If Influence Functions are the Answer, Then What is the Question?

Anonymous Author(s)
Affiliation
Address
email

Abstract

Influence functions efficiently estimate the effect of removing a single training data point on a model's learned parameters. While influence estimates align well with leave-one-out retraining for linear models, recent works have shown this alignment is often poor in neural networks. In this work, we investigate the specific factors that cause this discrepancy by decomposing it into five separate terms. We study the contributions of each term on a variety of architectures and datasets and how they vary with factors such as network width and training time. While practical influence function estimates may be a poor match to leave-one-out retraining for nonlinear networks, we show that they are often a good approximation to a different object we term the *proximal Bregman response function (PBRF)*. Since the PBRF can still be used to answer many of the questions motivating influence functions such as identifying influential or mislabeled examples, our results suggest that current algorithms for influence function estimation give more informative results than previous error analyses would suggest.

1 Introduction

The influence function [21, 13] is a classic technique from robust statistics that estimates the effect of deleting a single data example (or a group of data examples) from a training dataset. Formally, given a neural network with learned parameters $\boldsymbol{\theta}^\star$ trained on a dataset $\mathcal{D}$, we are interested in the parameters $\boldsymbol{\theta}^\star_{-\mathbf{z}}$ learned by training on a dataset $\mathcal{D} - \{\mathbf{z}\}$ constructed by deleting a single training example $\mathbf{z}$ from $\mathcal{D}$. By taking the second-order Taylor approximation to the cost function around $\boldsymbol{\theta}^\star$, influence functions approximate the parameters $\boldsymbol{\theta}^\star_{-\mathbf{z}}$ without the computationally prohibitive cost of retraining the model. Since Koh and Liang [30] first deployed influence functions in machine learning, influence functions have been used to solve various tasks such as explaining model's predictions [30, 22], relabelling harmful training examples [33], carrying out data poisoning attacks [31], and increasing fairness in models' predictions [11, 47].

When the training objective is strongly convex (e.g., as in logistic regression with L_2 regularization), influence functions are expected to align well with leave-one-out (LOO) or leave-k-out retraining [30, 32, 26]. However, Basu et al. [7] showed that influence functions in neural networks often do not accurately predict the effect of retraining the model and concluded that influence estimates are often erroneous. Because of the poor match between influence estimates and LOO retraining, influence function methods are often evaluated with alternative metrics such as the recovery rate of maliciously corrupted examples using influence scores [28, 30, 46, 27]. But such downstream metrics are very indirect. For purposes of validation and algorithmic research, it would be far preferable to have a well-defined quantity that influence functions are meant to approximate.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

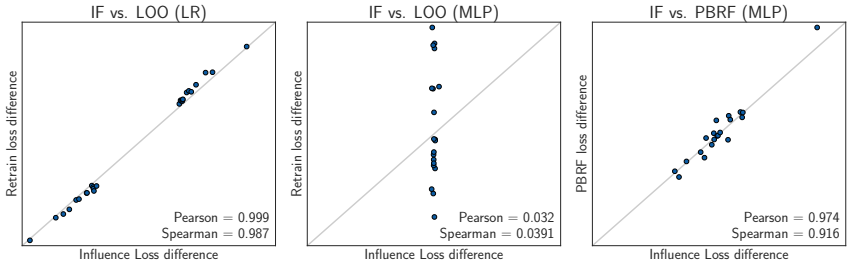


Figure 1: Comparison of test loss differences computed by influence function (IF), leave-one-out (LOO) retraining, and our proximal Bregman response function (PBRF). Each point corresponds to the individual effect of removing one training example. Influence estimates align well with true retraining for (**left**) logistic regression (LR) but poorly for (**middle**) multilayer perceptrons (MLP). While influence functions in neural networks do not accurately predict the effect of retraining the model, they are still a good approximation to (**right**) the PBRF.

In this work, we investigate the source of the discrepancy between influence functions and LOO retraining in neural networks. We decompose the discrepancy into five components: (1) the difference between cold-start and warm-start response functions (a concept elaborated on below), (2) an implicit proximity regularizer, (3) influence estimation on non-converged parameters, (4) linearization, and (5) approximate solution of a linear system. We empirically evaluate the contributions of each component on binary classification, regression, image classification, and language modeling tasks and show that, across all tasks, components (1–3) are most responsible for the discrepancy between influence functions and LOO retraining. We further investigate how the contribution of each component changes in response to the change in network width and depth, weight decay, training time, damping, and the number of data points being removed.

Moreover, we show that while influence functions for neural networks are often a poor match to LOO retraining, they are a much better match to what we term the *proximal Bregman response function (PBRF)*. Intuitively, the PBRF approximates the effect of removing a data point while otherwise trying to keep the predictions consistent with those of the (partially) trained model. From this perspective, we reframe misalignment components (1–3) as simply reflecting the difference between LOO retraining and the PBRF. The gap between the influence function estimate and the PRBF comes from only sources (4) and (5), which we found empirically to be at least an order of magnitude smaller for most neural networks. As a result, on a wide variety of tasks, we show that influence functions closely align with the PBRF while failing to approximate the effect of retraining the model, as shown in Figure 1.

The PBRF can be used for many of the same use cases that have motivated influence functions, such as finding influential or mislabeled examples and carrying out data poisoning attacks [30, 31], and can therefore be considered an alternative to LOO retraining as a gold standard for evaluating influence functions. Hence, we conclude that influence functions applied to neural networks are not inherently "fragile" as is often believed [7], but instead can be seen as giving accurate answers to a different question than is normally assumed.

2 Related Work

Instance-based interpretability methods are a class of techniques that explain a model's predictions in terms of the examples on which the model was trained. Methods of this type include `TracIn` [45], Representer Point Selection [58], Grad-Cos and Grad-Dot [12, 23], `MMD-critic` [29], unconditional counterfactual explanations [55], and of central focus in this paper, influence functions [30, 21, 13]. Since its adoption in machine learning by Koh and Liang [30], multiple extensions and improvements upon influence functions have also been proposed, such as variants that use Fisher kernels [28], higher-order approximations [8], tricks for faster and scalable inference [20, 46], group influence formulations [32, 8], and relative local weighting [5]. However, many of these methods rely on the same strong assumptions made in the original influence function derivation that the objective needs to be strongly convex and influence functions must be computed on the optimal parameters.

Measuring the quality of influence functions is often performed by measuring the correlation between LOO retraining and influence function estimations [30, 7, 8, 57]. The effectiveness of influence functions can also be evaluated by retraining the model without the most influential data points [28, 20] or attempting to identify mislabelled or poisoned data [28, 30, 46, 27]. In general, influence functions are assumed to approximate the effects of leave-out retraining from scratch. However, recent empirical analyses have demonstrated the fragility of influence functions and a fundamental misalignment between their assumed and actual effect [7, 18, 27]. For example, Basu et al. [7] argued that the accuracy of influence functions in deep networks is highly sensitive to network width and depth, weight decay strength, inverse-Hessian vector product estimation methodology, and test query point by measuring the alignment between influence functions and LOO retraining. Although these results were interpreted as a failure of influence functions, we claim that many of these apparent failures simply reflect that influence functions answer a different question than is usually assumed.

3 Background

Consider a prediction task from an input space $\mathcal{X}$ to a target space $\mathcal{T}$ where we are given a finite training dataset $\mathcal{D}_{\text{train}} = \{(\mathbf{x}^{(i)}, \mathbf{t}^{(i)})\}_{i=1}^{N}$. Given a data point $\mathbf{z} = (\mathbf{x}, \mathbf{t})$, let $\mathbf{y} = f(\boldsymbol{\theta}, \mathbf{x})$ be the prediction of the network parameterized by $\boldsymbol{\theta} \in \mathbb{R}^d$ and $\mathcal{L}(\mathbf{y}, \mathbf{t})$ be the loss (e.g., squared error or cross-entropy). We aim to solve the following optimization problem:

$$\boldsymbol{\theta}^\star = \underset{\boldsymbol{\theta}\in\mathbb{R}^d}{\arg\min}\, \mathcal{J}(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}\in\mathbb{R}^d}{\arg\min}\, \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(f(\boldsymbol{\theta}, \mathbf{x}^{(i)}), \mathbf{t}^{(i)}), \tag{1}$$

where $\mathcal{J}(\cdot)$ is the cost function. We summarize the notation used in this paper in Appendix A.

3.1 Downweighting a Training Example

The training objective in Eqn. 1 aims to find the parameters that minimize the average loss on all training examples. Herein, we are interested in studying the change in optimal model parameters when a particular training example $\mathbf{z} = (\mathbf{x}, \mathbf{t}) \in \mathcal{D}_{\text{train}}$ is removed from the training dataset, or more generally, when the data point $\mathbf{z}$ is downweighted by an amount $\epsilon \in \mathbb{R}$. Formally, this corresponds to minimizing the following downweighted objective:

$$\boldsymbol{\theta}^\star_{-\mathbf{z},\epsilon} = \underset{\boldsymbol{\theta}\in\mathbb{R}^d}{\arg\min}\, \mathcal{Q}_{-\mathbf{z}}(\boldsymbol{\theta}, \epsilon) = \underset{\boldsymbol{\theta}\in\mathbb{R}^d}{\arg\min}\, \mathcal{J}(\boldsymbol{\theta}) - \mathcal{L}(f(\boldsymbol{\theta}, \mathbf{x}), \mathbf{t})\epsilon. \tag{2}$$

When $\epsilon = 1/N$, the downweighted objective reduces to the cost over the dataset with the example $\mathbf{z}$ removed, up to a constant factor. To see how the optimum of the downweighted objective responds to changes in the downweighting factor ϵ, we define the *response function* $r^\star_{-\mathbf{z}} \colon \mathbb{R} \to \mathbb{R}^d$ by:

$$r^\star_{-\mathbf{z}}(\epsilon) = \underset{\boldsymbol{\theta}\in\mathbb{R}^d}{\arg\min}\, \mathcal{Q}_{-\mathbf{z}}(\boldsymbol{\theta}, \epsilon), \tag{3}$$

where we assume that the downweighted objective is strongly convex and the solution to the downweighted objective is unique given some factor ϵ. Note that $r^\star_{-\mathbf{z}}(0) = \boldsymbol{\theta}^\star$ and the response function is differentiable at 0 by the Implicit Function Theorem [34, 19] when the downweighted objective is further twice continuously differentiable. Influence functions approximate the response function by performing a first-order Taylor expansion around $\epsilon_0 = 0$:

$$r^\star_{-\mathbf{z},\text{lin}}(\epsilon) = r^\star_{-\mathbf{z}}(\epsilon_0) + \left.\frac{\mathrm{d}r^\star_{-\mathbf{z}}}{\mathrm{d}\epsilon}\right|_{\epsilon=\epsilon_0} (\epsilon - \epsilon_0) = \boldsymbol{\theta}^\star + (\nabla^2_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta}^\star))^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}(f(\boldsymbol{\theta}^\star, \mathbf{x}), \mathbf{t})\epsilon. \tag{4}$$

We refers readers to Van der Vaart [52] and Appendix B for a detailed derivation. The optimal parameters trained without $\mathbf{z}$ can then be approximated by plugging in $\epsilon = 1/N$ to Eqn. 4.

3.2 Influence Function Estimation in Neural Networks

Influence functions face two main challenges when deployed on neural networks. First, the influence estimation (shown in Eqn. 4) requires computing an inverse Hessian-vector product (`iHVP`). Unfortunately, storing and inverting the Hessian requires $O(d^3)$ operations and is infeasible to compute

for modern neural networks. Instead, Koh and Liang [30] tractably approximate the `iHVP` using truncated non-linear conjugate gradient (`CG`) [43] or the `LiSSA` algorithm [1]. Both approaches avoid explicit computation of the Hessian inverse (see Appendix G for details) and only require $O(Nd)$ operations to approximate the influence function.

Second, the derivation of influence functions assumes a strongly convex objective, which is often not satisfied for neural networks. The Hessian may be singular, especially when the parameters have not fully converged, due to non-positive eigenvalues. To enforce positive-definiteness of the Hessian, Koh and Liang [30] add a damping term in the `iHVP`. Teso et al. [51] further approximate the Hessian with the Fisher information matrix (which is equivalent to the Gauss-Newton Hessian [41]) as follows:

$$r^{\star}_{-\mathbf{z},\text{damp},\text{lin}}(\epsilon) \approx \boldsymbol{\theta}^{\star} + (\mathbf{J}^{\top}_{\mathbf{y}\boldsymbol{\theta}^{\star}}\mathbf{H}_{\mathbf{y}}\mathbf{J}_{\mathbf{y}\boldsymbol{\theta}^{\star}} + \lambda\mathbf{I})^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}(f(\boldsymbol{\theta}^{\star},\mathbf{x}),\mathbf{t})\epsilon, \tag{5}$$

where $\mathbf{J}_{\mathbf{y}\boldsymbol{\theta}^{\star}}$ is the parameter-output Jacobian and $\mathbf{H}_{\mathbf{y}}$ is the Hessian of the cost with respect to the network outputs. Here, $\lambda > 0$ is a damping term to ensure the invertibility of Hessian or Gauss-Newton Hessian (GNH). Unlike the Hessian, the GNH is guaranteed to be positive semidefinite as long as the loss function is convex as a function of the network outputs [43].

4 Understanding the Discrepancy between Influence Function and LOO Retraining in Neural Networks

In this section, we investigate several factors responsible for the misalignment between influence functions and LOO retraining. Specifically, we decompose the misalignment into five separate terms: (1) the warm-start gap, (2) damping gap, (3) non-convergence gap, (4) linearization error, and (5) solver error. This decomposition captures all approximations and assumption violations when deploying influence functions in neural networks. We use the term "gap" rather than "error" for the first three terms to emphasize that they reflect differences between solutions to different influence-related questions, rather than actual errors. For all models we investigate, we find that the first three sources dominate the misalignment, indicating that the misalignment reflects not algorithmic errors but rather the fact that influence function estimators are answering a different question from what is normally assumed. All proximal objectives are summarized in Table 1.

(1) Warm-Start Gap: Non-Strongly Convex Training Objective. By taking a first-order Taylor approximation of the response function at $\epsilon_0 = 0$ (Eqn. 4), influence functions approximate the effect of removing a data point $\mathbf{z}$ at a local neighborhood of the optimum $\boldsymbol{\theta}^{\star}$. Hence, influence approximation has a more natural connection to the retraining scheme that initializes the network at the current optimum $\boldsymbol{\theta}^{\star}$ (*warm-start retraining*) than the scheme that initializes the network randomly (*cold-start retraining*). The warm-start optimum is equivalent to the cold-start optimum when the objective is strongly convex (where the solution to the response function is unique), making the influence estimation close to the LOO retraining on logistic regression with L_2 regularization.

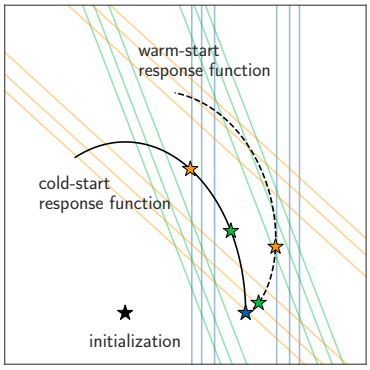


Figure 2: Cold-start (initialized from black star) and warm-start (initialized from blue star) response functions for quadratic cost function. Each contour represents the cost function at some ϵ. Because gradient descent converges to a minimum-norm solution, the warm-start and cold-start optima are not equivalent.

However, the equivalence between warm-start and cold-start optima is not typically guaranteed in neural networks [53]. Particularly, in the overparametrized regime ($N < d$), neural networks exhibit multiple global optima, and their converged solutions depend highly on the specifics of the optimization dynamics [38, 4, 6, 3]. For quadratic cost functions, gradient descent with initialization $\boldsymbol{\theta}^0$ converges to the optimum that achieves the minimum L_2 distance from $\boldsymbol{\theta}^0$ [24]. This phenomenon of the converged parameters being dependent on the initialization hinders influence functions from accurately predicting the effect of retraining the model from scratch as shown in Figure 2. We denote the discrepancy between cold-start and warm-start optima as **warm-start gap**.

(2) Proximity Gap: Addition of Damping Term in `iHVP`. In practical settings, we often impose a damping term (Eqn. 5) in influence approximations to ensure that the cost Hessian is positive-definite and hence invertible. As adding a damping term in influence estimations is equivalent to adding L_2 regularization to the cost function [43], when damping is used, influence functions can be seen as linearizing the following *proximal response function* at $\epsilon_0 = 0$:

$$r^{\star}_{-\mathbf{z},\text{damp}}(\epsilon) = \underset{\boldsymbol{\theta}\in\mathbb{R}^d}{\arg\min}\, \mathcal{Q}_{-\mathbf{z}}(\boldsymbol{\theta}, \epsilon) + \frac{\lambda}{2}\|\boldsymbol{\theta} - \boldsymbol{\theta}^{\star}\|^2. \tag{6}$$

Note that our use of "proximal" is based on the notion of proximal equilibria [17]. Intuitively, the proximal objective in Eqn. 6 not only minimizes the downweighted objective but also encourages the parameters to stay close to the optimal parameters at $\epsilon_0 = 0$. Hence, when the damping term is used in the `iHVP`, influence functions aim at approximating the warm-start retraining scheme with a proximity term that penalizes the L_2 distance between the new estimate and the optimal parameters. We call the discrepancy between the warm-start and proximal warm-start optima the **proximity gap**.

Interestingly, past works have observed that for quadratic cost functions, early stopping has a similar effect to L_2 regularization [53, 2]. Therefore, the proximal response function can be thought of as capturing how gradient descent will respond to a dataset perturbation if it takes only a limited number of steps starting from the warm-start solution.

(3) Non-Convergence Gap: Influence Estimation on Non-Converged Parameters. Thus far, our analysis has assumed that influence functions are computed on fully converged parameters $\boldsymbol{\theta}^{\star}$ at which the gradient of the cost is $\mathbf{0}$. However, in neural network training, we often terminate the optimization procedure before reaching the exact optimum due to several reasons, including having limited computational resources or to avoid overfitting [9]. In such situations, much of the change in the parameters from LOO retraining simply reflects the effect of training for longer, rather than the effect of removing a training example, as illustrated in Figure 3. What we desire from influence functions is to understand the effect of removing the training example; the effect of extended training is simply a nuisance. Therefore, to the extent that this factor contributes to the misalignment between influence functions and LOO retraining, influence functions are arguably *more useful* than LOO retraining.

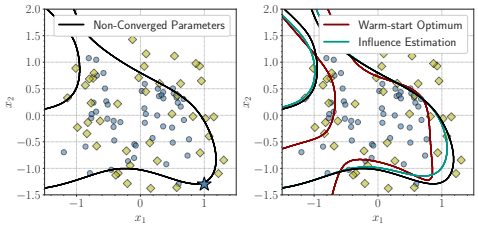


Figure 3: Decision boundaries for a partially trained binary classifier. We consider removing a data point located at right-bottom corner denoted as $\star$. While the influence estimation makes a local change on the data point of interest, the LOO retraining globally updates the parameters to better fit other data points (a nuisance from the perspective of understanding influence).

Since training the network to convergence may be impractical or undesirable, we instead modify the response function by replacing the original training objective with a similar one *for which the (possibly non-converged) final parameters $\boldsymbol{\theta}^s$ are optimal*. Here, we assume the loss function $\mathcal{L}(\cdot,\cdot)$ is convex as a function of the network outputs; this is true for commonly used loss functions such as squared error or softmax-cross-entropy. We replace the training loss with a term that penalizes mismatch to the predictions made by $\boldsymbol{\theta}^s$ (hence implying that $\boldsymbol{\theta}^s$ is optimal). Our *proximal Bregman response function (PBRF)* is defined as follows:

$$r^{b}_{-\mathbf{z},\text{damp}}(\epsilon) = \underset{\boldsymbol{\theta}\in\mathbb{R}^d}{\arg\min}\, \frac{1}{N}\sum_{i=1}^{N} D_{\mathcal{L}^{(i)}}(f(\boldsymbol{\theta}, \mathbf{x}^{(i)}), f(\boldsymbol{\theta}^s, \mathbf{x}^{(i)})) - \mathcal{L}(f(\boldsymbol{\theta}, \mathbf{x}), \mathbf{t})\epsilon + \frac{\lambda}{2}\|\boldsymbol{\theta} - \boldsymbol{\theta}^s\|^2, \tag{7}$$

where $D_{\mathcal{L}^{(i)}}(\cdot,\cdot)$ is the Bregman divergence defined as:

$$D_{\mathcal{L}^{(i)}}(\mathbf{y}, \mathbf{y}^s) = \mathcal{L}(\mathbf{y}, \mathbf{t}^{(i)}) - \mathcal{L}(\mathbf{y}^s, \mathbf{t}^{(i)}) - \nabla_{\mathbf{y}}\mathcal{L}(\mathbf{y}^s, \mathbf{t}^{(i)})^{\top}(\mathbf{y} - \mathbf{y}^s). \tag{8}$$

The PBRF defined in Eqn. 7 is composed of three terms. The first term measures the functional discrepancy between the current estimate and the parameters $\boldsymbol{\theta}^s$ in Bregman divergence, and its role is to prevent the new estimate from drastically altering the predictions on the training dataset. One

Error	Objective	Init
Cold-start	$\mathcal{J}(\boldsymbol{\theta}) - \mathcal{L}(f(\boldsymbol{\theta}, \mathbf{x}), \mathbf{t})\epsilon$	$\boldsymbol{\theta}^0$
+ Warm-start	$\mathcal{J}(\boldsymbol{\theta}) - \mathcal{L}(f(\boldsymbol{\theta}, \mathbf{x}), \mathbf{t})\epsilon$	$\boldsymbol{\theta}^s$
+ Proximity	$\mathcal{J}(\boldsymbol{\theta}) - \mathcal{L}(f(\boldsymbol{\theta}, \mathbf{x}), \mathbf{t})\epsilon + \frac{\lambda}{2}\lVert\boldsymbol{\theta} - \boldsymbol{\theta}^s\rVert^2$	$\boldsymbol{\theta}^s$
+ Non-Convergence	$\frac{1}{N}\sum_{i=1}^{N} D_{\mathcal{L}^{(i)}}(f(\boldsymbol{\theta}, \mathbf{x}^{(i)}), f(\boldsymbol{\theta}^s, \mathbf{x}^{(i)})) - \mathcal{L}(f(\boldsymbol{\theta}, \mathbf{x}), \mathbf{t})\epsilon + \frac{\lambda}{2}\lVert\boldsymbol{\theta} - \boldsymbol{\theta}^s\rVert^2$	$\boldsymbol{\theta}^s$
+ Linearization	$\frac{1}{N}\sum_{i=1}^{N} D_{\mathcal{L}^{(i)}_{\text{quad}}}(f_{\text{lin}}(\boldsymbol{\theta}, \mathbf{x}^{(i)}), f(\boldsymbol{\theta}^s, \mathbf{x}^{(i)})) - \nabla_{\boldsymbol{\theta}}\mathcal{L}(f(\boldsymbol{\theta}^s, \mathbf{x}), \mathbf{t})^\top\boldsymbol{\theta}\epsilon + \frac{\lambda}{2}\lVert\boldsymbol{\theta} - \boldsymbol{\theta}^s\rVert^2$	$\boldsymbol{\theta}^s$

Table 1: Summary of proximal objectives that influence functions aim to approximate when the network is non-strongly convex, a damping term is used, and influence functions are computed on non-converged parameters.

way of understanding this term in the cases of squared error or softmax-cross-entropy losses is that it is equivalent to the training error on a dataset where the original training labels are replaced with soft targets obtained from the predictions made by $\boldsymbol{\theta}^s$. The second term is the negative loss on the data point $\mathbf{z} = (\mathbf{x}, \mathbf{t})$, which aims to respond to the deletion of a training example. The final term is simply the proximity term described before. In Appendix B.3, we further show that the influence function on non-converged parameters is equivalent to the first-order approximation of PBRF instead of the first-order approximation of proximal response function for linear models.

Rather than computing the LOO retrained parameters by performing K additional optimization steps under the original training objective, we can instead perform K optimization steps under the proximal Bregman objective. The difference between the resulting parameter vectors is what we call the **non-convergence gap**.

(4) Linearization Error: A First-order Taylor Approximation of the Response Function. The key idea behind influence functions is the linearization of the response function. To simulate the local approximations made in influence functions, we define the linearized PBRF as:

$$r^b_{-\mathbf{z},\text{damp},\text{lin}}(\epsilon) = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^d} \frac{1}{N}\sum_{i=1}^{N} D_{\mathcal{L}^{(i)}_{\text{quad}}}(f_{\text{lin}}(\boldsymbol{\theta}, \mathbf{x}^{(i)}), f(\boldsymbol{\theta}^s, \mathbf{x}^{(i)})) - \nabla_{\boldsymbol{\theta}}\mathcal{L}(f(\boldsymbol{\theta}^s, \mathbf{x}), \mathbf{t})^\top\boldsymbol{\theta}\epsilon + \frac{\lambda}{2}\lVert\boldsymbol{\theta} - \boldsymbol{\theta}^s\rVert^2, \tag{9}$$

where $\mathcal{L}_{\text{quad}}(\cdot,\cdot)$ is the second-order expansion of the loss around $\mathbf{y}^s$ and $f_{\text{lin}}(\cdot,\cdot)$ is the linearization of the network outputs with respect to the parameters. The optimal solution to the linearized PBRF is equivalent to the influence estimation at the parameters $\boldsymbol{\theta}^s$ with the GNH approximation and a damping term λ (see Appendix B.4 for the derivation). As the linearized PBRF relies on several local approximations, the linearization error increases when the downweighting factor magnitude $|\epsilon|$ is large or the PBRF is highly non-linear. We refer to the discrepancy between the PBRF and linearized PBRF as the **linearization error**.

(5) Solver Error: A Crude Approximation of `iHVP`. As the precise computation of the `iHVP` is computationally infeasible, in practice, we use truncated `CG` or `LiSSA` to efficiently approximate influence functions [30]. Unfortunately, these efficient linear solvers introduce additional error by crudely approximating the `iHVP`. Moreover, different linear solvers can introduce specific biases in the influence estimation. For example, Vicol et al. [54] show that the truncated `LiSSA` algorithm implicitly adds an additional damping term in the `iHVP`. We use **solver error** to refer to the difference between the linearized PBRF and the influence estimation computed by a linear solver.

5 PBRF: The Question Influence Functions are Really Answering

The PBRF (Eqn. 7) approximates the effect of removing a data point while otherwise trying to keep the predictions consistent with those of the (partially) trained model. Since the discrepancy between the PBRF and influence function estimates is only due to the linearization and solver errors, the PBRF can be thought of as better representing the question that influence functions are trying to answer. Reframing influence functions in this way means that the PBRF can be regarded as a gold-standard ground truth for evaluating methods for influence function approximation. Existing analyses of

influence functions [7] rely on generating LOO retraining ground truth estimates by imposing strong L_2 regularization or training till convergence without early stopping. However, these conditions do not accurately reflect the typical way neural networks are trained in practice. In contrast, our PBRF formulation does not require the addition of any regularizers or modified training regimes and can be easily optimized.

In addition, although the PBRF may not necessarily align with LOO retraining due to the warm-start, proximity, and non-convergence gaps, the motivating use cases for influence functions typically do not rely on exact LOO retraining. This means that the PBRF can be used in place of LOO retraining for many tasks such as identifying influential or mislabelled examples, as demonstrated in Appendix D.2. In these cases, influence functions are still useful since they provide an efficient way of approximating PBRF estimates.

6 Experiments

Our experiments investigate the following questions: (1) What factors discussed in Section 4 contribute most to the misalignment between influence functions and LOO retraining? (2) While influence functions fail to approximate the effect of retraining, do they accurately approximate the PBRF? (3) How do changes in weight decay, damping, the number of total epochs, and the number of removed training examples affect each source of misalignment?

In all experiments, we first train the base network with the entire dataset to obtain the parameters $\boldsymbol{\theta}^s$. We repeat the training procedure 20 times with a different random training example deleted. The cold-start retraining begins from the same initialization used to train $\boldsymbol{\theta}^s$. All proximal objectives are trained with initialization $\boldsymbol{\theta}^s$ for 50% of the epochs used to train the base network. Lastly, we use the `LiSSA` algorithm with GNH approximation to compute influence functions. Since we are primarily interested in the effect of deleting a data point on model's predictions, we measure the discrepancy of each gap and error using the average L_2 distance between networks' outputs $\mathbb{E}_{(\mathbf{x},\cdot)\sim\mathcal{D}_{\text{train}}}[\|f(\boldsymbol{\theta},\mathbf{x}) - f(\boldsymbol{\theta}',\mathbf{x})\|]$ on the training dataset. We provide the full experimental set-up and additional experiments in Appendix C and D, respectively.

6.1 Influence Misalignment Decomposition

We first applied our decomposition to various models trained on a broad range of tasks covering binary classification, regression, image reconstruction, image classification, and language modeling. The summary of our results is provided in Figure 4 and Table 5 (Appendix E). Across all tasks, we found that the first three sources dominate the misalignment, indicating influence function estimators are answering a different question from what is normally assumed. Small linearization and solver errors indicates that influence functions accurately answers the modified question (PBRF).

Logistic Regression. We analyzed the logistic regression (LR) model trained on the Cancer and Diabetes classification datasets from the UCI collection [16]. We trained the model using `L-BFGS` [39] with L_2 regularization of 0.01 and damping term of $\lambda = 0.001$. As the training objective is strongly convex and the base model parameters were trained till convergence, we observed that each source of misalignment is significantly low. Hence, in case of logistic regression with L_2 regularization, influence functions accurately capture the effect of retraining the model without a data point.

Multilayer Perceptron. Next, we applied our analysis to the 2-hidden layer Multilayer Perceptron (MLP) with ReLU activations. We conducted the experiments in two settings: (1) regression on the Concrete and Energy datasets from the UCI collection and (2) image classification on 10% of the MNIST [15] and FashionMNIST [56] datasets, following the set-up from Koh and Liang [30] and Basu et al. [7]. We trained the networks for 1000 epochs using stochastic gradient descent (SGD) with a batch size of 128 and set a damping strength of $\lambda = 0.001$.

As opposed to linear models, MLPs violate the assumptions in the influence derivation and we observed an increase in gaps and errors on all five factors. We observed that warm-start, proximity, and the non-convergence gaps contribute more to the misalignment than linearization and solver errors. The average network's predictions for PBRF were similar to that computed by the `LiSSA` algorithm, demonstrating that influence functions are still a good approximation to PBRF.

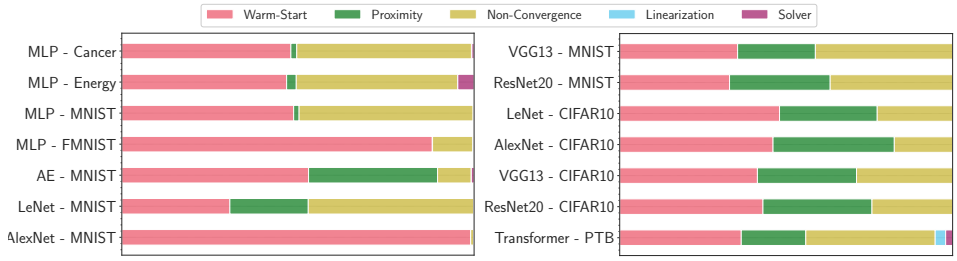


Figure 4: Decomposition of the discrepancy between influence functions and LOO retraining into (1) warm-start gap, (2) proximal gap, (3) non-convergence gap, (4) linearization error, and (5) solver error for each model and dataset. The size of each component is measured by the L_2 distance between the networks' outputs on the training dataset.

Autoencoder. Next, we applied our framework to an 8-layer autoencoder (AE) on the full MNIST dataset. We followed the experimental set-up from Martens and Grosse [42], where the encoder and decoder each consist of 4 fully-connected layers with sigmoid activation functions. We trained the network for 1000 epochs using SGD with momentum. We set the batch size to 1024, used L_2 regularization of 10^{-5} with a damping factor of $\lambda = 0.001$. In accordance with the findings from our MLP experiments, the warm-start, proximity, and non-convergence gaps were more significant than the linearization and solver errors, and influence functions accurately predicted the PBRF.

Convolutional Neural Networks. To investigate the source of discrepancy on larger-scale networks, we trained a set of convolutional neural networks of increasing complexity and size. Namely, LeNet [37], AlexNet [36], VGG13 [49], and ResNet-20 [25] were trained on 10% of the MNIST dataset and the full CIFAR10 [35] dataset. We trained the base network for 200 epochs on both datasets with a batch size of 128. For MNIST, we kept the learning rate fixed throughout training, while for CIFAR10, we decayed the learning rate by a factor of 5 at epochs 60, 120, and 160, following Zagoruyko and Komodakis [59]. We used L_2 regularization with strength $5 \cdot 10^{-4}$ and a damping factor of $\lambda = 0.001$. Consistent with the findings from our MLP and autoencoder experiments, the first three gaps were more significant than linearization and solver errors.

Model	Cold-Start		Warm-Start		PBRF	
	P	S	P	S	P	S
MLP	-0.55	0.01	0.22	0.35	**0.98**	**0.99**
LeNet	-0.19	0.12	0.32	0.25	**0.93**	**0.52**
AlexNet	-0.16	-0.08	0.51	0.58	**0.99**	**0.99**
VGG13	0.45	-0.07	-0.28	-0.51	**0.98**	**0.77**
ResNet-20	0.09	-0.06	0.02	0.09	**0.81**	**0.76**

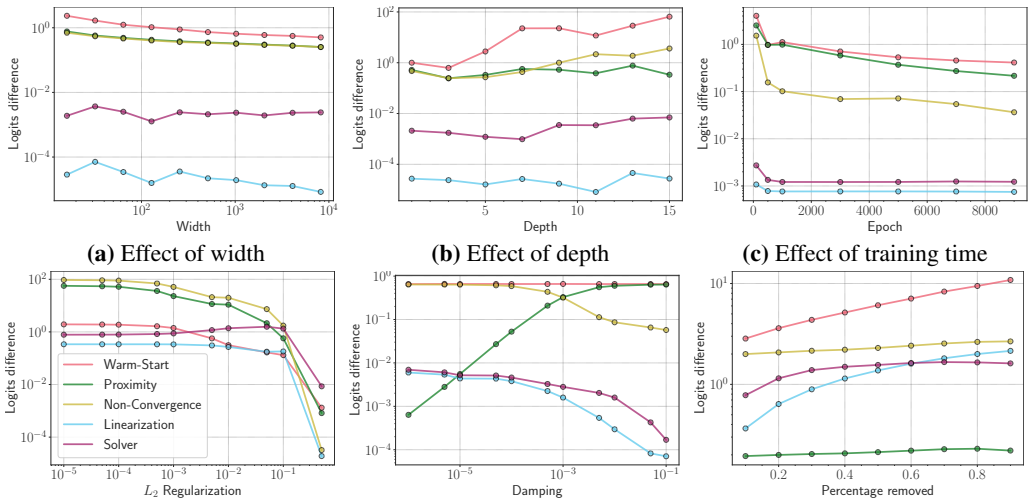
Table 2: Comparison of test loss differences computed by influence function, cold-start retraining, warm-start retraining, and PBRF on MNIST. We show Pearson (P) and Spearman rank-order (S) correlation when compared to influence estimates.

We further compared influence functions' approximations on the difference in test loss when a random training data point is removed with the value obtained from cold-start retraining, warm-start retraining, and PBRF in Table 2. We used both Pearson [48] and Spearman rank-order correlation [50] to measure the alignment. While the test loss predicted by influence functions do not align well with the values obtained by cold-start and warm-start retraining schemes, they show high correlations when compared to the estimates given by PBRF.

Transformer. Finally, we trained 2-layer Transformer language models on the Penn Treebank (PTB) [40] dataset. We set the number of hidden dimensions to 256 and the number of attention heads to 2. As we observed that model overfits after a few epochs of training, we trained the base network for 10 epochs using Adam. Notably, we observed that the non-convergence gap had the most considerable contribution to the discrepancy between influence functions and LOO retraining. Consistent with our previous findings, the first tree gaps had more impact on the discrepancy compared to linearization and solver errors.

6.2 Factors in Influence Misalignment

We further analyzed how the contribution of each component changes in response to changes in network width and depth, training time, weight decay, damping, and the percentage of data removed. We used an MLP trained on 10% of the MNIST dataset and summarized results in Figure 5.



(a) Effect of width **(b)** Effect of depth **(c)** Effect of training time

(d) Effect of L_2 regularization **(e)** Effect of damping strength **(f)** Effect of % data removed

Figure 5: Ablations on how various factors affect the contribution of the gaps and errors to the discrepancy between influence approximation and LOO retraining.

Width and Depth. We find that as we increase the width of the network, we observe a decrease in the linearization error. This is consistent with previous observations that networks behave more linearly as the width is increased [38]. In contrast to the findings from Basu et al. [7], we did not observe a strong relationship between the contribution of the components and the depth of the network.

Training Time. Unsurprisingly, as we increase the number of training epochs, we observe a decrease in the non-convergence gap.

Weight Decay. The weight decay allows the training objective to be better conditioned. Consequently, as weight decay increases, the training objective may act more as a strictly convex objective, resulting in a decrease in overall discrepancy for all components. Basu et al. [7] also found that the alignment between influence functions and LOO retraining increases as weight decay increases.

Damping. A higher damping term makes linear systems better conditioned, allowing solvers to find accurate solutions in fewer iterations [14], thereby reducing the solver error. Furthermore, the higher proximity term keeps the parameters close to $\boldsymbol{\theta}^s$, reducing the linearization error. However, increasing the effective proximity penalty directly increases the proximity gap.

Percentage of Training Examples Removed. As we remove more training examples from the dataset the PBRF becomes more non-linear and we observe a sharp increase in the linearization error. The cost landscape is also more likely to change as we remove more training examples, and we observe a corresponding increase in the warm-start gap.

7 Conclusion

In this paper, we investigate the sources of the discrepancy between influence functions and LOO retraining in neural networks. We decompose this difference into five distinct components: the warm-start gap, proximity gap, non-convergence gap, linearization error, and solver error. We empirically evaluate the contributions of each of these components on a wide variety of architectures and datasets and investigate how they change with factors such as network size and regularization. Our results show that the first three components are most responsible for the discrepancy between influence functions and LOO retraining. We further introduce the proximal Bregman response function (PBRF) to better capture the behavior of influence functions in neural networks. Compared to LOO retraining, the PBRF is more easily calculated and correlates better with influence functions, meaning it is an attractive alternative gold standard for evaluating influence functions. Although the PBRF may not necessarily align with LOO retraining, it can still be applied in many of the motivating use cases for influence functions. We conclude that influence functions in neural networks are not necessarily "fragile", but instead are giving accurate answers to a different question than is normally assumed.

References

[1] N. Agarwal, B. Bullins, and E. Hazan. Second-order stochastic optimization in linear time. *stat*, 1050:15, 2016.

[2] A. Ali, J. Z. Kolter, and R. J. Tibshirani. A continuous-time view of early stopping for least squares regression. In *The 22nd international conference on artificial intelligence and statistics*, pages 1370–1378. PMLR, 2019.

[3] S.-i. Amari, J. Ba, R. Grosse, X. Li, A. Nitanda, T. Suzuki, D. Wu, and J. Xu. When does preconditioning help or hurt generalization? *arXiv preprint arXiv:2006.10732*, 2020.

[4] S. Arora, S. Du, W. Hu, Z. Li, and R. Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning*, pages 322–332. PMLR, 2019.

[5] E. Barshan, M.-E. Brunet, and G. K. Dziugaite. RelatIF: Identifying explanatory training samples via relative influence. In S. Chiappa and R. Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 1899–1909. PMLR, 26–28 Aug 2020. URL `https://proceedings.mlr.press/v108/barshan20a.html`.

[6] P. L. Bartlett, P. M. Long, G. Lugosi, and A. Tsigler. Benign overfitting in linear regression. *Proceedings of the National Academy of Sciences*, 117(48):30063–30070, 2020.

[7] S. Basu, P. Pope, and S. Feizi. Influence functions in deep learning are fragile. *arXiv preprint arXiv:2006.14651*, 2020.

[8] S. Basu, X. You, and S. Feizi. On second-order group influence functions for black-box predictions. In *International Conference on Machine Learning*, pages 715–724. PMLR, 2020.

[9] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

[10] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

[11] M.-E. Brunet, C. Alkalay-Houlihan, A. Anderson, and R. Zemel. Understanding the origins of bias in word embeddings. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 803–811. PMLR, 09–15 Jun 2019. URL `https://proceedings.mlr.press/v97/brunet19a.html`.

[12] G. Charpiat, N. Girard, L. Felardos, and Y. Tarabalka. Input similarity from the neural network perspective. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/c61f571dbd2fb949d3fe5ae1608dd48b-Paper.pdf`.

[13] R. D. Cook. Influential observations in linear regression. *Journal of the American Statistical Association*, 74(365):169–174, 1979.

[14] J. W. Demmel. *Applied numerical linear algebra*. SIAM, 1997.

[15] L. Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[16] D. Dua and C. Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

[17] F. Farnia and A. Ozdaglar. Do gans always have nash equilibria? In *International Conference on Machine Learning*, pages 3029–3039. PMLR, 2020.

[18] A. Ghorbani, A. Abid, and J. Zou. Interpretation of neural networks is fragile. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):3681–3688, Jul. 2019. doi: 10.1609/aaai.v33i01.33013681. URL https://ojs.aaai.org/index.php/AAAI/article/view/4252.

[19] A. Griewank and A. Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

[20] H. Guo, N. Rajani, P. Hase, M. Bansal, and C. Xiong. FastIF: Scalable influence functions for efficient model interpretation and debugging. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10333–10350, Online and Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.808. URL https://aclanthology.org/2021.emnlp-main.808.

[21] F. R. Hampel. The influence curve and its role in robust estimation. *Journal of the american statistical association*, 69(346):383–393, 1974.

[22] X. Han, B. C. Wallace, and Y. Tsvetkov. Explaining black box predictions and unveiling data artifacts through influence functions. In D. Jurafsky, J. Chai, N. Schluter, and J. R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 5553–5563. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.492. URL https://doi.org/10.18653/v1/2020.acl-main.492.

[23] K. Hanawa, S. Yokoi, S. Hara, and K. Inui. Evaluation of similarity-based explanations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=9uvhpyQwzM_.

[24] T. Hastie, A. Montanari, S. Rosset, and R. J. Tibshirani. Surprises in high-dimensional ridgeless least squares interpolation. *The Annals of Statistics*, 50(2):949–986, 2022.

[25] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[26] Z. Izzo, M. A. Smart, K. Chaudhuri, and J. Zou. Approximate data deletion from machine learning models. In *International Conference on Artificial Intelligence and Statistics*, pages 2008–2016. PMLR, 2021.

[27] K. K and A. Søgaard. Revisiting methods for finding influential examples. *CoRR*, abs/2111.04683, 2021. URL https://arxiv.org/abs/2111.04683.

[28] R. Khanna, B. Kim, J. Ghosh, and S. Koyejo. Interpreting black box predictions using fisher kernels. In K. Chaudhuri and M. Sugiyama, editors, *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pages 3382–3390. PMLR, 16–18 Apr 2019. URL https://proceedings.mlr.press/v89/khanna19a.html.

[29] B. Kim, R. Khanna, and O. O. Koyejo. Examples are not enough, learn to criticize! criticism for interpretability. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/file/5680522b8e2bb01943234bce7bf84534-Paper.pdf.

[30] P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pages 1885–1894. PMLR, 2017.

[31] P. W. Koh, J. Steinhardt, and P. Liang. Stronger data poisoning attacks break data sanitization defenses. *Machine Learning*, 111(1):1–47, Jan 2022. ISSN 1573-0565. doi: 10.1007/s10994-021-06119-y. URL https://doi.org/10.1007/s10994-021-06119-y.

[32] P. W. W. Koh, K.-S. Ang, H. Teo, and P. S. Liang. On the accuracy of influence functions for measuring group effects. *Advances in neural information processing systems*, 32, 2019.

[33] S. Kong, Y. Shen, and L. Huang. Resolving training biases via influence-based data relabeling. In *International Conference on Learning Representations*, 2021.

[34] S. G. Krantz and H. R. Parks. *The implicit function theorem: history, theory, and applications*. Springer Science & Business Media, 2002.

[35] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[36] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.

[37] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

[38] J. Lee, L. Xiao, S. Schoenholz, Y. Bahri, R. Novak, J. Sohl-Dickstein, and J. Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *Advances in neural information processing systems*, 32, 2019.

[39] D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

[40] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993. URL https://aclanthology.org/J93-2004.

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

[42] J. Martens and R. Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417. PMLR, 2015.

[43] J. Martens et al. Deep learning via hessian-free optimization. In *ICML*, volume 27, pages 735–742, 2010.

[44] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[45] G. Pruthi, F. Liu, S. Kale, and M. Sundararajan. Estimating training data influence by tracing gradient descent. *Advances in Neural Information Processing Systems*, 33:19920–19930, 2020.

[46] A. Schioppa, P. Zablotskaia, D. Vilar, and A. Sokolov. Scaling up influence functions. *CoRR*, abs/2112.03052, 2021. URL https://arxiv.org/abs/2112.03052.

[47] P. Schulam and S. Saria. Can you trust this prediction? Auditing pointwise reliability after learning. In K. Chaudhuri and M. Sugiyama, editors, *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pages 1022–1031. PMLR, 16–18 Apr 2019. URL `https://proceedings.mlr.press/v89/schulam19a.html`.

[48] P. Sedgwick. Pearson's correlation coefficient. *Bmj*, 345, 2012.

[49] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[50] C. Spearman. The proof and measurement of association between two things. 1961.

[51] S. Teso, A. Bontempelli, F. Giunchiglia, and A. Passerini. Interactive label cleaning with example-based explanations. *Advances in Neural Information Processing Systems*, 34, 2021.

[52] A. W. Van der Vaart. *Asymptotic statistics*, volume 3. Cambridge university press, 2000.

[53] P. Vicol, J. Lorraine, D. Duvenaud, and R. Grosse. Implicit regularization in overparameterized bilevel optimization. 2022.

[54] P. Vicol, J. Lorraine, F. Pedregosa, D. Duvenaud, and R. Grosse. On implicit bias in overparameterized bilevel optimization. In *International Conference on Machine Learning*. PMLR, 2022.

[55] S. Wachter, B. D. Mittelstadt, and C. Russell. Counterfactual explanations without opening the black box automated decisions and the GDPR. *Harvard Journal of Law & Technology*, 31(2), 2018.

[56] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[57] Y.-Y. Yang and K. Chaudhuri. Understanding rare spurious correlations in neural networks. *arXiv preprint arXiv:2202.05189*, 2022.

[58] C.-K. Yeh, J. Kim, I. E.-H. Yen, and P. K. Ravikumar. Representer point selection for explaining deep neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper/2018/file/8a7129b8f3edd95b7d969dfc2c8e9d9d-Paper.pdf`.

[59] S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Section 5.
   (b) Did you describe the limitations of your work? [Yes] See Section 4 for ways in which the PBRF differs from LOO retraining and differs from influence estimates.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] Our work is fundamental in nature and does not enable immediate misuse.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See background in Section 3 and Appendix B.
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix B.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] We have not attached the code but we will release in the future.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We provide training details in section 6 and in C.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See full results in Table 5.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix C.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See dataset citations in section 6.1
   (b) Did you mention the license of the assets? [Yes] All datasets are licensed under MIT data license.
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

Our code will be made publicly available.

This appendix is structured as follows:

- In Section A, we provide an overview of the notation we use throughout the paper.
- In Section B, we provide derivations for influence functions, proximal response function, proximal Bregman response function, and linearized proximal response function.
- In Section C, we provide experimental details.
- In Section D, we provide additional experiment results.
- In Section E, we present the numerical results shown in Figure 4.
- In Section G, we provide an overview of `CG` and `LiSSA` algorithms.

A Table of Notation

Table 3 summarizes notations used in this paper.

Notation	Description
$\mathcal{D}_{\text{train}}$	Finite training dataset
N	Number of training examples, $N = \lvert\mathcal{D}_{\text{train}}\rvert$
$\mathbf{x}$	Input of a data point
$\mathbf{t}$	Target a data point
$\mathbf{z} = (\mathbf{x}, \mathbf{t})$	Data point composed of an input $\mathbf{x}$ and a target $\mathbf{t}$
$\mathbf{y} = f(\boldsymbol{\theta}, \mathbf{x})$	Prediction of the network f parameterized by the parameters $\boldsymbol{\theta}$
$\boldsymbol{\theta}$	Model parameters
$\boldsymbol{\theta}^\star$	Optimal model parameters on the full training dataset $\mathcal{D}_{\text{train}}$
$\boldsymbol{\theta}^\star_{-\mathbf{z}}$	Optimal model parameters on the dataset with a data point $\mathbf{z}$ removed $\mathcal{D}_{\text{train}} - \{\mathbf{z}\}$
$\mathcal{L}(\mathbf{y}, \mathbf{t})$	Loss function (e.g., squared error or cross-entropy)
$\mathcal{J}(\boldsymbol{\theta})$	Cost function on the full dataset $\mathcal{D}_{\text{train}}$
d	Number of model parameters
ϵ	Downweighting factor
$\mathcal{Q}_{-\mathbf{z}}(\boldsymbol{\theta}, \epsilon)$	Downweighted cost defined as $\mathcal{Q}_{-\mathbf{z}}(\boldsymbol{\theta}, \epsilon) = \mathcal{J}(\boldsymbol{\theta}) - \mathcal{L}(f(\boldsymbol{\theta}, \mathbf{x}), \mathbf{t})\epsilon$
$\boldsymbol{\theta}^\star_{-\mathbf{z},\epsilon}$	Optimal model parameters of the downweighted cost $\mathcal{Q}_{-\mathbf{z}}(\boldsymbol{\theta}, \epsilon)$, where ϵ is fixed
$r^\star_{-\mathbf{z}}(\epsilon)$	Response function
$\mathbf{J}_{\mathbf{y}\boldsymbol{\theta}^\star}$	Parameter-output Jacobian at $\boldsymbol{\theta}^\star$
$\mathbf{H}_{\mathbf{y}}$	Hessian of the cost with respect to the network outputs
λ	Damping strength
$r^\star_{-\mathbf{z},\text{damp}}(\epsilon)$	Proximal response function
$\boldsymbol{\theta}^s$	(Possibly non-converged) learned parameters
$\mathcal{D}_{\mathcal{L}^{(i)}}(\mathbf{y}, \mathbf{y}^s)$	The Bregman divergence $\mathcal{L}(\mathbf{y}, \mathbf{t}^{(i)}) - \mathcal{L}(\mathbf{y}^s, \mathbf{t}^{(i)}) - \nabla_{\mathbf{y}}\mathcal{L}(\mathbf{y}^s, \mathbf{t}^{(i)})^\top(\mathbf{y} - \mathbf{y}^s)$
$r^b_{-\mathbf{z},\text{damp}}(\epsilon)$	Proximal Bregman response function
$r^b_{-\mathbf{z},\text{damp,lin}}(\epsilon)$	Linearized proximal Bregman response function
$f_{\text{lin}}(\boldsymbol{\theta}, \mathbf{x})$	Linearized network outputs with respect to the parameters $f_{\text{lin}}(\boldsymbol{\theta}, \mathbf{x}) = f(\boldsymbol{\theta}^s, \mathbf{x}) + \mathbf{J}_{\mathbf{y}\boldsymbol{\theta}^s}(\boldsymbol{\theta} - \boldsymbol{\theta}^s)$
$\mathbf{y}^s = f(\boldsymbol{\theta}^s, \mathbf{x})$	Prediction of the network f parameterized by the parameters $\boldsymbol{\theta}^s$
$\mathcal{L}_{\text{quad}}(\mathbf{y}, \mathbf{t})$	Second-order Taylor expansion of the loss around $\mathbf{y}^s$

Table 3: A summary of the notation used in this paper.

B Derivations

B.1 Influence Function Derivation

We provide a derivation of influence functions using the response function. We refer readers to Van der Vaart [52] and Koh and Liang [30] for a more general derivation of influence functions.

Let $\mathbf{z} = (\mathbf{x}, \mathbf{t}) \in \mathcal{D}_{\text{train}}$ be a training example we are interested in downweighting. Recall that the downweighted objective is defined as:

$$\mathcal{Q}_{-\mathbf{z}}(\boldsymbol{\theta}, \epsilon) = \mathcal{J}(\boldsymbol{\theta}) - \mathcal{L}(f(\boldsymbol{\theta}, \mathbf{x}), \mathbf{t})\epsilon. \tag{10}$$

We further let $\epsilon_0 = 0$ and $\boldsymbol{\theta}^\star$ be the optimal parameters such that $\nabla_{\boldsymbol{\theta}} \mathcal{Q}_{-\mathbf{z}}(\boldsymbol{\theta}^\star, \epsilon_0) = \mathbf{0}$. Here, the optimal parameters $\boldsymbol{\theta}^\star$ is the solution that minimizes the cost function $\mathcal{J}(\cdot)$. We assume that the downweighted objective is twice continuously differentiable and strongly convex in the parameters $\boldsymbol{\theta}$ at ϵ_0. Note that if we assume the strong convexity of the loss function, the downweighted objective is only guaranteed to be strongly convex when $\epsilon \leq {}^1\!/_N$.

By the Implicit Function Theorem, these exists a unique continuously differentiable response function $r^\star_{-\mathbf{z}} \colon \mathcal{U}_0 \to \mathbb{R}^d$ defined on a neighborhood $\mathcal{U}_0$ of ϵ_0 such that $r^\star_{-\mathbf{z}}(\epsilon_0) = \boldsymbol{\theta}^\star$ and:

$$\nabla_{\boldsymbol{\theta}} \mathcal{Q}_{-\mathbf{z}}(r^\star_{-\mathbf{z}}(\epsilon), \epsilon) = \mathbf{0} \tag{11}$$

for all $\epsilon \in \mathcal{U}_0$. By taking the derivative with respect to the downweighting factor ϵ, we get:

$$\mathbf{0} = \frac{\mathrm{d}}{\mathrm{d}\epsilon}\left(\nabla_{\boldsymbol{\theta}} \mathcal{Q}_{-\mathbf{z}}(r^\star_{-\mathbf{z}}(\epsilon), \epsilon)\right) = \nabla^2_{\boldsymbol{\theta}} \mathcal{Q}_{-\mathbf{z}}(r^\star_{-\mathbf{z}}(\epsilon), \epsilon) \frac{\mathrm{d}r^\star_{-\mathbf{z}}}{\mathrm{d}\epsilon}(\epsilon) + \nabla^2_{\boldsymbol{\theta},\epsilon} \mathcal{Q}_{-\mathbf{z}}(r^\star_{-\mathbf{z}}(\epsilon), \epsilon) \tag{12}$$

for all $\epsilon \in \mathcal{U}_0$. The Jacobian of the response function at ϵ_0 can further be expressed as:

$$\mathbf{0} = \left(\nabla^2_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}^\star)\right) \left(\left.\frac{\mathrm{d}r^\star_{-\mathbf{z}}}{\mathrm{d}\epsilon}\right|_{\epsilon=\epsilon_0} \right) - \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\boldsymbol{\theta}^\star, \mathbf{x}), \mathbf{t}), \tag{13}$$

where we used these two equalities:

$$\nabla^2_{\boldsymbol{\theta}} \mathcal{Q}_{-\mathbf{z}}(r^\star_{-\mathbf{z}}(\epsilon_0), \epsilon_0) = \nabla^2_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}^\star) \tag{14}$$

$$\nabla^2_{\boldsymbol{\theta},\epsilon} \mathcal{Q}_{-\mathbf{z}}(r^\star_{-\mathbf{z}}(\epsilon_0), \epsilon_0) = -\nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\boldsymbol{\theta}^\star, \mathbf{x}), \mathbf{t}) \tag{15}$$

Rearranging Eqn. 13, we have:

$$\left.\frac{\mathrm{d}r^\star_{-\mathbf{z}}}{\mathrm{d}\epsilon}\right|_{\epsilon=\epsilon_0} = \left(\nabla^2_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}^\star)\right)^{-1} \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\boldsymbol{\theta}^\star, \mathbf{x}), \mathbf{t}), \tag{16}$$

where the Hessian $\nabla^2_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}^\star)$ is invertible by strong convexity of our downweighted objective at ϵ_0. Influence functions approximate the response function with a first-order Taylor expansion at ϵ_0:

$$r^\star_{-\mathbf{z},\text{lin}}(\epsilon) = r^\star_{-\mathbf{z}}(\epsilon_0) + \left.\frac{\mathrm{d}r^\star_{-\mathbf{z}}}{\mathrm{d}\epsilon}\right|_{\epsilon=\epsilon_0} (\epsilon - \epsilon_0) = \boldsymbol{\theta}^\star + (\nabla^2_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}^\star))^{-1} \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\boldsymbol{\theta}^\star, \mathbf{x}), \mathbf{t})\epsilon. \tag{17}$$

To approximate the optimal parameters trained without a data point $\mathbf{z}$, we can substitute $\epsilon = {}^1\!/_N$ as follows:

$$r^\star_{-\mathbf{z},\text{lin}}({}^1\!/_N) = \boldsymbol{\theta}^\star + \frac{1}{N} (\nabla^2_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}^\star))^{-1} \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\boldsymbol{\theta}^\star, \mathbf{x}), \mathbf{t}). \tag{18}$$

Influence functions can further approximate the loss at a particular test point $\mathbf{z}_{\text{test}} = (\mathbf{x}_{\text{test}}, \mathbf{t}_{\text{test}})$ (or test loss) when a training example $\mathbf{z}$ is eliminated from the training set using the chain rule [30]:

$$\begin{aligned}
&\mathcal{L}(f(r^\star_{-\mathbf{z},\text{lin}}({}^1\!/_N), \mathbf{x}_{\text{test}}), \mathbf{t}_{\text{test}}) \\
&\quad \approx \mathcal{L}(f(\boldsymbol{\theta}^\star, \mathbf{x}_{\text{test}}), \mathbf{t}_{\text{test}}) + \frac{1}{N} \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\boldsymbol{\theta}^\star, \mathbf{x}_{\text{test}}), \mathbf{t}_{\text{test}})^\top \left.\frac{\mathrm{d}r^\star_{-\mathbf{z}}}{\mathrm{d}\epsilon}\right|_{\epsilon=\epsilon_0} \\
&\quad \approx \mathcal{L}(f(\boldsymbol{\theta}^\star, \mathbf{x}_{\text{test}}), \mathbf{t}_{\text{test}}) + \frac{1}{N} \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\boldsymbol{\theta}^\star, \mathbf{x}_{\text{test}}), \mathbf{t}_{\text{test}})^\top (\nabla^2_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}^\star))^{-1} \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\boldsymbol{\theta}^\star, \mathbf{x}), \mathbf{t}).
\end{aligned} \tag{19}$$

B.2 Proximal Response Function Derivation

Let $\mathbf{z} = (\mathbf{x}, \mathbf{t}) \in \mathcal{D}_{\text{train}}$ be a training example we are interested in downweighting. Recall that the proximal response function (in Eqn. 6) is defined as:

$$r^\star_{-\mathbf{z},\text{damp}}(\epsilon) = \underset{\boldsymbol{\theta} \in \mathbb{R}^d}{\arg\min}\, \mathcal{Q}_{-\mathbf{z}}(\boldsymbol{\theta}, \epsilon) + \frac{\lambda}{2}\|\boldsymbol{\theta} - \boldsymbol{\theta}^\star\|^2, \tag{20}$$

where $\boldsymbol{\theta}^\star$ is the optimal parameters that minimize the cost function and $\lambda > 0$ is a damping term. Here, we show that influence estimations with a damping term correspond to a first-order Taylor approximation of the proximal response function. Let $\lambda > 0$ be some damping term and $\boldsymbol{\theta}^\star$ be the solution that minimizes the cost function. Let $\epsilon_0 = 0$ and assume that the downweighted objective is convex in the parameters $\boldsymbol{\theta}$ at ϵ_0. By the Implicit Function Theorem, we can guarantee the existence of the proximal response function $r^\star_{-\mathbf{z},\text{damp}} \colon \mathcal{U}_0 \to \mathbb{R}^d$ defined on a neighborhood $\mathcal{U}_0$ of ϵ_0 which satisfies $r^\star_{-\mathbf{z},\text{damp}}(\epsilon_0) = \boldsymbol{\theta}^\star$ and:

$$\nabla_{\boldsymbol{\theta}} \mathcal{Q}_{-\mathbf{z}}(r^\star_{-\mathbf{z},\text{damp}}(\epsilon), \epsilon) + \lambda \left(r_{-\mathbf{z},\text{damp}}(\epsilon)^\star - \boldsymbol{\theta}^\star\right) = \mathbf{0} \tag{21}$$

for all ϵ in some neighborhood $\mathcal{U}_0$ of ϵ_0. Then, differentiating Eqn. 21 with respect to the downweighting factor ϵ equates:

$$\mathbf{0} = \frac{\mathrm{d}}{\mathrm{d}\epsilon}\left(\nabla_{\boldsymbol{\theta}} \mathcal{Q}_{-\mathbf{z}}(r^\star_{-\mathbf{z},\text{damp}}(\epsilon), \epsilon) + \lambda\left(r^\star_{-\mathbf{z},\text{damp}}(\epsilon) - \boldsymbol{\theta}^\star\right)\right) \tag{22}$$

$$= \nabla^2_{\boldsymbol{\theta}} \mathcal{Q}_{-\mathbf{z}}(r^\star_{-\mathbf{z},\text{damp}}(\epsilon), \epsilon) \frac{\mathrm{d}r^\star_{-\mathbf{z},\text{damp}}}{\mathrm{d}\epsilon}(\epsilon) + \nabla^2_{\boldsymbol{\theta},\epsilon} \mathcal{Q}_{-\mathbf{z}}(r^\star_{-\mathbf{z},\text{damp}}(\epsilon), \epsilon) + \lambda \frac{\mathrm{d}r^\star_{-\mathbf{z},\text{damp}}}{\mathrm{d}\epsilon}(\epsilon) \tag{23}$$

for all $\epsilon \in \mathcal{U}_0$. Evaluating the response Jacobian at ϵ_0 and rearranging the terms in Eqn. 23:

$$\left.\frac{\mathrm{d}r^\star_{-\mathbf{z},\text{damp}}}{\mathrm{d}\epsilon}\right|_{\epsilon=\epsilon_0} = \left(\nabla^2_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}^\star) + \lambda \mathbf{I}\right)^{-1} \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\boldsymbol{\theta}^\star, \mathbf{x}), \mathbf{t}). \tag{24}$$

Hence, a first-order Taylor approximation of the proximal response function is equivalent to influence functions with a damping term $\lambda\mathbf{I}$ added:

$$r^\star_{-\mathbf{z},\text{damp,lin}}(\epsilon) = r^\star_{-\mathbf{z},\text{damp}}(\epsilon_0) + \left.\frac{\mathrm{d}r^\star_{-\mathbf{z},\text{damp}}}{\mathrm{d}\epsilon}\right|_{\epsilon=\epsilon_0} (\epsilon - \epsilon_0) \tag{25}$$

$$= \boldsymbol{\theta}^\star + (\nabla^2_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}^\star) + \lambda \mathbf{I})^{-1} \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\boldsymbol{\theta}^\star, \mathbf{x}), \mathbf{t})\epsilon. \tag{26}$$

When a damping term is used in influence functions, they approximate LOO retraining scheme with the proximity term added to the downweighted objective.

B.3 Proximal Bregman Response Function Derivation

As opposed to the derivation from Appendix B.1, we consider computing influence functions on parameters $\boldsymbol{\theta}^s$ that have not necessarily converged. When the parameters have not fully converged, a warm-start retraining with the downweighted objective defined in Eqn. 10 would simply minimize the cost function in the first term and reflect the effect of training longer, rather than the effect of removing a training example.

Let $\mathbf{z} = (\mathbf{x}, \mathbf{t}) \in \mathcal{D}_{\text{train}}$ be a training example we are interested in downweighting. We assume that the loss function is convex as a function of the network outputs which hold for commonly used loss functions. We replace the cost function in the downweighted objective with a term that penalizes mismatch to the predictions made by the current parameters $\boldsymbol{\theta}^s$ and define the proximal Bregman downweighted objective as:

$$\mathcal{Q}^b_{-\mathbf{z}}(\boldsymbol{\theta}, \epsilon) = \frac{1}{N}\sum_{i=1}^{N} D_{\mathcal{L}^{(i)}}(f(\boldsymbol{\theta}, \mathbf{x}^{(i)}), f(\boldsymbol{\theta}^s, \mathbf{x}^{(i)})) - \mathcal{L}(f(\boldsymbol{\theta}, \mathbf{x}), \mathbf{t})\epsilon + \frac{\lambda}{2}\|\boldsymbol{\theta} - \boldsymbol{\theta}^s\|, \tag{27}$$

where $D_{\mathcal{L}^{(i)}}(\cdot, \cdot)$ is the Bregman divergence defined as:

$$D_{\mathcal{L}^{(i)}}(\mathbf{y}, \mathbf{y}^s) = \mathcal{L}(\mathbf{y}, \mathbf{t}^{(i)}) - \mathcal{L}(\mathbf{y}^s, \mathbf{t}^{(i)}) - \nabla_{\mathbf{y}} \mathcal{L}(\mathbf{y}^s, \mathbf{t}^{(i)})^\top (\mathbf{y} - \mathbf{y}^s). \tag{28}$$

Because of our convexity assumption, the Bregman divergence term in Eqn. 27 is non-negative and is 0 when $\boldsymbol{\theta} = \boldsymbol{\theta}^s$. Hence, the parameters $\boldsymbol{\theta}^s$ is optimal at $\epsilon = 0$ on the proximal Bregman downweighted objective although it is not optimal on the cost function. Accordingly, we define the proximal Bregman response function (PBRF) as follows:

$$r^b_{-\mathbf{z},\text{damp}}(\epsilon) = \underset{\boldsymbol{\theta}\in\mathbb{R}^d}{\arg\min} \frac{1}{N}\sum_{i=1}^{N} D_{\mathcal{L}^{(i)}}(f(\boldsymbol{\theta}, \mathbf{x}^{(i)}), f(\boldsymbol{\theta}^s, \mathbf{x}^{(i)})) - \mathcal{L}(f(\boldsymbol{\theta}, \mathbf{x}), \mathbf{t})\epsilon + \frac{\lambda}{2}\|\boldsymbol{\theta} - \boldsymbol{\theta}^s\|, \tag{29}$$

where we assume that the proximal Bregman downweighted objective is strongly convex at ϵ_0 and the solution to the Bregman downweighted objective is unique. Letting $\epsilon_0 = 0$, the Bregman response function satisfies $r^b_{-\mathbf{z},\text{damp}}(\epsilon_0) = \boldsymbol{\theta}^s$ and:

$$\nabla_{\boldsymbol{\theta}} \mathcal{Q}^b_{-\mathbf{z}}(r^b_{-\mathbf{z},\text{damp}}(\epsilon), \epsilon) = \mathbf{0}. \tag{30}$$

for all downweighting factor ϵ in some neighborhood of ϵ_0. By taking the derivative with respect to the downweighting factor ϵ, we get:

$$\mathbf{0} = \frac{\mathrm{d}}{\mathrm{d}\epsilon}\left(\nabla_{\boldsymbol{\theta}} \mathcal{Q}^b_{-\mathbf{z}}(r^b_{-\mathbf{z},\text{damp}}(\epsilon), \epsilon)\right) \tag{31}$$

$$= \nabla^2_{\boldsymbol{\theta}} \mathcal{Q}^b_{-\mathbf{z}}(r^b_{-\mathbf{z},\text{damp}}(\epsilon), \epsilon)\frac{\mathrm{d}r^b_{-\mathbf{z},\text{damp}}}{\mathrm{d}\epsilon}(\epsilon) + \nabla^2_{\boldsymbol{\theta},\epsilon} \mathcal{Q}^b_{-\mathbf{z}}(r^b_{-\mathbf{z},\text{damp}}(\epsilon), \epsilon). \tag{32}$$

For linear models, where the parameter-output Jacobian is constant, the Jacobian of the response function at ϵ_0 can further be expressed as:

$$\mathbf{0} = \left(\nabla^2_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}^s) + \lambda \mathbf{I}\right)\left(\left.\frac{\mathrm{d}r^b_{-\mathbf{z},\text{damp}}}{\mathrm{d}\epsilon}\right|_{\epsilon=\epsilon_0}\right) - \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\boldsymbol{\theta}^s, \mathbf{x}), \mathbf{t}), \tag{33}$$

where we used these two equalities:

$$\nabla^2_{\boldsymbol{\theta}} \mathcal{Q}^b_{-\mathbf{z}}(r^b_{-\mathbf{z},\text{damp}}(\epsilon_0), \epsilon_0) = \nabla^2_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}^s) + \lambda \mathbf{I} \tag{34}$$

$$\nabla^2_{\boldsymbol{\theta},\epsilon} \mathcal{Q}^b_{-\mathbf{z}}(r^b_{-\mathbf{z},\text{damp}}(\epsilon_0), \epsilon_0) = -\nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\boldsymbol{\theta}^s, \mathbf{x}), \mathbf{t}) \tag{35}$$

Rearranging the terms in Eqn. 33, the Jacobian of the PBRF at ϵ_0 can be expressed as:

$$\left.\frac{\mathrm{d}r^b_{-\mathbf{z}}}{\mathrm{d}\epsilon}\right|_{\epsilon=\epsilon_0} = \left(\nabla^2_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}^s) + \lambda \mathbf{I}\right)^{-1} \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\boldsymbol{\theta}^s, \mathbf{x}), \mathbf{t}). \tag{36}$$

Note that both Hessian and gradient are computed on the final parameters $\boldsymbol{\theta}^s$ instead of the optimal parameters $\boldsymbol{\theta}^\star$. Hence, influence functions at the non-converged parameters $\boldsymbol{\theta}^s$ (with a damping) can be seen as an approximation to the PBRF rather than an approximation to LOO retraining.

B.4 Linearized Proximal Bregman Response Function Derivation

We show that the linearized proximal Bregman response function (PBRF) is equivalent to the influence estimation with the Gauss-Newton Hessian approximation and a damping term $\lambda > 0$. Let $\boldsymbol{\theta}^s \in \mathbb{R}^d$ be a possibly non-converged learned parameters, $\mathbf{z} = (\mathbf{x}, \mathbf{t}) \in \mathcal{D}_{\text{train}}$ be a training example we want to downweight, and $\lambda > 0$ be a damping term. Recall that the linearized PBRF is defined as:

$$\begin{aligned} r^b_{-\mathbf{z},\text{damp},\text{lin}}(\epsilon) = \underset{\boldsymbol{\theta}\in\mathbb{R}^d}{\arg\min} \frac{1}{N}\sum_{i=1}^{N} & D_{\mathcal{L}^{(i)}_{\text{quad}}}(f_{\text{lin}}(\boldsymbol{\theta}, \mathbf{x}^{(i)}), f(\boldsymbol{\theta}^s, \mathbf{x}^{(i)})) \\ & - \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\boldsymbol{\theta}^s, \mathbf{x}), \mathbf{t})^\top \boldsymbol{\theta}\epsilon + \frac{\lambda}{2}\|\boldsymbol{\theta} - \boldsymbol{\theta}^s\|^2, \end{aligned} \tag{37}$$

where $\mathcal{L}_{\text{quad}}(\cdot, \cdot)$ and $f_{\text{lin}}(\boldsymbol{\theta}, \mathbf{x})$ are defined as:

$$\mathcal{L}_{\text{quad}}(\mathbf{y}, \mathbf{t}) = \mathcal{L}(\mathbf{y}^s, \mathbf{t}) + \nabla_{\mathbf{y}} \mathcal{L}(\mathbf{y}^s, \mathbf{t})^\top(\mathbf{y} - \mathbf{y}^s) + (\mathbf{y} - \mathbf{y}^s)^\top \nabla^2_{\mathbf{y}} \mathcal{L}(\mathbf{y}^s, \mathbf{t})(\mathbf{y} - \mathbf{y}^s). \tag{38}$$

$$f_{\text{lin}}(\boldsymbol{\theta}, \mathbf{x}^{(i)}) = f(\boldsymbol{\theta}^s, \mathbf{x}^{(i)}) + \mathbf{J}_{\mathbf{y}^{(i)}\boldsymbol{\theta}^s}(\boldsymbol{\theta} - \boldsymbol{\theta}^s). \tag{39}$$

Here, $\mathbf{y}^s$ is the prediction of the network parameterized by $\boldsymbol{\theta}^s$ and $\mathbf{J}_{\mathbf{y}^{(i)}\boldsymbol{\theta}^s}$ is the parameter-output Jacobian of the i-th training example. We further assume that the loss function is convex as a function of the network outputs which hold for commonly used loss functions. The first Bregman divergence term in Eqn. 37 can be expressed as:

$$
\begin{aligned}
D_{\mathcal{L}^{(i)}_{\text{quad}}}(f_{\text{lin}}(\boldsymbol{\theta}, \mathbf{x}^{(i)}), \mathbf{y}^s) &= \nabla_{\mathbf{y}}\mathcal{L}(\mathbf{y}^s, \mathbf{t}^{(i)})^\top \mathbf{J}_{\mathbf{y}^{(i)}\boldsymbol{\theta}^s}(\boldsymbol{\theta} - \boldsymbol{\theta}^s) && (40)\\
&\quad + (\boldsymbol{\theta} - \boldsymbol{\theta}^s)^\top \mathbf{J}^\top_{\mathbf{y}^{(i)}\boldsymbol{\theta}^s} \nabla^2_{\mathbf{y}}\mathcal{L}(\mathbf{y}^s, \mathbf{t}^{(i)}) \mathbf{J}_{\mathbf{y}^{(i)}\boldsymbol{\theta}^s}(\boldsymbol{\theta} - \boldsymbol{\theta}^s) && (41)\\
&\quad - \nabla_{\mathbf{y}}\mathcal{L}(\mathbf{y}^s, \mathbf{t}^{(i)})^\top \mathbf{J}_{\mathbf{y}^{(i)}\boldsymbol{\theta}^s}(\boldsymbol{\theta} - \boldsymbol{\theta}^s) && (42)\\
&= (\boldsymbol{\theta} - \boldsymbol{\theta}^s)^\top \mathbf{J}^\top_{\mathbf{y}^{(i)}\boldsymbol{\theta}^s} \nabla^2_{\mathbf{y}}\mathcal{L}(\mathbf{y}^s, \mathbf{t}^{(i)}) \mathbf{J}_{\mathbf{y}^{(i)}\boldsymbol{\theta}^s}(\boldsymbol{\theta} - \boldsymbol{\theta}^s). && (43)
\end{aligned}
$$

Now, taking the gradient of linearized PBRF objective with respect to the parameters $\boldsymbol{\theta}$ and setting it equal to $\mathbf{0}$, we get:

$$
\begin{aligned}
\mathbf{0} &= \frac{1}{N}\sum_{i=1}^{N}\left(\mathbf{J}^\top_{\mathbf{y}^{(i)}\boldsymbol{\theta}^s}\nabla^2_{\mathbf{y}}\mathcal{L}(\mathbf{y}^s, \mathbf{t}^{(i)})\mathbf{J}_{\mathbf{y}^{(i)}\boldsymbol{\theta}^s}(\boldsymbol{\theta} - \boldsymbol{\theta}^s)\right) - \nabla_{\boldsymbol{\theta}}\mathcal{L}(f(\boldsymbol{\theta}^s, \mathbf{x}), \mathbf{t})\epsilon + \lambda(\boldsymbol{\theta} - \boldsymbol{\theta}^s)\\
&= \mathbf{J}^\top_{\mathbf{y}\boldsymbol{\theta}^s}\mathbf{H}^s_{\mathbf{y}}\mathbf{J}_{\mathbf{y}\boldsymbol{\theta}^s}(\boldsymbol{\theta} - \boldsymbol{\theta}^s) - \nabla_{\boldsymbol{\theta}}\mathcal{L}(f(\boldsymbol{\theta}^s, \mathbf{x}), \mathbf{t})\epsilon + \lambda(\boldsymbol{\theta} - \boldsymbol{\theta}^s),
\end{aligned} \tag{44}
$$

where $\mathbf{H}^s_{\mathbf{y}}$ is the Hessian of the loss with respect to the network outputs evaluated at $\mathbf{y}^s$. Rearranging the terms, we get:

$$
\boldsymbol{\theta}^\star_{\text{lin, PBRF}} = \boldsymbol{\theta}^s + \left(\mathbf{J}^\top_{\mathbf{y}\boldsymbol{\theta}^s}\mathbf{H}^s_{\mathbf{y}}\mathbf{J}_{\mathbf{y}\boldsymbol{\theta}^s} + \lambda\mathbf{I}\right)^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}(f(\boldsymbol{\theta}^s, \mathbf{x}), \mathbf{t})\epsilon, \tag{45}
$$

where $\boldsymbol{\theta}^\star_{\text{lin, PBRF}}$ is the optimal solution to the linearized PBRF objective. Note that the Gauss-Newton Hessian (GNH) $\mathbf{G}^s = \mathbf{J}^\top_{\mathbf{y}\boldsymbol{\theta}^s}\mathbf{H}^s_{\mathbf{y}}\mathbf{J}_{\mathbf{y}\boldsymbol{\theta}^s}$ is positive semidefinite (assuming that the loss function is convex as a function of network outputs) and an addition of damping term guarantees the invertibility. Therefore, the optimal solution to Eqn. 45 is equivalent to the influence estimation with the GNH approximation and a damping term λ.

C Experimental Details

C.1 Computing Environment

All experiments were implemented using the PyTorch [44] and JAX [10] frameworks and we ran all experiments on NVIDIA P100 GPUs.

C.2 Experiment Set-up

In all experiments, we first trained the base network (initialized with some $\boldsymbol{\theta}^0$) with the entire dataset for K epochs to obtain the base parameters $\boldsymbol{\theta}^s$. We then select 20 random data points $\mathbf{z}_t \in \mathcal{D}_{\text{train}}$ from the training dataset and computed the additional parameters for each t as follows:

1. **Cold optimum.** We retrained the network for $K + {}^K\!/_2$ epochs with the initialization $\boldsymbol{\theta}^0$. To minimize the effect of stochasticity in stochastic gradient-based optimizer, we further used the same batch order used for training the base network for the first K epochs.
2. **Warm optimum.** We retrained the network for ${}^K\!/_2$ epochs with the initialization $\boldsymbol{\theta}^s$.
3. **Proximal warm optimum.** We retrained the network for ${}^K\!/_2$ epochs with the initialization $\boldsymbol{\theta}^s$ using the objective defined in Eqn. 6.
4. **Proximal Bregman warm optimum.** We retrained the network for ${}^K\!/_2$ epochs with the initialization $\boldsymbol{\theta}^s$ using the objective defined in Eqn. 7.
5. **Linearized proximal Bregman warm optimum.** We retrained the network for ${}^K\!/_2$ epochs with the initialization $\boldsymbol{\theta}^s$ using the objective defined in Eqn. 9.
6. **Influence estimation.** We used the `LiSSA` algorithm on the base parameters $\boldsymbol{\theta}^s$ with the Gauss-Newton Hessian approximation (Eqn. 5).

The "warm-start gap" refers to the discrepancy between cold-start and warm-start optima. The "proximity gap" refers to the discrepancy between warm-start and proximal warm-start optima. The "non-convergence gap" denotes the discrepancy between proximal warm-start and proximal Bregman warm-start optima. The "linearization error" represents the discrepancy between proximal Bregman warm-start and linearized proximal Bregman warm-start optima. Lastly, the "solver error" denotes the discrepancy between linearized proximal Bregman warm optimum and influence estimation with the `LiSSA` algorithm.

We treated the scaling in the `LiSSA` algorithm as a separate hyperparameter [30] and tuned the scaling in the range {10, 25, 50, 100, 150, 200, 250, 300, 400, 500} so that the algorithm converges.

C.3 Influence Misalignment Decomposition

Logistic Regression. We used Cancer and Diabetes classification datasets from the UCI collection [16]. In training, we normalized the input features to have zero mean and unit variance. We trained the model using `L-BFGS` with L_2 regularization of 0.01 and damping term of $\lambda = 0.001$.

Multilayer Perceptron. For regression experiments, we used 2-hidden layer MLP with 128 hidden units. For both Concrete and Energy datasets, we normalized the input features and targets to have a zero mean and unit variance. For classification experiments with 10% of MNIST [15] and FashionMNIST [56] datasets, we used 2-hidden layer MLP with a hidden unit dimension of 1024. For both regression and classification experiments, we used a batch size of 128 and trained the base network for 1000 epochs using SGD. While we did not use any L_2 regularization, we set the damping strength to $\lambda = 0.001$.

We conducted the hyperparameter searches over the learning rates for the base model, making choices based on the final validation loss. We swept over the learning rates {1.0, 0.3, 0.1, 0.03, 0.01, 0.003, 0.001}. We used a learning rate decayed by a factor of 10 for computing PBRF and linearized PBRF. We set the recursion depth to 5000 and the number of repeat to 5 for the `LiSSA` algorithm.

Autoencoder. We used the same experimental set-up from Martens and Grosse [42]. The loss function was the binary cross-entropy and the L_2 regularization with strength $5 \cdot 10^{-5}$ was added to the cost function. The layer widths for the autoencoder were [784, 1000, 500, 250, 30, 250, 500, 1000, 784] and we used sigmoid activation functions. We trained the network for 1000 epochs on the full MNIST dataset with SGDm (SGD with momentum) and set the batch size to 1024. The damping term was set to $\lambda = 0.001$.

We conducted the hyperparameter searches for the base model making choices based on the final validation loss. We kept the momentum to 0.9 and swept over the learning rates 1, 0.3, 0.1, 0.03, 0.01, 0.003, 0.001. We used a learning rate decayed by a factor of 10 for computing PBRF and linearized PBRF. We set the recursion depth to 10000 and the number of repeat to 5 for the `LiSSA` algorithm.

Convolution Neural Networks. We trained LeNet [37], AlexNet [36], VGG13 [49], and ResNet-20 [25] on 10% of MNIST dataset and the full CIFAR10 [35] dataset. For the MNIST experiment, we kept the learning rate fixed, while for CIFAR10 experiment, we decayed the initial learning rate by a factor of 5 at epochs 60, 120, 160. We used L_2 regularization of $5 \cdot 10^{-4}$ and a damping factor of $\lambda = 0.001$. For both datasets, we trained the base network for 200 epochs with the batch size of 128.

We used 10% of the MNIST dataset for the test loss correlation experiments in Table 2 and computed the approximated test loss with Eqn. 19 on randomly selected test examples. We conducted the hyperparameter searches for the base model making choices based on the final validation accuracy. We fixed the momentum to 0.9 and swept over the initial learning rates {1.0, 0.3, 0.1, 0.03, 0.01, 0.003, 0.001}. We used a learning rate decayed by a factor of 10 for computing PBRF and linearized PBRF. We set the recursion depth to 10000 and the number of repeat to 5 for the `LiSSA` algorithm.

Transformer. We trained a 2-layer Transformer language model on the Penn Treebank (PTB) dataset [40]. The number of hidden dimensions was set to 256 and the number of attention heads was set to 2. We trained the model with Adam for 10 epochs. We set the batch size to 20 and a damping

Model	Dataset	Warm-Start	Proximity	Non-Convergence	Linearization	Solver
MLP	Concrete	0.079 ± 0.008	0.002 ± 0.000	$\mathbf{0.082} \pm 0.001$	0.000 ± 0.000	0.017 ± 0.026
	Energy	$\mathbf{0.016} \pm 0.001$	0.001 ± 0.000	$\mathbf{0.016} \pm 0.001$	0.000 ± 0.000	0.000 ± 0.000
	MNIST	$\mathbf{0.208} \pm 0.732$	0.006 ± 0.024	0.211 ± 0.745	0.000 ± 0.000	0.003 ± 0.006
	FashionMNIST	$\mathbf{2.968} \pm 0.545$	0.385 ± 0.028	0.408 ± 0.033	0.000 ± 0.001	0.009 ± 0.014

Table 4: Decomposition of the discrepancy between influence functions (without the Gauss-Newton Hessian approximation) and LOO retraining into (1) warm-start gap, (2) proximal gap, (3) non-convergence gap, (4) linearization error, and (5) solver error for each model and dataset. Different from Table 5, we computed influence functions **without** the Gauss-Newton Hessian approximation. The size of each component is measured by the L_2 distance between the networks' outputs on the training dataset.

term of $\lambda = 0.01$. We conducted the hyperparameter searches for the base model making choices based on the final validation perplexity. We swept over the learning rates {0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001} for training the base model and set the recursion depth to 5000 and the number of repeat to 5 for the `LiSSA` algorithm.

C.4 Factors in Influence Misalignment

The experiment in Section 6.2 was conducted using 10% of the MNIST dataset. We trained 2-hidden layer MLP composed of 1024 hidden units for 1000 epochs using SGD. We used the batch size of 128 and conducted the hyperparameter searches for each model making choices based on the final validation loss. We swept over the learning rates {1.0, 0.3, 0.1, 0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001}.

To see how the gaps and errors change as we increase the width of the network, we repeated the experiments with all widths in a set {16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192} while keeping the depth fixed to 2. To see the effect of increase in the depth of the network, we fixed the width to 1024 and computed gaps and errors with depth {1, 3, 5, 7, 9, 11, 13, 15}.

While keeping all configurations the same (2-hidden layer MLP with 1024 hidden units), we computed the decomposition by changing the total number of epochs to train the base model in the range {100, 500, 1000, 3000, 5000, 7000, 9000}, changing the strength of the weight decay {0.5, 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001, 0.00005, 0.00001}, and changing the strength of the damping term in the range {0.5, 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001, 0.00005, 0.00001, 0.000005, 0.000001}. Lastly, we modified the downweighted objective to remove a group of training examples (rather than a single training example) and altered the percentage we remove the training dataset in range {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9}.

D Additional Results

D.1 Influence Functions without the Gauss-Newton Hessian Approximation

In all experiments, we computed influence functions using the Gauss-Newton Hessian (GNH) approximation. As the previous error analysis was conducted without the GNH approximation [7], we repeated a subset of our experiments without the GNH approximation (using the Hessian matrix). The results are summarized in Table 4.

As the warm-start gap, proximity gap, non-convergence gap, and linearization error do not depend on the way influence estimates are computed, these numerical values are identical to the results in Table 5. However, as the linearized PBRF optimum is equivalent to influence estimations with the GNH approximation, the solver error slightly increased when we computed influence functions with the Hessian for all datasets. Nevertheless, the solver error is still significantly lower than other decomposition terms.

Furthermore, we investigated how the test loss difference on randomly selected test examples approximated by influence functions correlates with the actual value computed using cold-start

Model	Dataset	Warm-Start	Proximity	Non-Convergence	Linearization	Solver
LR	Cancer	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
	Diabetes	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
MLP	Concrete	0.079 ± 0.008	0.002 ± 0.000	$\mathbf{0.082} \pm 0.001$	0.000 ± 0.000	0.001 ± 0.002
	Energy	$\mathbf{0.016} \pm 0.001$	0.001 ± 0.000	$\mathbf{0.016} \pm 0.001$	0.000 ± 0.000	0.000 ± 0.000
	MNIST	$\mathbf{0.208} \pm 0.732$	0.006 ± 0.024	0.211 ± 0.745	0.000 ± 0.000	0.001 ± 0.004
	FashionMNIST	$\mathbf{2.968} \pm 0.545$	0.385 ± 0.028	0.408 ± 0.033	0.000 ± 0.001	0.007 ± 0.010
Autoencoder	MNIST	$\mathbf{20.743} \pm 0.522$	14.330 ± 0.184	11.409 ± 0.430	0.000 ± 0.000	0.307 ± 0.088
LeNet	MNIST	$\mathbf{7.434} \pm 1.068$	5.393 ± 0.547	3.748 ± 0.324	0.000 ± 0.001	0.001 ± 0.001
AlexNet		$\mathbf{13.403} \pm 1.289$	0.001 ± 0.000	0.118 ± 0.000	0.000 ± 0.000	0.013 ± 0.002
VGG13		7.389 ± 0.744	4.872 ± 1.052	$\mathbf{8.601} \pm 0.478$	0.000 ± 0.000	0.001 ± 0.001
ResNet-20		4.433 ± 0.059	4.061 ± 0.157	$\mathbf{4.940} \pm 0.155$	0.001 ± 0.000	0.002 ± 0.001
LeNet	CIFAR10	$\mathbf{10.668} \pm 0.162$	6.520 ± 0.442	5.032 ± 0.799	0.001 ± 0.000	0.000 ± 0.000
AlexNet		$\mathbf{7.530} \pm 0.233$	5.956 ± 0.102	2.864 ± 0.367	0.000 ± 0.000	0.000 ± 0.000
VGG13		$\mathbf{8.410} \pm 1.926$	6.279 ± 0.708	6.031 ± 0.660	0.000 ± 0.000	0.073 ± 0.161
ResNet-20		$\mathbf{5.827} \pm 0.152$	4.435 ± 0.669	3.280 ± 0.429	0.000 ± 0.000	0.003 ± 0.001
Transformer	PTB	57.926 ± 5.055	30.756 ± 0.673	$\mathbf{61.675} \pm 1.042$	5.002 ± 1.556	3.316 ± 2.630

Table 5: Decomposition of the discrepancy between influence functions and LOO retraining into (1) warm-start gap, (2) proximal gap, (3) non-convergence gap, (4) linearization error, and (5) solver error for each model and dataset. The size of each component is measured by the L_2 distance between the networks' outputs on the training dataset.

Dataset	Cold-Start		Warm-Start		PBRF	
	P	S	P	S	P	S
Concrete	-0.11	0.12	0.09	0.11	**0.93**	**0.94**
Energy	0.03	0.04	0.09	0.13	**0.97**	**0.91**
MNIST	-0.10	0.01	0.22	0.35	**0.98**	**0.91**
FashionMNIST	0.16	0.00	0.06	0.07	**0.90**	**0.92**

Table 6: Comparison of test loss differences computed by influence function (**with** the Gauss-Newton Hessian approximation), cold-start retraining, warm-start retraining, and PBRF. We show Pearson (P) and Spearman rank-order (S) correlation when compared to influence estimates.

retraining, warm-start retraining, and the PBRF. Table 6 shows the correlations with influence approximations using the GNH approximation. While influence estimates do not accurately predict the effect of retraining the model, they closely align the values obtained by the PBRF. Moreover, we conducted the same experiment with influence approximations without the GNH approximation (using the Hessian matrix) and show the results in Table 7. Similar to the previous results, the test loss differences predicted by influence functions align with the value obtained by the PBRF while failing to capture the effect of retraining the model for both regression and classification datasets. Hence, in both settings, the PBRF better captures the behavior of influence functions than LOO retraining.

D.2 Mislabelled Examples Detection

While the PBRF may not necessarily align with LOO retraining because of warm-start, proximity, and non-convergence gaps, the motivating use cases for influence functions typically do not rely on exact LOO retraining. Hence, the PBRF can be used instead of LOO retraining for many tasks, such as identifying influential or mislabelled examples. We conducted an additional experiment to verify that the PBRF (and influence function) is still helpful in detecting mislabelled training data points.

We used 10% of the MNIST dataset and randomly corrupted 10% of the training examples by assigning random labels. Then, we simulated the scenario where we manually inspect a fraction of training examples, correcting them if they were mislabelled. We trained 2-hidden layer MLP with 1024 hidden units using SGD with a batch size of 128. We used the damping term of $\lambda = 0.001$ for PBRF and influence functions. For each PBRF and influence estimation, we measured the influence

Dataset	Cold-Start		Warm-Start		PBRF	
	P	S	P	S	P	S
Concrete	-0.14	0.32	0.24	0.37	**0.85**	**0.71**
Energy	-0.01	-0.24	0.44	0.57	**0.95**	**0.83**
MNIST	-0.16	-0.12	0.21	-0.14	**0.98**	**0.81**
FashionMNIST	-0.19	0.01	0.05	0.24	**0.91**	**0.88**

Table 7: Comparison of test loss differences computed by influence function (**without** the Gauss-Newton Hessian approximation), cold-start retraining, warm-start retraining, and PBRF. We show Pearson (P) and Spearman rank-order (S) correlation when compared to influence estimates.

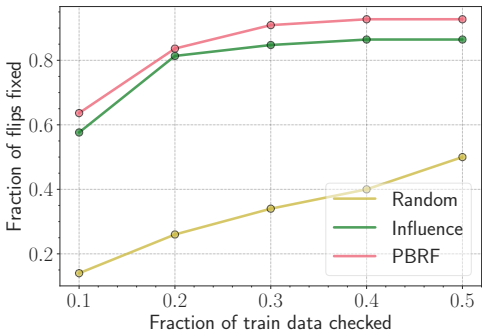


Figure 6: Effectiveness of PBRF and influence functions on fixing mislabeled training examples on corrupted MNIST dataset. We examine the fraction of the training data to fix the mislabelling while prioritizing the data examples with higher influential scores produced by PBRF and influence functions. The PBRF and influence functions help detect mislabelled examples.

of removing a single training training example on the total training loss (self-influence scores [30, 28]) to identify the top influential training examples.

We prioritized inspecting training examples that obtained high scores generated by the PBRF and influence functions. The results are summarized in Figure 6. Using the self-influence score generated by the PBRF, it is possible to detect over 80% of the mislabelled training examples by only examining 20% of training examples. Similarly, as influence functions closely align with the PBRF, influence functions can provide an efficient tool to help fix mislabelled training examples. Both PBRF and influence functions outperform the baseline of randomly selecting a subset of training examples to inspect if there are any mislabelled training examples.

E Influence Misalignment Decomposition Table

In Table 5, we present the numerical results shown in Figure 4.

G Efficient `iHVP` Computation

One of the major challenges in applying influence functions to neural networks (Eqn. 4) is that they involve the computation of an inverse-Hessian vector product (`iHVP`). However, for large networks, computing `iHVP`s exactly via storing and inverting the Hessian is intractable. To circumvent this, Koh and Liang [30] consider alternative methods for approximating `iHVP`s, namely, the method of conjugate gradients (`CG`) [43] and the Linear time Stochastic Second-Order Algorithm (`LiSSA`) [1].

Conjugate Gradient. Given a positive-definite damped Hessian $\nabla^2_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta}^s) + \lambda\mathbf{I}$ (where $\boldsymbol{\theta}^s$ are the potentially-suboptimal parameters at which the Hessian is taken and $\lambda > 0$ is a damping factor) and vector $\mathbf{v} \in \mathbb{R}^d$, `CG` arrives at the `iHVP` by solving an equivalent convex quadratic optimization

problem:

$$(\nabla^2_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta}^s) + \lambda\mathbf{I})^{-1}\mathbf{v} = \underset{\mathbf{t}\in\mathbb{R}^d}{\arg\min}\, \frac{1}{2}\mathbf{t}^\top(\nabla^2_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta}^s) + \lambda\mathbf{I})\mathbf{t} - \mathbf{v}^\top\mathbf{t}. \tag{46}$$

The CG algorithm starts with an initial guess $\mathbf{v}_0 \in \mathbb{R}^d$ and iteratively updates it, with the bottleneck at each step being an $O(Nd)$ Hessian-vector product. Although an exact solution is only guaranteed after d CG iterations, in practice, Koh and Liang [30] use truncated CG with fewer iterations and achieve a sufficiently close approximation.

LiSSA. The LiSSA algorithm approximates the iHVP using a truncated Neumann series. Given a positive-definite damped Hessian $\nabla^2_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta}^s) + \lambda\mathbf{I}$ and vector $\mathbf{v} \in \mathbb{R}^d$, we have:

$$(\nabla^2_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta}^s) + \lambda\mathbf{I})^{-1}\mathbf{v} \approx \sigma^{-1}\sum_{t=1}^{T}((1-\sigma^{-1}\lambda)\mathbf{I} - \sigma^{-1}\nabla^2_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta}^s))^t\mathbf{v}, \tag{47}$$

which becomes exact as the recursion depth T approaches ∞. Here, $\sigma > 0$ is a scaling hyperparameter that is chosen sufficiently large to ensure convergence of the series. Eqn. 47 can be recursively computed over T iterations, with each step requiring an $O(Nd)$ Hessian-vector product. In practice, the computation is further optimized by estimating $\nabla^2_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta}^s)$ using a randomly-sampled batch $\mathcal{B} \subseteq \mathcal{D}_{\text{train}}$ of size $|\mathcal{B}| \ll N$, so that the Hessian-vector product is reduced to $O(d)$ cost. Then, to accommodate for the added stochasticity, the iHVP is estimated by averaging Eqn. 47 over R trial repeats. Hence, the LiSSA algorithm estimates:

$$(\nabla^2_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta}^s) + \lambda\mathbf{I})^{-1}\mathbf{v} \approx \frac{1}{R}\sum_{r=1}^{R}\left(\sigma^{-1}\sum_{t=1}^{T}((1-\sigma^{-1}\lambda)\mathbf{I} - \sigma^{-1}\nabla^2_{\boldsymbol{\theta}}\mathcal{J}^{(r,t)}(\boldsymbol{\theta}^s))^t\mathbf{v}\right), \tag{48}$$

where $\mathcal{J}^{(r,t)}(\boldsymbol{\theta}^s)$ is the average loss over the (r,t)-th sampled batch of data.

The s_{test} trick. Finally, we note that another simple trick can be made when using influence functions to predict the change in test loss at a particular test point (Eqn. 19). It is often the case that we wish to compute the influence scores for the pairwise interactions of the entire training dataset on the loss at a comparatively smaller number $N_{\text{test}} \ll N$ of test points. Since the Hessian is symmetric, the order of multiplication in the second term of Eqn. 19 can be permuted as follows:

$$\frac{1}{N}\mathbf{v}_{\text{test}}^\top\left[(\nabla^2_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta}^s) + \lambda\mathbf{I})^{-1}\mathbf{v}\right] = \frac{1}{N}\mathbf{v}^\top\left[(\nabla^2_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta}^s) + \lambda\mathbf{I})^{-1}\mathbf{v}_{\text{test}}\right], \tag{49}$$

where $\mathbf{v} = \nabla_{\boldsymbol{\theta}}\mathcal{L}(f(\boldsymbol{\theta}^s, \mathbf{x}), \mathbf{t})$ and $\mathbf{v}_{\text{test}} = \nabla_{\boldsymbol{\theta}}\mathcal{L}(f(\boldsymbol{\theta}^s, \mathbf{x}_{\text{test}}), \mathbf{t}_{\text{test}})$. This means that we can precompute $\mathbf{s}_{\text{test}} = (\nabla^2_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta}^s) + \lambda\mathbf{I})^{-1}\mathbf{v}_{\text{test}}$ over all N_{test} test points of interest, and then cheaply compute influence scores over all N training points by simply taking dot products of the form $\mathbf{v}^\top\mathbf{s}_{\text{test}}$. We refer readers to Koh and Liang [30] for details.