Efficient Bilevel Optimization with KFAC-Based Hypergradients

Disen Liao
University of Waterloo
Vector Institute

Felix Dangel
Vector Institute

Yaoliang Yu
University of Waterloo
Vector Institute

Abstract

Bilevel optimization (BO) is widely applicable to many machine learning problems. Scaling BO, however, requires repeatedly computing hypergradients, which involves solving inverse Hessian-vector products (IHVPs). In practice, these operations are often approximated using crude surrogates such as one-step gradient unrolling or identity/short Neumann expansions, which discard curvature information. We build on implicit function theorem-based algorithms and propose to incorporate Kronecker-factored approximate curvature (KFAC), yielding curvature-aware hypergradients with a better performance efficiency trade-off than Conjugate Gradient (CG) or Neumann methods and consistently outperforming unrolling. We evaluate this approach across diverse tasks, including meta-learning and AI safety problems. On models up to BERT, we show that curvature information is valuable at scale, and KFAC can provide it with only modest memory and runtime overhead. Our implementation is available at `https://github.com/liaodisen/NeuralBo`.

1 INTRODUCTION

Bilevel optimization (BO), originally studied in economics (von Stackelberg, 1934), models interactions between two decision-makers: a leader who acts first and a follower who optimizes in response. This leader-follower structure captures scenarios where one optimization problem depends on another. In machine learning, BO provides a unifying framework for diverse tasks: in meta-learning to acquire transferable inductive biases (Bertinetto et al., 2019; Rajeswaran et al., 2019), in hyper-parameter and data optimization to adapt learning pipelines (Franceschi et al., 2018; Shu et al., 2019; Feng et al., 2024; Hu et al., 2023; Lorraine et al., 2020), and in neural architecture search to guide model design (Liu et al., 2019). BO also plays a central role in AI safety, covering adversarial problems such as data poisoning (Lu et al., 2022; Radiya-Dixit et al., 2022; Huang et al., 2020) and unlearnable examples (Huang et al., 2021; Liu et al., 2024). BO's appeal lies in its breadth: a single framework to reason about dependencies across learning, optimization, and safety.

Broadly speaking, algorithms for BO fall into three families: *gradient unrolling* (GU), *implicit function theorem* (IFT), and *value-function* (VF). GU methods directly differentiate through a sequence of lower-level optimization steps (Franceschi et al., 2017; Shaban et al., 2019; Shen et al., 2024). While conceptually simple, they demand substantial memory and computation, and short unrolling horizons introduce truncation bias. IFT methods avoid explicit unrolling by leveraging the implicit function theorem to characterize the sensitivity of the lower-level solution (Lorraine et al., 2020; Grazzi et al., 2020; Ji et al., 2021; Choe et al., 2023a). These approaches circumvent the need to store the intermediate iterates and are therefore more memory-efficient than unrolling, but hinge on computing inverse Hessian-vector products (IHVPs), which becomes a major computational bottleneck in high dimensions. VF methods instead sidestep second-order computations by reformulating BO as a constrained single-level problem (Liu et al., 2021; Kwon et al., 2023; Giovannelli et al., 2025). Despite avoiding explicit second-order operations, their practical scalability remains less clear: solving the resulting single-level formulation often introduces additional optimization variables and constraints, which can increase complexity and make them less competitive in large-scale stochastic learning settings (Zhang et al., 2024).

Among the three, IFT-based approaches have recently gained popularity due to their ability to capture accurate hypergradients (Ghadimi & Wang, 2018; Ji et al., 2021). Several efforts have pushed them toward larger-scale applications (Lorraine et al., 2020; Choe et al., 2023b,a). Since exact inversion of the Hessian is infeasible for modern models, practitioners often resort to crude approximations. A common strategy truncates the Neumann series after a few

Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300. Copyright 2026 by the author(s).

terms (Lorraine et al., 2020; Choe et al., 2023b). However, full convergence is usually computationally out of reach at this problem size, and the neural net's inner problem is frequently ill-conditioned (Sagun et al., 2016), making short truncations inaccurate. Others go further by replacing the Hessian with the identity matrix (Choe et al., 2023a; Hong et al., 2023; Liu et al., 2019), effectively collapsing IFT into a one-step unrolling method and discarding curvature information altogether. As a result, existing methods either degrade under poor conditioning or rely on overly simplistic surrogates that produce inaccurate hypergradients.

Our work builds squarely within the IFT territory, but seeks to address its central bottleneck: scalable IHVPs. To this end, we explore Kronecker-Factored Approximate Curvature (KFAC, Martens & Grosse, 2015), which provides an efficient block-diagonal curvature approximation using the (uncentered) covariance of layer inputs and gradients. Although rarely used in BO, recent large-scale results on influence functions in language models (Grosse et al., 2023) suggests that KFAC offers a practical middle ground between crude approximations and iterative curvature-aware methods. In contrast to influence functions, each outer step of IFT-based BO needs an IHVP, each of which is often obtained by iterative solvers via many HVPs. KFAC replaces these repeated HVPs with a structured curvature estimate, enabling efficient approximate inversion.

We propose a KFAC-based hypergradient method for BO that directly plugs into existing algorithms. It provides rich curvature information at low computational cost—illustrated in Figure 1 where KFAC reaches lower test loss with less per-iteration time than iterative IHVP approximations, with only modest memory overhead over HVP-based methods—and is effective on large-scale models across diverse applications. Our main contributions are:

- We propose using inverse KFAC-vector products (IKVPs) for hypergradient computation in IFT-based BO and use them to replace IHVPs in existing methods, yielding a scalable curvature-aware approximation that improves wall-clock efficiency, convergence, and stability under small batch sizes.
- We scale data hypercleaning to BERT and show that incorporating curvature improves large-scale performance with only modest time and memory overhead.
- We demonstrate broad applicability by applying KFAC (including an empirical variant) to two meta-learning tasks and two ML safety problems, consistently improving performance across domains.

2 BACKGROUND

Problem Setup. We formulate BO as a nested problem with an outer variable $\boldsymbol{\lambda} \in \mathbb{R}^m$ (e.g., hyperparameters) and

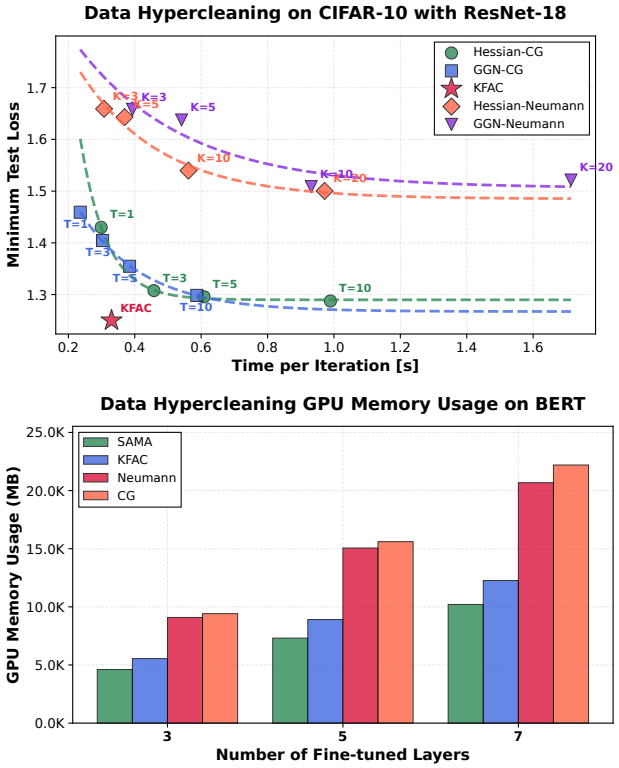


Figure 1: **Top**: KFAC balances accuracy and efficiency better than Hessian/GGN-based IHVP approximations. T is the number of CG iterations and K the number of truncated Neumann terms. **Bottom**: On BERT, KFAC incurs only a small memory overhead compared to the curvature-free SAMA, while using less memory than CG/Neumann.

an inner variable $\boldsymbol{\theta} \in \mathbb{R}^d$ (e.g., model parameters). The goal is to minimize an outer objective $\mathcal{J}_{\text{out}}$ that depends on the optimal solution of the inner problem $\mathcal{J}_{\text{in}}$:

$$\min_{\boldsymbol{\lambda} \in \mathbb{R}^m} \Phi(\boldsymbol{\lambda}) := \mathcal{J}_{\text{out}}\big(\boldsymbol{\lambda}, \boldsymbol{\theta}^\star(\boldsymbol{\lambda})\big)$$
$$\text{s.t.} \quad \boldsymbol{\theta}^\star(\boldsymbol{\lambda}) = \arg\min_{\boldsymbol{\theta} \in \mathbb{R}^d} \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}).$$

Here, the outer objective $\mathcal{J}_{\text{out}}$ depends on the inner problem's optimal solution $\boldsymbol{\theta}^*$. In machine learning, the inner problem typically corresponds to training a neural network $\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta}$, which processes an input $\boldsymbol{x}$ into a prediction $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta}) \in \mathbb{R}^C$. The prediction is compared to the ground truth $\boldsymbol{y}$ with a criterion function $c : \boldsymbol{f}, \boldsymbol{y} \mapsto c(\boldsymbol{f}, \boldsymbol{y}) \in \mathbb{R}$. Finally, we form the empirical risk by evaluating, then averaging, the risk over a dataset $\mathcal{D}$: $\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}) = {}^1\!/_N \sum_n c(\boldsymbol{f}(\boldsymbol{x}_n, \boldsymbol{\theta}), \boldsymbol{y}_n) := {}^1\!/_N \sum_n \ell_n(\boldsymbol{\theta})$ with the per-sample loss $\ell_n = c_n \circ \boldsymbol{f}_n$ with $c_n = c|_{\boldsymbol{y}=\boldsymbol{y}_n}, \boldsymbol{f}_n = \boldsymbol{f}|_{\boldsymbol{x}=\boldsymbol{x}_n}$. One simple example is $\mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}) = \mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}) = {}^1\!/_N \sum_n \ell_n(\boldsymbol{\theta})$, where we may identify $\boldsymbol{\lambda} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}$.

Hypergradient. To optimize the outer problem, we require the gradient of $\Phi(\boldsymbol{\lambda})$ with respect to $\boldsymbol{\lambda}$. By the im-

plicit function theorem (Steven G. Krantz, 2002), the hypergradient is:

$$\nabla\Phi(\boldsymbol{\lambda}) = \nabla_1 \mathcal{J}_{\text{out}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^\star(\boldsymbol{\lambda})) - \nabla^2_{12}\mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^\star(\boldsymbol{\lambda}))\boldsymbol{v}^\star. \quad (1)$$

Here ∇_1 denotes the gradient w.r.t the outer variable $\boldsymbol{\lambda}$, and ∇_2 denotes the gradient w.r.t the inner variable $\boldsymbol{\theta}$. The subscript $_{12}$ indicates mixed second derivatives, while $_{22}$ denotes the Hessian w.r.t $\boldsymbol{\theta}$ and $\boldsymbol{v}^\star$ is the solution of the linear system

$$\nabla^2_{22}\mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^\star(\boldsymbol{\lambda}))\boldsymbol{v} = \nabla_2 \mathcal{J}_{\text{out}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^\star(\boldsymbol{\lambda})). \quad (2)$$

In practice, $\boldsymbol{v}^\star$ corresponds to an IHVP, which is the main bottleneck in IFT-based bilevel optimization methods. The IFT expression above, Equation (2), holds under standard smoothness and non-degeneracy assumptions: the inner objective must be twice continuously differentiable, and its Hessian at the optimum $\boldsymbol{\theta}^\star(\boldsymbol{\lambda})$ must be non-singular, which implies local strong convexity (Steven G. Krantz, 2002). In neural networks, however, the inner optimization is rarely carried to exact optimality because of the cost, so IFT-based hypergradients are usually computed approximately at the current iterate rather than at the true inner solution.

IHVP approximations. Let $\boldsymbol{H} \coloneqq \nabla^2_{22}\mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^\star(\boldsymbol{\lambda}))$ denote the inner problem's Hessian at $\boldsymbol{\theta}^\star$, and let $\boldsymbol{b} \coloneqq \nabla_2 \mathcal{J}_{\text{out}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^\star(\boldsymbol{\lambda}))$. Computing the hypergradient via Equation (1) requires solving the linear system $\boldsymbol{H}\boldsymbol{v} = \boldsymbol{b}$, whose exact solution corresponds to the IHVP $\boldsymbol{v} = \boldsymbol{H}^{-1}\boldsymbol{b}$. Since explicitly forming or inverting $\boldsymbol{H}$ is infeasible in high dimensions, iterative methods are commonly used. They iteratively solve the linear system and require one Hessian-vector product (HVP) per iteration, but their convergence is guaranteed only when $\boldsymbol{H}$ is positive semi-definite. An alternative view is to solve the quadratic problem

$$\boldsymbol{v}^* = \arg\min_{\boldsymbol{v}} \frac{1}{2}\boldsymbol{v}^\top \boldsymbol{H}\boldsymbol{v} - \boldsymbol{b}^\top \boldsymbol{v}\,,$$

on which iterative solvers such as conjugate gradient (CG) or gradient descent can be applied; this formulation underlies algorithms such as AmIGO (Arbel & Mairal, 2022; Dagréou et al., 2022). Other works use the Neumann series

$$\boldsymbol{H}^{-1} = \sum_{k=0}^{\infty} (\boldsymbol{I} - \boldsymbol{H})^k\,,$$

and truncate it after K terms to obtain an approximate IHVP with K HVPs (Chen et al., 2021; Ji et al., 2021), though accuracy deteriorates under ill-conditioning. Finally, one-step unrolling methods such as DARTS (Liu et al., 2019) and SAMA (Choe et al., 2023a) approximate $\boldsymbol{H}$ by the identity (or equivalently $K = 0$), yielding $\boldsymbol{v} \approx \boldsymbol{b}$, ignoring curvature in exchange for efficiency.

Hessian, GGN, and Fisher. Consider the inner loss $\mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}) = \mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta})$. To calculate the Hessian, applying the chain rule to ℓ_n, the empirical risk's Hessian decomposes into the generalized Gauss-Newton (GGN) matrix $\boldsymbol{G}(\boldsymbol{\theta})$ (Schraudolph, 2002) and a residual term $\boldsymbol{R}(\boldsymbol{\theta})$:

$$\underbrace{\nabla^2_{\boldsymbol{\theta}}\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta})}_{\coloneqq \boldsymbol{H}(\boldsymbol{\theta})} = \underbrace{\frac{1}{N}\sum_n [\boldsymbol{J}_{\boldsymbol{\theta}}\boldsymbol{f}_n]^\top (\nabla^2_{\boldsymbol{f}} c_n)[\boldsymbol{J}_{\boldsymbol{\theta}}\boldsymbol{f}_n]}_{\coloneqq \boldsymbol{G}(\boldsymbol{\theta})} + \underbrace{\frac{1}{N}\sum_n \sum_{i=1}^{C} [\nabla_{\boldsymbol{f}} c_n]_i \nabla^2_{\boldsymbol{\theta}}[\boldsymbol{f}_n(\boldsymbol{\theta})]_i}_{\coloneqq \boldsymbol{R}(\boldsymbol{\theta})}\,.$$

Here, $\boldsymbol{J}_{\boldsymbol{\theta}}\boldsymbol{f}_n \in \mathbb{R}^{C\times d}$ is the Jacobian containing the partial derivatives of $\boldsymbol{f}_n$ w.r.t. $\boldsymbol{\theta}$. The GGN is often used instead of the Hessian as it is positive semi-definite. Moreover, for common losses such as mean-squared error and softmax cross-entropy, the GGN coincides with the Fisher information matrix (FIM, Martens, 2020)

$$\boldsymbol{F}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}, \boldsymbol{y}\sim p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})}\left[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})\, \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})^\top\right]$$

where $p_{\text{data}}(\boldsymbol{x})$ is the input distribution and $p_\theta(\boldsymbol{y}|\boldsymbol{x})$ is the model's predictive distribution. The Fisher view underlies natural gradient descent (Amari, 1998) and motivates KFAC as a preconditioner for neural network optimization (Martens & Grosse, 2015). While both the Hessian and GGN are prohibitively expensive to store, automatic differentiation allows cheap evaluation of matrix-vector products at the same complexity as computing the gradient (Pearlmutter, 1994; Schraudolph, 2002; Dagréou et al., 2024). However, this computation is still expensive for BO as it requires applying an inverse at every outer iteration.

3 METHOD

Hypergradient computation in BO requires IHVPs, which are expensive or unstable with iterative solvers. We replace them with IKVPs, enabled by a layer-wise Kronecker-factored approximation of the GGN/Fisher. Section 3.1 recalls KFAC, Section 3.2 shows how to integrate IKVPs into BO, and Section 3.3 provides a diagnostic study that motivates and illustrates its robustness under ill-conditioning.

3.1 KFAC: a Layer-wise Curvature Approximation

The Hessian $\boldsymbol{H}$ or GGN $\boldsymbol{G}$ is $d \times d$, which is prohibitively large for modern architectures. KFAC reduces this cost by adopting a layer-wise approximation, treating each layer independently and ignoring cross-layer interactions. This yields a block-diagonal structure, where each block is further approximated with a Kronecker product. This structure is particularly attractive for BO, since hypergradients require repeated curvature inverses: KFAC replaces expensive iterative solvers with direct structured inverses, enabling curvature information at low cost.

KFAC for a single fully-connected layer. Here we consider the case for a fully-connected layer inside a neural net. The layer's weights are $\boldsymbol{W} \in \mathbb{R}^{d_1 \times d_2}$ (we omit biases for simplicity) and it processes an input $\boldsymbol{a}_n \in \mathbb{R}^{d_2}$ into an output $\boldsymbol{z}_n = \boldsymbol{W}\boldsymbol{a}_n \in \mathbb{R}^{d_1}$. Further, let $\operatorname{vec}(\boldsymbol{W}) \in \mathbb{R}^{d_1 d_2}$ denote the flattened weights from stacking the rows into a vector. We can then write down the GGN matrix $\boldsymbol{G}(\operatorname{vec}\boldsymbol{W})$ w.r.t. to the layer's parameters analytically. To do so, we use the chain rule ($\boldsymbol{J}_{\operatorname{vec}\boldsymbol{W}}\boldsymbol{f}_n = (\boldsymbol{J}_{\boldsymbol{z}_n}\boldsymbol{f}_n)(\boldsymbol{J}_{\operatorname{vec}\boldsymbol{W}}\boldsymbol{z}_n)$) with analytical output-parameter Jacobian $\boldsymbol{J}_{\operatorname{vec}\boldsymbol{W}}\boldsymbol{z}_n = \boldsymbol{I}_{d_1} \otimes \boldsymbol{a}_n^\top$ (Dangel et al., 2020a), yielding that each datum contributes a single Kronecker product:

$$\begin{aligned}\boldsymbol{G}(\operatorname{vec}\boldsymbol{W}) &= \frac{1}{N}\sum_n (\boldsymbol{J}_{\boldsymbol{z}_n}\boldsymbol{f}_n)^\top (\nabla^2_{\boldsymbol{f}} c_n)(\boldsymbol{J}_{\boldsymbol{z}_n}\boldsymbol{f}_n) \otimes \boldsymbol{a}_n\boldsymbol{a}_n^\top \\ &= \hat{\mathbb{E}}\left[(\boldsymbol{J}_{\boldsymbol{z}}\boldsymbol{f})^\top (\nabla^2_{\boldsymbol{f}} c)(\boldsymbol{J}_{\boldsymbol{z}}\boldsymbol{f}) \otimes \boldsymbol{a}\boldsymbol{a}^\top\right]\end{aligned}$$

where we use the notation $\hat{\mathbb{E}}[\bullet] = {}^1\!/_N \sum_n \bullet_n$ to indicate the expectation over the dataset. To obtain a single Kronecker product, KFAC makes the expectation approximation $\hat{\mathbb{E}}[\bullet \otimes \star] \approx \hat{\mathbb{E}}[\bullet] \otimes \hat{\mathbb{E}}[\star]$. In summary, this yields the following approximation of the GGN:

$$\boldsymbol{G}(\operatorname{vec}\boldsymbol{W}) \approx \boldsymbol{B}_{\text{KFAC}} \otimes \boldsymbol{A}_{\text{KFAC}}$$

where

$$\begin{aligned}\boldsymbol{B}_{\text{KFAC}} &= \frac{1}{N}\sum_n (\boldsymbol{J}_{\boldsymbol{z}_n}\boldsymbol{f}_n)^\top (\nabla^2_{\boldsymbol{f}} c_n)(\boldsymbol{J}_{\boldsymbol{z}_n}\boldsymbol{f}_n), \\ \boldsymbol{A}_{\text{KFAC}} &= \frac{1}{N}\sum_n \boldsymbol{a}_n\boldsymbol{a}_n^\top.\end{aligned}$$

This structural approximation reduces the storage cost of $\boldsymbol{G}(\operatorname{vec}\boldsymbol{W}) \in \mathbb{R}^{d_1 d_2 \times d_1 d_2}$ to the much smaller Kronecker factors $\boldsymbol{A}_{\text{KFAC}} \in \mathbb{R}^{d_2 \times d_2}$ and $\boldsymbol{B}_{\text{KFAC}} \in \mathbb{R}^{d_1 \times d_1}$ for each layer. Inverting KFAC resorts to inverting the two Kronecker factors, since $(\boldsymbol{B} \otimes \boldsymbol{A})^{-1} = \boldsymbol{B}^{-1} \otimes \boldsymbol{A}^{-1}$. For a network with multiple layers, KFAC applies this construction layerwise and combines the resulting factors into a block-diagonal approximation of the full network's curvature.

Computational details. The (uncentered) covariance of the layer inputs, $\boldsymbol{A}_{\text{KFAC}}$, is easy to compute in a forward pass. For the first factor $\boldsymbol{B}_{\text{KFAC}}$, there exist multiple variants that differ in computational cost (see Dangel et al. 2025b for a detailed introduction). The most expensive one (Botev et al., 2017) employs a symmetric factorization of the criterion's Hessian into $\nabla^2_{\boldsymbol{f}} c_n = {}^1\!/_C \sum_{i=1}^C \boldsymbol{s}_{n,i}\boldsymbol{s}_{n,i}^\top$, then computes the covariance of the pseudo-gradients $\tilde{\boldsymbol{g}}_{n,i} \coloneqq (\boldsymbol{J}_{\boldsymbol{z}_n}\boldsymbol{f}_n)^\top \boldsymbol{s}_{n,i}$, i.e. $\boldsymbol{B}_{\text{KFAC}} = {}^1\!/_{NC} \sum_n \sum_i \tilde{\boldsymbol{g}}_{n,i}\tilde{\boldsymbol{g}}_{n,i}^\top$. This incurs C backpropagations per datum. We will follow the original approach of Martens & Grosse (2015), which introduces a randomization via $p(\boldsymbol{s}_n)$ such that $\nabla^2_{\boldsymbol{f}} c_n = \mathbb{E}_{\boldsymbol{s}_{n,i} \sim p(\boldsymbol{s}_n)}[\boldsymbol{s}_{n,i}\boldsymbol{s}_{n,i}^\top]$, then approximates the expectation with a Monte-Carlo estimate involving M samples, i.e. $\nabla^2_{\boldsymbol{f}} c_n \approx {}^1\!/_M \sum_i \boldsymbol{s}_{n,i}\boldsymbol{s}_{n,i}^\top$ where $\boldsymbol{s}_{n,i} \overset{\text{i.i.d.}}{\sim} p(\boldsymbol{s}_n)$. This reduces the number of backpropagations from C to M, and we will use $M = 1$ as is common practice. The distribution $p(\boldsymbol{s}_n)$ is determined by the probabilistic interpretation of the loss function (e.g., Gaussian for MSE; the explicit forms are given in Appendix A).

We also adopt the empirical variant of KFAC (KFAC EMP), which constructs $\boldsymbol{B}_{\text{KFAC}}$ from empirical, rather than sampled, gradients. Its main computational advantage is that it reuses the standard backward pass already needed for gradient computation, while Monte-Carlo GGN variants typically require an additional backward pass with sampled curvature probes. The empirical $\boldsymbol{B}_{\text{KFAC}}$ is estimated as

$$\boldsymbol{B}_{\text{KFAC}} = {}^1\!/_N \sum_n \boldsymbol{g}_n\boldsymbol{g}_n^\top, \qquad \boldsymbol{g}_n = \nabla_{\boldsymbol{z}_n} \ell_n(\boldsymbol{\theta}),$$

with $\boldsymbol{g}_n$ denoting the per-example gradient of the loss with respect to the layer pre-activations. The remaining components of KFAC, namely the Kronecker factorization, damping, and matrix inversion, are unchanged. Empirical Fisher approximations of this form are widely used in practical second-order optimization (George et al., 2018; Lin et al., 2024; Zhang et al., 2022). We compare KFAC EMP with standard KFAC in Section 5, and use KFAC EMP in all experiments in Section 6 due to the size of problems.

KFAC approximation accuracy. For linear networks with square loss, the Hessian, GGN, and KFAC are identical (Petersen et al., 2023), yielding exact natural-gradient updates and IHVPs. This exactness further extends to deep linear networks: despite nonlinearity in parameters, KFAC recovers the exact natural gradient up to a constant factor, and is equivalent to the block-diagonal GGN in this setting (Bernacchia et al., 2018). Beyond these regimes, KFAC provides a structured approximation to the GGN for deep nonlinear networks, preserving curvature information at layerwise granularity while remaining computattionally tractable. Moreover, in the infinite-width in the NTK regime (Jacot et al., 2018), approximate natural gradient descent via KFAC achieves the same convergence rate as exact gradient methods (Karakida & Osawa, 2020). Our work focuses on practical large-scale regimes where exact solvers are infeasible, and evaluates how effectively this approximation supports hypergradient computation in BO.

3.2 Replacing IHVP with IKVP

We focus on BO problems where hypergradients can be expressed, under the implicit function theorem, as in equation 1. Instead of approximating IHVP $\boldsymbol{H}^{-1}\boldsymbol{b}$, we follow the common practice of approximating $\boldsymbol{H}$ with the GGN matrix $\boldsymbol{G}$, and approximating $\boldsymbol{G}^{-1}\boldsymbol{b}$ instead.

To scale this computation, we further approximate $\boldsymbol{G}$ with a Kronecker-factored block-diagonal estimate $\hat{\boldsymbol{G}}$ obtained via KFAC. Suppose we would like to apply $\hat{\boldsymbol{G}}^{-1}\boldsymbol{v}$ for some parameter vector $\boldsymbol{v}$. Since $\hat{\boldsymbol{G}}$ is block diagonal, for simplicity, we can just consider a single layer case: for $\boldsymbol{W} \in \mathbb{R}^{d_1 \times d_2}$, let $\boldsymbol{A}_{\text{KFAC}} \in \mathbb{R}^{d_2 \times d_2}$ denote the uncentered covari-

ance matrix of the activation and $\boldsymbol{B}_{\text{KFAC}} \in \mathbb{R}^{d_1 \times d_1}$ denote the gradient covariances. By construction, the KFAC block takes the form $\hat{\boldsymbol{G}} = \boldsymbol{B}_{\text{KFAC}} \otimes \boldsymbol{A}_{\text{KFAC}}$. Given a reshaped vector $\boldsymbol{v} = \text{vec}(\boldsymbol{V})$, where $\boldsymbol{V}$ matches the shape of the layer's weights, the IKVP follows from Kronecker identities as

$$\hat{\boldsymbol{G}}^{-1}\boldsymbol{v} = (\boldsymbol{B}_{\text{KFAC}}^{-1} \otimes \boldsymbol{A}_{\text{KFAC}}^{-1})\boldsymbol{v} = \text{vec}\left(\boldsymbol{B}_{\text{KFAC}}^{-1}\boldsymbol{V}\boldsymbol{A}_{\text{KFAC}}^{-1}\right).$$

Thus, instead of applying the inverse of a full $d_1 d_2 \times d_1 d_2$ matrix, IKVP reduces to two smaller inversions and a sequence of matrix multiplications.

Computational Cost of IKVP. For each layer, inverting $\boldsymbol{A}_{\text{KFAC}}$ and $\boldsymbol{B}_{\text{KFAC}}$ requires $O(d_1^3 + d_2^3)$ operations. Once inverted, applying an IKVP to a vector costs $O(d_1^2 d_2 + d_1 d_2^2)$, corresponding to two matrix-matrix multiplications. Performing forward and backward passes is $O(B d_1 d_2)$, where B is the batch size, so this operation has similar complexity to backpropagation when d_1 and d_2 are similar to B.

Damping. Curvature approximations are known to be singular for neural nets (Sagun et al., 2016). To stabilize inversion, we use the factored Tikhonov damping, which adds scaled identity to each Kronecker factor, rather than to the full block, which preserves efficiency while improving conditioning. In our implementation, we adopt the *heuristic damping* strategy from Martens & Grosse (2015), where the relative scaling between the activation covariance $\boldsymbol{A}_{\text{KFAC}}$ and the gradient covariance $\boldsymbol{B}_{\text{KFAC}}$ is controlled by a parameter π derived from their traces. Concretely, the damping applied to $\boldsymbol{A}_{\text{KFAC}}$ and $\boldsymbol{B}_{\text{KFAC}}$ is

$$\boldsymbol{A}_{\text{KFAC}} \to \boldsymbol{A}_{\text{KFAC}} + \pi\sqrt{\lambda}\boldsymbol{I}, \quad \boldsymbol{B}_{\text{KFAC}} \to \boldsymbol{B}_{\text{KFAC}} + \tfrac{1}{\pi}\sqrt{\lambda}\boldsymbol{I},$$

with $\pi = \sqrt{d_2 \text{Tr}(\boldsymbol{A}_{\text{KFAC}}) / d_1 \text{Tr}(\boldsymbol{B}_{\text{KFAC}})}$ balancing the two factors. Heuristic damping has been shown to perform more robustly than naive Tikhonov damping (Martens & Grosse, 2015), while adding negligible computational overhead.

there exist many public source implementations of KFAC (Dangel et al., 2020b; Osawa et al., 2023; Grosse et al., 2023; Botev & Martens, 2022). We use the `curvlinops` package (Dangel et al., 2025a), which provides easy access to VPs products with KFAC and its damped inverse.

3.3 Diagnostic Study of IHVP Approximations

A key practical advantage of KFAC arises from its robustness to ill-conditioning in the inner-level curvature. To isolate this effect, we consider a controlled setting where the exact inverse Hessian is available and compare different IHVP approximations under increasing dimensionality.

We study linear regression with square loss, where both design matrix $\boldsymbol{X} \in \mathbb{R}^{d \times N}$ and targets $\boldsymbol{y} \in \mathbb{R}^N$ are generated with i.i.d. standard Gaussian entries. We fit $\boldsymbol{w} \in \mathbb{R}^d$ using the square loss. The Hessian is known in closed form and only depends on the data matrix:

$$\boldsymbol{H} = \frac{1}{N}\boldsymbol{X}\boldsymbol{X}^\top.$$

We vary the parameter dimension d while keeping the dataset size N to 100. In this setting, KFAC is an exact approximation of the Hessian (Petersen et al., 2023). As d grows relative to N, the Hessian becomes increasingly ill-conditioned and eventually rank-deficient, a regime commonly encountered in over-parameterized models.

We compare Monte-Carlo KFAC against truncated Neumann series and CG under fixed computational budgets. Accuracy is measured by the relative operator-norm error

$$\min_{\alpha} \|\alpha(\widehat{\boldsymbol{H}} + \lambda\boldsymbol{I})^{-1} - (\boldsymbol{H} + \lambda\boldsymbol{I})^{-1}\|_2 / \|(\boldsymbol{H} + \lambda\boldsymbol{I})^{-1}\|_2,$$

where α is searched to account for the step-size in training and $\lambda = 10^{-5}$ is the damping factor.

Table 1: Relative error for different IHVP approximations as the parameter dimension d increases. Neu-K denotes a truncated Neumann series with K terms. CG-T denotes conjugate gradient with T iterations. Identity corresponds to the identity approximation used in one-step unrolling.

Method	$d = 10$	$d = 100$	$d = 500$
KFAC	7.6×10^{-3}	7.6×10^{-3}	7.6×10^{-3}
Neu-3	1.1×10^{-1}	2.5×10^{-1}	8.0×10^{-1}
Neu-20	2.2×10^{-2}	4.5×10^{-2}	2.4×10^{-1}
Neu-50	1.0×10^{-4}	5.6×10^{-3}	1.4×10^{-1}
CG-3	1.1×10^{-3}	7.5×10^{-2}	3.0×10^{-1}
CG-5	6.3×10^{-4}	7.7×10^{-3}	2.0×10^{-1}
CG-10	6.3×10^{-4}	8.7×10^{-4}	7.8×10^{-2}
Identity ($\boldsymbol{H} = \boldsymbol{I}$)	1.7×10^{-1}	3.9×10^{-1}	4.9×10^{-1}

Table 1 highlights two consistent trends. First, while CG and Neumann solvers can achieve high accuracy at small d given sufficient iterations, their error degrades rapidly as d increases under fixed iteration budgets. This behavior is expected, as the convergence rate of iterative solvers depends on the Hessian's condition number (Bakushinsky & Kokurin, 2004). Second, KFAC maintains stable accuracy as d increases, achieving orders-of-magnitude lower error than budget-limited CG and Neumann solvers.

4 ENHANCING BO ALGORITHMS WITH KFAC

Here, we demonstrate that our KFAC-based IKVP solver can serve as a direct, drop-in replacement for the expensive IHVP solver used in existing IFT-based BO algorithms. Our method yields faster convergence (in wall-clock time) and improved performance, especially at larger scales.

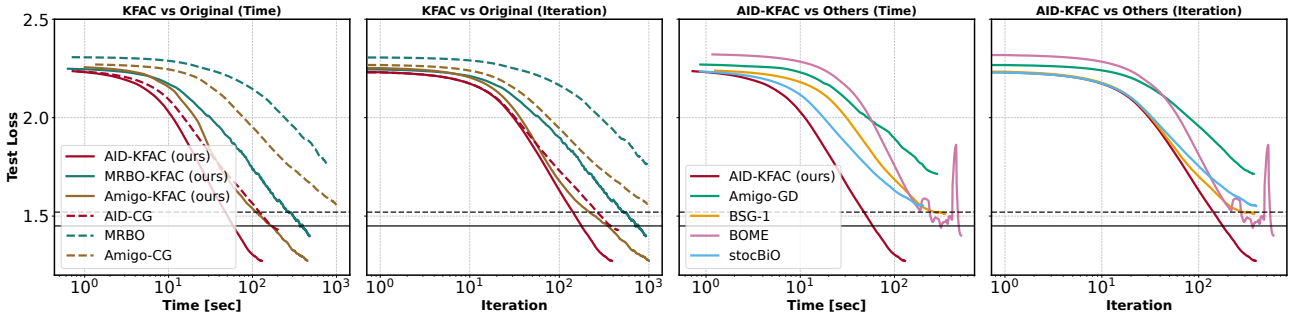


Figure 2: Results on data hypercleaning. **Left two panels:** existing BO algorithms improve when their IHVP solvers are replaced with IKVPs. **Right two panels:** KFAC-based methods compared against other non-IFT baselines, showing faster convergence in both time and iterations. Curves are truncated at the lowest test loss reached. Solid black line is trained on the validation dataset, while dashed blackline is trained on the validation dataset and the noisy dataset together.

4.1 Data Hypercleaning with ResNet-18

We first test our method on data hypercleaning, a standard and widely used benchmark for evaluating BO convergence (Liu et al., 2022; Ji et al., 2021; Dagréou et al., 2022). Our goal is to isolate the benefit of replacing the IHVP subroutine in IFT-based methods with our IKVP solver.

Given a corrupted training set $\mathcal{D}^{\text{tr}} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$ (random label noise) and a clean validation set $\mathcal{D}^{\text{val}}$, we learn per-example weights $\boldsymbol{\lambda} \in [0,1]^N$ by the bilevel objective:

$$\begin{aligned} \min_{\boldsymbol{\lambda},\boldsymbol{\theta}} \quad & \mathcal{L}^{\text{val}}(\boldsymbol{\theta}^\star), \\ \text{s.t.} \quad & \boldsymbol{\theta}^\star \in \arg\min_{\boldsymbol{\theta}} \left\{ \mathcal{L}^{\text{tr}}(\boldsymbol{\theta}, \boldsymbol{\lambda}) + \alpha \|\boldsymbol{\theta}\|^2 \right\}, \end{aligned} \tag{3}$$

where $\mathcal{L}^{\text{val}}(\boldsymbol{\theta}^\star)$ is the cross-entropy loss evaluated on the validation set. $\mathcal{L}^{\text{tr}}(\boldsymbol{\theta}, \boldsymbol{\lambda})$ is the weighted training loss, defined as: $\mathcal{L}^{\text{tr}}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \sum_{n=1}^N \sigma(\lambda_n) \ell_n(\boldsymbol{x}_n, y_n, \boldsymbol{\theta})$, where $\sigma(\lambda_n) = \text{Clip}(\lambda_n, [0,1])$ ensures that $\lambda_n \in [0,1]$. The training loss $\ell(\boldsymbol{x}_n, y_n, \boldsymbol{\theta})$ is also cross-entropy, and $\alpha\|\boldsymbol{\theta}\|^2$ adds an L_2 regularization to prevent overfitting. Since the GGN scales with the sample weight σ_n, in the Monte Carlo approximation, we rescale each pseudo-loss by $\sqrt{\sigma_n}$ for unbiased curvature estimation (details in Appendix A.2).

Previous works focusing on theoretical guarantees typically use MNIST with a linear model, far from practical scenarios. We instead evaluate on CIFAR-10 with a three-layer CNN and ResNet-18 to stress scalability. We adapt AmIGO (Arbel & Mairal, 2022), MRBO (Ji et al., 2021), and AID-CG (Grazzi et al., 2020), all IFT-based methods that approximate the IHVP. We replace their solver with IKVPs and compare against other methods including the IFT-based stocBiO (Ji et al., 2021), and first-order methods BOME (Liu et al., 2022), BSG-1 (Giovannelli et al., 2025), stocBiO (Ji et al., 2021). We also replace the Hessian with the GGN and apply Neumann and CG. For fairness, we use CG with 3 iterations and Neumann with 10 terms, which strikes a balance between runtime and accuracy. We run every method for fixed iterations to ensure they reach the minimum validation loss. Hyperparameters, including inner/outer learning rates, momentum, and damping values, were selected via grid search on a dedicated validation set.

Figure 2 highlights our main experimental finding: replacing IHVPs with IKVPs yields clear efficiency gains. The plots, which track the lowest test loss achieved within 1000 iterations, show that KFAC adapted methods (AID-KFAC, MRBO-KFAC, Amigo-KFAC) converge faster in wall-clock time and improve iteration-wise performance compared to their original IHVP-based counterparts. In Figure 1, we vary the number of CG iterations and Neumann terms within the AID algorithm. While the GGN is slightly more efficient than the Hessian, both remain weaker than KFAC in terms of the balance between accuracy and runtime. We detail the experimental results for a three-layer CNN and a linear model in Appendix E, where the performance gap between KFAC and the baseline solvers is less pronounced on these simpler architectures than on the more complex ResNet-18.

4.2 KFAC performs better across different batch sizes

Using a smaller batch size is a common technique to reduce computational cost. However, small batches introduce noise, which can destabilize curvature estimates and harm solvers like CG that are sensitive to ill-conditioning (Granziol et al., 2022). We want to test the robustness of KFAC compared to CG as the batch size is varied.

We study this under the data hypercleaning setting with a linear model and ResNet-18. We solve the BO problem for 1000 iterations across varying batch sizes and record the minimum test loss and time taken to reach that loss (Figure 3). For the linear model, full convergenc is achievable: both methods perform poorly with small batches but improve as batch size grows, eventually closing the gap between KFAC and CG, likely due to instability in hypergradient estimates under small batches. We give more details in Appendix G.2. For ResNet-18, where full conver-

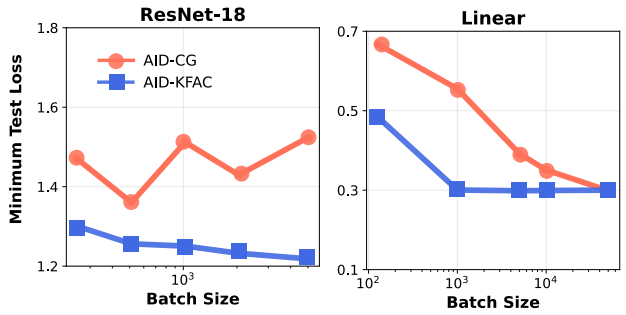


Figure 3: Batch size effects on data hypercleaning. KFAC consistently achieves lower minimum test loss than CG across models and batch sizes. For linear model, CG will converge early to a worse point than KFAC.

gence is unattainable, no monotonic improvement is seen, yet KFAC consistently outperforms CG at all batch sizes.

4.3 EKFAC and Amortization

In second-order optimization, it is common to amortize KFAC's curvature factors across iterations to save cost. Also, Eigenvalue-Corrected KFAC (EKFAC, George et al. 2018) is a popular refinement that improves KFAC's accuracy. We study both effects on ResNet-18 data hypercleaning by comparing KFAC to EKFAC under factor-update intervals $\tau \in \{1, 5, 10, 50\}$: gradients are recomputed every step but curvature is frozen within an interval.

We summarize the trade-off between computational efficiency and model performance in Section 4.3. As expected, amortizing curvature computation drastically reduces the per-iteration runtime. Specifically, increasing the update interval from $\tau = 1$ to $\tau = 10$ reduces KFAC's runtime by approximately 62% (0.69s to 0.26s). Consistent with our analysis of the computational overhead, EKFAC is significantly slower than KFAC at low intervals due to the costly eigendecomposition, though this gap narrows at larger τ. Crucially, the test loss remains robust under amortization; KFAC achieves its lowest test loss at $\tau = 5$ (1.33), suggesting that moderate amortization ($\tau = 5$ or 10) yields an optimal balance between efficiency and accuracy. Besides EKFAC and amortization, we also ablate the commonly used EMA technique (Martens & Grosse, 2015) in Appendix G.1, and we show that EMA can help stabilize the convergence for the linear model under small batch size.

5 CURVATURE MATTERS AT SCALE

Scaling BO to transformers poses unique challenges: curvature matrices such as the Hessian or GGN become prohibitively large, making direct solvers infeasible. Scalable baselines like SAMA (Choe et al., 2023a) sidestep this by ignoring curvature, achieving low cost but sacrificing accuracy and stability. In this section, we address two critical

τ	Best Test Loss		Avg. iter Time (s)	
	KFAC	EKFAC	KFAC	EKFAC
1	1.24	1.21	0.69	1.83
5	1.33	1.24	0.28	0.62
10	1.35	1.30	0.26	0.36
50	1.36	1.33	0.22	0.28

Table 2: Best test loss and average per-iteration runtime for KFAC and EKFAC with varying update intervals (τ) on ResNet-18. Amortization significantly reduces computational cost while maintaining competitive performance.

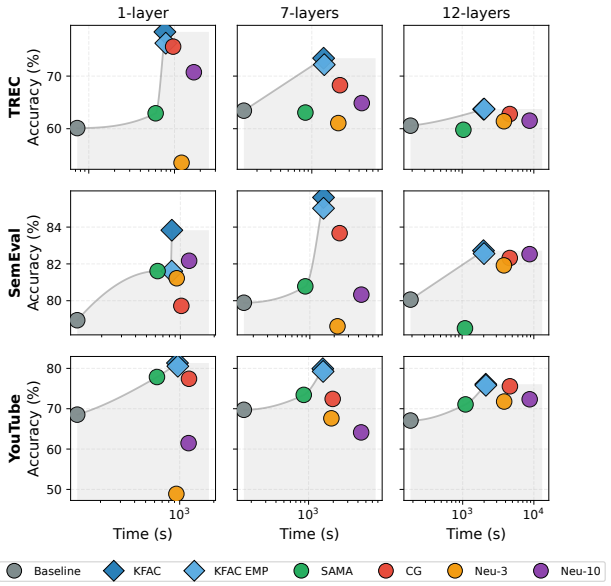


Figure 4: Results on BERT data hypercleaning. We report test accuracy (y-axis) versus total training time (x-axis) on three datasets when fine-tuning 1, 7, 12 encoder layers (columns). Each marker corresponds to one hypergradient solver. The gray region highlights the Pareto frontier.

questions for scaling BO to massive models like BERT:

1. Is curvature still beneficial at this scale, compared to gradient-only approximations (like SAMA)?
2. Can KFAC provide curvature information scalably and efficiently, without becoming prohibitively expensive?

To answer this, we design an experiment on three text classification datasets from the WRENCH benchmark (Zhang et al., 2021). We use the same data hypercleaning formulation as equation 3, but remove the L_2 regularization (for a large-scale model like BERT, this sums the squares of over 100 million parameters, which can become numerically unstable or dominate the inner task loss). To systematically vary the inner problem size and to make Hessian-based baselines feasible, we freeze different numbers of encoder layers in BERT and gradually unfreeze them starting

Table 3: ResNet-32 accuracy (%) on long-tailed CIFAR-10/100 with different imbalanced factors (IFs), best is in **bold**.

Dataset	CIFAR-10					CIFAR-100				
IF	200	100	50	20	10	200	100	50	20	10
Base Model	65.3	70.0	74.2	81.8	86.1	34.4	38.5	43.4	50.7	55.5
Focal Loss	65.2	69.9	74.3	81.5	85.9	34.9	38.4	43.7	51.8	55.6
MWN + T_1-T_2	66.3	72.8	78.6	84.8	87.5	**36.0**	40.2	46.0	52.0	58.6
MWN + SAMA	66.3	72.9	**79.5**	85.4	87.6	35.8	40.0	45.7	52.5	59.7
MWN + KFAC (ours)	**67.0**	**73.1**	78.9	**85.6**	**88.4**	35.6	**40.9**	**46.2**	**53.2**	**60.8**

Table 4: Experiment results for auxiliary learning with the continued pretraining task. Following Choe et al. (2023a), we report test micro-F1 for ChemProt and macro-F1 for other datasets; the result is averaged over 3 runs.

	ChemProt	HyperPartisan	ACL-ARC	SciERC
Baseline	$82.54_{\pm 0.32}$	$90.05_{\pm 1.24}$	$67.85_{\pm 2.03}$	$79.87_{\pm 0.59}$
TARTAN-MT	$83.97_{\pm 0.25}$	$94.64_{\pm 0.91}$	$72.86_{\pm 3.01}$	$80.49_{\pm 0.82}$
SAMA	$83.75_{\pm 0.24}$	$\mathbf{95.18}_{\pm 0.03}$	$71.75_{\pm 1.65}$	$81.06_{\pm 0.09}$
KFAC (ours)	$\mathbf{85.04}_{\pm 0.23}$	$95.13_{\pm 0.05}$	$\mathbf{72.98}_{\pm 1.58}$	$\mathbf{81.22}_{\pm 0.14}$

from the last layer. This avoids out-of-memory issues for Neumann and CG and allows us to study solver behaviour as more layers are trained, from only the last layer (≈6% of parameters) up to the full 12-layer model. We compare SAMA, Neumann with 3 or 10 terms, CG, and two variants of our approach, KFAC and KFAC EMP, trained for 1000 bilevel iterations under the AID-style double-loop framework. We report validation accuracy, runtime, and memory footprint in Figures 1 and 4.

Across all tuning depths, KFAC consistently attains the best accuracy, showing that curvature information remains beneficial at BERT scale. At the same time, its runtime stays in the same overall range as competing baselines, while its memory use remains manageable. As more layers are unfrozen, KFAC continues to perform reliably, whereas CG and Neumann become less practical due to memory limits or degraded performance. Interestingly, the layer-wise trend suggests that restricting the inner problem to a single layer can yield higher accuracy than full fine-tuning, meaning that the regularization provided by a smaller parameter space can improve the stability of hypergradient estimation. Further details can be found in Appendix D.

6 BROADER APPLICABILITY

Thus far, we have focused on the data hypercleaning problem, but BO underlies many other machine learning tasks. To test generality, we integrate KFAC into existing frameworks for meta-learning and AI safety. Across these domains, we used KFAC EMP. This section shows our final experimental contribution: our method seamlessly applies across domains and achieves better performance. All experimental details of this section are in Appendix F.

6.1 Meta Learning Applications

Meta-learning seeks to acquire inductive biases, such as sample reweighting or task-specific initialization, that enable downstream models to adapt efficiently to user-specific objectives. These problems naturally admit a bilevel structure and form an ideal testbed for our method. We integrate KFAC into the *Betty* library (Choe et al., 2023b) and study two tasks:

(1) Meta-Weight-Net (MWN) for Class Imbalance. This task addresses training under long-tailed class distributions, where some classes have far fewer examples than others, by learning a meta-network that assigns adaptive sample weights to balance training. Following Shu et al. (2019), we construct long-tailed CIFAR-10 and CIFAR-100 by exponentially decaying the number of samples per class. A small validation set of 10 images per class is used as meta-data. The original MWN work employed one-step unrolling ($T_1 - T_2$) to approximate the bilevel objective. We compare this baseline to alternatives where the inner problem is solved with different hypergradient approximations: MWN with SAMA and MWN with our proposed KFAC approximated hypergradients. We also compare to another baseline trained using the focal loss (Lin et al., 2020). Table 3 summarizes the results. On CIFAR-10, KFAC achieves the best accuracy for most imbalances (e.g., 88.4% vs. 87.6% for MWN+SAMA at imbalance factor 10). On CIFAR-100, the advantage is clearer: KFAC improves over SAMA by up to +1.1% under severe imbalance. This suggests that additional curvature information stabilizes MWN training under extreme data imbalance.

(2) Continued Pretraining of Large Language Models. We next evaluate continued pretraining on four domain-specific corpora, where BO is used to balance auxiliary and end-task objectives. Baselines include TARTAN (Dery et al., 2022) and SAMA (Choe et al., 2023a). As shown in Table 4, KFAC achieves the strongest or tied-best performance on three datasets: +1.1% on ChemProt compared to TARTAN, and +0.2–1.2% on ACL-ARC compared to

Table 5: Accuracy drop (%) and training time of different data poisoning algorithms on LR, MLP, and CNN. Results are averaged over 5 runs. Label flipping requires no training, so time is omitted.

Model	Dataset	Clean Acc	TGDA Hessian → KFAC (Ours)		BOME (VF)		Label Flip	
			Acc Drop	time	Acc Drop	time	Acc Drop	time
LR	MNIST	92.34	$2.78_{\pm 0.04} \rightarrow \mathbf{5.40}_{\pm 0.06}$	0.7 → 0.4 hrs	$2.22_{\pm 0.04}$	0.4 hrs	$1.52_{\pm 0.03}$	-
MLP	MNIST	98.02	$1.48_{\pm 0.04} \rightarrow \mathbf{3.99}_{\pm 0.05}$	7.3 → 3.2 hrs	$1.63_{\pm 0.05}$	3.0 hrs	$0.05_{\pm 0.01}$	-
CNN	CIFAR-10	64.30	$2.04_{\pm 0.03} \rightarrow \mathbf{3.44}_{\pm 0.07}$	21.5 → 6.2 hrs	$2.15_{\pm 0.03}$	5.9 hrs	$0.17_{\pm 0.02}$	-

Table 6: Unlearnable examples with different BO algorithms and non-bilevel algorithms. The unlearnable examples trained on CNN and tested on ResNet18 and CNN.

	ResNet18	CNN	Time	Bilevel?
None (clean)	92.36	86.89	-	-
AID-CG	78.23	57.32	11.5h	✓
BOME	48.09	26.34	2.2h	✓
stocBiO	84.92	75.13	6.7h	✓
AID-KFAC (ours)	**42.33**	**22.96**	3.8h	✓
EM	**32.71**	**21.83**	<0.5h	✗

SAMA. On HyperPartisan, all methods exceed 95%, leaving little room for improvement. These results indicate that KFAC can be seamlessly integrated into large-scale language model pretraining pipelines, where even small accuracy improvements are valuable.

6.2 AI Safety Applications

Many safety-critical machine learning problems can be seen as BO, often in the form of an attacker-defender game. The outer-level variable controls adversarial perturbations or poisoned data, while the inner-level learner attempts to minimize training loss on potentially corrupted inputs. Such problems are particularly challenging due to their adversarial nature and the need to scale to larger models like CNNs and ResNets. We evaluate KFAC on two standard testbeds: data poisoning and unlearnable examples.

Data Poisoning. We evaluate on MNIST (LR/MLP) and CIFAR-10 (CNN) with a small poisoning ratio ($\epsilon = 3\%$), comparing against TGDA (Lu et al., 2022), BOME, and label flipping ($y \rightarrow 10 - y$). Following Lu et al. (2022), evaluation involves attacker pretraining, joint training, and testing via defender retraining on poisoned data. As shown in Table 5, our KFAC-based method consistently surpasses baselines, matching BOME's efficiency while achieving stronger attacks. Label flipping is nearly cost-free but ineffective, while the smaller drops on CNN/MLP versus LR indicate that more complex models are less vulnerable.

Unlearnable Examples. We adopt the GUE framework (Liu et al., 2024), where the attacker is a U-Net trained with a CNN backbone to generate imperceptible perturbations that render training data unexploitable. We evaluate on CIFAR-10 with both CNN and ResNet-18 defenders, comparing KFAC-enhanced bilevel solvers against BOME, stocBiO, AID-CG, and error minimization (Huang et al., 2021, EM). Results are shown in Table 6. While bilevel approaches can reduce classifier accuracy, they remain computationally expensive. AID-KFAC improves both efficiency and attack strength relative to AID-CG, but EM is still faster and more effective overall. This underscores the strengths and limitations of BO: KFAC narrows the efficiency gap, yet in some large-scale adversarial settings, direct perturbation optimization may remain preferable.

7 CONCLUSION

We introduced KFAC for BO, replacing expensive IHVPs with IKVPs. Across diverse tasks (data hypercleaning, meta-learning for class imbalance, large-scale BERT finetuning, continued pretraining, and AI safety problems), our method consistently accelerates convergence and improves performance. Compared to curvature-free methods such as SAMA, our KFAC-based approach preserves richer second-order information with modest overhead while avoiding the prohibitive cost of iterative solvers such as CG and Neumann. These results establish KFAC as a practical and general middle ground, enabling curvature-aware BO at previously infeasible scales. Looking ahead, important directions include developing a theoretical quantification of the extent to which KFAC can benefit BO, refining our approximation by solving the inner problem more efficiently (e.g., Dong et al. 2025), designing systematic strategies for damping, and exploring alternative structured approximations across broader application domains.

Acknowledgement

We gratefully acknowledge funding support from NSERC, the Canada CIFAR AI Chairs program and the Ontario Early Researcher 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. We thank Yihan Wang for some early discussions.

References

Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998. URL https://doi.org/10.1162/089976698300017746.

Michael Arbel and Julien Mairal. Amortized implicit differentiation for stochastic bilevel optimization. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=3PN4iyXBeF.

A. B. Bakushinsky and M. Yu. Kokurin. *Iterative Methods for Approximate Solution of Inverse Problems*. Springer, 2004. URL https://doi.org/10.1007/978-1-4020-3122-9.

Alberto Bernacchia, Máté Lengyel, and Guillaume Hennequin. Exact natural gradient in deep linear networks and its application to the nonlinear case. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, 2018. URL https://papers.nips.cc/paper_files/paper/2018/hash/7f018eb7b301a66658931cb8a93fd6e8-Abstract.html.

Luca Bertinetto, Joao F. Henriques, Philip Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=HyxnZh0ct7.

Aleksandar Botev and James Martens. KFAC-JAX, 2022. URL https://github.com/google-deepmind/kfac-jax.

Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical Gauss-Newton optimisation for deep learning. In *International Conference on Machine Learning (ICML)*, pp. 557–565, 2017. URL https://proceedings.mlr.press/v70/botev17a.html.

Tianyi Chen, Yuejiao Sun, and Wotao Yin. Closing the gap: Tighter analysis of alternating stochastic gradient methods for bilevel problems. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. URL https://openreview.net/forum?id=OItvP2-i9j.

Sang Choe, Sanket Vaibhav Mehta, Hwijeen Ahn, Willie Neiswanger, Pengtao Xie, Emma Strubell, and Eric Xing. Making scalable meta learning practical. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 26271–26290, 2023a. URL https://openreview.net/forum?id=Xazhn0JoNx.

Sang Keun Choe, Willie Neiswanger, Pengtao Xie, and Eric Xing. Betty: An automatic differentiation library for multilevel optimization. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023b. URL https://openreview.net/forum?id=LV_MeMS38Q9.

Mathieu Dagréou, Pierre Ablin, Samuel Vaiter, and Thomas Moreau. A framework for bilevel optimization that enables stochastic and global variance reduction algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 26698–26710, 2022. URL https://openreview.net/forum?id=wlEOsQ917F.

Mathieu Dagréou, Pierre Ablin, Samuel Vaiter, and Thomas Moreau. How to compute Hessian-vector products? In *The Third Blogpost Track at ICLR*, 2024. URL https://openreview.net/forum?id=rTgjQtGP3O.

Felix Dangel, Stefan Harmeling, and Philipp Hennig. Modular block-diagonal curvature approximations for feedforward architectures. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020a. URL https://proceedings.mlr.press/v108/dangel20a.html.

Felix Dangel, Frederik Kunstner, and Philipp Hennig. BackPACK: Packing more into backprop. In *International Conference on Learning Representations (ICLR)*, 2020b. URL https://openreview.net/forum?id=BJlrF24twB.

Felix Dangel, Runa Eschenhagen, Weronika Ormaniec, Andres Fernandez, Lukas Tatzel, and Agustinus Kristiadi. Position: Curvature matrices should be democratized via linear operators. arXiv Preprint arXiv:2501.19183, 2025a. URL https://arxiv.org/abs/2501.19183.

Felix Dangel, Bálint Mucsányi, Tobias Weber, and Runa Eschenhagen. Kronecker-factored approximate curvature (KFAC) from scratch. code at https://github.com/f-dangel/kfac-tutorial, 2025b. URL https://arxiv.org/abs/2507.05127.

Erik Daxberger, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig. Laplace redux - effortless Bayesian deep learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. URL https://openreview.net/forum?id=gDcaUj4Myhn.

Lucio M. Dery, Paul Michel, Ameet Talwalkar, and Graham Neubig. Should we be pre-training? An argument for end-task aware training as an alternative. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=2bO2x8NAIMB.

Youran Dong, Junfeng Yang, Wei Yao, and Jin Zhang. Efficient curvature-aware hypergradient approximation for bilevel optimization. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=oah1fwo6Wv.

Runa Eschenhagen, Alexander Immer, Richard E. Turner, Frank Schneider, and Philipp Hennig. Kronecker-

factored approximate curvature for modern neural network architectures. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://openreview.net/forum?id=Ex3oJEKS53.

Yunzhen Feng, Shanmukha Ramakrishna Vedantam, and Julia Kempe. Embarrassingly simple dataset distillation. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=PLoWVP7Mjc.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*, pp. 1126–1135, 2017. URL https://proceedings.mlr.press/v70/finn17a.html.

Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Forward and reverse gradient-based hyperparameter optimization. In *International Conference on Machine Learning (ICML)*, pp. 1165–1173, 2017. URL https://proceedings.mlr.press/v70/franceschi17a.html.

Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *International conference on Machine Learning (ICML)*, pp. 1568–1577, 2018. URL https://proceedings.mlr.press/v80/franceschi18a.html.

Thomas George, César Laurent, Xavier Bouthillier, Nicolas Ballas, and Pascal Vincent. Fast approximate natural gradient descent in a Kronecker factored eigenbasis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/48000647b315f6f00f913caa757a70b3-Abstract.html.

Saeed Ghadimi and Mengdi Wang. Approximation methods for bilevel programming. arXiv Preprint arXiv:1802.02246, 2018. URL https://arxiv.org/abs/1802.02246.

Tommaso Giovannelli, Griffin Kent, and Luis Nunes Vicente. Inexact bilevel stochastic gradient methods for constrained and unconstrained lower-level problems. *Journal of Global Optimization*, 92:569–614, 2025. URL https://doi.org/10.1007/s10898-025-01502-8.

Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. Recasting gradient-based meta-learning as hierarchical Bayes. In *International Conference on Learning Representations (ICLR)*, 2018. URL https://openreview.net/forum?id=BJ_UL-k0b.

Diego Granziol, Stefan Zohren, and Stephen Roberts. Learning Rates as a Function of Batch Size: A Random Matrix Theory Approach to Neural Network Training. *Journal of Machine Learning Research*, 23(173):1–65, 2022. URL https://www.jmlr.org/papers/v23/20-1258.html.

Riccardo Grazzi, Luca Franceschi, Massimiliano Pontil, and Saverio Salzo. On the iteration complexity of hypergradient computation. In *International Conference on Machine Learning (ICML)*, pp. 3748–3758, 2020. URL https://proceedings.mlr.press/v119/grazzi20a.html.

Roger Grosse and James Martens. A Kronecker-factored approximate Fisher matrix for convolution layers. In *International Conference on Machine Learning (ICML)*, 2016. URL https://proceedings.mlr.press/v48/grosse16.html.

Roger Grosse, Juhan Bae, Cem Anil, Nelson Elhage, Alex Tamkin, Amirhossein Tajdini, Benoit Steiner, Dustin Li, Esin Durmus, Ethan Perez, et al. Studying large language model generalization with influence functions. arXiv Preprint arXiv:2308.03296, 2023. URL https://arxiv.org/abs/2308.03296.

Mingyi Hong, Hoi-To Wai, Zhaoran Wang, and Zhuoran Yang. A two-timescale stochastic algorithm framework for bilevel optimization: Complexity analysis and application to actor-critic. *SIAM Journal on Optimization*, 33 (1):147–180, 2023. URL https://doi.org/10.1137/20M1387341.

Nathan Hu, Eric Mitchell, Christopher D Manning, and Chelsea Finn. Meta-learning online adaptation of language models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4418–4432, 2023. URL https://aclanthology.org/2023.emnlp-main.268/.

Hanxun Huang, Xingjun Ma, Sarah Monazam Erfani, James Bailey, and Yisen Wang. Unlearnable examples: Making personal data unexploitable. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=iAmZUo0DxC0.

W. Ronny Huang, Jonas Geiping, Liam Fowl, Gavin Taylor, and Tom Goldstein. Metapoison: Practical general-purpose clean-label data poisoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 12080–12091, 2020. URL https://papers.nips.cc/paper/2020/hash/8ce6fc704072e351679ac97d4a985574-Abstract.html.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems (NeurIPS)*, volume 31, 2018. URL https://papers.nips.cc/paper_files/paper/2018/hash/5a4be1fa34e62bb8a6ec6b91d2462f5a-Abstract.html.

Kaiyi Ji, Junjie Yang, and Yingbin Liang. Bilevel optimization: Convergence analysis and enhanced design. In

International Conference on Machine Learning (ICML), pp. 4882–4892, 2021. URL https://proceedings.mlr.press/v139/ji21c.html.

Ryo Karakida and Kazuki Osawa. Understanding approximate Fisher information for fast convergence of natural gradient descent in wide neural networks. In *Advances in neural information processing systems (NeurIPS)*, volume 33, pp. 10891–10901, 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/hash/7b41bfa5085806dfa24b8c9de0ce567f-Abstract.html.

Yeongjoon Kim, Donggoo Kang, Yeongheon Mok, Sunkyu Kwon, and Joonki Paik. Bidirectional meta-kronecker factored optimizer and Hausdorff distance loss for few-shot medical image segmentation. *Scientific Reports*, 13 (1):8088, 2023. URL https://doi.org/10.1038/s41598-023-35276-4.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. URL https://arxiv.org/abs/1412.6980.

Frederik Kunstner, Philipp Hennig, and Lukas Balles. Limitations of the empirical Fisher approximation for natural gradient descent. In *Advances in neural information processing systems (NeurIPS)*, volume 32, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/46a558d97954d0692411c861cf78ef79-Abstract.html.

Jeongyeol Kwon, Dohyun Kwon, Stephen Wright, and Robert D. Nowak. A fully first-order method for stochastic bilevel optimization. In *International Conference on Machine Learning (ICML)*, pp. 18083–18113, 2023. URL https://proceedings.mlr.press/v202/kwon23c.html.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2):318–327, 2020. URL https://doi.org/10.1109/TPAMI.2018.2858826.

Wu Lin, Felix Dangel, Runa Eschenhagen, Kirill Neklyudov, Agustinus Kristiadi, Richard E Turner, and Alireza Makhzani. Structured inverse-free natural gradient descent: Memory-efficient & numerically-stable KFAC. In *roceedings of the 41st International Conference on Machine Learning*, 2024. URL https://proceedings.mlr.press/v235/lin24f.html.

Bo Liu, Mao Ye, Stephen Wright, Peter Stone, and Qiang Liu. BOME! Bilevel optimization made easy: A simple first-order approach. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://openreview.net/forum?id=DTsCy9Lyj5-.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=S1eYHoC5FX.

Risheng Liu, Xuan Liu, Xiaoming Yuan, Shangzhi Zeng, and Jin Zhang. A value-function-based interior-point method for non-convex bi-level optimization. In *International conference on machine learning*, pp. 6882–6892, 2021. URL https://proceedings.mlr.press/v139/liu21o.html.

Shuang Liu, Yihan Wang, and Xiao-Shan Gao. Game-theoretic unlearnable example generator. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 21349–21358, 2024. URL https://doi.org/10.1609/aaai.v38i19.30130.

Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 1540–1552, 2020. URL https://proceedings.mlr.press/v108/lorraine20a.html.

Yiwei Lu, Gautam Kamath, and Yaoliang Yu. Indiscriminate data poisoning attacks on neural networks. *Transactions on Machine Learning Research*, 2022. URL https://openreview.net/forum?id=x4hmIsWu7e.

James Martens. New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, 21(146):1–76, 2020. URL https://jmlr.org/papers/v21/17-678.html.

James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *International Conference on Machine Learning (ICML)*, pp. 2408–2417, 2015. URL https://proceedings.mlr.press/v37/martens15.html.

James Martens, Jimmy Ba, and Matt Johnson. Kronecker-factored curvature approximations for recurrent neural networks. In *International Conference on Learning Representations (ICLR)*, 2018. URL https://openreview.net/forum?id=HyMTkQZAb.

Kazuki Osawa, Yohei Tsuji, Yuichiro Ueno, Akira Naruse, Rio Yokota, and Satoshi Matsuoka. Large-scale distributed second-order optimization using Kronecker-factored approximate curvature for deep convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. URL https://doi.org/10.1109/CVPR.2019.01264.

Kazuki Osawa, Satoki Ishikawa, Rio Yokota, Shigang Li, and Torsten Hoefler. ASDL: A unified interface for gradient preconditioning in PyTorch, 2023. URL https://arxiv.org/abs/2305.04684.

Eunbyung Park and Junier B Oliva. Meta-curvature. In *Advances in Neural Information Processing Systems*

(NeurIPS), 2019. URL https://papers.nips.cc/paper_files/paper/2019/hash/57c0531e13f40b91b3b0f1a30b529a1d-Abstract.html.

Barak A. Pearlmutter. Fast exact multiplication by the Hessian. *Neural Computation*, 6(1):147–160, 1994. URL https://doi.org/10.1162/neco.1994.6.1.147.

Felix Petersen, Tobias Sutter, Christian Borgelt, Dongsung Huh, Hilde Kuehne, Yuekai Sun, and Oliver Deussen. ISAAC Newton: Input-based approximate curvature for newton's method. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=0paCJSFW7j.

Evani Radiya-Dixit, Sanghyun Hong, Nicholas Carlini, and Florian Tramer. Data poisoning won't save you from facial recognition. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=B5XahNLmna.

Aravind Rajeswaran, Chelsea Finn, Sham M. Kakade, and Sergey Levine. Meta-learning with implicit gradients. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. URL https://papers.nips.cc/paper/8306-meta-learning-with-implicit-gradients.

Levent Sagun, Leon Bottou, and Yann LeCun. Eigenvalues of the Hessian in deep learning: Singularity and beyond. *arXiv preprint arXiv:1611.07476*, 2016. URL https://arxiv.org/abs/1611.07476.

Nicol N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738, 2002. URL https://doi.org/10.1162/08997660260028683.

Amirreza Shaban, Ching-An Cheng, Nathan Hatch, and Byron Boots. Truncated back-propagation for bilevel optimization. In *The 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 1723–1732, 2019. URL https://proceedings.mlr.press/v89/shaban19a.html.

Qianli Shen, Yezhen Wang, Zhouhao Yang, Xiang Li, Haonan Wang, Yang Zhang, Jonathan Scarlett, Zhanxing Zhu, and Kenji Kawaguchi. Memory-efficient gradient unrolling for large-scale bi-level optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 90934–90964, 2024. URL https://openreview.net/forum?id=MI8Z9gutIn.

Jun Shu, Qi Xie, Lixuan Yi, Qian Zhao, Sanping Zhou, Zongben Xu, and Deyu Meng. Meta-weight-net: Learning an explicit mapping for sample weighting. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/e58cc5ca94270acaceed13bc82dfedf7-Abstract.html.

Harold R. Parks Steven G. Krantz. *The Implicit Function Theorem*. Birkhäuser New York, NY, 2002. URL https://doi.org/10.1007/978-1-4614-5981-1.

Heinrich von Stackelberg. *Market structure and equilibrium*. Springer, 1934. URL https://doi.org/10.1007/978-3-642-12586-7.

Chaoqi Wang, Roger Grosse, Sanja Fidler, and Guodong Zhang. Eigendamage: Structured pruning in the Kronecker-factored eigenbasis. In *International Conference on Machine Learning (ICML)*, 2019. URL https://proceedings.mlr.press/v97/wang19g.html.

Ce Zhang, Xiao Yao, Changfeng Shi, and Min Gu. Kronecker-factored approximate curvature with adaptive learning rate for optimizing model-agnostic meta-learning. *Multimedia Systems*, 29(6):3169–3177, 2023. URL https://doi.org/10.1007/s00530-023-01159-x.

Jieyu Zhang, Yue Yu, Yinghao Li, Yujing Wang, Yaming Yang, Mao Yang, and Alexander Ratner. WRENCH: A comprehensive benchmark for weak supervision. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=Q9SKS5k8io.

Lin Zhang, Shaohuai Shi, Wei Wang, and Bo Li. Scalable K-FAC training for deep neural networks with distributed preconditioning. *IEEE Transactions on Cloud Computing*, 11(3):2365–2378, 2022. URL https://doi.org/10.1109/TCC.2022.3205918.

Yihua Zhang, Prashant Khanduri, Ioannis Tsaknakis, Yuguang Yao, Mingyi Hong, and Sijia Liu. An introduction to bilevel optimization: Foundations and applications in signal processing and machine learning. *IEEE Signal Processing Magazine*, 41(1):38–59, 2024. URL https://doi.org/10.1109/MSP.2024.3358284.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Not Applicable] This work is mainly empirical and we make no theoretical claim.
   (b) Complete proofs of all theoretical results. [Not Applicable]
   (c) Clear explanations of any assumptions. [Not Applicable]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Not Applicable]
   (b) The license information of the assets, if applicable. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

A Probabilistic Interpretation of Losses and Monte-Carlo KFAC

This section explains how the random vector distribution $p(\boldsymbol{s}_n)$ used in the main text arises from the probabilistic interpretation of the loss and how it is instantiated in Monte-Carlo KFAC in practice.

A.1 How to implement Monte-Carlo KFAC

Many supervised learning objectives can be written as negative log-likelihoods. Given an input-target pair $(\boldsymbol{x}, \boldsymbol{y})$, let the network output be $\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}) \in \mathbb{R}^C$ that parametrizes a predictive distribution $q(\boldsymbol{y} \mid \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}))$. This viewpoint covers two standard losses used in practice.

In regression, the loss is

$$\ell(\boldsymbol{f}, \boldsymbol{y}) = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{f}\|^2,$$

which corresponds, up to an additive constant, to the negative log-likelihood of the Gaussian model

$$q(\boldsymbol{y} \mid \boldsymbol{f}) = \mathcal{N}(\boldsymbol{y}; \boldsymbol{f}, \boldsymbol{I}).$$

In multiclass classification, if $\boldsymbol{f} \in \mathbb{R}^C$ are logits and $\boldsymbol{p} = \mathrm{softmax}(\boldsymbol{f})$, then the cross-entropy loss is

$$\ell(\boldsymbol{f}, \boldsymbol{y}) = -\log \boldsymbol{p}_{\boldsymbol{y}}$$

is the negative log-likelihood of the categorical model

$$q(\boldsymbol{y} \mid \boldsymbol{f}) = \mathrm{Cat}(\boldsymbol{p}).$$

Given the above probabilistic view of loss functions, we now explain how Monte-Carlo KFAC is implemented. For a training example $(\boldsymbol{x}_n, y_n)$, let

$$\boldsymbol{f}_n = \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_n), \qquad c_n = \ell(\boldsymbol{f}_n, \boldsymbol{y}_n).$$

In Section 3.1, we introduced a distribution $p(\boldsymbol{s}_n)$ such that

$$\nabla^2_{\boldsymbol{f}_n} c_n = \mathbb{E}_{\boldsymbol{s}_n \sim p(\boldsymbol{s}_n)}[\boldsymbol{s}_n \boldsymbol{s}_n^\top].$$

This identity says that the Hessian w.r.t. $\boldsymbol{f}_n$, $\nabla^2_{\boldsymbol{f}_n} c_n$, can be represented as the second moment of a suitable random vector $\boldsymbol{s}_n$. In MC-KFAC, this random vector is constructed by sampling a pseudo-target

$$\tilde{\boldsymbol{y}}_n \sim q(\cdot \mid \boldsymbol{f}_n),$$

from the model's predictive distribution, and then differentiating the corresponding pseudo-loss

$$\tilde{c}_n = -\log q(\tilde{\boldsymbol{y}}_n \mid \boldsymbol{f}_n).$$

The random vector is therefore

$$\boldsymbol{s}_n := \nabla_{\boldsymbol{f}_n} \tilde{c}_n.$$

Equivalently, $p(\boldsymbol{s}_n)$ is the distribution induced by first drawing $\tilde{\boldsymbol{y}}_n \sim q(\cdot \mid \boldsymbol{f}_n)$, and then mapping it to the gradient

$$\boldsymbol{s}_n = \nabla_{\boldsymbol{f}_n}(-\log q(\tilde{\boldsymbol{y}}_n \mid \boldsymbol{f}_n)).$$

With this definition,

$$\mathbb{E}_{\tilde{y}_n \sim q(\cdot \mid \boldsymbol{f}_n)}[\boldsymbol{s}_n \boldsymbol{s}_n^\top] = \mathbb{E}_{\boldsymbol{s}_n \sim p(\boldsymbol{s}_n)}[\boldsymbol{s}_n \boldsymbol{s}_n^\top],$$

and this expectation equals the output-space GGN term used by KFAC. In practice, we draw M i.i.d. samples $\boldsymbol{s}_{n,i} \sim p(\boldsymbol{s}_n)$:

$$\nabla^2_{\boldsymbol{f}_n} c_n \approx \frac{1}{M} \sum_{i=1}^{M} \boldsymbol{s}_{n,i} \boldsymbol{s}_{n,i}^\top.$$

In essence, we applied the following identity:

$$\mathbb{E}_{\tilde{y}_n\sim q(\cdot|\boldsymbol{f}_n)}[\nabla_{\boldsymbol{f}_n}\log q(\tilde{y}_n|\boldsymbol{f}_n)\cdot\left(\nabla_{\boldsymbol{f}_n}\log q(\tilde{y}_n|\boldsymbol{f}_n)\right)^\top] = -\mathbb{E}_{\tilde{y}_n\sim q(\cdot|\boldsymbol{f}_n)}[\nabla^2_{\boldsymbol{f}_n}\log q(\tilde{y}_n|\boldsymbol{f}_n)] \tag{4}$$

$$= -\nabla^2_{\boldsymbol{f}_n}\log q(y_n|\boldsymbol{f}_n) =: \nabla^2_{\boldsymbol{f}_n} c_n, \tag{5}$$

where the first equality is a familiar consequence of integration by parts while the second equality holds if the Hessian $\nabla^2_{\boldsymbol{f}_n}\log q(y_n|\boldsymbol{f}_n)$ does not depend on y_n, e.g., for a simple exponential family where

$$\log q(y_n|\boldsymbol{f}_n) = h(y_n) + T(y_n)\cdot\boldsymbol{f}_n - A(\boldsymbol{f}_n).$$

We now make the distribution $p(\boldsymbol{s}_n)$ explicit for two commonly used loss functions.

Cross-entropy loss. Let $\boldsymbol{f}_n \in \mathbb{R}^C$ be the logits for example n, and let $\boldsymbol{p}_n = \mathrm{softmax}(\boldsymbol{f}_n)$, For a lable $y_n \in \{1,\cdots,C\}$, the softmax cross-entropy loss is

$$\ell(\boldsymbol{f}_n, y_n) = -\log p_{y_n} = -\boldsymbol{f}_{n,y_n} + \log\sum_{c=1}^{C} e^{\boldsymbol{f}_{n,c}}.$$

Its Hessian w.r.t. $\boldsymbol{f}_n$ is

$$\nabla^2_{\boldsymbol{f}_n}\ell(\boldsymbol{f}_n, y_n) = \mathrm{diag}(\boldsymbol{p}_n) - \boldsymbol{p}_n\boldsymbol{p}_n^\top,$$

which is the standard output-space GGN term for softmax classification (Kunstner et al., 2019). To realize this Hessian as a second memoent, we sample a pseudo-target

$$\tilde{y}_n \sim \mathrm{Cat}(\boldsymbol{p}_n),$$

and define the corresponding pseudo-loss

$$\tilde{\ell}_n = -\log p_{n,\tilde{y}_n}.$$

If $\boldsymbol{e}_j$ denotes the one-hot vector of class j, then when $\tilde{y}_n = j$,

$$\boldsymbol{s}_n := \nabla_{\boldsymbol{f}_n}\tilde{\ell}_n = \boldsymbol{p}_n - \boldsymbol{e}_j.$$

Hence $p(\boldsymbol{s}_n)$ is the distribution induced by $j \sim \mathrm{Cat}(\boldsymbol{p})$ and the map

$$\boldsymbol{s}_n = \boldsymbol{p}_n - \boldsymbol{e}_j.$$

Since

$$\mathbb{E}[\boldsymbol{e}_j] = \boldsymbol{p}_n, \quad \mathbb{E}[\boldsymbol{e}_j\boldsymbol{e}_j^\top] = \mathrm{diag}(\boldsymbol{p}_n),$$

we obtain that

$$\mathbb{E}[\boldsymbol{s}_n] = 0,$$

and

$$\mathbb{E}_{\boldsymbol{s}_n\sim p(\boldsymbol{s}_n)}[\boldsymbol{s}_n\boldsymbol{s}_n^\top] = \mathbb{E}[(\boldsymbol{p}_n - \boldsymbol{e}_j)(\boldsymbol{p}_n - \boldsymbol{e}_j)^\top] = \mathrm{diag}(\boldsymbol{p}_n) - \boldsymbol{p}_n\boldsymbol{p}_n^\top.$$

Therefore, for cross-entropy, the random vector $\boldsymbol{s}_n$ is obtained by sampling a pseudo-label from the model predictive distribution and differentiating the corresponding pseudo-loss, yielding an unbiased rank-one estimator of the output Hessian.

Mean squared error. For MSE,

$$\ell(\boldsymbol{f}, \boldsymbol{y}) = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{f}\|^2, \qquad \nabla^2_{\boldsymbol{f}}\ell(\boldsymbol{f}, \boldsymbol{y}) = \boldsymbol{I}.$$

In this case, $p(\boldsymbol{s}_n)$ can be taken as a standard Gaussian

$$\boldsymbol{s}_n \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}),$$

so that

$$\mathbb{E}_{\boldsymbol{s}_n\sim p(\boldsymbol{s}_n)}[\boldsymbol{s}_n\boldsymbol{s}_n^\top] = \boldsymbol{I}.$$

Thus, for MSE, the random vector $\boldsymbol{s}_n$ is not obtained by sampling a discrete pseudo-label, but by sampling a Gaussian output perturbation whose covariance matches the output curvature.

A.2 Weighted cross-entropy and Fisher

In data hypercleaning, each training example $(\boldsymbol{x}_n, y_n)$ is assigned a learnable nonnegative weight $\sigma_n \geq 0$, yielding the weighted objective

$$\mathcal{L}_\sigma(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \sigma_n \, \ell(\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_n), y_n).$$

For a fixed example n, define the weighted per-example loss

$$\ell_{n,\sigma} := \sigma_n \, \ell(\boldsymbol{f}_n, y_n), \qquad \boldsymbol{f}_n = \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_n).$$

Since σ_n does not depend on $\boldsymbol{f}_n$, its Hessian with respect to the network output is simply

$$\nabla^2_{\boldsymbol{f}_n} \ell_{n,\sigma} = \sigma_n \, \nabla^2_{\boldsymbol{f}_n} \ell(\boldsymbol{f}_n, y_n).$$

For softmax cross-entropy, let $\boldsymbol{p} = \mathrm{softmax}(\boldsymbol{f}_n)$. Then the Hessian w.r.t. $\boldsymbol{f}_n$ is

$$\nabla^2_{\boldsymbol{f}_n} \ell(\boldsymbol{f}_n, y_n) = \mathrm{diag}(\boldsymbol{p}_n) - \boldsymbol{p}_n \boldsymbol{p}_n^\top,$$

and therefore the weighted output-space curvature is

$$\nabla^2_{\boldsymbol{f}_n} \ell_{n,\sigma} = \sigma_n \big(\mathrm{diag}(\boldsymbol{p}_n) - \boldsymbol{p}_n \boldsymbol{p}_n^\top\big).$$

To realize this matrix as a second moment, we sample a pseudo-label

$$j \sim \mathrm{Cat}(\boldsymbol{p}_n),$$

and define the vector $\boldsymbol{s}_n$

$$\boldsymbol{s}_n := \sqrt{\sigma_n} \, (\boldsymbol{p}_n - \boldsymbol{e}_j),$$

where $\boldsymbol{e}_j$ is the one-hot vector for class j. Since

$$\mathbb{E}_{j \sim \mathrm{Cat}(\boldsymbol{p}_n)} \big[(\boldsymbol{p}_n - \boldsymbol{e}_j)(\boldsymbol{p}_n - \boldsymbol{e}_j)^\top \big] = \mathrm{diag}(\boldsymbol{p}) - \boldsymbol{p}\boldsymbol{p}^\top,$$

we obtain

$$\mathbb{E}[\boldsymbol{s}_n \boldsymbol{s}_n^\top] = \sigma_n \big(\mathrm{diag}(\boldsymbol{p}_n) - \boldsymbol{p}_n \boldsymbol{p}_n^\top\big) = \nabla^2_{\boldsymbol{f}_n} \ell_{n,\sigma}.$$

Thus, in the weighted setting, the correct Monte-Carlo estimator is obtained by multiplying the usual cross-entropy vector by $\sqrt{\sigma_n}$. Equivalently, the corresponding pseudo-loss can be written as

$$\tilde{\ell}_{n,\sigma} = \sqrt{\sigma_n} \, (-\log p_{n,j}), \qquad j \sim \mathrm{Cat}(\boldsymbol{p}_n),$$

whose gradient with respect to the logits is exactly

$$\nabla_{\boldsymbol{f}_n} \tilde{\ell}_{n,\sigma} = \sqrt{\sigma_n} \, (\boldsymbol{p}_n - \boldsymbol{e}_j).$$

This ensures that the rank-one estimator $\boldsymbol{s}_n \boldsymbol{s}_n^\top$ is unbiased for the weighted output Hessian. Scaling the pseudo-loss directly by σ_n instead would produce

$$\mathbb{E}[\boldsymbol{s}_n \boldsymbol{s}_n^\top] = \sigma_n^2 \big(\mathrm{diag}(\boldsymbol{p}_n) - \boldsymbol{p}_n \boldsymbol{p}_n^\top\big),$$

which would over-scale the curvature term.

B Related Work

B.1 More on KFAC

Originially introduced by Martens & Grosse (2015) for MLP, KFAC has been extended to a range of modern architectures, including convolutional networks (Grosse & Martens, 2016), recurrent networks (Martens et al., 2018), and transformers (Eschenhagen et al., 2023). There are also refinements of the basic approximation, such as eigenvalue-corrected KFAC (George et al., 2018, EKFAC). KFAC has been successfully applied in the context of optimization (Osawa et al., 2019), model sparsification (Wang et al., 2019), Laplace approximations (Daxberger et al., 2021) and influence functions (Grosse et al., 2023). We are unaware of works applying this curvature approximation in the context of bilevel optimization.

B.2 Meta Learning and KFAC

Several prior works have incorporated curvature information into meta-learning through KFAC or related approximations, but in fundamentally different ways from ours. We will discuss the difference between our work and theirs.

MAML (Finn et al., 2017) is a model-agnostic algorithm to train a model with the greatest sensitivity to the loss function of the new task. Suppose there are tasks obeying $T \sim p(T)$ distribution. We can also formulate MAML as a bilevel optimization problem, if we choose the initialization parameter $\boldsymbol{\theta} \in \mathbb{R}^d$.

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{T\sim p(T)}[\mathcal{L}^{\text{val}}(\phi_T^*(\boldsymbol{\theta}); T)], \quad \text{s.t.} \quad \phi_T^*(\boldsymbol{\theta}) = \arg\min_{\phi} \mathcal{L}^{\text{tr}}(\phi; T, \boldsymbol{\theta}),$$

where ϕ is the task-adapted model parameter. That is, we want to find the task-specific parameter $\phi_T^*(\boldsymbol{\theta})$ which will minimize the expected post-adaptation loss across tasks. In practice, MAML does not solve the inner problem to full convergence, but uses one step of SGD from $\boldsymbol{\theta}$ to approximate it, and the objective becomes:

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{T\sim p(T)}[\mathcal{L}^{\text{val}}(\boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\text{tr}}(\boldsymbol{\theta}; T), T)],$$

for the learning rate α. There are several works that have adapted KFAC to improve the algorithms for MAML. Zhang et al. (2023) replace the inner optimizer with natural gradient descent preconditioned by KFAC, focusing on improving convergence speed rather than on BO. Grant et al. (2018) reinterpret MAML as hierarchical Bayes and employ KFAC only to approximate Hessian determinants within Laplace correction to the marginal likelihood, rather than to compute hypergradients. Kim et al. (2023) propose BM-KFP, which applies Kronecker factorization to the parameter of a learned meta-optimizer, a structural efficiency technique unrelated to Hessian approximation. Meta-Curvature (Park & Oliva, 2019) learns gradient transformation matrices for fast adaptation, and to ensure tractability, parameterizes them with low-rank and Kronecker-structured factors, which are conceptually related to KFAC's block-Kronecker approximation of curvature. Unlike KFAC, however, these operators are not derived from the GGN but are meta-learned directly from data. In contrast, we use KFAC as an efficient means for inverse Hessian-vector products.

C Connection between GU and IFT

The idea of MAML can be seen as a one-step gradient unrolling (GU), and we can show that GU is equivalent to IFT for the one-step case or the infinite step case. We consider the BO formulation:

$$\min_{\boldsymbol{\lambda}\in\mathbb{R}^m} \Phi(\boldsymbol{\lambda}) := \mathcal{J}_{\text{out}}\big(\boldsymbol{\lambda}, \boldsymbol{\theta}^\star(\boldsymbol{\lambda})\big), \quad \text{s.t.} \quad \boldsymbol{\theta}^\star(\boldsymbol{\lambda}) = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^d} \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}).$$

Here, we demonstrate how to solve it using GU and how this is connected to the hypergradient. Let the inner problem be optimized by T steps of gradient descent with step size $\eta > 0$,

$$\boldsymbol{\theta}^{t+1} \;=\; \boldsymbol{\theta}^t - \eta\, \nabla_2\, \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^t), \qquad t = 0, \ldots, T-1, \tag{6}$$

with initialization $\boldsymbol{\theta}^0$ (typically independent of $\boldsymbol{\lambda}$). Define the unrolled outer objective $\Phi_T(\boldsymbol{\lambda}) := \mathcal{J}_{\text{out}}(\boldsymbol{\lambda}, \theta^T(\boldsymbol{\lambda}))$. By the chain rule,

$$\nabla_{\boldsymbol{\lambda}} \Phi_T(\boldsymbol{\lambda}) \;=\; \nabla_1\, \mathcal{J}_{\text{out}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^T) \;+\; \big(\tfrac{\mathrm{d}\boldsymbol{\theta}^T}{\mathrm{d}\boldsymbol{\lambda}}\big)^\top \nabla_2\, \mathcal{J}_{\text{out}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^T). \tag{7}$$

Let $\boldsymbol{S}^t := \frac{\mathrm{d}\boldsymbol{\theta}^t}{\mathrm{d}\boldsymbol{\lambda}} \in \mathbb{R}^{d\times m}$ and $\boldsymbol{H}_t := \nabla_{22}^2 \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^t)$. If $\boldsymbol{\theta}^0$ does not depend on $\boldsymbol{\lambda}$ (i.e., $\boldsymbol{S}^0 = \frac{\mathrm{d}\boldsymbol{\theta}^0}{\mathrm{d}\boldsymbol{\lambda}} = 0$), differentiating the update in eq. (6) and expanding the recursion, we can get the explicit formula for $\boldsymbol{S}^{t+1}$:

$$\begin{aligned}
\boldsymbol{S}^{t+1} &= \boldsymbol{S}^t - \eta\Big[\nabla_{21}^2 \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^t) \;+\; \nabla_{22}^2 \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^t)\, \boldsymbol{S}^t\Big] \\
&= -\eta \nabla_{21}^2 \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^t) + (\boldsymbol{I} - \eta \boldsymbol{H}_t)\boldsymbol{S}^t \\
&= -\eta \nabla_{21}^2 \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^t) + (\boldsymbol{I} - \eta \boldsymbol{H}_t)\big((\boldsymbol{I} - \eta \boldsymbol{H}_{t-1})\boldsymbol{S}^{t-1} - \eta \nabla_{21}^2 \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^{t-1})\big) \\
&= -\eta \nabla_{21}^2 \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^t) - \eta(\boldsymbol{I} - \eta \boldsymbol{H}_t) \nabla_{21} \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^{t-1}) + \big(\prod_{j=t-1}^{t} (\boldsymbol{I} - \eta \boldsymbol{H}_j)\big)\boldsymbol{S}^{t-1} \\
&= \ldots \\
&= -\eta \sum_{s=0}^{t} \left(\prod_{j=s+1}^{t} (\boldsymbol{I} - \eta \boldsymbol{H}_j)\right) \nabla_{21}^2 \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^s).
\end{aligned}$$

Plugging back to equation 7, the T-step unrolled hypergradient is

$$\boxed{\nabla_{\boldsymbol{\lambda}} \Phi_T(\boldsymbol{\lambda}) = \nabla_1 \mathcal{J}_{\text{out}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^T) \; - \; \eta \sum_{t=0}^{T-1} \Big[\nabla_{21}^2 \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^t)^\top \Big(\prod_{j=t+1}^{T-1} \big(\boldsymbol{I} - \eta \boldsymbol{H}_j\big) \Big) \nabla_2 \mathcal{J}_{\text{out}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^T) \Big],}$$

where products over an empty index set are the identity.

Now, recall the expression of IFT based hypergradient:

$$\boxed{\nabla_{\boldsymbol{\lambda}} \Phi(\boldsymbol{\lambda}) = \nabla_1 \mathcal{J}_{\text{out}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^\star) \; - \; \nabla_{12}^2 \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^\star) \big(\boldsymbol{H}\big)^{-1} \nabla_2 \mathcal{J}_{\text{out}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^\star),}$$

where $\boldsymbol{\theta}^*$ is the optimal $\boldsymbol{\theta}$ for the inner problem and $\boldsymbol{H} = \nabla_{22}^2 \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^*)$, and this value is usually estimated by running a few steps of an optimization algorithm. For $T = 1$, we have $\boldsymbol{S}^1 = -\eta \nabla_{21}^2 \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^0)$, and thus

$$\nabla_\lambda \Phi_1(\boldsymbol{\lambda}) = \nabla_1 \mathcal{J}_{\text{out}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^1) \; - \; \eta \left[\nabla_{21}^2 \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^0) \right]^\top \nabla_2 \mathcal{J}_{\text{out}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^1).$$

Comparing to the IFT form, if we (i) approximate $\boldsymbol{\theta}^* \approx \boldsymbol{\theta}^0 \approx \boldsymbol{\theta}^1$ and (ii) approximate the inner curvature by the identity, $\boldsymbol{H} \approx \boldsymbol{I}$ (so $\boldsymbol{H}^{-1} \nabla_2 \mathcal{J}_{\text{out}} \approx \nabla_2 \mathcal{J}_{\text{out}}$), then with $\eta \approx 1$ we recover the same algebraic structure:

$$\nabla_{\boldsymbol{\lambda}} \Phi(\boldsymbol{\lambda}) \; \approx \; \nabla_1 \mathcal{J}_{\text{out}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^\star) \; - \; \nabla_{12}^2 \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^\star) \nabla_2 \mathcal{J}_{\text{out}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^\star).$$

Lorraine et al. (2020) shows that as $T \to \infty$, under the assumption $\boldsymbol{\theta}^0 = \boldsymbol{\theta}^1 = \cdots = \boldsymbol{\theta}^*$ and $\boldsymbol{I} - \eta \boldsymbol{H}_t$ is contractive, GU can again happen to align with IFT. To see this, consider $\boldsymbol{S}^T$:

$$\begin{aligned}
\lim_{T \to \infty} \boldsymbol{S}^T &= \lim_{T \to \infty} \Big[-\eta \sum_{t=0}^{T-1} \big(\prod_{j=t+1}^{T-1} (\boldsymbol{I} - \eta \boldsymbol{H}_j) \big) \nabla_{21}^2 \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^t) \Big] \\
&= \lim_{T \to \infty} \Big[-\eta \sum_{t=0}^{T-1} \Big(\prod_{j=t+1}^{T-1} (\boldsymbol{I} - \eta \boldsymbol{H}) \Big) \nabla_{21}^2 \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^*) \Big] && \boldsymbol{\theta}^0 = \boldsymbol{\theta}^* = \boldsymbol{\theta}^t \\
&= \lim_{T \to \infty} \Big[-\eta \sum_{t=0}^{T-1} \Big(\boldsymbol{I} - \eta \boldsymbol{H} \Big)^{T-1-t} \nabla_{21}^2 \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^*) \Big] \\
&= \lim_{T \to \infty} \Big[-\eta \sum_{t=0}^{T-1} \Big(\boldsymbol{I} - \eta \boldsymbol{H} \Big)^{t} \nabla_{21}^2 \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^*) \Big] && \text{re-index} \\
&= -\boldsymbol{H}^{-1} \nabla_{21}^2 \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \boldsymbol{\theta}^*). && \text{Neumann series and contractive}
\end{aligned}$$

Plugging back to equation 7, we will get the same expression as the hypergradient. Therefore, under the local optimality and contraction assumption, unrolling to infinity is exactly implicit differentiation: the Neumann series induced by unrolling sums to the inverse Hessian, turning GU into IFT. Our KFAC-based hypergradient can be seen as a middle ground between $\boldsymbol{I}$ and $\boldsymbol{H}$, balancing the hypergradient accuracy and the computational cost.

D BERT Data Hypercleaning

We provide additional experimental details and results for the BERT data hypercleaning experiments described in the main text. The bilevel formulation is the same as in Appendix E, with the difference that the inner model is a pretrained BERT classifier and we vary the number of trainable layers.

We consider text classification tasks from the WRENCH benchmark (Zhang et al., 2021), namely SEMEVAL, YOUTUBE, and TREC. These datasets are weak-supervision benchmarks with noisy labeling sources and are large enough to show clear performance differences between bilevel optimization methods while remaining computationally manageable for repeated BERT fine-tuning.

We use the pretrained BERT-base model (110M parameters) from `HuggingFace`. In the inner problem, we optimize with AdamW using a fixed learning rate of 10^{-5} and a cosine learning-rate scheduler. In the outer problem, we use SGD with

momentum 0.9 and perform a grid search over learning rates $\{0.01, 0.1, 1, 10, 100\}$. Each outer iteration contains 10 inner updates. For KFAC and KFAC EMP, we use damping 10^{-3}; for CG, we use 3 conjugate-gradient iterations; for Neumann, we use either 3 or 10 truncation steps; and for SAMA, we set $\alpha = 0.1$. All experiments use batch size 128 and maximum sequence length 100. We run each experiment for 2000 iterations and report averages over multiple random seeds on a single NVIDIA RTX 4090 GPU. Full numerical results are shown in Table 7.

Table 7: Comparison of Methods on TREC, SEMEVAL, and YOUTUBE at Different Tuning Levels. Each entry shows Accuracy and Training Time. Each result is averaged over 5 runs. These are the same results as Figure 4 but in numbers.

Method	**Trec**		**SemEval**		**Youtube**	
	Acc(%)	Time (s)	Acc(%)	Time (s)	Acc(%)	Time (s)
1 Layer (6.5% Params)						
Baseline	$60.12_{\pm 0.43}$	$73.3_{\pm 0.1}$	$78.94_{\pm 0.87}$	$121.2_{\pm 0.3}$	$68.53_{\pm 1.24}$	$124.6_{\pm 1.2}$
KFAC	$\mathbf{77.62}_{\pm 1.28}$	$772.8_{\pm 3.4}$	$\mathbf{83.83}_{\pm 0.47}$	$840.5_{\pm 3.8}$	$\mathbf{81.33}_{\pm 1.55}$	$955.7_{\pm 4.4}$
KFAC EMP	$74.25_{\pm 2.57}$	$791.5_{\pm 3.5}$	$81.61_{\pm 0.75}$	$838.6_{\pm 4.5}$	$80.53_{\pm 1.32}$	$962.8_{\pm 3.2}$
SAMA	$62.93_{\pm 0.84}$	$\mathbf{602.7}_{\pm 2.2}$	$81.61_{\pm 1.46}$	$\mathbf{628.0}_{\pm 5.7}$	$77.86_{\pm 1.15}$	$\mathbf{630.1}_{\pm 5.3}$
CG	$75.60_{\pm 1.85}$	$963.7_{\pm 3.8}$	$79.72_{\pm 0.91}$	$1020.3_{\pm 4.5}$	$77.42_{\pm 1.58}$	$1207.9_{\pm 3.8}$
Neumann (K=3)	$53.53_{\pm 0.75}$	$1208.8_{\pm 15.3}$	$81.22_{\pm 1.32}$	$928.5_{\pm 2.4}$	$48.89_{\pm 1.20}$	$934.7_{\pm 7.5}$
Neumann (K=10)	$70.73_{\pm 3.43}$	$1681.5_{\pm 3.6}$	$82.17_{\pm 0.36}$	$1196.8_{\pm 1.8}$	$61.47_{\pm 1.54}$	$1198.0_{\pm 3.3}$
7 Layers (45.3% Params)						
Baseline	$63.43_{\pm 0.68}$	$135.9_{\pm 0.1}$	$79.89_{\pm 0.42}$	$152.8_{\pm 0.1}$	$69.73_{\pm 0.19}$	$162.3_{\pm 3.6}$
KFAC	$\mathbf{73.42}_{\pm 1.08}$	$1414.3_{\pm 2.6}$	$\mathbf{85.61}_{\pm 0.08}$	$1475.2_{\pm 2.9}$	$\mathbf{79.92}_{\pm 1.94}$	$1491.0_{\pm 6.9}$
KFAC EMP	$72.17_{\pm 1.09}$	$1442.1_{\pm 3.0}$	$85.02_{\pm 1.03}$	$1475.4_{\pm 2.0}$	$79.24_{\pm 0.82}$	$1504.3_{\pm 5.4}$
SAMA	$63.07_{\pm 0.52}$	$\mathbf{828.3}_{\pm 4.1}$	$80.78_{\pm 0.64}$	$\mathbf{878.4}_{\pm 7.3}$	$73.43_{\pm 1.68}$	$\mathbf{873.1}_{\pm 11.9}$
CG	$68.27_{\pm 0.50}$	$2285.6_{\pm 12.2}$	$83.67_{\pm 0.62}$	$2332.4_{\pm 7.4}$	$72.43_{\pm 1.39}$	$1967.3_{\pm 6.2}$
Neumann (K=3)	$61.07_{\pm 0.34}$	$2177.0_{\pm 5.5}$	$78.61_{\pm 1.13}$	$2195.4_{\pm 9.9}$	$67.62_{\pm 4.27}$	$1897.2_{\pm 7.8}$
Neumann (K=10)	$64.87_{\pm 2.86}$	$4336.1_{\pm 8.8}$	$80.33_{\pm 2.43}$	$4327.5_{\pm 10.9}$	$64.13_{\pm 4.15}$	$4375.6_{\pm 31.0}$
12 Layers (100% Params)						
Baseline	$60.58_{\pm 0.59}$	$189.6_{\pm 0.2}$	$80.06_{\pm 1.14}$	$190.6_{\pm 0.1}$	$67.07_{\pm 0.50}$	$184.8_{\pm 0.1}$
KFAC	$63.67_{\pm 1.43}$	$1988.1_{\pm 2.9}$	$\mathbf{82.72}_{\pm 1.51}$	$1986.9_{\pm 5.4}$	$\mathbf{76.08}_{\pm 1.94}$	$2110.8_{\pm 5.5}$
KFAC EMP	$\mathbf{63.74}_{\pm 1.17}$	$2027.6_{\pm 3.1}$	$82.56_{\pm 0.83}$	$2020.9_{\pm 2.6}$	$75.80_{\pm 1.28}$	$2117.2_{\pm 6.4}$
SAMA	$59.80_{\pm 0.65}$	$\mathbf{1038.7}_{\pm 6.1}$	$78.50_{\pm 0.89}$	$\mathbf{1096.2}_{\pm 3.6}$	$71.07_{\pm 0.94}$	$\mathbf{1096.2}_{\pm 3.6}$
CG	$62.79_{\pm 1.61}$	$4609.8_{\pm 4.3}$	$82.32_{\pm 1.23}$	$4618.3_{\pm 5.2}$	$73.56_{\pm 1.15}$	$4607.2_{\pm 5.9}$
Neumann (K=3)	$61.42_{\pm 2.01}$	$3823.2_{\pm 6.2}$	$81.92_{\pm 0.92}$	$3828.4_{\pm 5.3}$	$71.78_{\pm 1.42}$	$3825.4_{\pm 1.37}$
Neumann (K=10)	$61.53_{\pm 1.75}$	$8729.2_{\pm 6.3}$	$82.53_{\pm 1.24}$	$8727.9_{\pm 5.7}$	$72.34_{\pm 1.24}$	$8735_{\pm 6.2}$

E Data Hypercleaning

The training dataset, denoted as $\mathcal{D}^{\text{tr}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$, is corrupted by randomly assigning labels to a proportion of the data points. The objective is to assign weights to the training data points such that a model trained on the weighted training set performs optimally on the validation dataset $\mathcal{D}^{\text{val}}$. This process can be formulated as:

$$\min_{\boldsymbol{\lambda}, \boldsymbol{\theta}} \mathcal{L}^{\text{val}}(\boldsymbol{\theta}^\star), \quad \text{s.t.} \quad \boldsymbol{\theta}^\star \in \arg\min_{\boldsymbol{\theta}} \left\{ \mathcal{L}^{\text{tr}}(\boldsymbol{\theta}, \boldsymbol{\lambda}) + \alpha \|\boldsymbol{\theta}\|^2 \right\},$$

where $\mathcal{L}^{\text{val}}(\boldsymbol{\theta}^\star)$ is the cross-entropy loss evaluated on the validation set. $\mathcal{L}^{\text{tr}}(\boldsymbol{\theta}, \boldsymbol{\lambda})$ is the weighted training loss, defined as: $\mathcal{L}^{\text{tr}}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \sum_{n=1}^N \sigma(\lambda_n) \ell(\boldsymbol{x}_n, y_n, \boldsymbol{\theta})$,where $\sigma(\lambda_n) = \text{Clip}(\lambda_n, [0, 1])$ ensures that the weights λ_n remain in the range $[0, 1]$. The training loss $\ell(\boldsymbol{x}_n, y_n, \boldsymbol{\theta})$ is also computed using cross-entropy. The term $\alpha \|\boldsymbol{\theta}\|^2$ adds an L_2 regularization to prevent overfitting. The goal is to optimize both the data weights $\boldsymbol{\lambda}$ and the model parameters $\boldsymbol{\theta}$ to achieve the best validation performance.

The overall structure of our code was adapted from `https://github.com/Cranial-XIX/BOME`. However, in BOME's implementation, their model was implemented as a tensor using Pytorch. In our implementation, we imple-

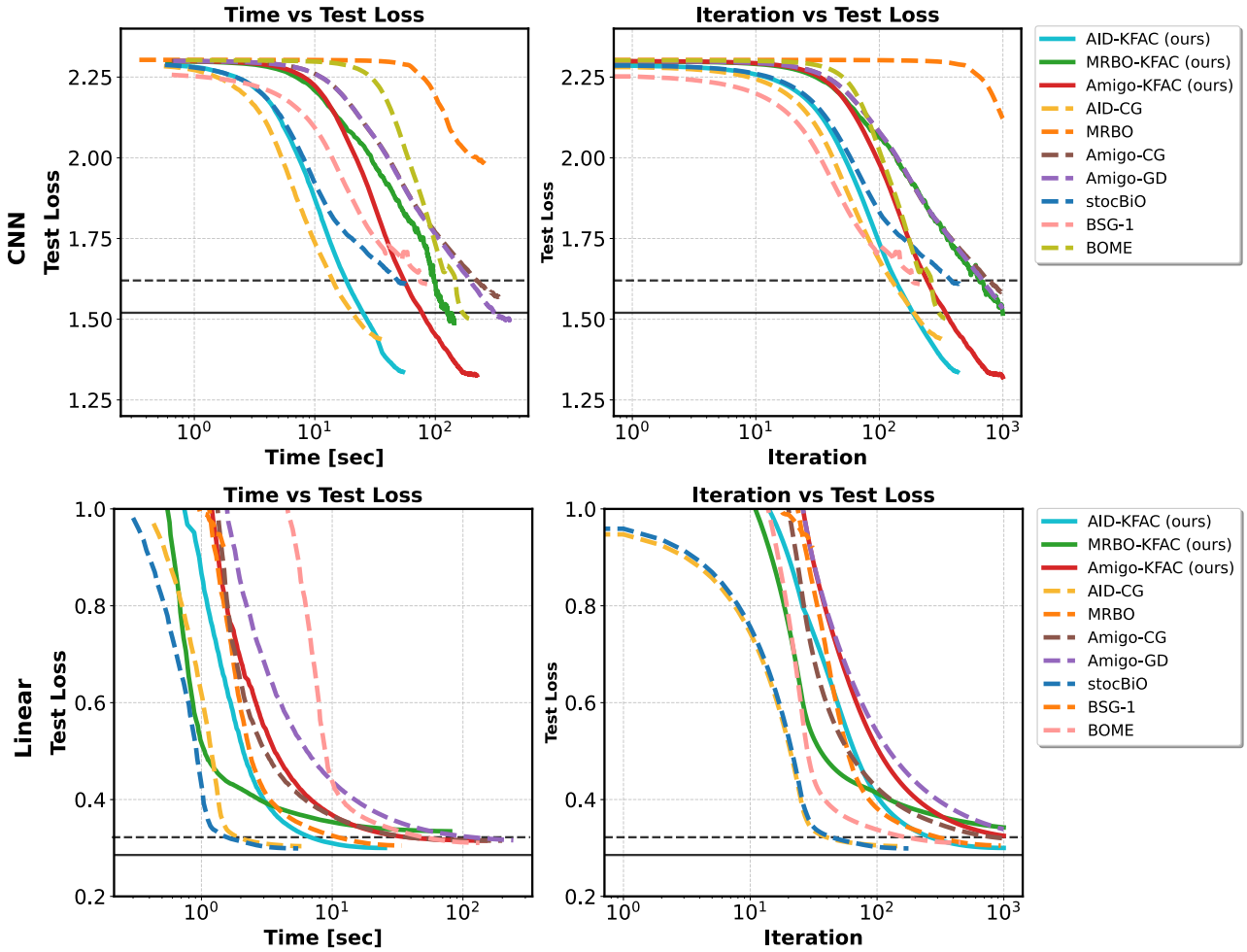


Figure 5: Data hypercleaning results for CNN on CIFAR-10 and linear on MNIST. For the linear model, Solid black line is the test loss reached on the cleaned dataset, while the dashed black line is trained only on validation set. For CNN, solid black line is test loss reached on validation dataset, while dashed line is trained on the validation dataset and noisy dataset.

mented LR, MLP and CNN using `torch.nn.Module`, which is more practical. However, this will slow down first-order methods like BSG-1 and BOME since we need to convert each layer of the model to a vector.

In the experiment, we split the dataset (both MNIST and CIFAR-10) into 4 parts. The training data size is 50000, and the validation data 1 size is 5000, the validation data 2 size is 5000, and the test data size is 10000. We train the models on training data, update the weight v according to validation data 1, and tune the other hyperparameters on validation data 2. We train on MNIST (linear) and CIFAR-10 (three-layer CNN, ResNet-18) with 50% label corruption, and we report the results on the test data. We fix the number of inner steps to 10. For the step sizes, we conducted a grid search over inner learning rate $\eta_{\boldsymbol{\theta}} \in \{0.1, 0.01, 0.001\}$ and outer learning rate $\eta_{\mathbf{w}} \in \{0.1, 1, 10, 100, 500, 1000\}$, and whether to apply momentum by searching the momentum value over $\{0, 0.9\}$. For BOME, BVSFM and BSG-1, we also search their hyperparameter over $\{0.001, 0.01, 0.1, 1\}$. For KFAC, we also grid search the damping value over $\{0.001, 0.01, 0.1, 1, 10, 100\}$. We report the results for MNIST and CIFAR in Figure 5. We can see that our method not only results in a faster convergence rate but also helps to converge to a better optimum point.

F More Applications

In this section, we introduce the detailed problem formulations and experimental details for the four applications we consider. For the class imbalance and the continued pretraining problem, our experiment code is based on the `Betty` library (Choe et al., 2023b) , which provides a template to add more hypergradient estimation methods.

F.1 Class Imbalance

Following Shu et al. (2019), we aim to train a Meta-Weight-Net (MWN) that assigns weights to different training samples to achieve good performance on the validation set. We can formulate the problem as:

$$\min_{\boldsymbol{\lambda}} \mathcal{L}^{\text{meta}}(\boldsymbol{\theta}^*),\ \text{s.t. } \boldsymbol{\theta}^* \in \arg\min_{\boldsymbol{\theta}} \sum_n w(\ell_n(\boldsymbol{x}_n, y_n, \boldsymbol{\theta}), \boldsymbol{\lambda}) \cdot \ell_n(\boldsymbol{x}_n, y_n, \boldsymbol{\theta}),$$

where $w(\cdot, \boldsymbol{\lambda})$ is MWN parametrized by $\boldsymbol{\lambda}$, taking the training loss of the n-th data point, and returning a weight. The $\mathcal{L}^{\text{meta}}$ is the loss on a small meta-set. We construct the long-tailed CIFAR dataset by reducing the number of training samples per class using an exponential function $n = n_i \mu^i$, where i is the class index, n_i is the original number of training samples. We train ResNet-32 with softmax cross-entropy loss and SGD, with momentum 0.9, a weight decay of 5×10^{-4} and a learning rate 0.1. The outer learning rate is set to 10^{-5} with the Adam optimizer. For SAMA and KFAC, we train the inner problem for 10 steps per outer iteration and set SAMA $\alpha = 0.1$ and damping value for KFAC to be 10^{-3}. We train the model for 100 epochs, and we randomly select 10 images per class on the validation set as the meta dataset. For the $T_1 - T_2$ method, we use the code in `https://github.com/xjtushujun/Meta-weight-net_class-imbalance`.

F.2 Continued Pretraining

We further evaluate our method on the continued pretraining task for large language models, following Dery et al. (2022) and Choe et al. (2023a). Continued pretraining aims to adapt a pretrained language model to domain-specific data while avoiding negative transfer from low-quality samples. Choe et al. (2023a) formulate the continued pretraining task as BO:

$$\min_{\boldsymbol{\lambda}} \mathcal{L}^{\text{ft}}(\mathcal{D}_{\text{ft}}, \boldsymbol{\theta}^*), \quad \text{s.t.} \quad \boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \Big[\mathcal{L}^{\text{ft}}(\mathcal{D}_{\text{ft}}, \boldsymbol{\theta}) + \frac{1}{|\mathcal{D}_{\text{pt}}|} \sum_{\boldsymbol{x} \in \mathcal{D}_{\text{pt}}} w(\boldsymbol{x}, \boldsymbol{\lambda}) \mathcal{L}^{\text{pt}}(\boldsymbol{x}; \boldsymbol{\theta}) \Big],$$

where $\mathcal{L}^{\text{ft}}$ and $\mathcal{L}^{\text{pt}}$ are finetuning and pretraining loss functions (respectively), $\mathcal{D}_{\text{ft}}$ and $\mathcal{D}_{\text{pt}}$ are finetuning and pretraining datasets (respectively), and $w(\cdot, \boldsymbol{\lambda})$ is the MWN similar to class imbalance, which dynamically reweighs samples from the pretraining corpus $\mathcal{D}_{\text{pt}}$, mitigating the negative effect of noisy or low-quality data.

We use RoBERTa-base as the backbone for both the downstream and auxiliary pretraining tasks; each task is optimized with Adam at an initial learning rate of 2×10^{-5} and a linear decay schedule with 0.6 warmup proportion. No weight decay is applied. The meta-network, implemented as a two-layer MLP, is optimized with Adam at a learning rate of 1×10^{-5} without a scheduler. For SAMA and our KFAC-integrated variant, we use 10 inner steps per outer iteration and set SAMA $\alpha = 0.3$ and KFAC damping value to be 10^{-3}. All experiments were conducted on the ChemProt, HyperPartisan, ACL-ARC, and SciERC datasets, with a batch size of 16 and a maximum sequence length of 256.

F.3 Data Poisoning

Data poisoning attacks aim to inject a small fraction of poisoned samples into the training dataset, thereby degrading the performance of the model trained on the corrupted data. As described by Lu et al. (2022), this problem can be expressed as a BO problem:

$$\max_{\mathcal{D}^{\text{p}}} \mathcal{L}^{\text{val}}(\mathcal{D}^{\text{val}}, \boldsymbol{\theta}^*), \quad \text{s.t.} \quad \boldsymbol{\theta}^* \in \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{\text{tr}}(\mathcal{D}^{\text{tr}} \cup \mathcal{D}^{\text{p}}, \boldsymbol{\theta}),$$

Where $\mathcal{L}^{\text{tr}}$ and $\mathcal{L}^{\text{val}}$ represent the training and validation losses, which can be implemented as the cross-entropy loss, $\mathcal{D}^{\text{p}}$ denotes the poisoned data injected into the training set. The goal is to optimize the poisoned data $\mathcal{D}^{\text{p}}$ such that a model trained on the combined dataset $\mathcal{D}^{\text{tr}} \cup \mathcal{D}^{\text{p}}$ performs poorly on the validation dataset $\mathcal{D}^{\text{val}}$. The implementation of TGDA was adapted from the repo: `https://github.com/watml/TGDA-Attack`.

During training time, we use generator models (attackers) to generate poisoned samples from random noise. The attacker model for the MNIST dataset is a three-layer neural network, with three fully connected layers and leaky ReLU activations; for the CIFAR-10 dataset, we use an autoencoder with three convolutional layers as encoder and three conv transpose layers as decoder. For the classifier on the MNIST dataset, we used logistic regression (LR) and Multi-layer Perceptron (MLP), while on the CIFAR-10 dataset, we used a convolutional neural network (CNN) with two convolutional layers, maxpooling and one fully connected layer.

In the data poisoning training setup, we used a batch size of 1000 for all methods on the MNIST dataset. Even for deterministic methods such as BOME AID-CG, batch processing was employed due to the computational expense of

alternative approaches. For AID-KFAC, the damping value was set to 10^{-3}. The models were trained for 200 epochs, with the attacker's learning rate set to 0.1 and the classifier's learning rate set to 0.01. Following Lu et al. (2022), for each update of the attacker, the classifier was updated 20 times. On the CIFAR-10 dataset, we used a batch size of 256 for all methods and trained for 200 epochs. The learning rates were set to 0.1 for the upper level (attacker) and 0.01 for the classifier. Both the attacker and classifier optimizers utilized SGD. For stocBiO, we set the Neumann series parameter $K = 3$. For BOME, the upper-level learning rate was 0.1, while the lower-level learning rate was 0.01. As outlined by Lu et al. (2022), pretraining the attacker and classifier was necessary to ensure convergence. The attacker was pretrained using SGD for 100 epochs with a learning rate of 0.1, utilizing MSE reconstruction loss. Similarly, the classifier was pretrained using SGD with a learning rate of 0.1 for 100 epochs.

F.4 Unlearnable Example

We follow the problem formulation of GUE (Liu et al., 2024). On a dataset $\mathcal{D}^{\text{tr}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$, an attacker (leader) employs a generator model to produce perturbations, and a classifier (follower) mimics the victim, with only access to the poisoned data, namely clean data plus the imperceptible perturbations. The classifier minimizes the cross-entropy loss on the poisoned data, but the resulting model exhibits poor performance on clean validation data. Formally, the objective is as follows:

$$\min_{\boldsymbol{\lambda},\boldsymbol{\theta}} \mathcal{L}^{\text{val}}(\boldsymbol{\theta}), \quad \text{s.t.} \quad \boldsymbol{\theta}^* \in \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{\text{tr}}(\boldsymbol{\theta}, \boldsymbol{\lambda}),$$

where $\mathcal{L}^{\text{val}}(\boldsymbol{\theta}) = -\frac{1}{m}\sum_n \ell_n(\boldsymbol{x}_n, y_n, \boldsymbol{\theta})$, $\mathcal{L}^{\text{tr}}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \frac{1}{m}\sum_n \ell_n(\boldsymbol{x}_n + g_{\boldsymbol{\lambda}}(\boldsymbol{x}_n), y_n; \boldsymbol{\theta})$, ℓ_n is the cross-entropy loss, and $g_{\boldsymbol{\lambda}}$ represents the generator that produces small perturbations for each training example $\boldsymbol{x}_i$. To address gradient explosion issues caused by the unbounded outer objective function, Liu et al. (2024) proposed replacing $\mathcal{L}^{\text{val}}$ by the following surrogate loss during training, for each pair of samples $(\boldsymbol{x}, y)$:

$$\mathcal{L}(\boldsymbol{x}, y; \boldsymbol{\theta}) \coloneqq \max_{y' \neq y} \ell_n(\boldsymbol{x}, y'; \boldsymbol{\theta}).$$

Minimizing this surrogate loss means minimizing the cross-entropy with respect to all other labels instead of y. Liu et al. (2024) also showed this loss is bounded and has the same monotonicity as regular cross-entropy.

The framework implementation and BOME implementation were adapted from the repo: `https://github.com/hong-xian/gue/tree/master/attacks`, while the implementation of AID-CG and AID-KFAC is the same as in data poisoning.

The dataset is CIFAR-10, with 50000 clean training samples and 10000 test samples. For unlearnable examples, we use generator models to generatee a small perturbation for each input image. We used the U-Net as a poison generator during training. We clip the perturbation radius to $\epsilon = 25/255$, since below this threshold we found that most BO algorithms fail to generate effective perturbations that can significantly downgrade the performance of the classifier. For the classifier model, we used a three-layer CNN with ReLU activation and two fully connected layers.

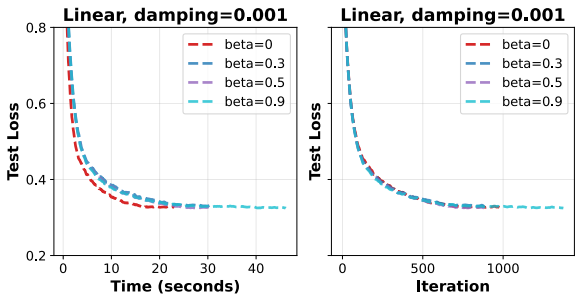
During training, we train 50 epochs for attackers which generate small perturbations. We used a batch size of 512 for training, with the attacker's learning rate set to 1 and the classifier's learning rate set to 0. We used the SGD optimizer for both the attacker and classifer. And we update the attacker 10 times after updating the classifier 10 times. However, we found that for BOME, the batch size 512 fails to generate perturbations that can significantly downgrade the performance of the classifier, so we used batch size 256 for BOME. For the inner approximation of BOME, we used Adam (Kingma & Ba, 2015) with a learning rate of 0.001 for $T = 10$ iterations. For evaluation, we train the model on a poisoned dataset for 100 epochs using an SGD optimizer with an initial learning rate of 0.01 and test the classification accuracy on the test set.

G Ablation Study

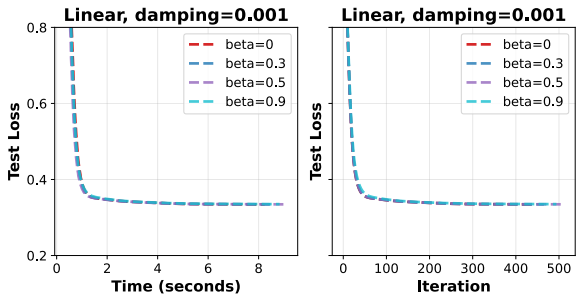
G.1 Exponential Moving Average (EMA)

Martens & Grosse (2015) suggest maintaining running estimates of $\boldsymbol{A}_{\text{KFAC}}$ and $\boldsymbol{B}_{\text{KFAC}}$ using an exponential moving average (EMA), which can smooth stochastic curvature estimates across iterations. After computing new mini-batch estimates $\boldsymbol{A}_{\text{KFAC}}^{\text{new}}$ and $\boldsymbol{B}_{\text{KFAC}}^{\text{new}}$, the running factors are updated as

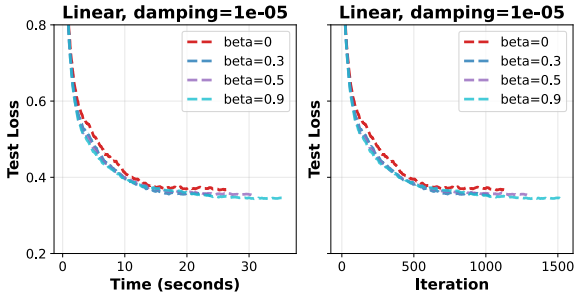
$$\boldsymbol{A}_{\text{KFAC}} \leftarrow \beta \boldsymbol{A}_{\text{KFAC}} + (1-\beta)\boldsymbol{A}_{\text{KFAC}}^{\text{new}}, \qquad \boldsymbol{B}_{\text{KFAC}} \leftarrow \beta \boldsymbol{B}_{\text{KFAC}} + (1-\beta)\boldsymbol{B}_{\text{KFAC}}^{\text{new}},$$



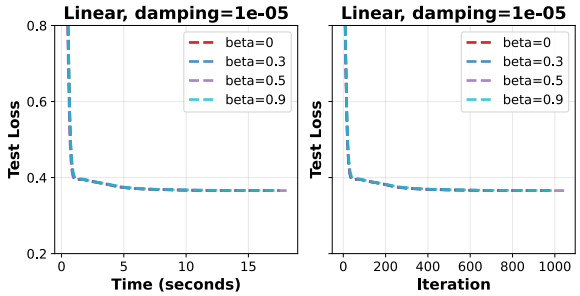
(a) damping = 0.001



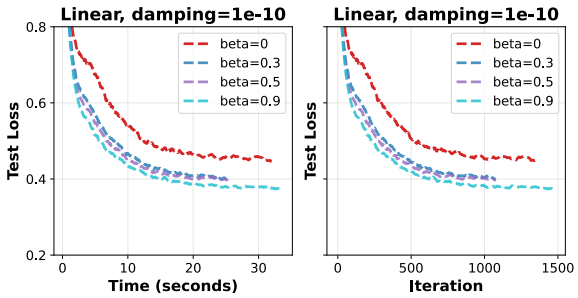
(b) damping = 0.001



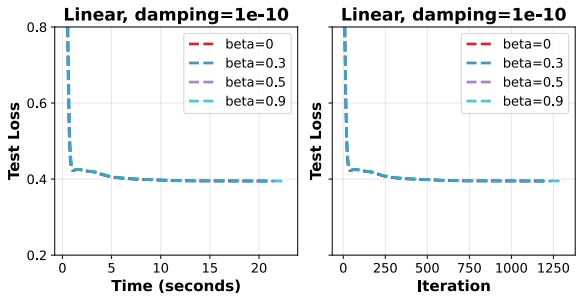
(c) damping = 10^{-5}



(d) damping = 10^{-5}



(e) damping = 10^{-10}



(f) damping = 10^{-10}

Figure 6: Effect of EMA coefficient β on convergence for data hypercleaning with logistic regression on MNIST. Each curve corresponds to a different $\beta \in \{0, 0.3, 0.5, 0.9\}$. **Left column**: batch size 1000. **Right column**: batch size 50000. In this setting, EMA is most helpful for small damping and small batch sizes, where curvature estimates are noisier, while its effect is modest for larger batches or stronger damping.

where $\beta \in [0, 1)$ controls the amount of temporal smoothing.

To study the role of EMA in BO, we evaluate data hypercleaning with logistic regression on MNIST, varying the batch size $B \in \{1000, 50000\}$, the EMA coefficient $\beta \in \{0, 0.3, 0.5, 0.9\}$, and the damping strength. The results are shown in Figure 6. In this setting, EMA is most beneficial when both the batch size and damping are small, where curvature estimates are noisier and temporal averaging can stabilize the KFAC factors. By contrast, for larger batches or stronger damping, the performance differences across β are small, suggesting that EMA provides only limited additional benefit once the curvature estimates are already stable.

Overall, these results support EMA as a useful stabilization mechanism in noisier BO regimes, rather than as a uniformly necessary component. Since our main experiments do not operate in this particularly noisy small-batch regime, we use the simpler non-EMA variant there. We note, however, that these conclusions are drawn from a small-scale linear model, and EMA may still be valuable in larger or more stochastic settings.

G.2 Batch Size

We compare AID-CG and AID-KFAC under varying batch sizes, following the same grid search setup described in Appendix E. Specifically, we consider batch size $B = \{256, 512, 1024, 2048, 5000\}$ for ResNet-18 and $B = \{128, 1000, 5000, 10000, 50000\}$ for the linear model and consider the lowest test loss achieved within 1000 optimization iterations. We report the extra results in Figure 7. The results show that batch size has a noticeable effect on BO performance. In both the ResNet-18 and linear settings, AID-KFAC consistently matches or outperforms AID-CG across all tested batch sizes, with the largest gains typically appearing in the smaller-batch regime. As the batch size increases, the performance gap narrows, especially for the linear model, but AID-KFAC remains competitive throughout. These results suggest that KFAC provides a robust curvature surrogate across a wide range of practical batch-size settings.

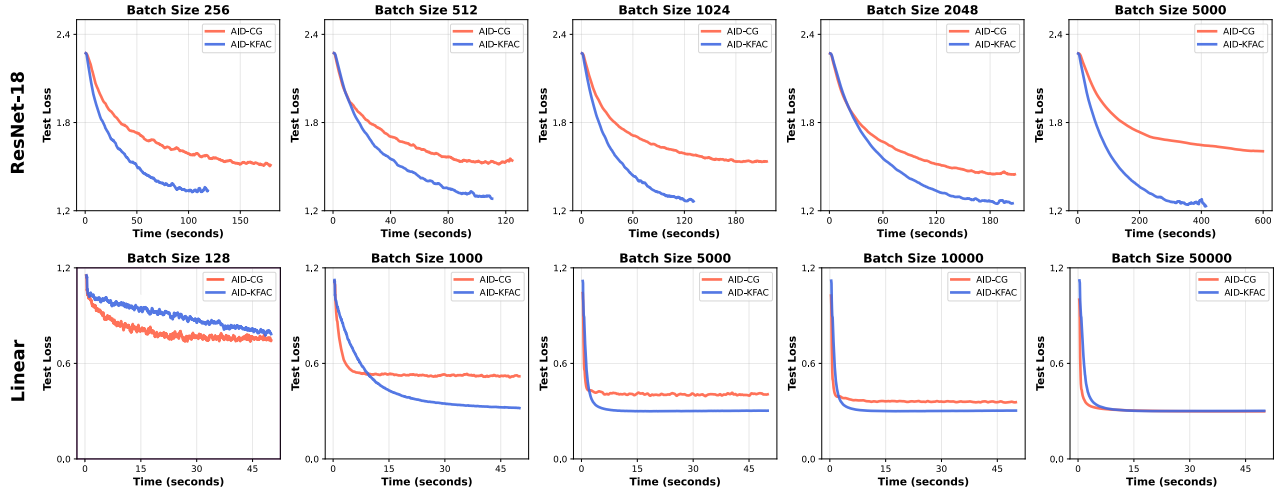


Figure 7: Convergence of AID-CG and AID-KFAC on data hypercleaning for different batch sizes. KFAC achieves faster and more stable convergence, particularly for smaller batches, where curvature estimates are noisier and the inner problem becomes ill-conditioned. Compared with Figure 3, this figure shows the evolution of the test loss for different batch sizes.