Better Training Data Attribution via Better Inverse Hessian-Vector Products

Andrew Wang[*1,2] **Elisa Nguyen**[3] **Runshi Yang**[1,2]
Juhan Bae[1,2] **Sheila A. McIlraith**[1,2,4] **Roger Grosse**[1,2,4]
[1]University of Toronto [2]Vector Institute for Artificial Intelligence
[3]Tübingen AI Center, University of Tübingen
[4]Schwartz Reisman Institute for Technology and Society

Abstract

Training data attribution (TDA) provides insights into which training data is responsible for a learned model behavior. Gradient-based TDA methods such as influence functions and unrolled differentiation both involve a computation that resembles an inverse Hessian-vector product (iHVP), which is difficult to approximate efficiently. We introduce an algorithm (ASTRA) which uses the EKFAC-preconditioner on Neumann series iterations to arrive at an accurate iHVP approximation for TDA. ASTRA is easy to tune, requires fewer iterations than Neumann series iterations, and is more accurate than EKFAC-based approximations. Using ASTRA, we show that improving the accuracy of the iHVP approximation can significantly improve TDA performance.

1 Introduction

Machine learning systems derive their behavior from the data they are trained on. *Training data attribution* (TDA) is a family of techniques that help uncover how individual training examples influence model predictions. As such, TDA is a valuable tool with applications in data valuation and curation [1–4], interpreting model behavior [5–11], building more equitable and transparent machine learning systems [12, 13] and investigating questions of intellectual property and copyright by tracing outputs back to specific data sources [11, 14, 15], among other applications.

Influence functions (IF) [16, 17] and unrolled differentiation [4, 18–21] are two gradient-based TDA methods that involve, or can be approximated as computing inverse Hessian-vector products (iHVPs).[1] Inverting the Hessian is infeasible but for the smallest of neural networks, so the iHVP is typically computed without explicitly constructing the Hessian. There are a number of choices available: Koh and Liang [17], who first introduced influence functions to deep learning, use the iterative algorithm LiSSA [23], which is a method based on stochastic Neumann series iterations[2] (SNI) [24, 25] that can take *thousands* of iterations to converge to an unbiased solution [7, 17]. Alternatively, Grosse et al. [7] adopt a tractable parametric approximation to the Hessian [1, 11, 19] using Eigenvalue-corrected Kronecker Factorization (EKFAC) [26, 27]. EKFAC makes several simplifying assumptions that hold only approximately in practice, but they dramatically lower both computational and memory cost, making it feasible to scale to billion-parameter language models [7]. However, EKFAC influence functions (EKFAC-IF) computed on converged models only correlate modestly with ground truth

[*]Correspondence to andrewwang@cs.toronto.edu.

[1]Other gradient-based TDA methods such as TRAK [22] or LOGRA [1] also involve iHVPs but use additional techniques such as random/PCA gradient projection.

[2]LiSSA was introduced in the context of optimization and contains other components, but we refer to the component that computes the iHVP, as found in the Koh and Liang [17] implementation. Henceforth, we will use the terms LiSSA and SNI interchangeably, with their slight difference described in Appendix B.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

 https://doi.org/10.52202/085713-3408

over a variety of datasets and model architectures [19] and tend to struggle for architectures involving convolution, suggesting further room for improvement. We aim to improve EKFAC-based TDA methods in this paper by improving the iHVP approximation.

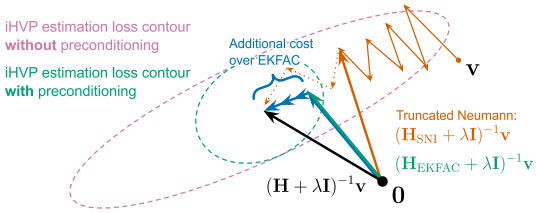

Figure 1: The objective is to compute the damped iHVP $(\mathbf{H} + \lambda\mathbf{I})^{-1}\mathbf{v}$. Preconditioning Stochastic Neumann Iterations (SNI) with EKFAC (ASTRA) improves the convergence speed of the iHVP approximation. Initialized at $\mathbf{0}$, it results in the same approximation as using EKFAC after one iteration. SNI may require thousands of iterations to converge, and truncating early results in undesirable implicit damping.

Computing iHVPs in the context of TDA can be seen as finding the minimizer to high-dimensional quadratic optimization problems in parameter space [25, 28, 29]. Curvature matrices for large neural networks are known to be ill-conditioned [30, 31], causing slow convergence for iterative methods such as SNI. While costly and rather difficult to tune [17, 32, 33], the upside of SNI is that it produces a *consistent* estimator – the algorithm converges to the iHVP in the limit as more compute is used [23]. In contrast, while EKFAC often provides a better cost vs. accuracy tradeoff for TDA [7], there is no simple way to improve its accuracy by applying more compute. Our central insight is that we can combine the best of both worlds: we can repurpose the EKFAC decomposition — which needs to be computed for EKFAC-based influence functions and unrolled differentiation anyways — as a preconditioner for SNI, yielding a cost effective procedure for improving the iHVP accuracy (Figure 1). Our contributions are as follows:

First, **we present an algorithm called ASTRA which uses EKFAC as a preconditioner for SNI with the aim of computing cost-effective and accurate iHVPs**. ASTRA can be applied to both influence functions (ASTRA-IF) and unrolled differentiation via an approximation called SOURCE (ASTRA-SOURCE) [19] among other applications. To the best of our knowledge, no prior work has applied the EKFAC preconditioner to SNI for the TDA setting. In our experiments, the incremental cost of ASTRA-IF and ASTRA-SOURCE was only *hundreds* of iterations, compared to EKFAC-IF and EKFAC-SOURCE, respectively. In contrast to other settings in which we would encounter iHVPs, we are often willing to pay this extra computational cost to obtain a more accurate iHVP for TDA.

Second, while past papers have questioned the reliability of influence functions [34, 35], **we show that influence functions computed with accurate iHVPs have strong predictive power**, even in settings in which the assumptions in its derivations may not hold [35, 36]. In our experiments, ASTRA-IF was able to achieve a Spearman correlation score [37] with ground-truth retraining of around 0.5 across many settings,[3] and ensembling these predictions often raised performance to 0.6. ASTRA provides an accurate approximation of the iHVP, which significantly increases the efficacy of ensembling [19, 22] compared to EKFAC and also significantly improves TDA performance for architectures involving convolution layers. For the experiment involving a convolution architecture, performance of ensembled ASTRA-IF was 0.6, a large increase from EKFAC-IF's 0.25.

Finally, **leveraging EKFAC's eigendecomposition, we show that low curvature directions are essential for high quality influence estimates**. Truncating Neumann series has an implicit damping effect [29, 38], which has a disproportional impact on low curvature directions. To quantify the downstream impact on influence functions performance, we perform an ablation study by using the EKFAC eigendecomposition to project the iHVP onto subspaces containing different levels of curvature during Neumann series iterations. We show that these low curvature components are essential for high quality influence estimates. This suggests that good influence function performance demands careful hyperparameter tuning, which can be costly, especially for off-the-shelf iterative solvers. In contrast, ASTRA requires much less hyperparameter tuning; we use a set of hyperparameters determined by simple heuristics which worked well for all of our experiments. We also note that while EKFAC was introduced as a preconditioner in second-order optimization [26], its compact representation of the eigendecomposition remains relatively underappreciated — the method in this ablation study in which we use the EKFAC eigendecomposition to analyze the quadratic objective could therefore be of independent interest.

[3] More precisely, we use a widely-used evaluation metric called Linear Datamodeling Score (LDS) [22], which measures the rank correlation between ground-truth retraining outcomes and a TDA algorithm's predictions. LDS is defined and performance comparisons are provided in Section 5.

2 Preliminaries

This section briefly introduces preliminaries relating to 1) computing the iHVP and 2) influence functions. To help the reader navigate the mathematical objects, we have provided a table of notations and acronyms in Appendix A. Further background and expanded discussion is provided in Appendix B.

Given a training dataset $\mathcal{D} = \{\boldsymbol{z}_i\}_{i=1}^N$ where $\boldsymbol{z}_i = (\mathbf{x}^{(i)}, \mathbf{t}^{(i)})$ is an input-target pair, and a model parameterized by $\boldsymbol{\theta} \in \mathbb{R}^D$, let $g(\boldsymbol{\theta}, \mathbf{x}^{(i)})$ denote the model output on $\mathbf{x}^{(i)}$, let $\mathcal{L}(\mathbf{y}, \mathbf{t})$ be a convex loss function. We define a training objective $\mathcal{J}(\boldsymbol{\theta}, \mathcal{D}) \coloneqq \frac{1}{N}\sum_{i=1}^N \mathcal{L}(g(\boldsymbol{\theta}, \mathbf{x}^{(i)}), \mathbf{t}^{(i)})$ as the average loss over $\mathcal{D}$. Given a query data point $\boldsymbol{z}_q$ and a measurement function $f_{\boldsymbol{z}_q}(\boldsymbol{\theta})$, such as correct-class margin [22], an *idealized* objective of TDA is to approximate the impact of removing a training example $\boldsymbol{z}_m$ from the training dataset $\mathcal{D}$ on $f_{\boldsymbol{z}_q}$. A pointwise TDA method τ assigns a score $\tau(\boldsymbol{z}_m, \boldsymbol{z}_q, \mathcal{D})$ that measures the impact of removing $\boldsymbol{z}_m$ from $\mathcal{D}$ on $f_{\boldsymbol{z}_q}$. Since modern neural networks often exhibit multiple optima, the stochasticity in the optimization process from sources such as parameter initialization [39], sampled dropout masks [40], and mini-batch ordering [41] can result in different learned optima, which we denote $\boldsymbol{\theta}^s$. Let ξ be a random variable which captures this stochasticity in the training procedure [42, 43].

2.1 Computing Inverse Hessian-Vector Products for TDA

Inverse Hessian-vector products (iHVPs) are ubiquitous in many machine learning settings beyond TDA [25, 26, 44, 45], such as second-order optimization [26, 46, 47], but different settings have different considerations when trading off cost vs. accuracy. The canonical second-order optimization method – Newton's method – utilizes an inverse Hessian-gradient product to compute the Newton step, and may achieve faster local convergence than first-order methods [28]. However, computing an iHVP is substantially more expensive than a standard gradient computation. In optimization, there exists a tradeoff between devoting extra compute to obtain a better iHVP approximation versus taking more steps in the optimization procedure [48]. Because most deep learning optimizers rely on stochastic updates, the extra cost of a highly precise curvature estimate often is not justified and popular optimizers such as Adam [47] default to the much cheaper diagonal preconditioner. In contrast, we are typically willing to pay a higher cost for accurate iHVP approximations in the TDA context, since – as we will show – an accurate iHVP may significantly improve TDA performance. Two popular ways of computing the iHVP for TDA are EKFAC [7, 26, 27] and LiSSA [17, 23]. We now briefly describe each method.

Computing the iHVP with EKFAC KFAC [26] and EKFAC [27] make a block-diagonal parametric approximation of the Fisher Information Matrix[4] (FIM) so that its eigendecomposition can be done for each layer l independently, which is significantly cheaper than an often infeasible brute-force eigendecomposition. The FIM is defined as:

$$\mathbf{F} \coloneqq \frac{1}{N}\sum_{\mathbf{x}^{(i)}\in\mathcal{D}} \mathbb{E}_{\hat{\mathbf{y}}\sim p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}^{(i)})}\left[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\hat{\mathbf{y}}|\mathbf{x}^{(i)}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\hat{\mathbf{y}}|\mathbf{x}^{(i)})^\top\right], \tag{1}$$

where $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$ is the model's own distribution over targets. For multi-layer perceptrons, KFAC approximates the lth block of the FIM with:

$$\mathbf{F}_l \approx \underbrace{\mathbb{E}\left[\bar{\mathbf{a}}_{l-1}\bar{\mathbf{a}}_{l-1}^\top\right]}_{\mathbf{A}_{l-1}} \otimes \underbrace{\mathbb{E}\left[\mathcal{D}\mathbf{s}_l \mathcal{D}\mathbf{s}_l^\top\right]}_{\mathbf{S}_l}, \tag{2}$$

$$= (\mathbf{Q}_{\mathbf{A}_{l-1}} \otimes \mathbf{Q}_{\mathbf{S}_l}) \underbrace{(\mathbf{D}_{\mathbf{A}_{l-1}} \otimes \mathbf{D}_{\mathbf{S}_l})}_{\boldsymbol{\Lambda}_{l,\text{KFAC}}} (\mathbf{Q}_{\mathbf{A}_{l-1}} \otimes \mathbf{Q}_{\mathbf{S}_l})^\top \tag{3}$$

where $\bar{\mathbf{a}}_{l-1}$ are the activations of the $l-1$th layer[5] and $\mathcal{D}\mathbf{s}_l$ are the pseudogradients[6] of the loss with respect to the preactivations $\mathbf{S}_l$ of the lth layer, and $\mathbf{A}_{l-1} = \mathbf{Q}_{\mathbf{A}_{l-1}}\mathbf{D}_{\mathbf{A}_{l-1}}\mathbf{Q}_{\mathbf{A}_{l-1}}^\top$ and $\mathbf{S}_l =$

[4] For standard regression and classification tasks whose outputs can be seen as the natural parameters of an exponential family, the Fisher Information Matrix $\mathbf{F}$ and the Generalized Gauss-Newton Hessian $\mathbf{G}$ coincide, which we will use as a substitute for the Hessian $\mathbf{H}$. For details, see Appendix B.

[5] The bar indicates the use of the homogeneous vector notation $\bar{\mathbf{a}}_{l-1} = \left[\mathbf{a}_{l-1}^\top\ 1\right]^\top$.

[6] By pseudogradients, we mean the gradient of the loss using a sampled target with respect to the parameters.

https://doi.org/10.52202/085713-3408

$\mathbf{Q}_{\mathbf{S}_l}\mathbf{D}_{\mathbf{S}_l}\mathbf{Q}_{\mathbf{S}_l}^\top$ are eigendecompositions of $\mathbf{A}_{l-1}$ and $\mathbf{S}_l$ respectively. We denote the block-diagonal matrix approximated this way as $\mathbf{F}_{\text{KFAC}}$. We can then approximate the inverse FIM as $(\mathbf{F}+\lambda\mathbf{I})^{-1} \approx (\mathbf{Q}_{\mathbf{A}_{l-1}} \otimes \mathbf{Q}_{\mathbf{S}_l})(\mathbf{\Lambda}_{l,\text{KFAC}} + \lambda\mathbf{I})^{-1}(\mathbf{Q}_{\mathbf{A}_{l-1}} \otimes \mathbf{Q}_{\mathbf{S}_l})^\top$ where λ is a damping hyperparameter. The EKFAC approximation $\mathbf{F}_{\text{EKFAC}}$ is an improvement over KFAC using the diagonal matrix $\mathbf{\Lambda}_{l,\text{EKFAC}}$ instead, whose ith entry along the diagonal is:

$$\mathbf{\Lambda}_{l,\text{EKFAC},i} = \mathbb{E}\Big[\big((\mathbf{Q}_{\mathbf{A}_{l-1}} \otimes \mathbf{Q}_{\mathbf{S}_l})^\top \mathcal{D}\boldsymbol{\theta}_l\big)_i^2\Big], \tag{4}$$

where $\mathcal{D}\boldsymbol{\theta}_l$ are the pseudogradients with respect to the parameters in layer l.

To illustrate the compute and memory savings, for a P-layer multi-layer perceptron with $\tilde{D}$ input dimensions and $\tilde{D}$ output dimensions for all layers, due to the block-diagonal approximation, the EKFAC eigendecomposition only costs $O(P\tilde{D}^3) = O(D^{\frac{3}{2}})$ time and storing its statistics requires $O(P\tilde{D}^2) = O(D)$ memory [49].[7] Although EKFAC significantly reduces the time and space complexity of the eigendecomposition, it results in a biased iHVP. See Appendix B for a more detailed discussion of EKFAC.

Computing the iHVP with LiSSA In contrast to EKFAC, iterative methods such as LiSSA are based on Neumann series iterations (NI) [24]. NI do not require EKFAC's assumptions, and thus in principle can produce exact iHVPs as more compute is applied. For an invertible matrix $\mathbf{A} \in \mathbb{R}^{D\times D}$ with $\|\mathbf{I}-\mathbf{A}\|_2 < 1$, the Neumann series is approximated as $\mathbf{A}^{-1} = \sum_{j=0}^{\infty}(\mathbf{I}-\mathbf{A})^j$, which is a generalization of the geometric series $a^{-1} = \sum_{j=0}^{\infty}(1-a)^j$ for $|1-a| < 1$. By substituting $\mathbf{A}$ with a scaled positive-definite damped Generalized Gauss-Newton Hessian (GGN) $\alpha(\mathbf{G}+\lambda\mathbf{I})$, and multiplying both sides by any $\mathbf{v} \in \mathbb{R}^D$, we obtain: $\frac{1}{\alpha}(\mathbf{G}+\lambda\mathbf{I})^{-1}\mathbf{v} = \sum_{j=0}^{\infty}(\mathbf{I}-\alpha\mathbf{G}-\alpha\lambda\mathbf{I})^j\mathbf{v}$. Here λ is a positive scalar known as the damping term and $\alpha > 0$ is the learning rate hyperparameter. The learning rate must satisfy $\alpha < \frac{1}{\sigma_{\max}(\mathbf{G})+\lambda}$, where $\sigma_{\max}(\mathbf{G})$ is the largest eigenvalue of $\mathbf{G}$ so that $\|\mathbf{I}-\mathbf{A}\|_2 < 1$. We can then approximate the iHVP $\frac{1}{\alpha}(\mathbf{G}+\lambda\mathbf{I})^{-1}\mathbf{v}$ via the iterative update:

$$\mathbf{v}_{k+1} \leftarrow \mathbf{v}_k - \alpha(\mathbf{G}+\lambda\mathbf{I})\mathbf{v}_k + \mathbf{v}, \tag{5}$$

which satisfies the property $\mathbf{v}_k \to \frac{1}{\alpha}(\mathbf{G}+\lambda\mathbf{I})^{-1}\mathbf{v}$ as $k \to \infty$. Computing $\mathbf{G}$ requires two backward passes over the whole dataset,[8] so an unbiased estimate $\tilde{\mathbf{G}}_k$ of $\mathbf{G}$ is usually used instead by sampling a mini-batch with replacement, which we refer to as *stochastic Neumann series iterations* (SNI). Compared to other iterative methods like conjugate gradient (CG) [51], SNI is typically preferred to compute the iHVP for TDA since CG tends to struggle with stochastic gradients [17, 26, 52]. LiSSA reduces the variance in SNI by taking an average over multiple trials.

2.2 Training Data Attribution with Influence Functions

Influence functions [16, 53] are derived under the assumption that $\mathcal{J}$ is strictly convex in $\boldsymbol{\theta}$ and twice differentiable. Let $\boldsymbol{\theta}^\star \coloneqq \arg\min_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}, \mathcal{D})$ be the optimal parameters over $\mathcal{D}$. We can define the objective after downweighting a training example $\boldsymbol{z}_m$ by ϵ as: $\mathcal{Q}(\boldsymbol{\theta}, \epsilon) \coloneqq \mathcal{J}(\boldsymbol{\theta}, \mathcal{D}) - \frac{\epsilon}{N}\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{z}_m)$ where ϵ is a scalar that specifies the amount of downweighting. When $\epsilon = 0$, this corresponds to the original objective $\mathcal{J}$. We can then define the optimal parameters *after* downweighting as a function of ϵ: $r(\epsilon) = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \mathcal{Q}(\boldsymbol{\theta}, \epsilon)$. When $\boldsymbol{z}_m \in \mathcal{D}$ and $\epsilon = 1$, this corresponds to the downweighted objective in which $\boldsymbol{z}_m$ is removed from $\mathcal{D}$. Typically, ϵ is assumed to be small, and we can approximate the leave-one-out (LOO) parameter change as $\boldsymbol{\theta}^\star(\mathcal{D}\setminus\{\boldsymbol{z}_m\}) - \boldsymbol{\theta}^\star(\mathcal{D}) \approx \frac{\mathrm{d}r}{\mathrm{d}\epsilon}\Big|_{\epsilon=0}$, where $\frac{\mathrm{d}r}{\mathrm{d}\epsilon}\Big|_{\epsilon=0} = \frac{1}{N}\mathbf{H}^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}^\star, \boldsymbol{z}_m)$ and $\mathbf{H} \coloneqq \nabla^2_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta}^\star, \mathcal{D})$ denotes the Hessian of the training objective at the optimal parameters. To approximate the effect on the measurement function $f_{\boldsymbol{z}_q}$, we invoke the chain rule: $f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^\star(\mathcal{D}\setminus\{\boldsymbol{z}_m\})) - f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^\star(\mathcal{D})) \approx \frac{1}{N}\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^\star)^\top\mathbf{H}^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}^\star, \boldsymbol{z}_m)$, which also gives the first-order Taylor approximation of $f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^\star(\mathcal{D}\setminus\{\boldsymbol{z}_m\}))$ after rearranging terms. When applying this approximation to neural networks in which the convexity assumption does not

[7] The time and space complexity of ordinary eigendecomposition is $O(D^3)$ and $O(D^2)$ respectively, and $P\tilde{D}^3$ typically is much smaller than D^3. This analysis treats P as constant.

[8] An efficient implementation with $O(D)$ time and space complexity requires a Jacobian-vector product and a vector-Jacobian product [50].

hold, $\mathbf{H}$ may not be invertible, so $\mathbf{H}$ is typically approximated with the damped GGN $\mathbf{G} + \lambda\mathbf{I}$ [54], which is always positive definite for $\lambda > 0$, and tends to work well in practice [7, 11, 19, 55].[9] With this substitution, the influence functions *attribution score* is:

$$\tau_{\text{IF}}(\boldsymbol{z}_m, \boldsymbol{z}_q, \mathcal{D}) \coloneqq \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^\star)^\top (\mathbf{G} + \lambda\mathbf{I})^{-1} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^\star, \boldsymbol{z}_m). \tag{6}$$

When applying influence functions to neural networks, the model may not have fully converged, so we typically compute the gradients and the GGN in Equation 6 with the final parameters $\boldsymbol{\theta}^s$ instead of $\boldsymbol{\theta}^\star$. We can also ensemble influence functions for better TDA performance, by training models with various seeds ξ and averaging over τ_{IF} for each seed to get an ensembled score [19, 22] (details in Appendix B). Ensembling for other TDA methods, such as unrolled differentiation, can be done analogously.

The iHVP in Equation 6 was originally computed with LiSSA by Koh and Liang [17] and has the drawback that its iterative procedure must be carried out once for each vector in the iHVP. Fortunately, it is frequently the case that $|\mathcal{D}_{\text{query}}| \ll |\mathcal{D}|$ [7, 36, 56], so by choosing $\mathbf{v} \coloneqq \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^s)$ and first computing $\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^s)^\top (\mathbf{G} + \lambda\mathbf{I})^{-1}$ in Equation 6,[10] we can reduce the number of iterative procedures to $|\mathcal{D}_{\text{query}}|$ for the influence function computation. Plugging these values into Equation 5, we can compute $\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^s)^\top (\mathbf{G} + \lambda\mathbf{I})^{-1}$ via the iterative update:[11]

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \alpha(\tilde{\mathbf{G}}_k + \lambda\mathbf{I})\boldsymbol{\theta}_k + \alpha \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^s). \tag{7}$$

While $|\mathcal{D}_{\text{query}}|$ sounds like a large number of iterative procedures, in practice no $\mathcal{D}_{\text{query}}$ exists, and instead queries are run when the user wants to understand specific behavior pertaining to $\boldsymbol{z}_q$. Nevertheless, to get a good approximation of $\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^s)^\top (\mathbf{G} + \lambda\mathbf{I})^{-1}$ for any particular $\boldsymbol{z}_q$ may require *thousands* of iterations [7, 17], limiting its scalability. Furthermore, tuning the hyperparameter for SNI is difficult [17, 32, 33] due to the ill-conditioning and stochasticity of the gradients. Once the iHVP is computed, its dot product with $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^s, \boldsymbol{z}_m)$ for every $\boldsymbol{z}_m$ in consideration is taken. If the goal is to simply compute the influence of $\boldsymbol{z}_m$ on $\boldsymbol{z}_q$ for *given* $\boldsymbol{z}_m$ and $\boldsymbol{z}_q$, then the cost of the dot product is minimal. However, if the goal is to search for the most influential points in $\boldsymbol{z}_m \in \mathcal{D}$ on $\boldsymbol{z}_q$, then we must take the dot product with *every* training example gradient in $\mathcal{D}$, which amounts to a backward pass over the entire training dataset and can be a substantial component of the cost of computing influence functions.[12] Arguably, the latter goal is more prevalent in settings such as data-centric model debugging and interpretability, where insight into model behavior is given by retrieving highly influential training samples [57–61].

3 Introducing EKFAC-Accelerated Neumann Series Iterations for TDA

We now have laid sufficient groundwork to introduce a novel algorithm called **ASTRA**, which preconditions SNI updates with EKFAC. In this section, we present the ASTRA update rule, discuss its computational costs, and discuss extensions to unrolled-differentiation-based TDA.

Preconditioning Stochastic Neumann Series Iterations with EKFAC The SNI update rule in Equation 7 can be viewed as performing mini-batch gradient descent on the quadratic objective [23, 62]:

$$h_{f_{\boldsymbol{z}_q}}(\boldsymbol{\theta}) \coloneqq \frac{1}{2}\boldsymbol{\theta}^\top (\mathbf{G} + \lambda\mathbf{I})\boldsymbol{\theta} - \boldsymbol{\theta}^\top \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^s). \tag{8}$$

It is well-known that for a converged neural network, the curvature matrix in the objective in Equation 8 is typically ill-conditioned [30, 31], which presents challenges for iterative methods. To improve the conditioning of $h_{f_{\boldsymbol{z}_q}}(\boldsymbol{\theta})$, we introduce an algorithm which computes accurate iHVPs for use in TDA called EKFAC-**A**ccelerated Neumann **S**eries Iterations for **TR**aining Data **A**ttribution (ASTRA) by applying preconditioning. The resulting update rule is:

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \alpha(\mathbf{P} + \tilde{\lambda}\mathbf{I})^{-1}(\tilde{\mathbf{G}}_k + \lambda\mathbf{I})\boldsymbol{\theta}_k + \alpha(\mathbf{P} + \tilde{\lambda}\mathbf{I})^{-1} \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^s). \tag{9}$$

[9] We will use the approximation $\mathbf{G} \approx \mathbf{H}$ throughout, and our use of the term iHVP will generally refer to both the inverse Hessian-vector product and the inverse Gauss-Newton-Hessian-vector product.

[10] This uses the fact that $\mathbf{G}$ is symmetric. $(\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^s)^\top (\mathbf{G} + \lambda\mathbf{I})^{-1})^\top = (\mathbf{G} + \lambda\mathbf{I})^{-1} \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^s)$.

[11] We emphasize that $\tilde{\mathbf{G}}_k$ and $\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^s)$ are computed using the final parameters $\boldsymbol{\theta}^s$. We use $\boldsymbol{\theta}_k$ to denote the iterates for SNI since it has the same dimensions as the model's parameters.

[12] Procedures such as TF-IDF filtering exist to prune the potentially vast training dataset [7].

https://doi.org/10.52202/085713-3408

While a number of choices of preconditioners exist, we choose the FIM computed with EKFAC (i.e., $\mathbf{P} \coloneqq \mathbf{F}_{\text{EKFAC}}$) for the following reasons: First, the computation cost of the EKFAC eigendecomposition is usually much cheaper than full matrix inversion, and its eigendecomposition statistics can be stored compactly and shared across all $|\mathcal{D}_{\text{query}}|$ optimization problems, since the value of $\mathbf{G}$ is the same across all objectives. Second, this choice has a close connection with EKFAC influence functions: Equation 5 suggests initializing $\boldsymbol{\theta}_0$ as $\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^s)$, which is frequently done in public implementations [17]. Observe that if we initialize $\boldsymbol{\theta}_0 \leftarrow \mathbf{0}$, choose $\tilde{\lambda} = \lambda$ and a learning rate $\alpha = 1$, we arrive at the iHVP which would be approximated by EKFAC after one step of ASTRA, resulting in the same TDA prediction as EKFAC-IF.[13] We hypothesize that further training using the update rule in Equation 9 will improve the iHVP approximation, a claim we validate empirically in Section 5. This update rule assumes using mini-batch gradient descent, but our formulation is compatible with other optimization algorithms as well.

Time Complexity of ASTRA Computing the update in Equation 9 involves explicitly constructing neither $(\mathbf{P} + \tilde{\lambda}\mathbf{I})^{-1}$ nor $(\mathbf{G} + \lambda\mathbf{I})$. Instead, we use Hessian-vector products [50] and first compute $(\tilde{\mathbf{G}}_k + \lambda\mathbf{I})\boldsymbol{\theta}_k$. We can then compute $(\mathbf{P} + \tilde{\lambda}\mathbf{I})^{-1}(\tilde{\mathbf{G}}_k + \lambda\mathbf{I})\boldsymbol{\theta}_k$ using the EKFAC preconditioner $\mathbf{P} \coloneqq \mathbf{F}_{\text{EKFAC}}$, so the *incremental* time complexity of *each iteration* in our algorithm is $O(\mathcal{B}D)$ where $\mathcal{B}$ is the mini-batch size used to sample $\tilde{\mathbf{G}}_k$. For both EKFAC-IF and ASTRA-IF, computing $\mathbf{F}_{\text{EKFAC}}$ is necessary to compute the iHVP, which only needs to be done once per model and can be shared among all queries. When we need to search the entire dataset for highly influential training examples, both methods also need to compute the dot product of the resulting iHVP with the training example gradients over all $\boldsymbol{z}_m$ in consideration, as discussed in Section 2.2. In our experiments, ASTRA-IF required only a few hundred incremental iterations, so its iterative component is a relatively small additional cost per query compared to the total cost to compute EKFAC-IF.

Application to Unrolled Differentiation In addition to influence functions, we also study the use of ASTRA in the context of an unrolling-based TDA method. Influence functions make assumptions such as model convergence and unique optimal parameters in its derivation, which may not be satisfied in practice [17, 36]. In contrast, unrolled differentiation methods such as SOURCE sidestep this limitation by differentiating through the training trajectory. Here, we only sketch our approach to apply ASTRA to SOURCE, deferring the full discussion of the SOURCE derivation to Appendix B and the details of ASTRA-SOURCE to Appendix C. SOURCE approximates differentiating through the training trajectory by partitioning it into L segments, assuming stationary and independent GGNs and gradients within each. Its approximation of the first order effect of downweighting a training example contains L different finite series involving the GGN, each of which can be approximated with an iHVP. Similarly to ASTRA-IF, ASTRA-SOURCE improves this approximation by repurposing the EKFAC decompositions as preconditioners, which would have been needed to be computed to implement SOURCE anyways.

4 Relationship to Existing Works

TDA Methods using iHVPs Both influence functions and unrolled differentiation can be viewed as belonging to a family of gradient-based TDA methods (for a survey see [58]), which both have a connection with iHVPs. Many gradient-based TDA methods are variants of the influence functions method proposed by Koh and Liang [17] with aims to improve its computational cost [7, 22, 33], by using techniques such as EKFAC for iHVP approximation [7], Arnoldi iterations [33], gradient projection [1, 22], and rank-one updates [63], instead of iterative algorithms such as LiSSA [23] or CG [51, 64], since the former is expensive and hard to tune [17, 32, 33], and the latter struggles with stochastic gradients [65]. Unrolled differentiation addresses the key derivation assumptions underlying influence functions – namely, the convexity of the training objective and the convergence of the final model parameters [36]. Methods include SGD-influence [4], HYDRA [18], SOURCE [19], DVEmb [20] and MAGIC [21], which all differentiate through the training trajectory, and only differ in their approximations. Rather than comparing the TDA algorithms themselves, our goal in this paper is to show that substantial TDA performance improvements can be achieved by using better iHVP approximations. In some public comparisons, LiSSA-based influence functions perform poorly

[13] Going forward, we therefore directly initialize ASTRA with $\boldsymbol{\theta}_0 \leftarrow (\mathbf{P} + \tilde{\lambda}\mathbf{I})^{-1}\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^s)$.

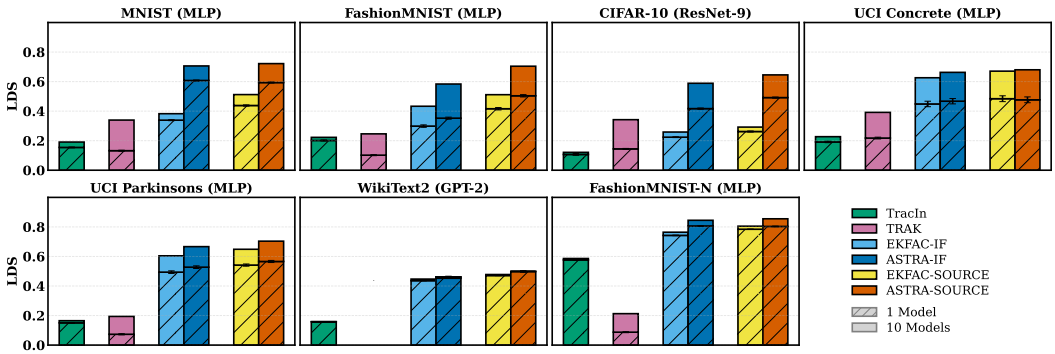

Figure 2: **TDA Performance.** Single model ASTRA-IF and ASTRA-SOURCE beat EKFAC-based counterparts in most settings, as well as other TDA methods such as TracIn [83] and TRAK [22] when measured by average LDS over the query set $\mathcal{D}_{\text{query}}$. ASTRA also enjoys a larger performance boost from ensembling. Improvement is particularly large for convolution architectures such as ResNet-9. Error bars (where available) indicate 1 standard error. We omit TRAK for GPT-2 due to lack of public implementations.

[22, 63, 66], sometimes even worse than dot products[14] [63, 67] and many have opted to instead use EKFAC-IF as their method or baseline of choice [1, 11, 19]. In principle, EKFAC-IF is only an *approximation* of what LiSSA-based influence functions attempts to compute, since the latter is only constrained by solver error while the former makes assumptions on the structure of the curvature matrix. We show in this paper that indeed, accurately solving the iHVP typically produces better TDA performance than the EKFAC solution, which is what our algorithm ASTRA addresses.

iHVPs beyond TDA iHVPs can also be found in higher-order optimization algorithms such as Newton's method [28], quasi-Newton methods [68–71], natural gradient descent [65, 72, 73], KFAC [26], and Hessian-free optimization [65, 74], which computes Hessian-vector products iteratively with CG [51, 64]. Influence functions can also be cast as a bilevel optimization problem [25, 29, 44, 75], which can be solved via implicit differentiation or unrolled differentiation. Since iHVPs show up frequently in machine learning, there is motivation to adapt and develop iHVP computation techniques such as SNI and EKFAC. While these two methods are well-established and preconditioning is a well-established technique in optimization [26, 72], to the best of our knowledge, no prior TDA method has combined these methods to compute the iHVP. For extended related works, see Appendix D.

5 Performance Comparisons

This section aims to answer the following questions: 1) Do ASTRA-IF and ASTRA-SOURCE outperform their EKFAC-based counterparts? 2) Is ASTRA substantially faster than vanilla SNI? To answer these questions, we run experiments in a number of settings. For regression tasks, we use the UCI datasets Concrete and Parkinsons [76] trained with a multi-layer perceptron (MLP). For classification tasks, we use CIFAR-10 [77] trained with ResNet-9 [78], MNIST [79] and FashionMNIST[80] trained with MLPs, and GPT-2 [81] fine-tuned with WikiText-2 [82]. We also include a non-converged setting, FashionMNIST-N, introduced by Bae et al. [19], for which SOURCE was specifically designed; in this setting, 30% of the training examples were randomly labeled, and the model was trained for only three epochs to avoid overfitting [19]. In addition to comparing against EKFAC-IF and EKFAC-SOURCE, we also compare against two popular TDA methods TracIn [83] and TRAK [22]. Details for all experiments can be found in Appendix F.

Evaluating Training Data Attribution Performance We evaluate the performance of our TDA algorithms on a popular evaluation metric called Linear Datamodeling Score (LDS) [22], and use mean absolute error as the measurement function for regression tasks and correct-class margin for classification tasks in line with past works [19, 22]. LDS measures a TDA algorithm's ability to predict the outcome of counterfactual retraining on a subset of data. Given a collection of uniformly

[14]This TDA method, sometimes called Hessian-free [63, 67], simply takes the dot product between the training gradient and the query gradient and is equivalent to influence functions if the damped GGN is set to the identity.

https://doi.org/10.52202/085713-3408

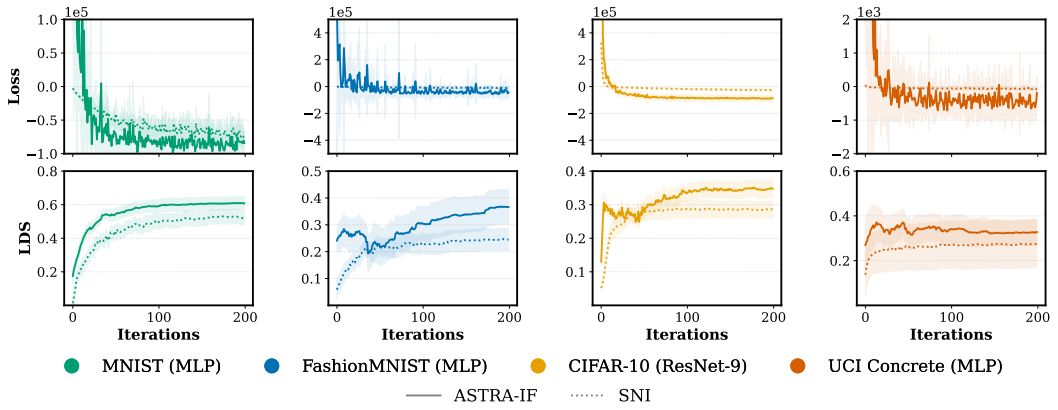

Figure 3: **Training Curves**. Loss and LDS curves for SNI and ASTRA measured over 10 seeds (shaded region = 1 standard error) on an arbitrary query point $\boldsymbol{z}_q$, using influence functions as the TDA method. SNI makes slower progress compared to ASTRA as measured by LDS.

randomly sampled subsets $\{\mathcal{S}_1, \ldots, \mathcal{S}_M : \mathcal{S}_i \subset \mathcal{D}\}$ of a fixed size, typically a fraction β of $\mathcal{D}$ and a measurement function $f_{\boldsymbol{z}_q}$, the LDS scores a TDA method τ as follows:

$$\text{LDS}(\tau, \boldsymbol{z}_q) := \boldsymbol{\rho}\Big(\underbrace{\big[\mathbb{E}_\xi[f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^s(\mathcal{S}_j;\xi))] : j \in [M]\big]}_{\substack{\textit{expected} \text{ ground-truth predictions on } \boldsymbol{z}_q \\ \text{using models trained on various } \mathcal{S}_j}}, \underbrace{[\Gamma_\tau(\mathcal{S}_j, \boldsymbol{z}_q; \mathcal{D}) : j \in [M]]}_{\substack{\text{TDA method's predictions on } \boldsymbol{z}_q \\ \text{for various } \mathcal{S}_j}}\Big), \tag{10}$$

where $\boldsymbol{\rho}$ denotes the Spearman correlation [37], and the group influence $\Gamma_\tau(\mathcal{S}_j, \boldsymbol{z}_q, \mathcal{D})$ is defined linearly as: $\Gamma_\tau(\mathcal{S}_j, \boldsymbol{z}_q; \mathcal{D}) = \sum_{z_i \in \mathcal{S}_j} \tau(\boldsymbol{z}_i, \boldsymbol{z}_q, \mathcal{D})$. To compute the ground-truth to which we compare our TDA method, we need to retrain a model many times both over various subsets S_i, and also over random seeds ξ for every subset to obtain a good estimate of the expectation in Equation 10, which can be quite noisy [19, 43]. For example, for the experiment involving GPT-2, computing ground-truth involved fine-tuning 1000 models. We report the *average* LDS $\frac{1}{|\mathcal{D}_{\text{query}}|}\sum_{\boldsymbol{z}_q \in \mathcal{D}_{\text{query}}} \text{LDS}(\tau, \boldsymbol{z}_q)$ over a test set $\mathcal{D}_{\text{query}}$ containing 100 query points. We randomly sample $M = 100$ subsets with a subsampling fraction of $\beta = 0.5$ in line with previous works [1, 19]. We discuss other evaluation methods in Appendix E.

LDS evaluation of ASTRA Figure 2 compares LDS across various TDA methods. We compare ASTRA-IF and ASTRA-SOURCE with their respective EKFAC-versions. For each setting, EKFAC-IF and ASTRA-IF use the same damping value implied by SOURCE for comparability (details in Appendix F). In almost all settings, ASTRA improves TDA performance as measured by LDS, strongly suggesting that better iHVP approximations are responsible. We also observe an especially large improvement over EKFAC for CIFAR-10 trained on ResNet-9 [78]. When applied to convolution layers, EKFAC makes additional simplifying assumptions,[15] which can cause EKFAC-IF and EKFAC-SOURCE to underperform on architectures involving convolution layers – an issue that ASTRA effectively addresses. Figure 2 also reveals increased benefits from ensembling when applying ASTRA, which computes an unbiased estimator of the iHVP. In some cases, such as FashionMNIST, the advantages of using precise iHVPs become much more pronounced in conjunction with ensembling. We hypothesize that this is due to the various bottlenecks in TDA methods: in some cases, the primary performance bottleneck lies in computing the iHVP accurately; in others, it stems from other factors such as the method's underlying assumptions (e.g., unique optimal parameters), which can be mitigated by ensembling.

ASTRA speeds up iHVP approximation The top row of Figure 3 shows the loss curves for SNI and ASTRA as each iHVP solver progresses. The bottom row corresponds to the LDS that influence functions achieves based on the current progress of each iHVP solver. For SNI, we follow public implementations [17], which typically initialize SNI using the query gradient $\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^s)$, and results

[15] In addition to layer-wise independence and independence of activations and pseudogradients [26], it assumes spatially uncorrelated derivatives and spatial homogeneity [84].

in an initial influence functions prediction that approximates the damped-GGN with the identity in Equation 6. For both methods, we conduct a hyperparameter sweep for the learning rate over 10^0, $\ldots$, 10^{-5} in steps of one order of magnitude, and use the best hyperparameter based on the average training loss performance on the same query point over the last 10 iterations. We find that EKFAC preconditioning reduces the notorious challenge of tuning learning rates [17, 32, 33] – in all of the settings reported in Figure 3 the learning rate for ASTRA used was 10^{-2}. In comparison, the reported (and best) SNI learning rates were $1, 0.1, 0.01, 0.1$ for MNIST, FashionMNIST, CIFAR-10, and UCI Concrete respectively and LDS performance was very sensitive to the learning rate. Figure 3 shows that SNI makes slow progress while ASTRA usually converges in fewer than 200 iterations.

6 Investigating the Role of Low Curvature Directions in Influence Functions Performance

Our results in the previous section lead us to hypothesize that preconditioning accelerates convergence in directions of low curvature, which is important for influence function performance. We can analyze how directions of low curvature are affected by NI (without preconditioning) when truncated early, something that is tempting to do as it usually takes long to converge. We derive the following expression for the truncated Neumann series with J iterations:

$$\alpha \sum_{j=0}^{J-1} (\mathbf{I} - \alpha\mathbf{G} - \alpha\lambda\mathbf{I})^j = (\mathbf{G} + \lambda\mathbf{I})^{-1}(\mathbf{I} - (\mathbf{I} - \alpha\mathbf{G} - \alpha\lambda\mathbf{I})^J) \tag{11}$$

$$\approx (\mathbf{G} + (\lambda + \alpha^{-1}J^{-1})\mathbf{I})^{-1}, \tag{12}$$

where the equality utilizes the definition for finite series, and the approximation is identical to that used in [19] (see Appendix G for the derivation). From the last equation, we can see that truncating Neumann series effectively adds an *implicit* damping term of $1/\alpha J$, which disproportionately affects directions of low curvature.

This insight prompts an investigation into the role of low curvature directions in influence functions performance since, in addition to the implicit damping effect mentioned above, some methods may discard them when projecting gradients into lower-dimensional subspaces [1, 33]. Let $\mathbf{F}_{\text{EKFAC}} = \hat{\mathbf{Q}}\hat{\mathbf{D}}\hat{\mathbf{Q}}^\top$ be the EKFAC eigendecomposition of the GGN at the final parameters. We study the importance of directions of varying curvature by doing the following: We bin the D eigenvalues given by $\mathbf{F}_{\text{EKFAC}}$ into 5 bins labeled $S_1, S_2, \ldots, S_5$, where each bin S_i holds all eigenvalues larger than 10^{-i}. Let $\hat{\mathbf{Q}}_{S_i} \in \mathbf{R}^{D \times |S_i|}$ be the projection matrix whose columns are the associated orthonormal eigenvectors of the eigenvalues in each bin. We investigate the values of $h^{S_i}_{f_{\boldsymbol{z}_q}}(\boldsymbol{\theta}_k)$ and the corresponding LDS for each S_i where $h^{S_i}_{f_{\boldsymbol{z}_q}}$ is defined as:

$$h^{S_i}_{f_{\boldsymbol{z}_q}}(\boldsymbol{\theta}) \coloneqq \frac{1}{2}(\hat{\mathbf{Q}}_{S_i}^\top\boldsymbol{\theta})^\top\hat{\mathbf{Q}}_{S_i}^\top(\mathbf{G} + \lambda\mathbf{I})\hat{\mathbf{Q}}_{S_i}(\hat{\mathbf{Q}}_{S_i}^\top\boldsymbol{\theta}) - (\hat{\mathbf{Q}}_{S_i}^\top\boldsymbol{\theta})^\top\hat{\mathbf{Q}}_{S_i}^\top\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^s). \tag{13}$$

We use the EKFAC eigendecomposition since the true eigendecomposition is intractable for the settings we report. We conduct the experiment in the MNIST and FashionMNIST settings and use a small damping hyperparameter of $\lambda = 10^{-4}$ to be able to observe the impact of directions of low curvature on influence functions performance (details in Appendix F).

The top row of Figure 4 shows the outcome when we run ASTRA and SNI on the objective in Equation 8 to obtain a sequence of $\boldsymbol{\theta}_k$, and plot the value of each $h^{S_i}_{f_{\boldsymbol{z}_q}}(\boldsymbol{\theta}_k)$ for each S_i. The bottom row shows the LDS, which are computed by projecting the iterates $\boldsymbol{\theta}_k$ into each curvature subspace defined by S_i and computing influence functions using these projected vectors, which we can write as:

$$\tau_{\text{PROJ-IF},k}(\boldsymbol{z}_m, \boldsymbol{z}_q, \mathcal{D}) \coloneqq (\hat{\mathbf{Q}}_{S_i}^\top\boldsymbol{\theta}_k)^\top\hat{\mathbf{Q}}_{S_i}^\top\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}^s, \boldsymbol{z}_m). \tag{14}$$

Figure 4 reveals that low curvature directions play a large role in the performance of influence functions: projecting to high-curvature eigenspaces degrades LDS performance, as evidenced by the large vertical gaps between lines representing different level of curvature. When the iHVP is computed via SNI, large eigenvalue directions converge quickly during training, but it takes longer for small eigenvalue directions to converge, as evidenced by the earlier plateau of the loss curves in

 https://doi.org/10.52202/085713-3408

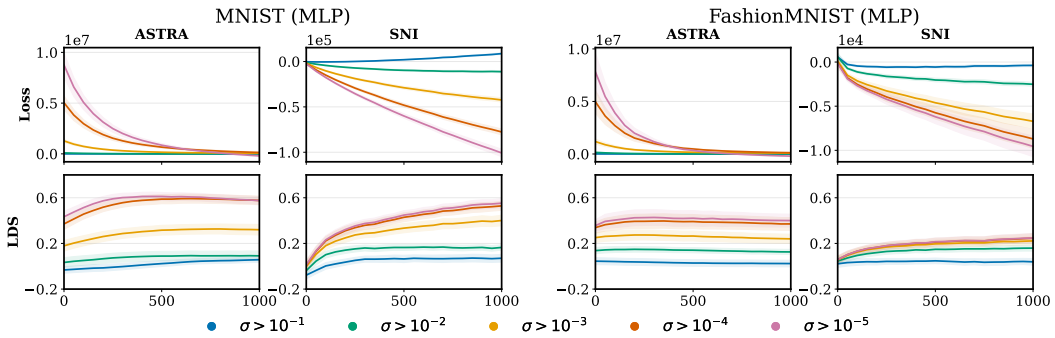

Figure 4: **Training Curves in Various Eigenspaces**. **Top:** The values $h^{S_i}_{f_{\boldsymbol{z}_q}}$ as ASTRA and SNI train on the objective in Equation 8 for an arbitrary $\boldsymbol{z}_q$. The subspaces represented by $S_1, \ldots, S_5$ are spanned by eigenvectors with eigenvalues $\sigma > 10^{-1}, \ldots, \sigma > 10^{-5}$ respectively. The loss of subspaces with large eigenvalue directions tend to plateau first, followed by subspaces with smaller eigenvalue directions, especially for SNI, which does not use preconditioning. **Bottom:** LDS of influence functions after projecting to corresponding subspace. The objective for *high* curvature directions plateaus first; continued training further decreases the objective in progressively lower curvature directions, yielding LDS gains even after high curvature directions have converged. Shaded region = 1 standard error.

the high-curvature subspaces in the top row of Figure 4. Nevertheless, as the solution progresses in low-curvature directions, LDS rises substantially, evidenced by the growing gap between lines representing large and small levels of curvature in the bottom row of Figure 4. While the slower convergence in low-curvature directions is present for ASTRA, it is substantially diminished due to the preconditioning. Our results highlight that the behavior of estimators in low-curvature subspaces may be a substantial factor in the performance of TDA methods.[16]

7 Conclusion

We presented an algorithm ASTRA that combines the EKFAC preconditioner with SNI for TDA. We compared ASTRA-IF and ASTRA-SOURCE with their EKFAC-based counterparts in a variety of settings. In many settings, TDA performance measured by LDS improved substantially, especially for convolution architectures. We find that in general, a more accurate iHVP approximation increases the efficacy of ensembling. ASTRA is easier to tune and converges faster than SNI. Compared with EKFAC, it only incrementally costs hundreds of iterations in our experiments as it leverages the same eigendecomposition. We conclude this paper by providing insights into how various curvature directions affect influence functions performance. We show that low curvature directions are important for good influence functions performance by using the EKFAC decomposition to analyze the quadratic objective, a technique that may be of independent interest outside of TDA. Overall, the technical contributions of this work should lead to improved performance in real-world problems such as data curation and interpreting model behavior, among other applications. We discuss limitations and broader implications of our work in Appendix H.

Acknowledgements

We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Canada CIFAR AI Chairs Program. Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute for Artificial Intelligence (https://vectorinstitute.ai/partnerships/). We thank the Schwartz Reisman Institute for Technology and Society for providing a rich multi-disciplinary research environment. RG and SM acknowledge support from the Canada CIFAR AI Chairs program and from the Natural Sciences

[16]We note that the difference in performance between ASTRA and SNI depends on the damping hyperparameter, and since the damping in this experiment is smaller, the performance boost from ASTRA compared to SNI is notably larger than in Figure 3.

and Engineering Research Council of Canada (NSERC). RG acknowledges support from Open Philanthropy and the Schmidt Sciences AI2050 Fellows Program.

References

[1] Sang Keun Choe, Hwijeen Ahn, Juhan Bae, Kewen Zhao, Minsoo Kang, Youngseog Chung, Adithya Pratapa, Willie Neiswanger, Emma Strubell, Teruko Mitamura, et al. What is your data worth to gpt? llm-scale data valuation with influence functions. *arXiv preprint arXiv:2405.13954*, 2024.

[2] Saachi Jain, Kimia Hamidieh, Kristian Georgiev, Andrew Ilyas, Marzyeh Ghassemi, and Aleksander Madry. Improving subgroup robustness via data selection. *Advances in Neural Information Processing Systems*, 37:94490–94511, 2024.

[3] Stefano Teso, Andrea Bontempelli, Fausto Giunchiglia, and Andrea Passerini. Interactive label cleaning with example-based explanations. *Advances in Neural Information Processing Systems*, 34:12966–12977, 2021.

[4] Satoshi Hara, Atsushi Nitanda, and Takanori Maehara. Data cleansing for models trained with sgd. *Advances in Neural Information Processing Systems*, 32, 2019.

[5] Pang Wei W Koh, Kai-Siang Ang, Hubert Teo, and Percy S Liang. On the accuracy of influence functions for measuring group effects. *Advances in neural information processing systems*, 32, 2019.

[6] Chih-Kuan Yeh, Joon Kim, Ian En-Hsu Yen, and Pradeep K Ravikumar. Representer point selection for explaining deep neural networks. *Advances in neural information processing systems*, 31, 2018.

[7] Roger Grosse, Juhan Bae, Cem Anil, Nelson Elhage, Alex Tamkin, Amirhossein Tajdini, Benoit Steiner, Dustin Li, Esin Durmus, Ethan Perez, Evan Hubinger, Kamilė Lukošiūtė, Karina Nguyen, Nicholas Joseph, Sam McCandlish, Jared Kaplan, and Samuel R. Bowman. Studying large language model generalization with influence functions, 2023. URL https://arxiv.org/abs/2308.03296.

[8] Ekin Akyürek, Tolga Bolukbasi, Frederick Liu, Binbin Xiong, Ian Tenney, Jacob Andreas, and Kelvin Guu. Tracing knowledge in language models back to the training data. *arXiv preprint arXiv:2205.11482*, 2022.

[9] Fulton Wang, Julius Adebayo, Sarah Tan, Diego Garcia-Olano, and Narine Kokhlikyan. Error discovery by clustering influence embeddings. *Advances in Neural Information Processing Systems*, 36:41765–41777, 2023.

[10] Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. Datamodels: Predicting predictions from training data. *arXiv preprint arXiv:2202.00622*, 2022.

[11] Bruno Mlodozeniec, Runa Eschenhagen, Juhan Bae, Alexander Immer, David Krueger, and Richard Turner. Influence functions for scalable data attribution in diffusion models. *arXiv preprint arXiv:2410.13850*, 2024.

[12] Marc-Etienne Brunet, Colleen Alkalay-Houlihan, Ashton Anderson, and Richard Zemel. Understanding the origins of bias in word embeddings. In *International conference on machine learning*, pages 803–811. PMLR, 2019.

[13] Haonan Wang, Ziwei Wu, and Jingrui He. Fairif: Boosting fairness in deep learning via influence functions with validation set sensitive attributes. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pages 721–730, 2024.

[14] Gerrit van den Burg and Chris Williams. On memorization in probabilistic deep generative models. *Advances in Neural Information Processing Systems*, 34:27916–27928, 2021.

 https://doi.org/10.52202/085713-3408

[15] Emanuele Mezzi, Asimina Mertzani, Michael P Manis, Siyanna Lilova, Nicholas Vadivoulis, Stamatis Gatirdakis, Styliani Roussou, and Rodayna Hmede. Who owns the output? bridging law and technology in llms attribution. *arXiv preprint arXiv:2504.01032*, 2025.

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

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

[18] Yuanyuan Chen, Boyang Li, Han Yu, Pengcheng Wu, and Chunyan Miao. Hydra: Hypergradient data relevance analysis for interpreting deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 7081–7089, 2021.

[19] Juhan Bae, Wu Lin, Jonathan Lorraine, and Roger Grosse. Training data attribution via approximate unrolled differentation. *arXiv preprint arXiv:2405.12186*, 2024.

[20] Jiachen T Wang, Dawn Song, James Zou, Prateek Mittal, and Ruoxi Jia. Capturing the temporal dependence of training data influence. *arXiv preprint arXiv:2412.09538*, 2024.

[21] Andrew Ilyas and Logan Engstrom. Magic: Near-optimal data attribution for deep learning. *arXiv preprint arXiv:2504.16430*, 2025.

[22] Sung Min Park, Kristian Georgiev, Andrew Ilyas, Guillaume Leclerc, and Aleksander Madry. Trak: Attributing model behavior at scale. *arXiv preprint arXiv:2303.14186*, 2023.

[23] Naman Agarwal, Brian Bullins, and Elad Hazan. Second-order stochastic optimization for machine learning in linear time. *Journal of Machine Learning Research*, 18(116):1–40, 2017.

[24] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

[25] Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *International conference on artificial intelligence and statistics*, pages 1540–1552. PMLR, 2020.

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

[27] Thomas George, César Laurent, Xavier Bouthillier, Nicolas Ballas, and Pascal Vincent. Fast approximate natural gradient descent in a kronecker factored eigenbasis. *Advances in Neural Information Processing Systems*, 31, 2018.

[28] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer Series in Operations Research and Financial Engineering. Springer Nature, 2006.

[29] Paul Vicol, Jonathan P Lorraine, Fabian Pedregosa, David Duvenaud, and Roger B Grosse. On implicit bias in overparameterized bilevel optimization. In *International Conference on Machine Learning*, pages 22234–22259. PMLR, 2022.

[30] Levent Sagun, Leon Bottou, and Yann LeCun. Eigenvalues of the hessian in deep learning: Singularity and beyond. *arXiv preprint arXiv:1611.07476*, 2016.

[31] Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An investigation into neural net optimization via hessian eigenvalue density. In *International Conference on Machine Learning*, pages 2232–2241. PMLR, 2019.

[32] Yegor Klochkov and Yang Liu. Revisiting inverse hessian vector products for calculating influence functions. *arXiv preprint arXiv:2409.17357*, 2024.

[33] Andrea Schioppa, Polina Zablotskaia, David Vilar, and Artem Sokolov. Scaling up influence functions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 8179–8186, 2022.

[34] Samyadeep Basu, Philip Pope, and Soheil Feizi. Influence functions in deep learning are fragile. *arXiv preprint arXiv:2006.14651*, 2020.

[35] Andrea Schioppa, Katja Filippova, Ivan Titov, and Polina Zablotskaia. Theoretical and practical perspectives on what influence functions do. *Advances in Neural Information Processing Systems*, 36, 2024.

[36] Juhan Bae, Nathan Ng, Alston Lo, Marzyeh Ghassemi, and Roger B Grosse. If influence functions are the answer, then what is the question? *Advances in Neural Information Processing Systems*, 35:17953–17967, 2022.

[37] C Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, 1904.

[38] L Lo Gerfo, Lorenzo Rosasco, Francesca Odone, E De Vito, and Alessandro Verri. Spectral algorithms for supervised learning. *Neural Computation*, 20(7):1873–1897, 2008.

[39] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[40] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[41] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 661–670, 2014.

[42] Jacob R Epifano, Ravi P Ramachandran, Aaron J Masino, and Ghulam Rasool. Revisiting the fragility of influence functions. *Neural Networks*, 162:581–588, 2023.

[43] Elisa Nguyen, Minjoon Seo, and Seong Joon Oh. A bayesian approach to analysing training data attribution in deep learning. *Advances in Neural Information Processing Systems*, 36, 2024.

[44] Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International conference on machine learning*, pages 2113–2122. PMLR, 2015.

[45] Hippolyt Ritter, Aleksandar Botev, and David Barber. A scalable laplace approximation for neural networks. In *6th international conference on learning representations, ICLR 2018-conference track proceedings*, volume 6. International Conference on Representation Learning, 2018.

[46] Tijmen Tieleman. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26, 2012.

[47] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[48] Richard H Byrd, Samantha L Hansen, Jorge Nocedal, and Yoram Singer. A stochastic quasi-newton method for large-scale optimization. *SIAM Journal on Optimization*, 26(2):1008–1031, 2016.

[49] Roger Grosse. Neural network training dynamics, 2021.

[50] Barak A Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1): 147–160, 1994.

[51] Magnus R Hestenes, Eduard Stiefel, et al. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49(6):409–436, 1952.

 https://doi.org/10.52202/085713-3408

[52] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

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

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

[55] James Martens. New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, 21(146):1–76, 2020.

[56] Myeongseob Ko, Feiyang Kang, Weiyan Shi, Ming Jin, Zhou Yu, and Ruoxi Jia. The mirrored influence hypothesis: Efficient data influence estimation by harnessing forward passes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 26286–26295, 2024.

[57] Andrea Brennen. What do people really want when they say they want" explainable ai?" we asked 60 stakeholders. In *Extended abstracts of the 2020 CHI conference on human factors in computing systems*, pages 1–7, 2020.

[58] Zayd Hammoudeh and Daniel Lowd. Training data influence analysis and estimation: A survey. *Machine Learning*, 113(5):2351–2403, 2024.

[59] Pouya Pezeshkpour and Sarthak Jain. Combining feature and instance attribution to detect artifacts. In *Proceedings of the Association for Computational Linguistics (ACL)*, 2022.

[60] Elisa Nguyen, Johannes Bertram, Evgenii Kortukov, Jean Y Song, and Seong Joon Oh. Towards user-focused research in training data attribution for human-centered explainable ai. *arXiv preprint arXiv:2409.16978*, 2024.

[61] Shreya Shankar, Rolando Garcia, Joseph M Hellerstein, and Aditya G Parameswaran. Operationalizing machine learning: An interview study. *arXiv preprint arXiv:2209.09125*, 2022.

[62] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[63] Yongchan Kwon, Eric Wu, Kevin Wu, and James Zou. Datainf: Efficiently estimating data influence in lora-tuned llms and diffusion models. *arXiv preprint arXiv:2310.00902*, 2023.

[64] Reeves Fletcher and Colin M Reeves. Function minimization by conjugate gradients. *The computer journal*, 7(2):149–154, 1964.

[65] James Martens et al. Deep learning via hessian-free optimization. In *Icml*, volume 27, pages 735–742, 2010.

[66] Junwei Deng, Ting-Wei Li, Shiyuan Zhang, Shixuan Liu, Yijun Pan, Hao Huang, Xinhe Wang, Pingbang Hu, Xingjian Zhang, and Jiaqi Ma. dattri: A library for efficient data attribution. *Advances in Neural Information Processing Systems*, 37:136763–136781, 2024.

[67] Zhe Li, Wei Zhao, Yige Li, and Jun Sun. Do influence functions work on large language models? *arXiv preprint arXiv:2409.19998*, 2024.

[68] R. Fletcher and M. J. D. Powell. A rapidly convergent descent method for minimization. *The Computer Journal*, 6(2):163–168, 1963.

[69] Charles G Broyden. A class of methods for solving nonlinear simultaneous equations. *Mathematics of computation*, 19(92):577–593, 1965.

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

[71] Jorge Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980.

[72] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2): 251–276, 1998.

[73] Guodong Zhang, Shengyang Sun, David Duvenaud, and Roger Grosse. Noisy natural gradient as variational inference. In *International conference on machine learning*, pages 5852–5861. PMLR, 2018.

[74] James Martens and Ilya Sutskever. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 1033–1040, 2011.

[75] Juhan Bae. *Beyond Gradients: Using Curvature Information for Deep Learning*. PhD thesis, University of Toronto (Canada), 2025.

[76] Dheeru Dua and C Graff. Uci machine learning repository. university of california, school of information and computer science, irvine, ca (2019), 2019.

[77] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images, 2009.

[78] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[79] Yann LeCun, Corinna Cortes, and CJ Burges. MNIST handwritten digit database. *ATT Labs*, 2, 2010.

[80] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[81] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[82] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[83] Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. *Advances in Neural Information Processing Systems*, 33:19920–19930, 2020.

[84] Roger Grosse and James Martens. A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, pages 573–582. PMLR, 2016.

[85] Anders Søgaard et al. Revisiting methods for finding influential examples. *arXiv preprint arXiv:2111.04683*, 2021.

[86] Yuzheng Hu, Pingbang Hu, Han Zhao, and Jiaqi W Ma. Most influential subset selection: Challenges, promises, and beyond. *arXiv preprint arXiv:2409.18153*, 2024.

[87] Nathan Ng, Roger Grosse, and Marzyeh Ghassemi. Measuring stochastic data complexity with boltzmann influence functions. *arXiv preprint arXiv:2406.02745*, 2024.

[88] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

[89] Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *Advances in neural information processing systems*, 32, 2019.

[90] Alexander Wei, Wei Hu, and Jacob Steinhardt. More than a toy: Random matrix models predict how real-world neural representations generalize. In *International Conference on Machine Learning*, pages 23549–23588. PMLR, 2022.

 https://doi.org/10.52202/085713-3408

[91] Frederik Kunstner, Philipp Hennig, and Lukas Balles. Limitations of the empirical fisher approximation for natural gradient descent. *Advances in neural information processing systems*, 32, 2019.

[92] Nikunj Saunshi, Arushi Gupta, Mark Braverman, and Sanjeev Arora. Understanding influence functions and datamodels via harmonic analysis. In *The Eleventh International Conference on Learning Representations*, 2022.

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

[94] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

[95] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

[96] Wesley J Maddox, Pavel Izmailov, Timur Garipov, Dmitry P Vetrov, and Andrew Gordon Wilson. A simple baseline for bayesian uncertainty in deep learning. *Advances in neural information processing systems*, 32, 2019.

[97] Andrew G Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. *Advances in neural information processing systems*, 33:4697–4708, 2020.

[98] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[99] Kelvin Guu, Albert Webson, Ellie Pavlick, Lucas Dixon, Ian Tenney, and Tolga Bolukbasi. Simfluence: Modeling the influence of individual training examples by simulating training runs. *arXiv preprint arXiv:2303.08114*, 2023.

[100] Paul W Holland. Statistics and causal inference. *Journal of the American statistical Association*, 81(396):945–960, 1986.

[101] Xiaochuang Han, Byron C Wallace, and Yulia Tsvetkov. Explaining black box predictions and unveiling data artifacts through influence functions. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5553–5563, 2020.

[102] Jinghan Yang, Sarthak Jain, and Byron C Wallace. How many and which training points would need to be removed to flip this prediction? *arXiv preprint arXiv:2302.02169*, 2023.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes] , [No] , or [NA] .
- [NA] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for NA).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will be asked to also include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While "[Yes] " is generally preferable to "[No] ", it is perfectly acceptable to answer "[No] " provided a proper justification is given (e.g., "error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering "[No] " or "[NA] " is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Yes. The main claims made in the abstract and introduction accurately reflect the paper's contributions and scope.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

https://doi.org/10.52202/085713-3408

Justification: We discuss the limitations in Appendix H and throughout the body of the paper. Specifically, Section 3 describes the additional computational cost from running ASTRA compared to EKFAC.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: We describe the derivation of ASTRA in Section 2 and Section 3.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We disclose the information to reproduce the main experimental results in Section 5, Section 6 and Appendix F.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: We do not provide open access to the data and code. We use publicly available data for our experiments, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.

https://doi.org/10.52202/085713-3408

- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide all the training and test details necessary to understand the results in Section 5, Section 6 and Appendix F.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We show error bars representing 1 standard error for one-model LDS performance comparisons and all other experiments. Details in Appendix F.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide information on the computer resources needed to reproduce the experiments in Appendix F.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: The research conducted in this paper conform, in every respect, with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss both potential positive societal impacts and negative societal impacts of the work performed in Appendix H.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

https://doi.org/10.52202/085713-3408

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: This paper poses no such risks.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We properly credit the creators or original owners of assets used in the paper.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [NA]

Justification: The paper does not release new assets.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: In addition to using LLMs as a writing, editing, formatting, and a programming aid, we conduct a training data attribution experiment using GPT-2.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (https://neurips.cc/Conferences/2025/LLM) for what should or should not be described.

https://doi.org/10.52202/085713-3408

A Notation & Acronyms

A.1 Notation

Notation	Description
B	Batch size in mini-batch gradient descent
$\mathcal{B}$	Batch size for stochastic Neumann series iterations
D	Number of parameters in neural network
J	Number of iterations for SNI
M	Number of subsets (masks) to sample for LDS computation
N	Number of training data points, $N = \lvert\mathcal{D}\rvert$
P	Number of layers in a neural network.
R	Number of trials (repeats) for LiSSA.
$\mathcal{D} = \{\boldsymbol{z}_i\}_{i=1}^N$	Training dataset
$\mathcal{D}_{\text{query}}$	Query dataset, used to benchmark TDA algorithms and small in practice
$\mathcal{S}$	Data subset of the training dataset
$\boldsymbol{z}_i$	An arbitrary i-th training example
$\boldsymbol{z}_m \in \mathcal{D}$	A training example from the dataset $\mathcal{D}$
$\boldsymbol{z}_q$	A query data point
$\mathbf{x}^{(i)}$	The inputs (feature vector) of the i-th training example
$\mathbf{z}^{(i)}$	The neural network output for the i-th training example
$\mathbf{t}^{(i)}$	The ground-truth target for the i-th example
$\hat{\mathbf{y}}^{(i)}$	Sampled target for the i-th example using model probabilities
ξ	Source of training procedure randomness
ξ_b	Randomness from batch ordering
Ξ	A set containing various seeds ξ
$\boldsymbol{\theta}$	The neural network parameters
$\boldsymbol{\theta}^\star$	Optimal parameters
$\boldsymbol{\theta}^\star(\mathcal{S})$	Optimal parameters trained on data subset $\mathcal{S} \subseteq \mathcal{D}$
$\boldsymbol{\theta}^s$	Final model parameters (not necessarily at optimum)
$\boldsymbol{\theta}^s(\xi)$	Final model parameters (not necessarily at optimum) which depends on randomness ξ
$\boldsymbol{\theta}_k$	The parameters of a network after k iterations of an algorithm, which depends on context
$g(\boldsymbol{\theta}, \mathbf{x})$	The output (logits) of a neural network with parameters $\boldsymbol{\theta}$ and input $\mathbf{x}$
$\mathcal{L}(\mathbf{z}, \mathbf{t})$	Loss function as a function of the neural network output and target (e.g., cross-entropy)
$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{z})$	Loss function as a function of the parameters and training example
$\mathcal{J}(\boldsymbol{\theta}, \mathcal{D})$	Cost function, $\mathcal{J}(\boldsymbol{\theta}, \mathcal{D}) = \frac{1}{N}\sum_{i=1}^N \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{z}_i)$
$f_{\boldsymbol{z}_q}(\boldsymbol{\theta})$	Measurement function on query point $\boldsymbol{z}_q$, typically correct-class margin or absolute error
$h_{f_{\boldsymbol{z}_q}}(\boldsymbol{\theta})$	The Neumann series iteration objective for the iHVP $\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^s)^\top (\mathbf{G} + \lambda \mathbf{I})^{-1}$
$h^{S_i}_{f_{\boldsymbol{z}_q}}(\boldsymbol{\theta})$	The Neumann series iteration objective projected onto the subspace corresponding to S_i
$\mathbf{F}$	The Fisher information matrix
$\mathbf{F}_{\text{EKFAC}}$	The Fisher Information Matrix approximated with EKFAC
$\mathbf{G}$	The Generalized Gauss-Newton Hessian (GGN) matrix
$\tilde{\mathbf{G}}_k$	An unbiased sample of the GGN computed with data at iteration k
$\mathbf{H}$	Hessian matrix $\mathbf{H} \coloneqq \nabla^2_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}^\star, \mathcal{D})$
$\mathbf{P}$	Preconditioning matrix used in ASTRA, chosen as the $\mathbf{G}_{\text{EKFAC}}$
β	Fraction of $\mathcal{D}$ used for subsampling when computing LDS
η	Learning rate when training the neural network
α	Learning rate to find the iHVP with SNI or ASTRA
ϵ	Downweighting amount used in influence functions and SOURCE formulation
λ	Damping parameter to compute iHVPs: a small positive scalar.
$\tilde{\lambda}$	Damping parameter added to the preconditioner: a small positive scalar
τ	Training data attribution method
σ	Eigenvalues in the decomposition of the curvature matrix.
Γ	Group influence of a training data attribution method
$\boldsymbol{\rho}$	The Spearman correlation coefficient [37]
$\otimes$	The Kronecker product

A.2 SOURCE specific notation

Notation	Description
δ_{ki}	Indicator variable used in SOURCE formulation
ℓ	An index variable indicating the current segment in SOURCE
K_ℓ	The number of iterations within segment ℓ in SOURCE
L	The number of segments in SOURCE
T	The number of optimization steps for the underlying model
$\overline{\mathbf{g}}_\ell$	The average gradient in segment ℓ
$\overline{\eta}_\ell$	The average learning rate in segment ℓ
$\overline{\mathbf{r}}_\ell$	Defined in Equation 20
$\overline{\mathbf{S}}_\ell$	Defined in Equation 20
$\tilde{\mathbf{r}}_\ell$	An approximation of $\overline{\mathbf{r}}_\ell$ introduced by Bae et al. [19]

A.3 Acronyms

Acronym	Description
CG	Conjugate Gradient [51]
EKFAC	Eigenvalue-corrected Kronecker-factored Approximate Curvature [27]
FIM	Fisher Information Matrix
iHVP	Inverse Hessian-vector product, or inverse Gauss-Newton-Hessian-vector product
IF	Influence functions [17]
KFAC	Kronecker-factored Approximate Curvature [26]
LiSSA	Linear time Stochastic Second-Order Algorithm [23]
LOO	Leave-one-out
LDS	Linear Datamodeling Score [22]
MLP	Multi-layer perceptron
NI	Neumann series iterations
PBRF	Proximal-Bregman Response Function [36]
SGD	Stochastic gradient descent
SOURCE	Segmented statiOnary UnRolling for Counterfactual Estimation [19]
SNI	Stochastic Neumann series iterations
TDA	Training data attribution

https://doi.org/10.52202/085713-3408

Algorithm 1 iHVP approximation with LiSSA

Require: $\mathbf{v} \in \mathbb{R}^D$, $\alpha > 0$ (learning rate), $J > 0$ (number of iterations), $R > 0$ (repeat size), $\lambda > 0$ (damping term), $\mathcal{B} > 0$ (batch size), $\mathcal{D}$ (training dataset)

$\mathbf{x} \leftarrow \mathbf{0}$ ▷ Initialize the accumulator for final estimation

for $r = 1$ to R **do**

  $\mathbf{v}_0 \leftarrow \mathbf{v}$ ▷ Initialize $\mathbf{v}_0$ as per the initial condition

  for $j = 0$ to $J - 1$ **do**

    $\boldsymbol{\mathcal{B}} \leftarrow \text{SampleWithReplacement}(\mathcal{D}, \mathcal{B})$ ▷ Sample a mini-batch of size $\mathcal{B}$ from $\mathcal{D}$

    $\mathbf{p} \leftarrow \tilde{\mathbf{G}}_{\boldsymbol{\mathcal{B}}} \mathbf{v}_j$ ▷ Compute HVP using mini-batch $\mathcal{B}$.

    $\mathbf{v}_{j+1} \leftarrow \mathbf{v}_j - \alpha(\mathbf{p} + \lambda \mathbf{v}_j) + \alpha \mathbf{v}$ ▷ SNI update rule

  end for

  $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{v}_J$ ▷ Accumulate the result of this repetition

end for

$\mathbf{x} \leftarrow \mathbf{x}/R$ ▷ Average the accumulated results over R repetitions

return $\mathbf{x}$ ▷ Return final iHVP estimation $\mathbf{H}^{-1}\mathbf{v}$

B Extended Preliminaries

B.1 LiSSA and SNI

LiSSA [23] is a second-order optimization algorithm which involves the computation of an iHVP. Koh and Liang [17] choose LiSSA as their iHVP solver, but LiSSA contains other components as well. In this paper, "LiSSA" refers to the iHVP component. Algorithm 1 shows that the primary difference between LiSSA and SNI is that the former repeats the SNI procedure multiple times to reduce variance (highlighted in red). We use $R = 1$ throughout this paper, so LiSSA's iHVP component is equivalent to SNI and thus we use the two terms interchangeably.

B.2 Influence Functions

Previous papers have shown that influence function estimates are often fragile due to the strong assumptions in the influence function derivation [34–36, 42, 85, 86]. In Table 1, we outline the main assumptions.

Table 1: Summary of assumptions of Influence Functions vs. Unrolled Differentiation.

Assumption	**Influence Functions**	**Unrolled Differentiation**
First order approximation	✓	✓
Objective convex with respect to parameters	✓	✗
Model trained to optimal parameters	✓	✗

Despite the strong assumptions in the derivation of influence functions, a poor iHVP approximation can make influence functions estimates appear less reliable than they are. To appreciate these assumptions, we refer the reader to Appendix B.1 of [36] for a well-presented derivation of influence functions.

Ensembling Influence Functions TDA scores can be typically ensembled over multiple training trajectories [19, 22] to mitigate the problem of noise in the training procedure [42, 43]. This is typically done by training models with various seeds $\xi \in \Xi$, and approximating the *expected* first-order downweighting effect with the empirical average of attribution scores τ:

$$\tau_{\text{IF-Ensemble}}(\boldsymbol{z}_m, \boldsymbol{z}_q, \mathcal{D}) := \frac{1}{|\Xi|} \sum_{\xi \in \Xi} \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^s(\xi))^\top (\mathbf{G}_\xi + \lambda \mathbf{I})^{-1} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^s(\xi), \boldsymbol{z}_m), \tag{15}$$

where $\mathbf{G}_\xi$ is the GGN computed at $\boldsymbol{\theta}^s(\xi)$. We perform ensembling in this paper using the procedure in Equation 15, and apply ensembling for SOURCE analogously. Note that Equation 15 ensembles the attribution scores [19, 22], while some other works ensemble the weights [19, 87].

B.3 Curvature Matrices

We explain the relationships between curvature matrices below for completeness, which is heavily based on Grosse [49] and Martens [55].

Approximating H with G Throughout this paper, we use the approximation $\mathbf{G} \approx \mathbf{H}$. Letting $\mathbf{z} = g(\boldsymbol{\theta}, \mathbf{x})$ denote the neural network output[17], the GGN is equal to the Hessian if we drop the second term from the following decomposition of $\mathbf{H}$ [49]:

$$\mathbf{H} = \frac{1}{N} \sum_{(\mathbf{x}^{(i)}, \mathbf{t}^{(i)}) \in \mathcal{D}} [\mathbf{J}_{\mathbf{z}^{(i)}\boldsymbol{\theta}}{}^{\top} \mathbf{H}_{\mathbf{z}^{(i)}} \mathbf{J}_{\mathbf{z}^{(i)}\boldsymbol{\theta}} + \sum_j \frac{\partial \mathcal{L}}{\partial \mathbf{z}_j^{(i)}} \nabla^2_{\boldsymbol{\theta}} g(\boldsymbol{\theta}, \mathbf{x}^{(i)})_j], \tag{16}$$

where $\mathbf{J}_{\mathbf{z}^{(i)}\boldsymbol{\theta}}$ is the Jacobian matrix of the neural network's outputs with respect to the parameters for the ith training example, $\mathbf{H}_{\mathbf{z}^{(i)}} := \nabla^2_{\mathbf{z}} \mathcal{L}\big(\mathbf{z}^{(i)}, \mathbf{t}^{(i)}\big)$ refers to the Hessian of the loss function with respect to the neural network outputs for the ith training example, $\frac{\partial \mathcal{L}}{\partial \mathbf{z}_j^{(i)}}$ refers to the derivative of the loss function with respect to the jth neural network output for the ith training example and $\nabla^2_{\boldsymbol{\theta}} g(\boldsymbol{\theta}, \mathbf{x}^{(i)})_j$ refers to the Hessian of the jth neural network output for the ith training example with respect to the parameters. For a linear neural network, $\nabla^2_{\boldsymbol{\theta}} g(\boldsymbol{\theta}, \mathbf{x}^{(i)})_j = 0$, so the GGN is equal to the Hessian if we linearize the neural networks with respect to the parameters and only capture the curvature in the loss function. Linearization of the neural network is an approximation documented in previous works [55, 88–90] and used frequently for influence functions if the damped GGN $\mathbf{G} + \lambda \mathbf{I}$ is used [7, 11, 22, 36].

Equivalence of F and G For the machine learning tasks that we consider, such as regression and classification tasks, the outputs of the neural network can be seen as the natural parameters of an exponential family. For these cases, the Fisher Information Matrix and the GGN coincide (i.e. $\mathbf{F} = \mathbf{G}$). We will illustrate this for softmax classification, but the case for regression can be derived similarly. The cross-entropy loss for a training example $\boldsymbol{z} = (\mathbf{x}, \mathbf{t})$ is $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{z}) = -\mathbf{t}^{\top} \log p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$ whose gradient is: $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{z}) = \mathbf{J}^{\top}_{\mathbf{z}\boldsymbol{\theta}} \big(p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}) - \mathbf{t}\big)$. Then the following equalities hold:

$$\begin{aligned}
\mathbf{F} &= \frac{1}{N} \sum_{(\mathbf{x}^{(i)}, \mathbf{t}^{(i)}) \in \mathcal{D}} \mathbb{E}_{\hat{\mathbf{y}} \sim p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}^{(i)})} \Big[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\hat{\mathbf{y}}|\mathbf{x}^{(i)}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\hat{\mathbf{y}}|\mathbf{x}^{(i)})^{\top} \Big] \\
&= \frac{1}{N} \sum_{(\mathbf{x}^{(i)}, \mathbf{t}^{(i)}) \in \mathcal{D}} \mathbb{E}_{\hat{\mathbf{y}} \sim p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}^{(i)})} \Big[\mathbf{J}^{\top}_{\mathbf{z}^{(i)}\boldsymbol{\theta}} (p_{\boldsymbol{\theta}}(\hat{\mathbf{y}} \mid \mathbf{x}^{(i)}) - \hat{\mathbf{y}}) (p_{\boldsymbol{\theta}}(\hat{\mathbf{y}} \mid \mathbf{x}^{(i)}) - \hat{\mathbf{y}})^{\top} \mathbf{J}_{\mathbf{z}^{(i)}\boldsymbol{\theta}} \Big] \\
&= \frac{1}{N} \sum_{(\mathbf{x}^{(i)}, \mathbf{t}^{(i)}) \in \mathcal{D}} \mathbf{J}^{\top}_{\mathbf{z}^{(i)}\boldsymbol{\theta}} \Big(\operatorname{diag}\big(p_{\boldsymbol{\theta}}(\hat{\mathbf{y}} \mid \mathbf{x}^{(i)})\big) \; - \; p_{\boldsymbol{\theta}}(\hat{\mathbf{y}} \mid \mathbf{x}^{(i)}) \, p_{\boldsymbol{\theta}}(\hat{\mathbf{y}} \mid \mathbf{x}^{(i)})^{\top} \Big) \mathbf{J}_{\mathbf{z}^{(i)}\boldsymbol{\theta}} \\
&= \frac{1}{N} \sum_{(\mathbf{x}^{(i)}, \mathbf{t}^{(i)}) \in \mathcal{D}} \mathbf{J}^{\top}_{\mathbf{z}^{(i)}\boldsymbol{\theta}} \nabla^2_{\mathbf{z}} \mathcal{L}\big(\mathbf{z}^{(i)}, \mathbf{t}^{(i)}\big) \mathbf{J}_{\mathbf{z}^{(i)}\boldsymbol{\theta}} = \mathbf{G}
\end{aligned}$$

B.4 Training Data Attribution with Unrolled Differentiation

Influence functions may struggle with models that have not converged [19, 35, 36, 83] (Table 1). Fortunately, unrolled differentiation methods [4, 18–21] do not rely on model convergence. Instead, they capture the effect of downweighting a training example by differentiating through the entire training trajectory. They can also capture the effect of other sources of randomness, such as batch ordering [41]. Assume our optimization algorithm is mini-batch gradient descent, which uses a learning rate η_k and batch size B, and let δ_{ki} be an indicator variable that equals 1 if and only if $\boldsymbol{z}_m = \boldsymbol{z}_{ki}$, where $\boldsymbol{z}_{ki}$ is the i-th training example in batch k.[18] Then the mini-batch gradient descent

[17] Note the difference between the bold font $\mathbf{z}$, which refers to neural network outputs, with italicized font $\boldsymbol{z}$, which refers to a data point $\boldsymbol{z} = (\mathbf{x}, \mathbf{t})$

[18] We note that $\boldsymbol{\theta}_k$ in this setting refers to the parameters at time step k when training the network and distinguish it from SNI or ASTRA iterations presented in Section 2 and Section 3.

https://doi.org/10.52202/085713-3408

update rule is:

$$\boldsymbol{\theta}_{k+1}(\epsilon) \leftarrow \boldsymbol{\theta}_k(\epsilon) - \frac{\eta_k}{B}\sum_{i=1}^{B}(1-\delta_{ki}\epsilon)\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_k(\epsilon), \boldsymbol{z}_{ki}), \tag{17}$$

where $\delta_k \coloneqq \sum_{i=1}^{B}\delta_{ki}$. Let ξ_b denote the randomness from batch ordering. Then the expected first-order effect on the parameters $\boldsymbol{\theta}_T$ after T steps of training with $\boldsymbol{z}_m$ downweighted by ϵ is:[19]

$$\mathbb{E}_{\xi_b}\left[\frac{\mathrm{d}\boldsymbol{\theta}_T}{\mathrm{d}\epsilon}\right] = -\mathbb{E}_{\xi_b}\left[\sum_{k=0}^{T-1}\frac{\eta_k}{B}\delta_k\left(\prod_{i=T-1}^{k+1}(\mathbf{I}-\eta_i\mathbf{G}_i)\right)\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_k, \boldsymbol{z}_m)\right], \tag{18}$$

Bae et al. [19] introduce an algorithm called SOURCE, which approximates Equation 18 much more cheaply by segmenting the trajectory into L segments, assuming stationary and independent GGNs and gradients within each segment. For the ℓth segment which starts at iteration $T_{\ell-1}$ and ends at iteration T_ℓ, and k satisfying $T_{\ell-1} \le k < T_\ell$, let $\overline{\mathbf{G}}_\ell \coloneqq \mathbb{E}_{\xi_b}[\mathbf{G}_k]$, $\overline{\mathbf{g}}_\ell \coloneqq \mathbb{E}_{\xi_b}[\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_k, \boldsymbol{z}_m)]$, and $\overline{\eta}_\ell$ refer to the average GGN, average gradient, and average learning rate in segment ℓ respectively and let $K_\ell \coloneqq T_\ell - T_{\ell-1}$ refer to the total number of iterations in segment ℓ. Then SOURCE approximates the first-order effect of downweighting parameters as:

$$\mathbb{E}_{\xi_b}\left[\frac{\mathrm{d}\boldsymbol{\theta}_T}{\mathrm{d}\epsilon}\right] \approx -\frac{1}{N}\sum_{\ell=1}^{L}\left(\prod_{\ell'=L}^{\ell+1}(\mathbf{I}-\overline{\eta}_{\ell'}\overline{\mathbf{G}}_{\ell'})^{K_{\ell'}}\right)\left(\sum_{k=T_{\ell-1}}^{T_\ell-1}\overline{\eta}_\ell(\mathbf{I}-\overline{\eta}_\ell\overline{\mathbf{G}}_\ell)^{T_\ell-1-k}\overline{\mathbf{g}}_\ell\right) \tag{19}$$

$$\approx -\frac{1}{N}\sum_{\ell=1}^{L}\left(\prod_{\ell'=L}^{\ell+1}\underbrace{\exp(-\overline{\eta}_{\ell'}K_{\ell'}\overline{\mathbf{G}}_{\ell'})}_{\overline{\mathbf{S}}_{\ell'}}\right)\underbrace{\left(\mathbf{I}-\exp(-\overline{\eta}_\ell K_\ell\overline{\mathbf{G}}_\ell)\right)\overline{\mathbf{G}}_\ell^{-1}\overline{\mathbf{g}}_\ell}_{\overline{\mathbf{r}}_\ell}. \tag{20}$$

The stationary and independent assumptions allow us to factor shared products resulting in the approximation in Equation 19, which can be then approximated with matrix exponentials in Equation 20 and computed with the EKFAC eigendecomposition. Bae et al. [19] provide an interpretation of the $\overline{\mathbf{r}}_\ell$ term, noticing that $\overline{\mathbf{r}}_\ell \approx \left(\overline{\mathbf{G}}_\ell + \overline{\eta}_\ell^{-1}K_\ell^{-1}\mathbf{I}\right)^{-1}\overline{\mathbf{g}}_\ell \coloneqq \tilde{\mathbf{r}}_\ell$, which is an iHVP that we can use to apply ASTRA. We discuss this term in more detail in Appendix C. We refer the reader to Bae et al. [19] for a full explanation of SOURCE.

B.5 Kronecker-Factored Approximate Curvature

The KFAC approximation was introduced in [26] in the context of second-order optimization and explained in [7] in the context of influence functions. To understand the cost of EKFAC and EKFAC-IF compared to ASTRA-IF, as well as the assumptions EKFAC make which results in a biased iHVP approximation, we present the derivation below which is heavily based on Grosse [49] and Grosse et al. [7] and refer readers to Martens and Grosse [26] and George et al. [27] for further reading.

Our goal is to compute the iHVP with the Fisher Information Matrix (FIM) $\mathbf{F}$ as an approximation for the Hessian $\mathbf{H}$. The FIM is defined as:

$$\mathbf{F} \coloneqq \frac{1}{N}\sum_{\mathbf{x}^{(i)}\in\mathcal{D}}\mathbb{E}_{\hat{\mathbf{y}}\sim p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}^{(i)})}\left[\nabla_{\boldsymbol{\theta}}\log p_{\boldsymbol{\theta}}(\hat{\mathbf{y}}|\mathbf{x}^{(i)})\nabla_{\boldsymbol{\theta}}\log p_{\boldsymbol{\theta}}(\hat{\mathbf{y}}|\mathbf{x}^{(i)})^\top\right] \tag{21}$$

where $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$ is the model's own distribution over targets. We omit the random variables in the expectation's subscripts going forward to reduce clutter. Using the model's own distribution over targets (as opposed to actual targets) is rather important since using the actual targets rather than the model's distribution results in a matrix called the Empirical Fisher, which does not have the same properties as the FIM [91].

We now describe KFAC for multilayer perceptrons. We refer readers to Grosse and Martens [84] for the derivation for convolution layers. Consider the l-th layer of a neural network,[20] which has input

[19] Unless otherwise stated, derivatives taken with respect to ϵ are evaluated at $\epsilon = 0$. The notation $\prod_{i=T-1}^{k+1}$ means taking products in decreasing order from $T-1$ to $k+1$.

[20] Note the difference between ℓ, which refers to a segment in SOURCE, and l, which is an index denoting a layer of a neural network.

activations $\mathbf{a}_{l-1} \in \mathbb{R}^I$, weights $\mathbf{W}_l \in \mathbb{R}^{O \times I}$, bias $\mathbf{b}_l \in \mathbb{R}^O$, and outputs $\mathbf{s}_l \in \mathbb{R}^O$. For convenience, we use the notation $\overline{\mathbf{a}}_{l-1} = \left[\mathbf{a}_{l-1}^\top \; 1\right]^\top$ and $\overline{\mathbf{W}}_l = [\mathbf{W}_l \; \mathbf{b}_l]$ to handle weights and biases together, and we write $\boldsymbol{\theta}_l = \mathrm{vec}(\overline{\mathbf{W}}_l)$ to denote the reshaped vector of layer l parameters. Then each layer computes:

$$\mathbf{s}_l \;=\; \overline{\mathbf{W}}_l\, \overline{\mathbf{a}}_{l-1}, \quad \mathbf{a}_l \;=\; \phi_l(\mathbf{s}_l), \tag{22}$$

where ϕ_l is the activation function. We will define the *pseudo-gradient* operator as:

$$\mathcal{D}v \coloneqq \nabla_v \, \log p_{\boldsymbol{\theta}}(\hat{\mathbf{y}} \mid \mathbf{x}) \tag{23}$$

for notational convenience. Notice that $\mathcal{D}v$ is a random variable whose randomness arises from sampling $\hat{\mathbf{y}}$. Using the properties of the Kronecker product, we can write the pseudo-gradient of $\boldsymbol{\theta}_l$ as:

$$\mathcal{D}\boldsymbol{\theta}_l = \mathrm{vec}(\mathcal{D}\overline{\mathbf{W}}_l) = \mathrm{vec}(\mathcal{D}\mathbf{s}_l \overline{\mathbf{a}}_{l-1}^\top) = \overline{\mathbf{a}}_{l-1} \otimes \mathcal{D}\mathbf{s}_l, \tag{24}$$

where $\otimes$ is the Kronecker product. Then the lth block of the FIM can be approximated as:

$$\begin{aligned}
\mathbf{F}_l &= \mathbb{E}\left[\mathcal{D}\boldsymbol{\theta}_l \mathcal{D}\boldsymbol{\theta}_l^\top\right] && (25)\\
&= \mathbb{E}\left[(\overline{\mathbf{a}}_{l-1} \otimes \mathcal{D}\mathbf{s}_l)(\overline{\mathbf{a}}_{l-1} \otimes \mathcal{D}\mathbf{s}_l)^\top\right] && (26)\\
&= \mathbb{E}\left[\overline{\mathbf{a}}_{l-1}\, \overline{\mathbf{a}}_{l-1}^\top \otimes \mathcal{D}\mathbf{s}_l \mathcal{D}\mathbf{s}_l^\top\right] && (27)\\
&\approx \mathbb{E}\left[\overline{\mathbf{a}}_{l-1}\, \overline{\mathbf{a}}_{l-1}^\top\right] \otimes \mathbb{E}\left[\mathcal{D}\mathbf{s}_l \mathcal{D}\mathbf{s}_l^\top\right] && (28)\\
&= \mathbf{A}_{l-1} \;\otimes\; \mathbf{S}_l, && (29)
\end{aligned}$$

where have applied Kronecker product identities on the third equality. Our final approximation $\widehat{\mathbf{F}}$ to the FIM is the block-diagonal matrix in which each block is $\mathbf{F}_l$. Here, $\mathbf{A}_{l-1} = \mathbb{E}[\overline{\mathbf{a}}_{l-1}\, \overline{\mathbf{a}}_{l-1}^\top]$ and $\mathbf{S}_l = \mathbb{E}[\mathcal{D}\mathbf{s}_l \mathcal{D}\mathbf{s}_l^\top]$ are the uncentered covariance matrices of the activations and the pre-activation pseudo-gradients with dimensions $(I+1) \times (I+1)$ and $O \times O$, respectively. Practically, we can estimate the expectations via an empirical estimate and store the resulting statistics $\widehat{\mathbf{A}}_{l-1} = \frac{1}{N} \sum_{\mathcal{D}} \overline{\mathbf{a}}_{l-1}\, \overline{\mathbf{a}}_{l-1}^\top$ and $\widehat{\mathbf{S}}_l = \frac{1}{N} \sum_{\mathcal{D}} \mathcal{D}\mathbf{s}_l \mathcal{D}\mathbf{s}_l^\top$, and we define $\widehat{\mathbf{F}}_l \coloneqq \widehat{\mathbf{A}}_{l-1} \otimes \widehat{\mathbf{S}}_l$.

To approximate $(\mathbf{F} + \lambda \mathbf{I})^{-1}\mathbf{v}$ for a vector $\mathbf{v}$ as needed to compute influence functions, we can compute $(\mathbf{F}_l + \lambda \mathbf{I})^{-1}\mathbf{v}_l$ separately for each layer l. Let $\overline{\mathbf{V}}_l$ be the slice of $\mathbf{v}$ reshaped to match $\overline{\mathbf{W}}_l$, and define $\mathbf{v}_l = \mathrm{vec}(\overline{\mathbf{V}}_l)$. Applying the eigendecompositions $\mathbf{A}_{l-1} = \mathbf{Q}_{\mathbf{A}_{l-1}} \mathbf{D}_{\mathbf{A}_{l-1}} \mathbf{Q}_{\mathbf{A}_{l-1}}^\top$ and $\mathbf{S}_l = \mathbf{Q}_{\mathbf{S}_l} \mathbf{D}_{\mathbf{S}_l} \mathbf{Q}_{\mathbf{S}_l}^\top$, and using the Kronecker identity $\mathbf{U} \otimes \mathbf{V} = (\mathbf{Q}_\mathbf{U} \otimes \mathbf{Q}_\mathbf{V})\,(\mathbf{D}_\mathbf{U} \otimes \mathbf{D}_\mathbf{V})\,(\mathbf{Q}_\mathbf{U}^\top \otimes \mathbf{Q}_\mathbf{V}^\top)$ for two symmetric matrices $\mathbf{U} \otimes \mathbf{V}$, we can write:

$$\begin{aligned}
(\mathbf{F}_l + \lambda \mathbf{I})^{-1}\mathbf{v}_l &= (\mathbf{A}_{l-1} \otimes \mathbf{S}_l + \lambda \mathbf{I})^{-1}\mathbf{v}_l && (30)\\
&= (\mathbf{Q}_{\mathbf{A}_{l-1}} \otimes \mathbf{Q}_{\mathbf{S}_l}) \underbrace{(\mathbf{D}_{\mathbf{A}_{l-1}} \otimes \mathbf{D}_{\mathbf{S}_l}}_{\text{Scaling matrix } \boldsymbol{\Lambda}_{\text{KFAC}}} + \lambda \mathbf{I}_{\mathbf{A}_{l-1}} \otimes \mathbf{I}_{\mathbf{S}_l})^{-1} \underbrace{(\mathbf{Q}_{\mathbf{A}_{l-1}} \otimes \mathbf{Q}_{\mathbf{S}_l})^\top}_{\text{Orthonormal eigenbasis}} \mathbf{v}_l && (31)
\end{aligned}$$

where $\mathbf{I}_{\mathbf{A}_{l-1}}$ and $\mathbf{I}_{\mathbf{S}_l}$ represent identity matrices of the same shape as $\mathbf{A}_{l-1}$ and $\mathbf{S}_l$ respectively.

Eigenvalue Corrected Kronecker-Factored Approximate Curvature One simple adjustment to the KFAC approximation can yield material improvements to the curvature approximation as well as the influence approximation. The KFAC formulation in Equation 31 suggests that after expressing $\mathbf{v}_l$ in the eigenbasis $(\mathbf{Q}_{\mathbf{A}_{l-1}} \otimes \mathbf{Q}_{\mathbf{S}_l})^\top$ we scale each element with $(\mathbf{D}_{\mathbf{A}_{l-1}} \otimes \mathbf{D}_{\mathbf{S}_l} + \lambda \mathbf{I}_{\mathbf{A}_{l-1}} \otimes \mathbf{I}_{\mathbf{S}_l})^{-1}$. Observing that for any matrix $\mathbf{W} = \mathbb{E}[\mathbf{u}\mathbf{u}^\top] = \mathbf{U}\mathbf{S}\mathbf{U}^\top$, it is true that $\mathbf{S}_{ii} = \mathbb{E}[(\mathbf{U}^\top \mathbf{v})_i^2]$, George et al. [27] propose that a better scaling matrix is the diagonal matrix $\boldsymbol{\Lambda}_{\text{EKFAC}}$ with entries:

$$\boldsymbol{\Lambda}_{ii} = \mathbb{E}\left[\left((\mathbf{Q}_{\mathbf{A}_{l-1}} \otimes \mathbf{Q}_{\mathbf{S}_l})^\top \mathcal{D}\boldsymbol{\theta}_l\right)_i^2\right]. \tag{32}$$

In practice, this eigenvalue correction results in better influence estimates compared with KFAC.

Assumptions in EKFAC From the equations above, we can see that KFAC makes two critical approximations which EKFAC inherits: First, it assumes that correlations between $\mathcal{D}\boldsymbol{\theta}_i$ and $\mathcal{D}\boldsymbol{\theta}_j$ are zero if they belong to different layers, yielding a block-diagonal approximation of $\mathbf{F}$. Second, it treats the activations as independent from the pre-activation pseudo-gradients, the basis of Equation 28.

https://doi.org/10.52202/085713-3408

Furthermore, the matrix $\mathbf{S}_l$ depends on the *sampled* labels $\widehat{\mathbf{y}}$, which for efficiency reasons is usually only sampled once per input $\mathbf{x}$ which also may introduce some approximation error. In total, these assumptions cause $\mathbf{F}_{\text{EKFAC}}$ to differ from the true FIM $\mathbf{F}$, which is an error that ASTRA corrects. Grosse and Martens [84] introduce the KFAC approximation for convolution layers which introduces two further assumptions – spatially uncorrelated derivatives and spatial homogeneity. Consistent with past works [1, 19], we find that EKFAC-IF struggles for convolution architectures in comparison with MLPs. ASTRA-IF's TDA performance on ResNet-9 achieves a particularly large improvement compared to EKFAC-IF.

Cost of Computing EKFAC-IF The three main components of computing EKFAC-IF are: 1) collecting the statistics $\widehat{\mathbf{A}}_{l-1}$ and $\widehat{\mathbf{S}}_l$, which requires a backward pass over the whole dataset. 2) computing the eigendecompositions $\mathbf{A}_{l-1} = \mathbf{Q}_{\mathbf{A}_{l-1}}\mathbf{D}_{\mathbf{A}_{l-1}}\mathbf{Q}_{\mathbf{A}_{l-1}}^\top$ and $\mathbf{S}_l = \mathbf{Q}_{\mathbf{S}_l}\mathbf{D}_{\mathbf{S}_l}\mathbf{Q}_{\mathbf{S}_l}^\top$, whose total precise cost depends on the architecture [7]. 3) if we want to search the whole dataset $\mathcal{D}$ for influential examples, once we approximate $(\mathbf{F}+\lambda\mathbf{I})^{-1}\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^s)$ via the procedure above, we need to take a dot product with every training example gradient under consideration in $\mathcal{D}$. ASTRA adds an incremental iterative procedure to the cost of EKFAC-IF which results in stronger TDA performance.

C Introducing ASTRA-SOURCE

Understanding the iHVP We have discussed how to compute iHVPs and therefore influence functions with ASTRA. We can also apply ASTRA to SOURCE by making the substitution $\overline{\mathbf{r}}_\ell \approx \left(\overline{\mathbf{G}}_\ell + \overline{\eta}_\ell^{-1}K_\ell^{-1}\mathbf{I}\right)^{-1}\overline{\mathbf{g}}_\ell := \tilde{\mathbf{r}}_\ell$ described in Appendix B.4, which replaces the matrix exponential in $\overline{\mathbf{r}}_\ell$. To understand the approximation, observe that the following term (with some rearranging) found in Equation 19 $\overline{\eta}_\ell \sum_{k=T_{\ell-1}}^{T_\ell-1}(\mathbf{I}-\overline{\eta}_\ell\overline{\mathbf{G}}_\ell)^{T_\ell-1-k}\overline{\mathbf{g}}_\ell$ resembles applying the Neumann series iterations on the vector $\mathbf{v}=\overline{\mathbf{g}}_\ell$ and the matrix $\mathbf{A}=\overline{\mathbf{G}}_\ell$ with a learning rate of $\overline{\eta}_\ell$ for a total of $K_\ell-1$ iterations. For large enough K_ℓ, the truncation can be seen as approximately running Neumann series iterations until convergence, which results in the iHVP $\overline{\mathbf{G}}_\ell^{-1}\overline{\mathbf{g}}_\ell$. However, each segment actually only involves K_ℓ iterations at an average learning rate of $\overline{\eta}_\ell$, and thus is more closely related to *truncated* Neumann series iterations (Appendix G), in which the parameters make less progress in the low eigenvalue directions. Bae et al. [19] provide an interpretation that this can be approximated with damping as follows: $\overline{\eta}_\ell \sum_{k=T_{\ell-1}}^{T_\ell-1}(\mathbf{I}-\overline{\eta}_\ell\overline{\mathbf{G}}_\ell)^{T_\ell-1-k}\overline{\mathbf{g}}_\ell \approx (\overline{\mathbf{G}}+\overline{\eta}_\ell^{-1}K_\ell^{-1}\mathbf{I})^{-1}\overline{\mathbf{g}}_\ell$, and that over a wide range of eigenvalues, the qualitative behavior matches well. The transformation of this term into an iHVP allows us to apply ASTRA to SOURCE.

Practical Implementation Recall that the final goal in SOURCE is to approximate $\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^s)^\top \mathbb{E}_{\xi_b}\left[\frac{\mathrm{d}\boldsymbol{\theta}_T}{\mathrm{d}\epsilon}\right]$. To do this, we follow Equation 20 from left to right: for all $\ell = 1,\ldots,L$ segments, we first compute $-\frac{1}{N}\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^s)^\top \prod_{\ell'=L}^{\ell+1}\overline{\mathbf{S}}_{\ell'}$ in the same manner as SOURCE. This will be the vector in our iHVP. ASTRA differs from SOURCE in that instead of multiplying this vector by another matrix exponential, we multiply it by the inverse damped GGN $\left(\overline{\mathbf{G}}_\ell + \overline{\eta}_\ell^{-1}K_\ell^{-1}\mathbf{I}\right)^{-1}$, which we can do using ASTRA in the same manner described previously. Finally, we multiply by the average gradient, $\overline{\mathbf{g}}_\ell$ and accumulate the result over L segments, which is done in the same manner as SOURCE. Compared to ASTRA-IF, there is an additional detail introduced: how to obtain the average GGN $\overline{\mathbf{G}}_\ell$. There are a number of options available, but we find that using the average weights in the segment works well, which we discuss below. The full procedure of applying ASTRA to SOURCE requires L iHVPs per query. In many cases, the number of segments L is likely to be small.[21] Since the preconditioners used by ASTRA would need to be computed if we were to run EKFAC-SOURCE anyways, the incremental cost of ASTRA-SOURCE compared to EKFAC-SOURCE is no more than L times the number of iterations for each iHVP, which is hundreds of iterations in our experiments.

Approximation of Other Terms We have discussed how to improve the approximation of $\overline{\mathbf{r}}_\ell$ with ASTRA, but have not discussed $\overline{\mathbf{S}}_{\ell'}$. We found evidence that improving the quality of the term $\overline{\mathbf{r}}_\ell$ improved TDA performance, but could not find the same evidence for $\overline{\mathbf{S}}_{\ell'}$. Therefore we spent

[21] Part of the motivation for SOURCE is to devise a scalable TDA algorithm for multi-stage training procedures, in which the number of segments is likely to be modest. Bae et al. [19] use $L=3$.

additional compute on improving the iHVP approximation. As a result, we will leave $\overline{\mathbf{S}}_{\ell'}$ as the approximation involving EKFAC.

Computing the Average Gauss-Newton Hessian There are a number of options in computing the average GGN $\overline{\mathbf{G}}_\ell$. **Option 1)**: since $\overline{\mathbf{G}}_\ell := \mathbb{E}_{\xi_b}[\mathbf{G}_k]$ for $T_{\ell-1} \leq k < T_\ell$, one can compute the matrix-vector product involving $\overline{\mathbf{G}}_\ell$ and $\mathbf{v} \in \mathbb{R}^D$ simply by taking an empirical average over samples within the segment: $\overline{\mathbf{G}}_\ell \mathbf{v} \approx \frac{1}{K_\ell} \sum_{T_{\ell-1} \leq k < T_\ell} \mathbb{E}_{\xi_b}[\mathbf{G}_k]\mathbf{v}$. This might be costly since one would have to load multiple checkpoints into memory just to compute a forward pass. **Option 2)**: Instead of sampling *every* checkpoint in the segment, we could reduce the number of samples and take the empirical average of that instead. This is consistent with SOURCE as it uses only a subset of checkpoints in each segment anyways. If the segments chosen in SOURCE are indeed stationary as the derivation approximates, then in the extreme, we could take one checkpoint in each segment as the representative. **Option 3)**: Bae et al. [19] provides an alternative averaging scheme called FAST-SOURCE, in which one averages the parameters rather than the gradients, and shows that it works approximately as well as SOURCE. We tested Option 2 and Option 3 for a few settings and could not find meaningful differences, so opted to present the results for Option 3, using the average weights.

D Extended Related Works

Understanding Influence Functions A number of previous works [34–36, 42, 85, 86, 92] have studied influence functions accuracy in various modern neural network settings. [34] show that influence functions do not approximate LOO retraining well. [36] discover that the derivation of influence functions actually approximate the *Proximal Bregman Response Function* (PBRF) rather than LOO retraining, which can be seen as an objective which tries to *maximize* loss on the removed training example, subject to constraints in function space and weight space measured from the final parameters. Two large contributions to influence functions error are the *warm-start retraining* assumption, which assumes that the counterfactual model is initialized at the final parameters, and the *non-convergence gap*, which relates to the fact that the derivation assumes the model has converged to an optimal solution. Ensembling can help address the warm-start retraining assumption, while the non-convergence gap is addressed by [19]. Others have focused on whether influence functions can accurately approximate group influence [5, 10, 92, 93] as it makes a linearity assumption due to the fact that it is a first-order approximation. While LOO influence is very noisy [19, 43], Ilyas et al. [10] discover that model predictions are approximately linear with respect to training example inclusion, which provides the justification for LDS [22]. Overall, these works typically aim to address the fundamental assumptions surrounding influence functions (Table 1). In contrast, our work shows that a poorly approximated iHVP can cause substantial performance degradation.

Ensembling in Influence Functions Ensembling combines multiple models for improved generalization, uncertainty estimation, and calibration. It is a common approach to estimate the model posterior $p(\boldsymbol{\theta}|\mathcal{D})$ in Bayesian deep learning. Different strategies for sampling models as members of an ensemble exist: For example, deep ensembles sample models from varying random initializations [94] to represent variations stemming from possible training trajectories, while stochastic weight averaging (SWA) approaches sample model parameters from the final iterations of model training [95, 96], which has the advantage of reduced training cost. Other methods may combine these ideas [97], or use different approaches (e.g., Dropout [98]). Taking the average across several runs with varying sources of training process randomness is a common approach to account for the variability of model training in TDA estimation [10, 19, 22]. This can be seen as sampling from the distribution of true TDA to estimate the average treatment effect of excluding a training subset [43] and has been effective in stabilizing estimations as well as evaluations (e.g. the LDS [22]). The size of the ensemble is generally connected to improved estimation quality of TDA scores, as shown in [22]. Our results show that ASTRA enjoys a larger performance boost from ensembling.

https://doi.org/10.52202/085713-3408

E Evaluating TDA

In this paper, we evaluate the performance of TDA methods with the LDS [22], a widely used metric which we shortly described in Section 5. Besides LDS, other evaluations for measuring TDA performance also exist. In this section, we present alternate methods of TDA evaluation.

Expected leave-one-out retraining. TDA methods usually define the influence of a training sample $\boldsymbol{z}_m$ on the model as the change in the model's predictions if $\boldsymbol{z}_m$ were not part of the training set. Hence, a straightforward way to compute the ground-truth to compare against is leave-one-out (LOO) retraining, as done in [6, 17, 99]. However, since the stochasticity inherent to model training makes LOO a noisy measure [42, 43], the LOO score should be considered in expectation over the training process stochasticity ξ. We can then define the expected leave-one-out (ELOO) score as:

$$\text{ELOO}(\boldsymbol{z}_m, \boldsymbol{z}_q, \mathcal{D}) \coloneqq \mathbb{E}_\xi[f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^s(\mathcal{D} \setminus \{\boldsymbol{z}_m\}; \xi))] - \mathbb{E}_\xi[f_{\boldsymbol{z}_q}(\boldsymbol{\theta}^s(\mathcal{D}; \xi))]. \tag{33}$$

This score can be viewed as the ground-truth average treatment effect (ATE) [100] of the removal of $\boldsymbol{z}_m$ from $\mathcal{D}$. While principled, in practice, the effect of removing a single point is highly noisy [19, 43] so that a stable estimate of ELOO may only be achieved with an extremely large number of samples to compute the empirical expectation. In contrast, LDS considers the ATE of excluding a *group* of training samples from training, which has been shown to be more stable in expectation [19, 22].

Top-k removal and retraining. The sign of the TDA scores indicates whether the excluded training samples are positively or negatively influential [19], also referred to as proponents and opponents [83], helpful and harmful samples [17], or excitatory and inhibitory [6], respectively [19]. The idea behind this evaluation is that the removal of positively influential samples $\{\boldsymbol{z}_m\}$ removes support for the query sample $\boldsymbol{z}_q$ and consequently should lead to a change in prediction confidence on $\boldsymbol{z}_q$ [101]. [101] conduct this evaluation by removing the top and bottom 10% of training samples ranked by influence functions and compare against the removal of the least influential (i.e., smallest influence scores by magnitude) and random samples to see if the resulting models change their predictions in the expected way. Similar to top-k removal and retraining, previous work has tested how many highly influential samples need to be removed to *flip* a prediction [19, 102], called subset removal counterfactual evaluation. Similar to LDS, removing top-k samples is based on counterfactual retraining with excluded groups. However, one core difference is that since LDS involves a sum over all the attribution scores for every $\boldsymbol{z}_m$ in each subset (Equation 10), poorly *calibrated* attribution scores resulting in one outlier score $\tau(\boldsymbol{z}_m, \boldsymbol{z}_q, \mathcal{D})$ may result in poor LDS, demanding that the TDA method assigns calibrated scores across all points $\boldsymbol{z}_m$ in consideration.

F Experiment Details

F.1 Choice of Measurement Function

While in principle the derivation of our influence scores does not restrict what measurement function $f_{\boldsymbol{z}_q}$ is used, in practice some choices of measurement functions work better than others. In our experiments, for regression problems, we use the measurement function:

$$f_{\boldsymbol{z}_q}(\boldsymbol{\theta}) = |g(\boldsymbol{\theta}, \mathbf{x}_q) - \mathbf{t}_q| \tag{34}$$

where $g(\boldsymbol{\theta}, \mathbf{x}_q)$ denotes the last layer output of the neural network when $\mathbf{x}_q$ is the input and $\mathbf{t}_q$ is a scalar output. For all classification problems with the exception of GPT-2, we use the measurement function:

$$f_{\boldsymbol{z}_q}(\boldsymbol{\theta}) = -\log \frac{\sigma(g(\boldsymbol{\theta}, \mathbf{x}_q))_{\mathbf{t}_q}}{1 - \sigma(g(\boldsymbol{\theta}, \mathbf{x}_q))_{\mathbf{t}_q}} \tag{35}$$

$$= -g(\boldsymbol{\theta}, \mathbf{x}_q)_{\mathbf{t}_q} + \log \left(\sum_i \exp g(\boldsymbol{\theta}, \mathbf{x}_q)_i - \exp g(\boldsymbol{\theta}, \mathbf{x}_q)_{\mathbf{t}_q} \right), \tag{36}$$

where σ denotes the softmax function and the subscript $\mathbf{t}_q$ refers to the taking the entry corresponding to the correct class. This measurement function is identical to the one found in [22]. For GPT-2, we use the training loss as the measurement function.

F.2 Experiment Details

Table 2 shows the architecture and hyperparameters used to compute the final parameters $\boldsymbol{\theta}^s$, which we use to run EKFAC-IF, EKFAC-SOURCE, ASTRA-IF and ASTRA-SOURCE. The first column shows the size of the training dataset and the number of query examples in each setting. The third column shows the hyperparameters used to train the models in each setting; we use the same hyperparameters as reported in Bae et al. [19]. We estimate the expected ground-truth in the left-hand-side of Equation 10 with an empirical average – the last column in Table 2 shows the number of repeats *per mask* $\mathcal{S}_j$ (i.e., number of ξ sampled) to estimate this value. All settings use a constant learning rate, with the exception of CIFAR-10, which uses a cyclic learning rate schedule.

Table 2: **Summary of training details.**

Dataset	Architecture	Hyperparameters	Ground-truth Retraining
UCI Concrete 896 training examples 103 query examples	MLP - 4 Layers (128, 128, 128) Hidden Units ReLU activation	SGD w/ momentum Learning rate: 3×10^{-2} Weight decay: 10^{-5} Momentum: 0.9 Batch size: 32 Epochs: 20	Repeats: 100 Masks: 100 Total: 10,000
UCI Parkinsons 5,280 training examples 100 query examples	MLP - 4 Layers (128, 128, 128) Hidden Units ReLU activation	SGD w/ momentum Learning rate: 10^{-2} Weight decay: 3×10^{-5} Momentum: 0.9 Batch size: 32 Epochs: 20	Repeats: 100 Masks: 100 Total: 10,000
MNIST (Subset) 6,144 training examples 100 query examples	MLP - 4 Layers (512, 256, 128) Hidden Units ReLU activation	SGD w/ momentum Learning rate: 3×10^{-2} Weight decay: 10^{-3} Momentum: 0.9 Batch size: 64 Epochs: 20	Repeats: 50 Masks: 100 Total: 5,000
FashionMNIST (Subset) 6,144 training examples 100 query examples	MLP - 4 Layers (512, 256, 128) Hidden Units ReLU activation	SGD w/ momentum Learning rate: 3×10^{-2} Weight decay: 10^{-3} Momentum: 0.9 Batch size: 64 Epochs: 20	Repeats: 50 Masks: 100 Total: 5,000
CIFAR-10 (Subset) 3,072 training examples 100 query examples	ResNet-9 [78]	SGD w/ momentum Peak learning rate: 0.4 Weight decay: 10^{-3} Momentum: 0.9 Batch size: 512 Epochs: 25	Repeats: 50 Masks: 100 Total: 5,000
WikiText-2 4,656 training sequences 512 sequence length 100 query sequences	GPT-2 [81]	AdamW Learning rate: 3×10^{-5} Weight decay: 10^{-2} Batch size: 8 Epochs: 3	Repeats: 10 Masks: 100 Total: 1,000
FashionMNIST-N 6,144 training examples 100 query examples 30% of the dataset randomly relabeled	MLP - 4 Layers (512, 256, 128) Hidden Units ReLU activation	SGD w/ momentum Learning rate: 10^{-2} Weight decay: 3×10^{-5} Momentum: 0.9 Batch size: 64 Epochs: 3	Repeats: 50 Masks: 100 Total: 5,000

Table 3 shows the details for the various TDA algorithms we use in our experiments.

For EKFAC-IF, we use the same damping as the corresponding ASTRA-IF for comparability. SOURCE provides a natural value for damping from its derivation: $\lambda_\ell = 1/\overline{\eta}_\ell K_\ell$ [19], allowing us to sidestep tuning the damping term. For both EKFAC-IF and ASTRA-IF, we use the damping value implied by SOURCE for comparability between influence functions and SOURCE: we take the average λ_ℓ implied by SOURCE for each segment and weigh them by the total iterations K_ℓ in segment ℓ as our influence functions damping value.

EKFAC-IF, EKFAC-SOURCE, ASTRA-IF, and ASTRA-SOURCE all compute influence on the same set of layers. For MLP architectures, we compute influence on all layers. For ResNet-9, we

https://doi.org/10.52202/085713-3408

compute influence on MLP and convolution layers. For GPT-2, we compute influence only on MLP layers in line with Grosse et al. [7].

Table 3: **Summary of TDA Algorithm Details**

Dataset	ASTRA-IF Details	ASTRA-SOURCE Details
UCI Concrete	We run ASTRA for 200 iterations, and use GGN damping factor $\lambda = 0.0017$, preconditioner damping factor $\tilde{\lambda} = 0.0017$, learning rate $\alpha = 0.1\lambda$, batch-size 256, SGD w/ 0.9 momentum, and decay the learning rate by 0.5 every 50 iterations.	We use 3 segments. For each segment, we run ASTRA for 200 iterations, and use preconditioner damping factor equal to the GGN damping factor $\tilde{\lambda}_\ell = \lambda_\ell = 1/\overline{\eta}_\ell K_\ell$, learning rate $\alpha_\ell = 0.1\lambda_\ell$, batch-size 256, SGD w/ 0.9 momentum, and decay the learning rate by 0.5 every 50 iterations.
UCI Parkinsons	We run ASTRA for 200 iterations, and use GGN damping factor $\lambda = 0.00091$, preconditioner damping factor $\tilde{\lambda} = 0.00091$, learning rate $\alpha = 0.1\lambda$, batch-size 256, SGD w/ 0.9 momentum, and decay the learning rate by 0.5 every 50 iterations.	We use 3 segments. For each segment, we run ASTRA for 200 iterations, and use preconditioner damping factor equal to the GGN damping factor $\tilde{\lambda}_\ell = \lambda_\ell = 1/\overline{\eta}_\ell K_\ell$, learning rate $\alpha_\ell = 0.1\lambda_\ell$, batch-size 256, SGD w/ 0.9 momentum, and decay the learning rate by 0.5 every 50 iterations.
MNIST	We run ASTRA for 200 iterations, and use GGN damping factor $\lambda = 0.0052$, preconditioner damping factor $\tilde{\lambda} = 0.0052$, learning rate $\alpha = 0.1\lambda$, batch-size 256, SGD w/ 0.9 momentum, and decay the learning rate by 0.5 every 50 iterations.	We use 3 segments. For each segment, we run ASTRA for 200 iterations, and use preconditioner damping factor equal to the GGN damping factor $\tilde{\lambda}_\ell = \lambda_\ell = 1/\overline{\eta}_\ell K_\ell$, learning rate $\alpha_\ell = 0.1\lambda_\ell$, batch-size 256, SGD w/ 0.9 momentum, and decay the learning rate by 0.5 every 50 iterations.
FashionMNIST	We run ASTRA for 200 iterations, and use GGN damping factor $\lambda = 0.0052$, preconditioner damping factor $\tilde{\lambda} = 0.0052$, learning rate $\alpha = 0.1\lambda$, batch-size 256, SGD w/ 0.9 momentum, and decay the learning rate by 0.5 every 50 iterations.	We use 3 segments. For each segment, we run ASTRA for 200 iterations, and use preconditioner damping factor equal to the GGN damping factor $\tilde{\lambda}_\ell = \lambda_\ell = 1/\overline{\eta}_\ell K_\ell$, learning rate $\alpha_\ell = 0.1\lambda_\ell$, batch-size 256, SGD w/ 0.9 momentum, and decay the learning rate by 0.5 every 50 iterations.
CIFAR-10	We run ASTRA for 200 iterations, and use GGN damping factor $\lambda = 0.014$, preconditioner damping factor $\tilde{\lambda} = 0.014$, learning rate $\alpha = 0.01\lambda$, batch-size 128, SGD w/ 0.9 momentum, and decay the learning rate by 0.5 every 50 iterations.	We use 3 segments. For each segment, we run ASTRA for 200 iterations, and use preconditioner damping factor equal to the GGN damping factor $\tilde{\lambda}_\ell = \lambda_\ell = 1/\overline{\eta}_\ell K_\ell$, learning rate $\alpha_\ell = 0.01\lambda_\ell$, batch-size 128, SGD w/ 0.9 momentum, and decay the learning rate by 0.5 every 50 iterations.
WikiText-2	We run ASTRA for 300 iterations, and use GGN damping factor $\lambda = 0.011$, preconditioner damping factor $\tilde{\lambda} = 0.011$, learning rate $\alpha = \lambda$, batch-size 16, SGD w/ 0.9 momentum, and decay the learning rate by 0.9 every 100 iterations.	We use 3 segments. For each segment, we run ASTRA for 300 iterations, and use preconditioner damping factor equal to the GGN damping factor $\tilde{\lambda}_\ell = \lambda_\ell = 1/\overline{\eta}_\ell K_\ell$, learning rate $\alpha_\ell = \lambda_\ell$, batch-size 8, SGD w/ 0.9 momentum, and decay the learning rate by 0.9 every 100 iterations.
FashionMNIST-N	We run ASTRA for 200 iterations, and use GGN damping factor $\lambda = 0.10$, preconditioner damping factor $\tilde{\lambda} = 0.10$, learning rate $\alpha = 0.1\lambda$, batch-size 256, SGD w/ 0.9 momentum, and decay the learning rate by 0.5 every 50 iterations.	We use 3 segments. For each segment, we run ASTRA for 200 iterations, and use preconditioner damping factor equal to the GGN damping factor $\tilde{\lambda}_\ell = \lambda_\ell = 1/\overline{\eta}_\ell K_\ell$, learning rate $\alpha_\ell = 0.1\lambda_\ell$, batch-size 256, SGD w/ 0.9 momentum, and decay the learning rate by 0.5 every 50 iterations.

Other Baselines In Figure 2, in addition to the EKFAC-based baselines, we also compare ASTRA against TracIn [83] and TRAK [22]. We use the same checkpoints for TracIn as SOURCE for comparability. TRAK applies random projections to reduce the computation and memory footprint – for MNIST, FashionMNIST(-N) and UCI Parkinsons, we use projection dimensions of 4,096 and for UCI Concrete, we use projection dimension of 512, consistent with [19]. For CIFAR-10, we do a hyperparameter sweep among [128, 256, 512, 1024, 2048, 4096, 8192, 20480] for the best hyperparameter (512). We omit TRAK's results on GPT-2 due to lack of publicly available implementations.

Training Curve Details In Figure 3, we compare ASTRA-IF and SNI on an arbitrary $\boldsymbol{z}_q$, using 10 seeds and report mean values along with shaded regions representing 1 standard error. To compute both ASTRA-IF and the baseline SNI in Figure 3, we conduct a hyperparameter sweep for the learning rate from 10^0 to 10^{-5} in increments of one order of magnitude with momentum set to zero, and use the best learning rate based on the average training objective over the last ten iterations. Both SNI and ASTRA-IF use the same damping values and batch sizes for Figure 3, as listed in Table 3 for comparability. The learning rates used were 10^{-2} for all settings for ASTRA-IF, and the best learning rates for SNI were $1, 0.1, 0.01, 0.1$ for MNIST, FashionMNIST, CIFAR-10, and UCI Concrete respectively.

Eigendecomposition Details In Figure 4, we compare the performance of influence functions computed with ASTRA and SNI after projecting to various subspaces. This experiment uses a smaller damping of 10^{-4} for both MNIST and FashionMNIST to be able to discern the impact of directions of low curvature on influence functions performance. We use the same batch sizes as disclosed in Table 3 and no momentum. For ASTRA-IF, we used a learning rate of 10^{-4} for both settings. For the baseline SNI, the learning rates were 0.1 and 0.01 respectively for MNIST and FashionMNIST, which we found to give the strongest LDS for the subspace corresponding to S_5.

Compute Resources A shared cluster was used (both internal and external), consisting of a mix of A6000 (48GB), A100 (80GB) and H100 (80GB) GPUs, which were used to conduct all experiments. Computing the ground truth of the LDS experiments in Figure 1 is an expensive component of the overall compute to replicate the experiments. For the most expensive setting, fine-tuning GPT-2 with WikiText-2, computing ground-truth costs at most 500 hours of compute with the resources listed above. Similarly, running EKFAC-IF, EKFAC-SOURCE, ASTRA-IF and ASTRA-SOURCE for the GPT-2 setting is the most expensive of all the settings. Running ASTRA-SOURCE for the GPT-2 setting costs at most 40 hours with the compute resources listed above.

Statistical Significance For LDS experiments, we provide error bars indicating 1 standard error for all one-model predictions estimated with 10 seeds. For the training curve (Figure 3) and the eigendecomposition experiments (Figure 4), we show 1 standard error estimated with 10 seeds. The standard error is computed by taking the standard deviation s of the computed metric, divided by $\sqrt{10}$.

Assets We use `pytorch` 2.5.0 and the publicly available `kronfluence` package for our experiments, which can be found at `https://github.com/pomonam/kronfluence`.

G Implications of Truncated Neumann Series

In Section 2.1, we introduced the connection between NI and the iHVP approximation. Iterative iHVP approximation algorithms like LiSSA [23] usually requires a large number of iterations to achieve good iHVP approximations [7, 17]. In practice, the number of iterations in the algorithm is usually set with the assumption that the approximation converges within the iterations (e.g., 5000 in [17]). While a large number of iterations makes convergence likely, it is not a guarantee, and raising the number of iterations implies additional computational cost. We demonstrate that using fewer iterations and stopping before convergence can be interpreted as adding an additional damping term to the iHVP approximation. There is existing work noting that the truncation of Neumann series can be viewed as increased damping [29], but here we present it with the derivation technique found in

 https://doi.org/10.52202/085713-3408

[19]. We derive the following expression for the truncated Neumann series with J iterations:

$$\alpha \sum_{j=0}^{J-1} (\mathbf{I} - \alpha\mathbf{G} - \alpha\lambda\mathbf{I})^j = (\mathbf{G} + \lambda\mathbf{I})^{-1}(\mathbf{I} - (\mathbf{I} - \alpha\mathbf{G} - \alpha\lambda\mathbf{I})^J) \tag{37}$$

$$\approx (\mathbf{G} + \lambda\mathbf{I})^{-1}(\mathbf{I} - \exp(-\alpha J\mathbf{G} - \alpha J\lambda\mathbf{I})) \tag{38}$$

$$\approx (\mathbf{G} + \hat{\lambda}\mathbf{I})^{-1}, \tag{39}$$

where $\hat{\lambda} = \lambda + \alpha^{-1}J^{-1}$. Equation 37 and Equation 38 utilize the finite series and the matrix exponential definition, respectively. Given that the matrices in Equation 38 commute, we can express it as the following matrix function:

$$F(\sigma) \coloneqq \frac{1 - \exp(-\alpha J\sigma - \alpha J\lambda)}{\sigma + \lambda}. \tag{40}$$

Let $\mathbf{G} = \mathbf{Q}\mathbf{D}\mathbf{Q}^\top$ be the eigendecomposition, where σ_j denotes the jth eigenvalue of $\mathbf{G}$. The expression in Equation 38 can be interpreted as applying the function $F(\sigma)$ to each eigenvalue σ of $\mathbf{G}$. In high-curvature directions, this term asymptotically approaches $1/\sigma + \lambda$, whereas in low-curvature directions with small values of λ, it tends toward αJ.

The qualitative behavior of the function F can be captured by F_{inv}, defined as:

$$F_{\text{inv}}(\sigma) \coloneqq \frac{1}{\sigma + \lambda + \alpha^{-1}J^{-1}} \tag{41}$$

Applying F_{inv} to the Hessian results in approximating Equation 39 with a modified damping term $\hat{\lambda} \coloneqq \lambda + 1/\alpha J$. Hence, the truncated version can be interpreted as incorporating a larger damping term, by $1/\alpha J$. The implicitly larger damping term affects the iterative updates and adds additional difficulty to tuning the hyperparameters of iterative iHVP solvers. In ASTRA, we leverage the EKFAC approximation as a preconditioner for the iHVP approximation to improve the conditioning of the problem, which mitigates this issue.

H Limitations & Broader Impact

Limitations We have addressed the problem of computing accurate iHVPs for TDA. As we outlined in Section 2.2, in addition to the computing the iHVP, a large component of the cost in computing influence functions for both EKFAC-IF and ASTRA-IF when scanning the dataset for highly influential examples is taking the dot product with all the training example gradients, which is an orthogonal but important issue that we do not address. We also noted in our experiments that in some cases, the bottleneck for influence function performance in TDA is not an inaccurate iHVP, but other factors such as violating the fundamental assumptions involved in the influence functions derivation. In these cases, the benefits from an improved iHVP approximation may not materialize until other bottlenecks are resolved. For example, the FashionMNIST experiments show a large increase in performance after ensembling relative to the 1-model scores obtained by an improved iHVP approximation, which suggests that stochasticity in the training procedure may be the dominant factor hindering TDA performance. Finally, for ASTRA-SOURCE, we present a way in which we can apply the iHVP computation, but we leave to future work to explore various ways to compute the average GGN $\overline{\mathbf{G}}_\ell$, which may further improve performance.

Broader impact Our work improves the accuracy of the iHVP approximation for use in TDA. The algorithmic improvements we present do not have direct societal impact. However, improved TDA can provide insights into the relation between training data and model behavior. From this perspective, our work has similar broader impact to other work in TDA, sharing similar potential benefits – namely, enhanced interpretability, transparency, and fairness in AI systems and share similar risks. Specifically, advancing TDA can help us understand models through the lens of training data, which can be applied to many domains such as building more equitable and transparent machine learning systems [12, 13], and investigating questions of intellectual property and copyright [11, 14, 15]. On the flip side, TDA can also be used to maliciously to craft data poisoning attacks and could result in models with undesirable behavior. It is important that TDA algorithms are improved with mitigation of the risks.