epochs. For better visualization, we first sort the examples according to the sensitivities of the converged model, sorted in decreasing order of sensitivity. Then, we plot the average sensitivies of examples, where we grouped examples with similar sensitivities together into



(a) MNIST, Bayesian logistic regression

(b) MNIST, iBLR with MLP

(c) FMNIST, iBLR with LeNet5

(d) CIFAR-10, iBLR with ResNet–20

Figure 6: We show the evolution of sensitivies during training for Bayesian logistic regression in panel (a) and the iBLR optimizer in the panels (b), (c) and (d). We see that, with training, the model becomes more and more sensitive to a small fraction of data. We also show some examples of high and low sensitivies.

overall 200 groups. In Fig. 6(a), we see that in the beginning the sensitivities of all examples are almost the same (5th iteration), but as training proceeds, the model becomes more and more sensitive to a small fraction of examples. In the end, the distribution of sensitivities is heavy-tailed where only around 20% of data has reasonable sensitivities of values > 1, while $< 2\%$ are extremely sensitive with values > 8 as indicated by the vertical dashed lines. The experiments with iBLR on MNIST, FMNIST and CIFAR-10 all exhibit the same increasing heavy-tailedness of the sensitivities over the training. Notably, for the ResNet–20 on CIFAR-10, the distribution is very heavy-tailed after the 250-th epoch. For all experiments we show images of high and low sensitivity examples, where we find highly sensitive examples to be the odd ones while low sensitivity examples to be regular looking.

Predicting generalization during the training: We predict the test loss during training by evaluating Eq. 19. We use the Train-LOO estimate with MPE (Train-LOO-MPE) and the influence function as a special case of the MPE. Both estimates are based on the training data. In Fig. 5 we show the results for a LeNet5 trained on FMNIST. The experiments indicate that the estimate obtained from MPE can predict the test loss during training faithfully. Such information can be used as a diagnostic for stopping training to avoid overfitting. In the MPE experiments, we compare Adam and iBLR for the computation of the variance in Eq. 19. For details we refer to App. F. The results suggest that improved variance computation leads to a better quality of the prediction of generalization. The heuristic estimate based on the Adam deep-learning optimizer [24] in Fig. 5(b) reflects the general trend of the test loss, but exhibits inaccuracies, for instance in the end of training. The variance estimation with the iBLR optimizer [27] in Fig. 5(c) leads to faithful prediction of the test loss, despite a diagonal covariance structure. iBLR avoids overfitting in the end of training. Application of the influence function in Fig. 5(a) captures the trend of the test loss. There is, however, a difference in magnitude between the test loss and the prediction. We use a generalized Gauss-Newton approxi-

mation to the Hessian with diagonal matrix structure. Details on the implementation of the influence function can be found in App. G.5. Fig. 10 additionally reports the test errors over the training for the FMNIST experiment. In Fig. 1(b) and Fig. 11 we show a result using MPE with iBLR for a Resnet–20 on CIFAR10. The method faithfully predicts the test loss. Further results on MNIST (MLP, LeNet5) and CIFAR10 (CNN) are reported in Fig. 8 and Fig. 9, respectively.

5 Discussion

We present the memory-perturbation equation by building upon the BLR framework. The equation suggests to take a step in the direction of the natural gradient of the perturbed examples. Using the MPE framework, we unify existing influence measures, generalize them to a wide variety of problems, and unravel useful properties regarding sensitivity. We also show that sensitivity estimation can be done cheaply and use this to predict generalization performance.

An interesting avenue for future research is to apply the method to larger models and real-world problems. We also need to understand how our generalization measure compares to other methods, such as those considered in [18]. We would also like to understand the effect of various posterior approximations. Another interesting direction is to apply the method to non-Gaussian cases, for example, to study ensemble methods in deep learning with mixture models.

References

[1] Vincent Adam, Paul Chang, Mohammad Emtiyaz E Khan, and Arno Solin. Dual parameterization of sparse variational Gaussian processes. *Advances in Neural Information Processing Systems*, 34:11474–11486, 2021. 16

[2] Chirag Agarwal, Daniel D'Souza, and Sara Hooker. Estimating example difficulty using variance of gradients. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2022. 3

[3] Devansh Arpit, Stanisław Jastrzębski, 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 *International Conference on Machine Learning*, 2017. 3

[4] Gregor Bachmann, Thomas Hofmann, and Aurélien Lucchi. Generalization through the lens of leave-one-out error. In *International Conference on Learning Representations*, 2022. 2, 7

[5] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. 3

[6] R Dennis Cook. Detection of influential observation in linear regression. *Technometrics*, 19(1):15–18, 1977. 1, 3

[7] R Dennis Cook and Sanford Weisberg. Characterizations of an empirical influence function for detecting influential cases in regression. *Technometrics*, 22(4):495–508, 1980. 1, 2, 3

[8] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995. 4

[9] Erik Daxberger, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig. Laplace redux-effortless bayesian deep learning. *Advances in Neural Information Processing Systems*, 34:20089–20103, 2021. 18, 21

[10] Gintare Karolina Dziugaite and Daniel M Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2017. 2, 7, 9

[11] Vitaly Feldman and Chiyuan Zhang. What neural networks memorize and why: Discovering the long tail via influence estimation. In *Advances in Neural Information Processing Systems*, 2020. 3

[12] Wing K Fung and CW Kwan. A note on local influence based on normal curvature. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 59(4):839–843, 1997. 1, 3

[13] Satoshi Hara, Atsushi Nitanda, and Takanori Maehara. Data cleansing for models trained with SGD. In *Advances in Neural Information Processing Systems*, 2019. 3, 6

[14] Hrayr Harutyunyan, Alessandro Achille, Giovanni Paolini, Orchid Majumder, Avinash Ravichandran, Rahul Bhotika, and Stefano Soatto. Estimating informativeness of samples with smooth unique information. In *International Conference on Learning Representations*, 2021. 3

[15] James Hensman, Nicolo Fusi, and Neil D Lawrence. Gaussian processes for big data. In *Uncertainty in Artificial Intelligence*, page 282. Citeseer, 2013. 16

[16] Alexander Immer, Matthias Bauer, Vincent Fortuin, Gunnar Rätsch, and Khan Mohammad Emtiyaz. Scalable marginal likelihood estimation for model selection in deep learning. In *International Conference on Machine Learning*, 2021. 2, 7, 9

[17] Louis A Jaeckel. *The infinitesimal jackknife*. Bell Telephone Laboratories, 1972. 1, 3

[18] Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic generalization measures and where to find them. In *International Conference on Learning Representations*, 2020. 2, 7, 9, 11

[19] Angelos Katharopoulos and Francois Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *International Conference on Machine Learning*, 2018. 3

[20] Mohammad Khan, Didrik Nielsen, Voot Tangkaratt, Wu Lin, Yarin Gal, and Akash Srivastava. Fast and scalable Bayesian deep learning by weight-perturbation in Adam. In *International Conference on Machine Learning*, 2018. 5, 6, 16, 17

[21] Mohammad Emtiyaz Khan and Wu Lin. Conjugate-computation variational inference: Converting variational inference in non-conjugate models to inferences in conjugate models. In *International Conference on Artificial Intelligence and Statistics*, 2017. 16

[22] Mohammad Emtiyaz Khan and Håvard Rue. The Bayesian learning rule. *arXiv:2107.04562*, 2021. 1, 4, 5, 6, 17

[23] George S Kimeldorf and Grace Wahba. A correspondence between bayesian estimation on stochastic processes and smoothing by splines. *The Annals of Mathematical Statistics*, 41(2):495–502, 1970. 4

[24] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. 6, 10, 17

[25] Pang Wei Koh, Kai-Siang Ang, Hubert Teo, and Percy S Liang. On the accuracy of influence functions for measuring group effects. In *Advances in Neural Information Processing Systems*, 2019. 2, 3

[26] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, 2017. 1, 3, 4, 5, 6, 7, 17, 21

[27] Wu Lin, Mark Schmidt, and Mohammad Emtiyaz Khan. Handling the positive-definite constraint in the Bayesian learning rule. In *International Conference on Machine Learning*, 2020. 6, 10, 17

[28] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *ICLR*, 2019. 19, 21

[29] James Martens. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014. 21

[30] Roman Novak, Yasaman Bahri, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and generalization in neural networks: an empirical study. In *International Conference on Learning Representations*, 2018. 1, 3

[31] Kazuki Osawa, Siddharth Swaroop, Mohammad Emtiyaz E Khan, Anirudh Jain, Runa Eschenhagen, Richard E Turner, and Rio Yokota. Practical deep learning with Bayesian principles. In *Advances in Neural Information Processing Systems*, 2019. 6

[32] Mansheej Paul, Surya Ganguli, and Gintare Karolina Dziugaite. Deep learning on a data diet: Finding important examples early in training. In *Advances in Neural Information Processing Systems*, 2021. 3

[33] Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. In *Advances in Neural Information Processing Systems*, 2020. 3

[34] Hugh Salimbeni, Stefanos Eleftheriadis, and James Hensman. Natural gradients in practice: Non-conjugate variational inference in Gaussian process models. In *International Conference on Artificial Intelligence and Statistics*, pages 689–697. PMLR, 2018. 16

[35] Frank Schneider, Lukas Balles, and Philipp Hennig. DeepOBS: A deep learning optimizer benchmark suite. In *International Conference on Learning Representations*, 2019. 18

[36] Bernhard Schölkopf, Ralf Herbrich, and Alex J Smola. A generalized representer theorem. In *International conference on computational learning theory*, pages 416–426. Springer, 2001. 4

[37] Saurabh Singh and Shankar Krishnan. Filter response normalization layer: Eliminating batch dependence in the training of deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020. 20

[38] Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J. Gordon. An empirical study of example forgetting during deep neural network learning. In *International Conference on Learning Representations*, 2019. 3

[39] Hongtu Zhu, Joseph G. Ibrahim, Sikyum Lee, and Heping Zhang. Perturbation selection and influence measures in local influence analysis. *The Annals of Statistics*, 35(6):2565 – 2588, 2007. 1

A Influence Function for linear regression

We define a perturbation model as follows with $\epsilon_i \in \mathbb{R}$:

$$\boldsymbol{\theta}_*^{\epsilon_i} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) - \epsilon_i \ell_i(\boldsymbol{\theta}).$$

The solution has a closed-form expression,

$$\boldsymbol{\theta}_*^{\epsilon_i} = \left(\boldsymbol{\Sigma}_*^{-1} - \epsilon_i \mathbf{x}_i \mathbf{x}_i^\top\right)^{-1} \left(\mathbf{X}^\top \mathbf{y} - \epsilon_i y_i \mathbf{x}_i\right). \tag{21}$$

To derive Eq. 2, we express this in terms of $\boldsymbol{\theta}_*$ by using the Sherman-Morrison formula,

$$\begin{aligned}
\boldsymbol{\theta}_*^{\epsilon_i} &= \left(\boldsymbol{\Sigma}_* + \frac{\epsilon_i \boldsymbol{\Sigma}_* \mathbf{x}_i \mathbf{x}_i^\top \boldsymbol{\Sigma}_*}{1 - \epsilon_i \mathbf{x}_i^\top \boldsymbol{\Sigma}_* \mathbf{x}_i}\right) \left(\mathbf{X}^\top \mathbf{y} - \epsilon_i y_i \mathbf{x}_i\right) \\
&= \boldsymbol{\Sigma}_* \mathbf{X}^\top \mathbf{y} + \epsilon_i \boldsymbol{\Sigma}_* \mathbf{x}_i \left[\frac{\mathbf{x}_i^\top \boldsymbol{\Sigma}_* \mathbf{X}^\top \mathbf{y}}{1 - \epsilon_i \mathbf{x}_i^\top \boldsymbol{\Sigma}_* \mathbf{x}_i} - \frac{\epsilon_i y_i \mathbf{x}_i^\top \boldsymbol{\Sigma}_* \mathbf{x}_i}{1 - \epsilon_i \mathbf{x}_i^\top \boldsymbol{\Sigma}_* \mathbf{x}_i} - y_i\right] \\
&= \boldsymbol{\theta}_* + \epsilon_i \boldsymbol{\Sigma}_* \mathbf{x}_i \left[\frac{\mathbf{x}_i^\top \boldsymbol{\theta}_*}{1 - \epsilon_i v_i} - \frac{\epsilon_i y_i v_i}{1 - \epsilon_i v_i} - y_i\right] \\
&= \boldsymbol{\theta}_* + \epsilon_i \boldsymbol{\Sigma}_* \mathbf{x}_i \left[\frac{\mathbf{x}_i^\top \boldsymbol{\theta}_* - y_i}{1 - \epsilon_i v_i}\right] \\
&= \boldsymbol{\theta}_* + \boldsymbol{\Sigma}_* \mathbf{x}_i \frac{\epsilon_i e_i}{1 - \epsilon_i v_i}.
\end{aligned} \tag{22}$$

where in line 3 we substitute $v_i = \mathbf{x}_i^\top \boldsymbol{\Sigma}_* \mathbf{x}_i$ and $\boldsymbol{\theta}_* = \boldsymbol{\Sigma}_* \mathbf{X}^\top \mathbf{y}$ and in line 5 we use $e_i = \mathbf{x}_i^\top \boldsymbol{\theta}_* - y_i$.

To derive Eq. 2, we use $\epsilon_i = 1$ and simply write,

$$\boldsymbol{\theta}_*^{\backslash i} - \boldsymbol{\theta}_* = \boldsymbol{\Sigma}_* \mathbf{x}_i \frac{e_i}{1 - v_i} = \boldsymbol{\Sigma}_* \mathbf{x}_i e_i^{\backslash i} \tag{23}$$

$$f_i(\boldsymbol{\theta}_*^{\backslash i}) - f_i(\boldsymbol{\theta}_*) = \mathbf{x}_i^\top (\boldsymbol{\theta}_*^{\backslash i} - \boldsymbol{\theta}_*) = \mathbf{x}_i^\top \boldsymbol{\Sigma} \mathbf{x}_i \frac{e_i}{1 - v_i} = v_i \frac{e_i}{1 - v_i} = v_i e_i^{\backslash i}, \tag{24}$$

where we used the relation $e_i^{\backslash i} = e_i / (1 - v_i)$ which we proceed to show,

$$e_i^{\backslash i} = \mathbf{x}_i^\top \boldsymbol{\theta}_*^{\backslash i} - y_i = \mathbf{x}_i^\top \left(\boldsymbol{\theta}_* + \boldsymbol{\Sigma}_* \mathbf{x}_i \frac{e_i}{1 - v_i}\right) - y_i = \mathbf{x}_i^\top \boldsymbol{\theta}_* + \frac{v_i}{1 - v_i} e_i - y_i = \frac{e_i}{1 - v_i}. \tag{25}$$

An alternate way to derive it (in a more general form) is by taking derivatives of $\boldsymbol{\theta}_*^{\epsilon_i}$ in Eq. 22:

$$\frac{\partial \boldsymbol{\theta}_*^{\epsilon_i}}{\partial \epsilon_i} = \boldsymbol{\Sigma}_* \mathbf{x}_* \frac{e_i}{(1 - \epsilon_i v_i)^2} \tag{26}$$

Taking $\epsilon_i = 1$ gives us Eq. 2, but we can get other influence measures by evaluating, for example, at $\epsilon_i = 0$, as shown below:

$$\left.\frac{\partial \boldsymbol{\theta}_*^{\epsilon_i}}{\partial \epsilon_i}\right|_{\epsilon_i = 0} = \boldsymbol{\Sigma}_* \mathbf{x}_i e_i, \qquad \left.\frac{\partial f_i(\boldsymbol{\theta}_*^{\epsilon_i})}{\partial \epsilon_i}\right|_{\epsilon_i = 0} = \mathbf{x}_i^\top \left.\frac{\partial \boldsymbol{\theta}_*^{\epsilon_i}}{\partial \epsilon_i}\right|_{\epsilon_i = 0} = \mathbf{x}_i^\top \boldsymbol{\Sigma}_* \mathbf{x}_i e_i = v_i e_i \tag{27}$$

where Eq. 27 follows from straightforward application of the chain rule. Here too we get a bi-linear relationship of the influence measure with respect to prediction variance v_i and prediction error e_i.

B Proof of Theorem 2

The exact deviation for removing an example i amounts to dividing out its likelihood $p(\mathcal{D}_i|\boldsymbol{\theta})$ from the exact posterior $p(\boldsymbol{\theta}|\mathcal{D})$ and renormalizing,

$$p(\boldsymbol{\theta}|\mathcal{D}^{\backslash i}) \propto \frac{p(\boldsymbol{\theta}|\mathcal{D})}{p(\mathcal{D}_i|\boldsymbol{\theta})}. \tag{28}$$

Now in the conjugate-setting of the theorem, our posterior $p(\boldsymbol{\theta}|\mathcal{D}) = q_{\boldsymbol{\lambda}}(\boldsymbol{\theta}) \propto \exp(\langle \mathbf{T}(\boldsymbol{\theta}), \boldsymbol{\lambda} \rangle)$ is an exponential family and our likelihoods $p(\mathcal{D}_i|\boldsymbol{\theta}) \propto \exp(-\ell_i(\boldsymbol{\theta}))$ are of the conjugate form $\ell_i(\boldsymbol{\theta}) = -\langle \tilde{\boldsymbol{\lambda}}_i, \mathbf{T}(\boldsymbol{\theta}) \rangle + \text{const}$. Under this assumption, Eq. 28 can be written as follows:

$$p(\boldsymbol{\theta}|\mathcal{D}^{\backslash i}) = q_{\boldsymbol{\lambda}^{\backslash i}}(\boldsymbol{\theta}) \propto \exp(\langle \boldsymbol{\lambda} - \tilde{\boldsymbol{\lambda}}_i, \mathbf{T}(\boldsymbol{\theta}) \rangle). \tag{29}$$

Since two minimal and regular exponential families are only equal if their natural-parameters are equal, this equation can be equivalently written as follows:

$$\boldsymbol{\lambda}^{\backslash i} - \boldsymbol{\lambda} = -\tilde{\boldsymbol{\lambda}}_i. \tag{30}$$

Now noticing that we have

$$-\tilde{\boldsymbol{\lambda}}_i = \nabla_{\boldsymbol{\mu}} \langle \boldsymbol{\mu}, -\tilde{\boldsymbol{\lambda}}_i \rangle = \nabla_{\boldsymbol{\mu}} \langle \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})}[\mathbf{T}(\boldsymbol{\theta})], -\tilde{\boldsymbol{\lambda}}_i \rangle = \tilde{\nabla} \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})}[\ell_i(\boldsymbol{\theta})], \tag{31}$$

we arrive exactly at our MPE in Eq. 4, where we used linearity of the expectation and the definition of ℓ_i in the last step of Eq. 31.

C MPE for Beta-Bernoulli conjugate model

Using the Beta-Bernoulli model as an example, we demonstrate that for conjugate models, the MPE recovers the exact deviations. Let us denote the prior as $p(\theta) = \text{Beta}(\theta|\alpha_0, \beta_0)$ and per-example likelihood as $p(y_i|\theta) = \text{Ber}(y_i|\theta)$. Then we are ready to introduce the posterior over the full dataset $\mathcal{D}$, $q_{\boldsymbol{\lambda}_*}(\theta) = \text{Beta}(\theta|\alpha_*, \beta_*)$, as well as the posterior over a perturbed dataset $\mathcal{D}^{\backslash i}$ where the i'th example is removed, $q_{\boldsymbol{\lambda}_*^{\backslash i}}(\theta) = \text{Beta}(\theta|\alpha_*^{\backslash i}, \beta_*^{\backslash i})$, whose parameters are given by:

$$\alpha_* = \alpha_0 + \sum_{j=1}^{N} y_j - 1, \qquad \beta_* = \beta_0 - \sum_{j=1}^{N} y_j + N - 1 \tag{32}$$

$$\alpha_*^{\backslash i} = \alpha_0 + \sum_{\substack{j=1,\\ j\neq i}}^{N} y_j - 1, \qquad \beta_*^{\backslash i} = \beta_0 - \sum_{\substack{j=1,\\ j\neq i}}^{N} y_j + N - 2 \tag{33}$$

Then the deviations can be simply obtained,

$$\alpha_*^{\backslash i} - \alpha_* = -y_i, \qquad \beta_*^{\backslash i} - \beta_* = y_i - 1 \tag{34}$$

We now proceed to show that Eq. 34 can be straightforwardly obtained using the MPE. For the Beta distribution $q_{\boldsymbol{\lambda}}(\theta) = \text{Beta}(\theta|\alpha, \beta)$, we have $\boldsymbol{\lambda} = (\alpha - 1, \beta - 1)$ and $\boldsymbol{\mu} = \mathbb{E}_{q_{\boldsymbol{\lambda}}}(\log\theta, \log(1-\theta))$. Due to conjugacy, the exact posterior is obtained by running a single update of the BLR with $\rho = 1$ in Eq. 3. It turns out for this model, the deviations in the natural-parameter space of Eq. 4 simplify to the deviations considered in Eq. 34:

$$\begin{aligned} \hat{\boldsymbol{\lambda}}_*^{\backslash i} - \boldsymbol{\lambda}_* = \boldsymbol{\lambda}_*^{\backslash i} - \boldsymbol{\lambda}_* &= \left(\alpha_*^{\backslash i} - 1 - (\alpha_* - 1),\ \ \beta_*^{\backslash i} - 1 - (\beta_* - 1)\right) \\ &= \left(\alpha_*^{\backslash i} - \alpha_*,\ \ \beta_*^{\backslash i} - \beta_*\right) \end{aligned} \tag{35}$$

The natural gradients can be easily obtained by exploiting linearity in $\boldsymbol{\mu}$ similar to Eq. 6. Defining $\ell_i(\theta) := -\log p(y_i|\theta)$, we obtain:

$$\begin{aligned} \widetilde{\nabla} \mathbb{E}_{q_{\boldsymbol{\lambda}_*}}[\ell_i(\theta)] &= \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\lambda}_*}}[-\log\left\{\theta^{y_i}(1-\theta)^{1-y_i}\right\}] \\ &= \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\lambda}_*}}[-y_i \log\theta - (1-y_i)\log(1-\theta)] \\ &= \nabla_{\boldsymbol{\mu}}[-\mathbb{E}_{q_{\boldsymbol{\lambda}_*}}[\log\theta]\, y_i + \mathbb{E}_{q_{\boldsymbol{\lambda}_*}}[\log(1-\theta)]\,(y_i - 1)] \\ &= (-y_i, y_i - 1)\,. \end{aligned} \tag{36}$$

Combining Eq. 35 and Eq. 36, we recover Eq. 34.

D MPE for Sparse Variational Gaussian Process

Sparse variational GP (SVGP) methods optimize the following variational objective to find a Gaussian posterior approximation $q_{\boldsymbol{\lambda}}(\mathbf{u})$ over function values $\mathbf{u} := (f(\mathbf{z}_1), f(\mathbf{z}_2), \ldots, f(\mathbf{z}_M))$ where $\mathcal{Z} := (\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_M)$ is the set of inducing inputs with $M \ll N$:

$$\underline{\mathcal{L}}(\mathbf{m}, \boldsymbol{\Sigma}, \mathcal{Z}, \boldsymbol{\phi}) := \sum_{i=1}^{N} \mathbb{E}_{q_{\boldsymbol{\lambda}}(f_i)} \left[\log p(y_i|f_i)\right] - \mathbb{D}_{\mathrm{KL}}(q_{\boldsymbol{\lambda}}(\mathbf{u}) \,\|\, p(\mathbf{u})) \tag{37}$$

where $p(\mathbf{u}) := \mathcal{N}(\mathbf{u}|\mathbf{0}, \mathbf{K_{uu}})$ is the prior with $\mathbf{K_{uu}}$ as the covariance function $\kappa(\cdot,\cdot')$ evaluated at $\mathcal{Z}$, $q_{\boldsymbol{\lambda}}(f_i) = \mathcal{N}(f_i|\mathbf{a}_i^\top \mathbf{m}, \mathbf{a}_i^\top \boldsymbol{\Sigma} \mathbf{a}_i + \sigma_i^2)$ is the posterior marginal of $f_i = f(\mathbf{x}_i)$ with $\mathbf{a}_i := \mathbf{K_{uu}}^{-1}\mathbf{k}_{\mathbf{u}i}$ and $\sigma_i^2 := \kappa_{ii} - \mathbf{a}_i^\top \mathbf{K_{uu}} \mathbf{a}_i$ as the noise variance of f_i conditioned on $\mathbf{u}$. The objective is also used to optimize hyperparameters $\boldsymbol{\phi}$ and inducing input set $\mathcal{Z}$.

We can optimize Eq. 37 using the BLR for which the resulting update is identical to the variational online-newton (VON) algorithm (see Eqs. 7 and 8 in [20]):

$$\mathbf{S}_{t+1} = (1-\rho)\mathbf{S}_t + \rho \left[\mathbf{A}^\top \mathrm{diag}(\boldsymbol{\beta}_t)\mathbf{A} + \mathbf{K_{uu}}^{-1}\right] \tag{38}$$

$$\begin{aligned}
\mathbf{m}_{t+1} &= \mathbf{S}_{t+1}^{-1}\left[(1-\rho)\mathbf{S}_t\mathbf{m}_t - \rho\left(\mathbf{A}^\top \mathbf{e}_t - \mathbf{A}^\top \mathrm{diag}(\boldsymbol{\beta}_t)\mathbf{A}\mathbf{m}_t\right)\right] \\
&= \mathbf{S}_{t+1}^{-1}\left[\left((1-\rho)\mathbf{S}_t + \rho\mathbf{A}^\top \mathrm{diag}(\boldsymbol{\beta}_t)\mathbf{A}\right)\mathbf{m}_t - \rho\mathbf{A}^\top \mathbf{e}_t\right] \\
&= \mathbf{S}_{t+1}^{-1}\left[\left(\mathbf{S}_{t+1} - \rho\mathbf{K_{uu}}^{-1}\right)\mathbf{m}_t - \rho\mathbf{A}^\top \mathbf{e}_t\right] \qquad (39) \\
&= \mathbf{S}_{t+1}^{-1}\left[\mathbf{S}_{t+1}\mathbf{m}_t - \rho\left(\mathbf{A}^\top \mathbf{e}_t + \mathbf{K_{uu}}^{-1}\mathbf{m}_t\right)\right] \\
&= \mathbf{m}_t - \rho\mathbf{S}_{t+1}^{-1}\left[\mathbf{A}^\top \mathbf{e}_t + \mathbf{K_{uu}}^{-1}\mathbf{m}_t\right]
\end{aligned}$$

where $\mathbf{A}$ is a matrix with $\mathbf{a}_i^\top$ as rows, and $\mathbf{e}_t$ and $\boldsymbol{\beta}_t$ are vectors of e_{it} and β_{it} respectively computed as follows:

$$e_{it} = \mathbb{E}_{q_{\boldsymbol{\lambda}_t}(f_i)}[-\nabla_{f_i} \log p(y_i|f_i)], \qquad \beta_{it} = \mathbb{E}_{q_{\boldsymbol{\lambda}_t}(f_i)}[-\nabla^2_{f_i f_i} \log p(y_i|f_i)] \tag{40}$$

These result from expressing the natural gradient in terms of the gradient and Hessian of the log-likelihood. For Gaussian likelihood, the updates in Eqs. 38 and 39 coincide with the method of [15], and for non-Gaussian likelihood they are similar to the natural-gradient method by [34], but we use the specific parameterization of [21]. An alternate update rule in terms of site parameters is given by [1] (see Eqs. 22-24).

We are now ready to write the MPE,

$$\widetilde{\nabla}\mathbb{E}_{q_{\boldsymbol{\lambda}_t}(f_i)}[-\log p(y_i|f_i)] = \left((e_{it} - \beta_{it}\mathbf{a}_i^\top \mathbf{m}_*)\mathbf{a}_i, \;\; \tfrac{1}{2}\beta_{it}\mathbf{a}_i\mathbf{a}_i^\top\right). \tag{41}$$

This used Eqs. 38 and 39 although it can also be derived starting from the tied parameterization in [1]. Next, plugging Eq. 41 into Eq. 4,

$$\hat{\mathbf{S}}_t^{\setminus i}\hat{\mathbf{m}}_t^{\setminus i} - \mathbf{S}_t\mathbf{m}_t = \mathbf{a}_i e_{it} - \beta_{it}\mathbf{a}_i\mathbf{a}_i^\top \mathbf{m}_t, \qquad -\tfrac{1}{2}\hat{\mathbf{S}}_t^{\setminus i} + \tfrac{1}{2}\mathbf{S}_t = \tfrac{1}{2}\beta_{it}\mathbf{a}_i\mathbf{a}_i^\top, \tag{42}$$

and then rearranging it to get the deviation over the mean (following the same steps as Eq. 13) and using the approximation $\hat{\boldsymbol{\Sigma}}_t^{\setminus i} \approx \boldsymbol{\Sigma}_t$, we obtain,

$$\hat{\mathbf{m}}_t^{\setminus i} - \mathbf{m}_t = \hat{\boldsymbol{\Sigma}}_t^{\setminus i}\mathbf{a}_i e_{it} \approx \boldsymbol{\Sigma}_t\mathbf{a}_i e_{it} \tag{43}$$

If we consider Bernoulli likelihood with $y \in \{0, 1\}$ and link function $\sigma(f) = 1/(1+e^{-f})$ (i.e., sigmoid) then $e_{it} = \mathbb{E}_{q_{\boldsymbol{\lambda}_t}(f_i)}[\sigma(f_i) - y_i]$ which can be interpreted as the prediction error. This also holds for categorical likelihood or any other GLM likelihood.

We next demonstrate the bi-linear relationship by considering the deviation in the mean of the posterior marginal $f_i(\mathbf{m}) := \mathbf{a}_i^\top \mathbf{m}$,

$$f_i(\mathbf{m}_t^{\setminus i}) - f_i(\mathbf{m}_t) \approx \mathbf{a}_i^\top(\hat{\mathbf{m}}_t^{\setminus i} - \mathbf{m}_t) = \mathbf{a}_i^\top \boldsymbol{\Sigma}_t \mathbf{a}_i e_{it} = v_{it} e_{it} \tag{44}$$

where $v_{it} = \mathbf{a}_i^\top \boldsymbol{\Sigma}_t \mathbf{a}_i$ is the marginal variance of f_i.

E Extension to Non-Differentiable Loss function

For non-differentiable cases, we can rewrite the BLR of Eq. 10 as,

$$\mathbf{m}_t = \mathbf{m}_{t-1} - \rho \mathbf{\Sigma}_t \nabla_{\mathbf{m}} \mathbb{E}_{q_{\boldsymbol{\lambda}_{t-1}}}[\mathcal{L}(\boldsymbol{\theta})], \qquad \mathbf{S}_t = (1-\rho)\mathbf{S}_{t-1} + 2\rho \nabla_{\mathbf{\Sigma}} \mathbb{E}_{q_{\boldsymbol{\lambda}_{t-1}}}[\mathcal{L}(\boldsymbol{\theta})]. \tag{45}$$

This follows by using [22, Eq. 10-11] where we take derivative outside the expectation instead of inside. This is valid because the expectation of a non-differentiable function is still differentiable (under some regularity conditions). The same technique can be applied to Eq. 11 to get

$$\widetilde{\nabla} \mathbb{E}_{q_{\boldsymbol{\lambda}}}[\ell_i] = \left(\nabla_{\mathbf{m}} \mathbb{E}_{q_{\boldsymbol{\lambda}}}[\ell_i] - 2\nabla_{\mathbf{\Sigma}} \mathbb{E}_{q_{\boldsymbol{\lambda}}}[\ell_i]\mathbf{m}, \;\; \nabla_{\mathbf{\Sigma}} \mathbb{E}_{q_{\boldsymbol{\lambda}}}[\ell_i]\right), \tag{46}$$

and proceeding in the same fashion we can write: $\hat{\mathbf{m}}_t^{\backslash i} - \mathbf{m}_t = \hat{\mathbf{\Sigma}}_t^{\backslash i} \nabla_{\mathbf{m}} \mathbb{E}_{q_{\boldsymbol{\lambda}_t}}[\ell_i(\boldsymbol{\theta})]$. This is the extension of Eq. 13 to non-differentiable loss functions. This is a much more principled approach than [26] who used an ad-hoc smoothing of the non-differentiable loss. The smoothing here is automatically handled using the Bayesian framework.

F MPE for Deep Learning Optimizers

F.1 Adam

The Adam and RMSprop deep-learning optimizers [24] can be derived from the BLR update in Eq. 10 as described in [20, 22]. The MPE from Eq. 4 therefore also applies to both RMSprop and Adam. Consider the following update equations of Adam:

$$\mathbf{r}_t = \beta_1 \mathbf{r}_{t-1} + (1-\beta_1)\mathbf{g}_t, \tag{47}$$

$$\mathbf{s}_t = \beta_2 \mathbf{s}_{t-1} + (1-\beta_2)\,(\mathbf{g}_t \circ \mathbf{g}_t), \tag{48}$$

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \rho\, \mathbf{r}_t / (\sqrt{\hat{\mathbf{s}}_t} + \epsilon). \tag{49}$$

The gradient of the loss $\mathbf{g}_t = \nabla \mathcal{L}(\boldsymbol{\theta}_{t-1})$ is evaluated for a minibatch of examples. $\mathbf{a} \circ \mathbf{b}$ denotes an element-wise product between vectors $\mathbf{a}$ and $\mathbf{b}$. β_1 and β_2 are coefficients for the running averages, ρ is a learning rate and ϵ a small damping to stabilize the method.

To evaluate Eq. 19 and predict generalization during the training, prediction variances v_{it} are required. Due to its connection to the BLR update Eq. 10, we can use the second moment vector $\mathbf{s}_t$ of the Adam optimizer in Eq. 48 to approximately estimate these variances. This vector maintains an exponential running average of the squared gradients over the training iterations. We construct a diagonal covariance matrix $\mathbf{\Sigma}_t = \text{diag}(\boldsymbol{\sigma}_t^2)$ with $\boldsymbol{\sigma}_t^2 = 1/(N\mathbf{s}_t + \delta)$ as the diagonal entries. N is the number of training examples, and δ is the regularization parameter of an L_2 regularizer $\mathcal{R}(\boldsymbol{\theta}) = \frac{1}{2}\delta\|\boldsymbol{\theta}\|^2$. Analogous to Eq. 15, we obtain the prediction variances as $v_{it} = \nabla f_i(\boldsymbol{\theta}_t)^\top \mathbf{\Sigma}_t \nabla f_i(\boldsymbol{\theta}_t)$. We evaluate the prediction errors as $e_{it} = \sigma(f_i(\boldsymbol{\theta}_t)) - y_i$. Depending on the application scenario, the sensitivities do not need to be evaluated at every step, and can for example be evaluated only periodically during training when needed.

F.2 Improved Bayesian Learning Rule (iBLR)

Rather than using Adam or RMSprop to estimate the covariances, we can also use a more faithful approximation of the BLR update in Eq. 10. This is expected to give better results. In practice, we use the iBLR optimizer [27], which implements an improved version of the BLR iterations Eq. 10. Similarly to Eq. 10, the method updates a Gaussian posterior over the weights during training, but the improved version makes it more easily applicable to nonconvex problems. Essentially, iBLR entails a slight modification of Eq. 10 to ensure that the estimated precision stays positive definite.

The iBLR [27] can be used to derive an Adam-like optimizer, see also [27, Fig. 1]:

$$\mathbf{r}_t = \beta_1 \mathbf{r}_{t-1} + (1-\beta_1)\mathbf{g}_t, \tag{50}$$

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \rho\, \mathbf{r}_t / \mathbf{s}_t, \tag{51}$$

$$\mathbf{s}_t = \mathbf{s}_{t-1} + (1-\beta_2)\,\mathbf{h}_t + \tfrac{1}{2}(1-\beta_2)^2 \mathbf{h}_t \circ \mathbf{s}^{-1} \circ \mathbf{h}_t, \tag{52}$$

where $\mathbf{g}_t \approx \mathbb{E}_{\mathcal{N}(\boldsymbol{\theta}_t, \mathbf{\Sigma}_t)}[\nabla \mathcal{L}]$ and $\mathbf{h}_t \approx \mathbb{E}_{\mathcal{N}(\boldsymbol{\theta}_t, \mathbf{\Sigma}_t)}[\nabla^2 \mathcal{L}]$ are approximations of the expected gradient and Hessian, and $\boldsymbol{\sigma}_t^2 = 1/(N\mathbf{s}_t)$ are the diagonal entries of $\mathbf{\Sigma}_t = \text{diag}(\boldsymbol{\sigma}_t^2)$. As in [27, Fig. 1]

we use one random sample to approximate the expectations and use the reparametrization trick to estimate the Hessian via gradients only.

The updates Eq. 50 – Eq. 52 essentially are a small modification of the Adam optimizer, but thanks to the random sampling the method performs approximate Bayesian inference. This, along with an arguably more accurate estimate of second-order information is expected to lead to improved variance estimates. The final prediction variances v_{it} and prediction errors e_{it} are computed from $\mathbf{s}_t$ and $\boldsymbol{\theta}_t$ in the same way as described for Adam in App. F.1.

G Experimental Details

G.1 Details of "Do estimated deviations correlate with the truth?"

For the experiments in Fig. 2, we train models on all training data to obtain $\boldsymbol{\theta}^*$. For $\boldsymbol{\theta}^{\setminus i}$, we exclude the i-th example and warmstart the training at $\boldsymbol{\theta}^*$. For both cases, we use SGD with momentum parameter of 0.9 and a cosine learning-rate scheduler. The regularization parameter δ is chosen by grid search. The remaining hyperparameters are shown in Table 1.

The details of the models are as follows. For MNIST, we use a fully-connected multilayer perceptron (MLP) with two hidden layers of 500 and 300 neurons each. It has 545810 parameters. For FMNIST, we use a LeNet5 architecture. It is a standard convolutional neural network (CNN) with three convolution layers followed by two fully-connected layers. It has 61706 parameters. For CIFAR10, we use a CNN architecture consisting of three convolution layers followed by three fully-connected layers. This architecture is used in benchmarks such as the DeepOBS suite [35]. It has a parameter count of 895210.

Dataset	Model	B	δ	E^*	LR^*	LR^*_{min}	$E^{\setminus i}$	$LR^{\setminus i}$	$LR^{\setminus i}_{\text{min}}$
MNIST	MLP (500, 300)	256	100	500	10^{-2}	10^{-3}	300	10^{-3}	10^{-4}
FMNIST	LeNet5	256	100	300	10^{-1}	10^{-3}	200	10^{-3}	10^{-4}
CIFAR10	CNN	512	250	500	10^{-2}	10^{-4}	300	10^{-4}	10^{-6}

Table 1: Hyperparameters for predicting true sensitivity in Fig. 2. B, E and LR denote batch size, training epochs and learning rates. The superscripts * and $^{\setminus i}$ indicate hyperparameters for training on all data and warmstarted leave-one-out training, respectively. The subscript $_{\text{min}}$ stands for the minimum learning rate of the learning rate scheduler.

We now describe the group removal experiments. For the MNIST result in Fig. 3(b) we use the model from Table 1 with the same hyperparameters. The exception are the last three hyperparameters in the table, which differ between leave-one-out and leave-group-out training. For binary USPS in Fig. 3(a), we train a small MLP with three hidden layers with 30 neurons. We classify the digits 3 and 5. For the training on all data we use 500 epochs with a learning rate of 10^{-3}. We use a batch-size of 32 and a regularization parameter of 5. For the leave-group-out training on both datasets we use 1000 epochs and the same learning rate as for training on all data. We initialize the MLPs at $\boldsymbol{\theta}^*$.

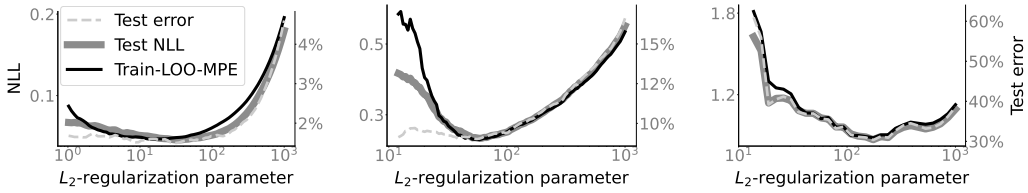
Variance computation requires inversion of matrices. For large problems, we use a Kronecker-factored Laplace variance approximation as implemented in the `laplace` package [9]. We observe a reasonable speed-accuracy trade-off on both fully-connected and convolutional neural networks.

G.2 Details of "Predicting the effect of class removal on generalization"

For the MNIST experiment in Fig. 3(c), we use the large MLP and LeNet5. For FMNIST in Fig. 1(c), we use the small MLP and LeNet5. The hyperparameters are given in Table 2. All models are trained with a regularization parameter of 100 and a batch size of 256. The leave-class-out training is done for 1000 epochs and is warmstarted at $\boldsymbol{\theta}^*$. We use the same Kronecker-factored Laplace variance approximation as before.

Dataset	Model	E^*	LR^*	LR^*_{min}	$LR^{\setminus i}$	$LR^{\setminus i}_{\text{min}}$
MNIST	MLP (500, 300)	500	10^{-2}	10^{-3}	10^{-4}	10^{-5}
MNIST	LeNet5	300	10^{-1}	10^{-3}	10^{-5}	10^{-6}
FMNIST	MLP (32, 16)	300	10^{-2}	10^{-3}	10^{-5}	10^{-6}
FMNIST	LeNet5	300	10^{-1}	10^{-3}	10^{-4}	10^{-5}

Table 2: Hyperparameters for the class removal experiments in Fig. 3(c) and Fig. 1(c). B, E and LR denote batch size, training epochs and learning rates. The superscripts * and $^{\setminus i}$ indicate hyperparameters for training on all data and warmstarted leave-class-out training, respectively. The subscript $_{\text{min}}$ stands for the minimum learning rate of the learning rate scheduler.



(a) MNIST, MLP (b) FMNIST, LeNet5 (c) CIFAR10, CNN

Figure 7: The MPE can accurately estimate the LOO-CV curve for predicting generalization and tuning of the L_2-regularization parameter on MNIST, FMNIST and CIFAR-10.

G.3 Details of "Estimating the leave-one-out cross-validation curves for hyperparameter tuning"

We show results on MNIST (large MLP) in Fig. 4(a), on FMNIST (LeNet5) in Fig. 4(b) and on CIFAR10 (CNN) in Fig. 4(c). Fig. 7 shows the same experiments with test errors included.. In We evaluate a certain number of values within a log-spaced range. For visualization purposes we calculate a moving average of the plotted lines with a smoothing window. The details are in Table 1.

Dataset	Model	Number of δs	Range	Smoothing
MNIST	MLP (500, 300)	96	$10^0 - 10^3$	3
FMNIST	LeNet5	96	$10^1 - 10^3$	5
CIFAR10	CNN	30	$10^1 - 10^3$	3

Table 3: Experimental settings for the leave-one-out cross-validation curves for hyperparameter tuning in Fig. 4. The table gives information on the datasets, the models, the number of values of the regularization parameter δ evaluated, the range of the values and the smoothing window for calculating a moving average for visualization.

All models are trained from scratch. As optimizers we use Adam for FMNIST, Adamw [28] for CIFAR10 and SGD with a momentum of 0.9 for MNIST. We use a cosine learning rate scheduler to anneal the learning rate. The other hyperparameters are equivalent to the settings of the models trained on all data from the leave-one-out experiments in Table 1. The exception is the number of epochs for CIFAR10. We train for 150 epochs. As before, we use a Kronecker-factored Laplace approximation for variance computation.

G.4 Details of "How do sensitivities evolve during training?"

Bayesian logistic regression: For the experiment in Fig. 6(a), we consider a generalized linear model $\mathbf{f}_i = \mathbf{\Theta}\mathbf{x}_i$ on the MNIST dataset using the raw pixels $\mathbf{x}_i \in \mathbb{R}^P$ as features. We denote the predicted logits by $\mathbf{f}_i \in \mathbb{R}^C$ and the parameters as $\mathbf{\Theta} \in \mathbb{R}^{C \times P}$ where $P = 785$ and $C = 9$ for the considered MNIST dataset. We use the negative log-likelihood of the categorical distribution as a loss function and use regularization parameter $\delta = 0.1$.

We run the iteration of Eq. 10 for 125 stochastic updates with batch-size 200, reaching around 91% when predicting at the mean $\mathbf{m}_t$. We use linear learning-rate decay from 0.005 to 0.001 for the mean $\mathbf{m}$ and a learning rate of 10^{-5} for the precision $\mathbf{S}$. The expectations in Eq. 10 are approximated using three samples drawn from the posterior.

Given the mean $\mathbf{m}_t$ and precision $\mathbf{S}_t$ we use the last equation in Eq. 13 to understand the sensitivity of the model at iteration t. In Fig. 6(a) at iteration $t = 5, 10, 25, 125$ we compute for each example

- its variance $\mathbf{v}_{i,t,c} = \mathbf{x}_i^\top \mathbf{\Sigma}_{t,c} \mathbf{x}_i$, where $\mathbf{\Sigma}_{t,c}$ is the c-th block of the covariance matrix $\mathbf{\Sigma}_t$ given as the inverse of the precision matrix $\mathbf{S}_t$,
- and its prediction error $\mathbf{e}_{i,t} = \mathbb{E}_{\mathbf{g} \sim \mathcal{N}(\mathbf{f}_{i,t}, \mathbf{v}_{i,t})}[\sigma(\mathbf{g}) - \mathbf{y}_i]$, $\mathbf{f}_{i,t} = \mathbf{m}_t \cdot \mathbf{x}_i$.

We used 150 samples to approximate the expectation for the prediction error. Analogous to Eq. 16, this gives us a sensitivity at iteration t for each data point: $\sum_{c=1}^{C} |\mathbf{v}_{i,t,c}| \cdot |\mathbf{e}_{i,t,c}| \cdot |\sigma'(\mathbf{m}\mathbf{x}_i)_c|$. For Fig. 6(a) we sorted all examples according to this sensitivity at iteration $t = 125$ in decreasing order and then plot the average sensitivities of examples in 60 groups with 100 examples in each group.

iBLR: For the experiments in Fig. 6(b), Fig. 6(c) and Fig. 6(d) we consider neural network models $\mathbf{f}_i(\boldsymbol{\theta}_t)$ on MNIST, FMNIST and CIFAR10. We obtain test accuracies of 0.985, 0.913 and 0.809, respectively. These values are standard for the used models if the data is not augmented. For CIFAR10 we here use a ResNet–20 with filter response normalization [37]. The number of classes and output neurons is $C = 10$ for all datasets. We use the negative log-likelihood of the categorical distribution as a loss function. We run the iterations of Eq. 10, adjusted according to the iBLR update equations (50) – (52). The expectations are approximated with one sample drawn from the posterior. We use a cosine learning rate scheduler with a start learning rate of 0.1 and anneal to zero over the epochs. Other details are in Table 4.

Given the mean $\mathbf{m}_t$ and precision $\mathbf{S}_t$ we use the last equation Eq. 13 to understand the sensitivity of the model at iteration t. In the figures, we compute for each example

- its variance $\mathbf{v}_{i,t} = \nabla f_i(\boldsymbol{\theta}_t)^T \mathbf{\Sigma}_t \nabla f_i(\boldsymbol{\theta}_t)$, where $\nabla f_i(\boldsymbol{\theta}_t)^T \in \mathbb{R}^{C \times P}$ and $\mathbf{\Sigma}_t \in \mathbb{R}^{C \times C}$ contains the per-class weight variances on the diagonal as obtained by inversion of the precision matrix $\mathbf{S}_t$,
- and its prediction error $\mathbf{e}_{i,t} = \mathbb{E}_{\mathbf{g} \sim \mathcal{N}(\mathbf{f}_{i,t}, \mathbf{v}_{i,t})}[\sigma(\mathbf{g}) - \mathbf{y}_i]$, $\mathbf{f}_{it} = \mathbf{f}_i(\mathbf{m}_t)$.

We used 150 samples to approximate the expectation for the prediction error. We sorted all examples according to this sensitivity at the last plotted epoch in the figures in decreasing order. We plot the sensitivities, where we group examples with similar sensitivity into overall 200 groups.

Dataset	Model	B	δ	E
MNIST	MLP (500, 300)	256	30	100
FMNIST	LeNet5	256	60	100
CIFAR10	ResNet–20	512	25	300

Table 4: Experimental settings for analyzing the evolution of sensitivities during training. B denotes the batch size and E are the training epochs. δ is the regularization parameter

G.5 Details of "Predicting generalization during the training: "

We conduct experiments to predict the generalization of models during the training. We use an estimate obtained from the MPE (Train-LOO-MPE) and the influence function, which is a special case of MPE. The estimate is based on training data alone and is calculated with Eq. 19. In Fig. 5, we show results with a LeNet5 on FMNIST. In Fig. 1(b) we use a ResNet–20 with Filter-Response-Normalization on the CIFAR10 dataset. In Fig. 10 and Fig. 11 we additionally report the test error over the training epochs for these experiments. Fig. 8 and Fig. 9 contain results for MNIST and CIFAR10 that are not included in the main text. For MNIST, we evaluate both on a small MLP and a LeNet5 architecure. For the additional CIFAR10 results, we use the previously described CNN

architecture. In Eq. 19 the computation of prediction variances and prediction errors is required. The quantities are evaluated periodically during the training, which is indicated with markers in the figures.

MPE with Adam and iBLR: For the MPE experiments, we use estimates based on the Adam optimizer and the iBLR optimizer. Details for Adam and iBLR are described in App. F.1 and App. F.2, respectively. The experimental results suggests that a more accurate variance computation improves the prediction.

Influence function: As described in the main text, the influence function as given by Eq. 14 is a special case of the MPE with Gaussian posterior in Eq. 13. The expectations from Eq. 13 are approximated at the mean using the Delta method.

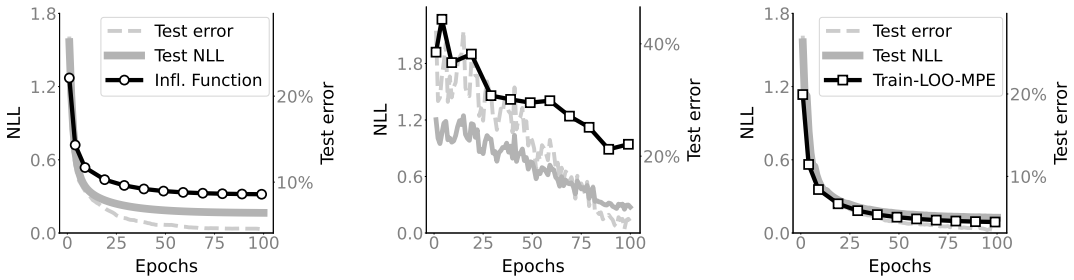
We highlight some differences of our implementation to the influence function commonly used in deep learning [26]. The right-hand expression in Eq. 14 is equivalent to the influence estimation in parameters at convergence of [26]. [26] also evaluate their influence function for non-converged models and non-convex objectives using a quadratic approximation of the loss. They do, however, only evaluate close to a local minimum. In contrast to that, we compute influence also throughout the training. We use the left-hand side of Eq. 14), $\hat{\boldsymbol{\theta}}_t^{\setminus i} - \boldsymbol{\theta}_t \approx \boldsymbol{\Sigma}_t \nabla f_i(\boldsymbol{\theta}_t) e_i(\boldsymbol{\theta}_t)$, and Taylor's approximation to obtain the deviation in function outputs in Eq. 15. That expression is required to evaluate the Train-LOO estimate in Eq. 19. The Train-LOO estimate can be used to approximate the average test loss based on all training examples. Opposed to that, [26] evaluate the effect of perturbing a training example on a single test example using the expression for the parameter-influence and the chain rule.

To estimate the Hessian, [26] use implicit Hessian-vector products and approximate solutions of a stochastic conjugate gradient solver. To deal with Hessian estimates that are not positive-definite, they add a damping term. We use a generalized Gauss-Newton approximation (GGN) [29] to the Hessian, based on an implementation from the `laplace` package [9]. The GGN ensures positive-definiteness when applied at approximately optimal points or even intermediate points in the loss landscape of nonconvex problems such as deep learning. The GGN scales quadratically with the number of parameters, which is computationally expensive for large neural networks. Structural approximations of the GGN matrix reduce both memory consumption and computational cost of inversion. We use a diagonal approximation that neglects off-diagonal elements of the Hessian for fast inversion. With the GGN approximation we get a Hessian estimate at the iterate t as $\nabla^2 \mathcal{L}(\boldsymbol{\theta}_t) = \sum_{i=1}^{N} \nabla f_i(\boldsymbol{\theta}_t) \sigma'(f_{it}) \nabla f_i(\boldsymbol{\theta}_t)^\top + \delta \mathbf{I}$. Using $\boldsymbol{\Sigma}_t = (\nabla^2 \mathcal{L}(\boldsymbol{\theta}_t))^{-1}$, the prediction variances and prediction errors in Eq. 13 can be computed as described for the Adam optimizer in App. F.1. For future work, we would consider relaxing some of the approximations.

Details of optimization procedures: The experimental details are listed in Table 5. In all experiments, we use a grid search to determine the regularization parameter δ. For the experiments with the influence function, we use the SGD optimizer. The exception is the experiment on the FMNIST dataset. There, we use the AdamW optimizer [28]. We use a weight decay factor of δ/N replacing the explicit L_2-regularization term in the loss Eq. 1. The regularizer $\mathcal{R}(\boldsymbol{\theta})$ is set to zero there.

Dataset	Model	Method	LR	LR_{min}	B	δ
MNIST	MLP (32, 16)	Infl. function (SGD)	10^{-3}	10^{-4}	256	80
		MPE (Adam)	10^{-1}	10^{-2}	256	80
		MPE (iBLR)	10^{-2}	10^{-4}	256	80
MNIST	LeNet5	Infl. function (SGD)	10^{-3}	10^{-4}	256	60
		MPE (Adam)	10^{-2}	10^{-6}	256	60
		MPE (iBLR)	10^{-2}	10^{-4}	256	60
FMNIST	LeNet5	Infl. function (AdamW)	10^{-3}	10^{-3}	256	60
		MPE (Adam)	10^{-1}	0	256	60
		MPE (iBLR)	10^{-1}	0	256	60
CIFAR10	CNN	Infl. function (SGD)	10^{-1}	0	512	250
		MPE (Adam)	10^{-3}	10^{-9}	512	50
		MPE (iBLR)	10^{-2}	10^{-4}	512	200
CIFAR10	ResNet–20	MPE (iBLR)	10^{-1}	0	512	250

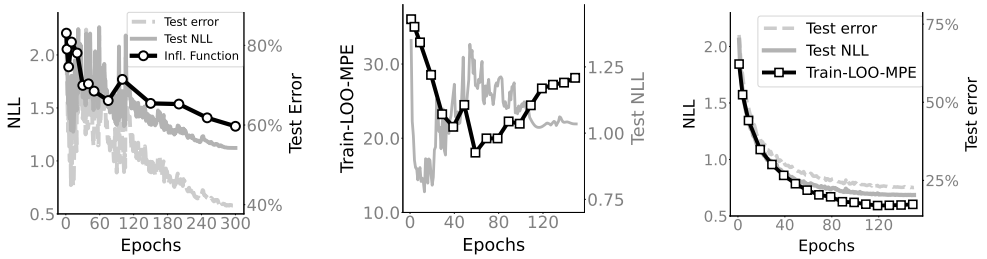
Table 5: Experimental settings for predicting generalization during the training. B and E denote the batch-size and training epochs, respectively. LR and LR_{min} are the start and end learning rates of the cosine learning rate scheduler. δ is the regularization parameter.



(a) Infl. Function with SGD, MLP (b) MPE with Adam, MLP (c) MPE with Adam, MLP

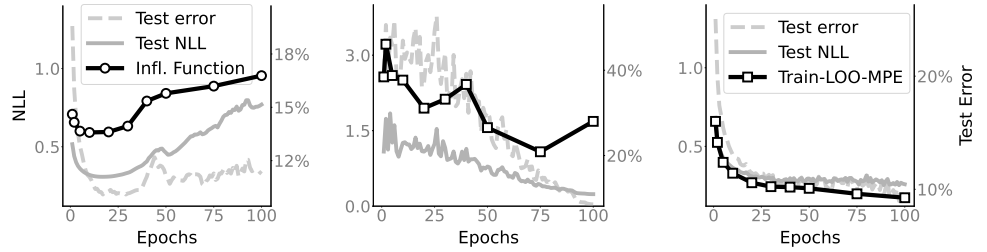
(d) Infl. Function with SGD, LeNet5 (e) MPE with Adam, LeNet5 (f) MPE with iBLR, LeNet5

Figure 8: Panels (a), (b) and (c) show estimates of the test NLL using influence function and MPE during training of a MLP on MNIST. Panels (d), (e) and (f) show results with a LeNet5. For influence function, we use a generalized Gauss-Newton approximation of the Hessian with diagonal matrix structure. The results suggest that MPE with iBLR can faithfully estimate the test NLL and improves upon applying influence function. It performs better than the estimation with MPE that computes variances heuristically based on the Adam optimizer.



(a) Infl. Function with SGD (b) MPE with Adam (c) MPE with iBLR

Figure 9: Panel (a) shows an estimate of the test NLL with influence function during training of a CNN on CIFAR10. We use a generalized Gauss-Newton approximation of the Hessian with diagonal matrix structure. Panels (b) and (c) display the Train-LOO estimate obtained from MPE. The results suggest that MPE with iBLR can faithfully estimate the test NLL and improves upon applying influence function. It performs better than the estimation with MPE that computes variances heuristically based on the Adam optimizer.



(a) Infl. Function with AdamW (b) MPE with Adam (c) MPE with iBLR

Figure 10: Panel (a) shows an estimate of the test NLL with influence function during training of a LeNet5 on FMNIST. We use a generalized Gauss-Newton approximation of the Hessian with diagonal matrix structure. Panels (b) and (c) display the Train-LOO estimate obtained from MPE. The results suggest that MPE with iBLR can faithfully estimate the test NLL and improves upon applying influence function. It performs better than the estimation with MPE that computes variances heuristically based on the Adam optimizer.

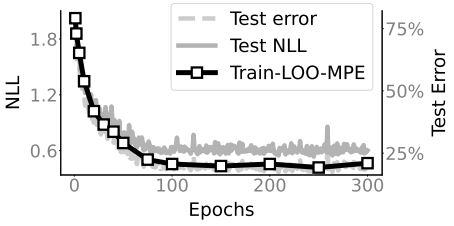


Figure 11: Train-LOO-MPE with the iBLR optimizer faithfully estimates the test NLL during training of a ResNet–20 on CIFAR10.