# Penalty Method for Inversion-Free Deep Bilevel Optimization

**Akshay Mehra**
Department of Computer Science
Tulane University
amehra@tulane.edu

**Jihun Hamm**
Department of Computer Science
Tulane University
jhamm3@tulane.edu

## Abstract

Bilevel optimization problems are at the center of several important machine learning problems such as hyperparameter tuning, data denoising, meta- and few-shot learning, and training-data poisoning. Different from simultaneous or multi-objective optimization, the steepest descent direction for minimizing the upper-level cost requires the inverse of the Hessian of the lower-level cost. In this paper, we propose a new method for solving bilevel optimization problems using the classical penalty function approach which avoids computing the inverse and can also handle additional constraints easily. We prove the convergence of the method under mild conditions and show that the exact hypergradient is obtained asymptotically. Our method's simplicity and small space and time complexities enable us to effectively solve large-scale bilevel problems involving deep neural networks. We present results on data denoising, few-shot learning, and training-data poisoning problems in a large scale setting and show that our method outperforms or is comparable to previously proposed methods based on automatic differentiation and approximate inversion in terms of accuracy, run-time and convergence speed.

## 1 Introduction

Solving a bilevel optimization problem is crucial for the fields of study which involve a competition between two parties or two objectives. Particularly, a bilevel problem arises if one party makes its choice first affecting the optimal choice for the second party, known as the Stackelberg model dating back to 1930's [34]. The general form of a bilevel optimization problem is

$$\min_{u\in\mathcal{U}} f(u,v) \quad \text{s.t.} \quad v = \arg\min_{v\in\mathcal{V}(u)} g(u,v) \tag{1}$$

The 'upper-level' problem $\min_{u\in\mathcal{U}} f(u,v)$ is a usual minimization problem except that $v$ is constrained to be the solution to the 'lower-level' problem $\min_{v\in\mathcal{V}(u)} g(u,v)$ which is dependent on $u$ (see [3] for a review of bilevel optimization). In this work we propose and analyze a new method for solving bilevel problems using the classical penalty function method. We also demonstrate that it outperforms existing methods in important machine learning applications including gradient-based hyperparameter tuning [7, 18, 16, 24, 9, 10], data denoising by importance learning [15, 35, 27], meta/few-shot learning [26, 28, 33, 9, 20, 31, 10, 25], and training-data poisoning [19, 21, 13, 30]. In the following we explain how these applications can be written as bilevel optimization problems.

**Gradient-based hyperparameter tuning.** Searching for optimal hyperparameters is an indispensable step for any machine learning problem and grid search is a popular method when domain of the hyperparameters is a discrete set or a range. However, when losses are differentiable functions of the hyperparameter(s), a continuous bilevel optimization problem can help find the optimal hyperparameters. Let $u$ and $w$ be hyperparameter(s) and parameter(s) for a class of learning algorithms, $h(x;u,w)$ be the hypothesis, $L_{\text{val}}(u,w) = \frac{1}{N_{\text{val}}}\sum_{(x_i,y_i)\in\mathcal{D}_{\text{val}}} l(h(x_i;u,w),y_i)$ and $L_{\text{train}}(u,w)$ be the loss on

Preprint. Under review.

validation and training sets, respectively. Then the best hyperparameter(s) $u$ is the solution to

$$\min_u \; L_{\text{val}}(u, w) \;\; \text{s.t.} \;\; w = \arg\min_w \; L_{\text{train}}(u, w). \tag{2}$$

**Data denoising by importance learning.** Most learning algorithms assume that the training set is an i.i.d. sample from the same distribution as the test set. However, if train and test distributions are not identical or if the training set is corrupted by noise or modified by adversaries, the assumption is violated. In such cases, changing the importance of each training example, before training, can reduce the discrepancy between the two distributions. For example, importance of the examples from the same distribution (non-corrupted examples) can be up-weighted in comparison to other examples. Determining the correct weight for each training example can be formulated as a bilevel problem. Let $u$ be the vector of non-negative importance values for each training example $u = [u_1, \cdots, u_N]^T$ where $N$ is the number of training examples, $w$ be the parameter(s) of a classifier $h(x; w)$. Assuming access to small set of validation data, from the same distribution as the test data, and $L_{\text{val}}(u, w) = \frac{1}{N_{\text{val}}} \sum_{(x_i, y_i) \in \mathcal{D}_{\text{val}}} l(h(x_i; u, w), y_i)$ be the loss on validation set, the importance learning problem is

$$\min_u \; L_{\text{val}}(u, w) \;\; \text{s.t.} \;\; w = \arg\min_w \; \frac{1}{\sum_i u_i} \sum_{(x_i, y_i) \in \mathcal{D}_{\text{train}}} u_i l(h(x_i; w), y_i). \tag{3}$$

**Meta-learning.** Meta-learning is a problem of learning a prior on the hypothesis classes (a.k.a. inductive bias) for a given set of tasks. Few-shot learning is an example of meta-learning, where a learner is trained on several related tasks, during the meta-training phase, so it generalizes well on unseen (but related) tasks during the meta-testing phase. An effective approach to this problem is to learn a common representation for various tasks and train task specific classifiers over this representation. Let $T$ be the map that takes raw features to a common representation $T : \mathcal{X} \to \mathbb{R}^d$ for all tasks and $h_i$ be the classifier for the $i$-th task, $i \in \{1, \cdots, M\}$ where $M$ is the total number of tasks for training. The goal is to learn both the representation map $T(\cdot\,; u)$ parameterized by $u$ and the set of classifiers $\{h_1, \cdots, h_M\}$ parameterized by $w = \{w_1, \cdots, w_M\}$. Let $L_{\text{val}}(u, w_i) := \frac{1}{N_{\text{val}}} \sum_{(x_i, y_i) \in \mathcal{D}_{\text{val}}} l(h_i(T(x_i; u); w_i), y_i)$ be the validation loss of task $i$ and $L_{\text{train}}(u, w_i)$ be the training loss defined similarly, then the bilevel problem for few-shot learning is

$$\min_u \; \sum_i L_{\text{val}}(u, w_i) \;\; \text{s.t.} \;\; w_i = \arg\min_{w_i} \; L_{\text{train}}(u, w_i), \;\; i = 1, \cdots, M. \tag{4}$$

At test time the common representation $T(\cdot\,; u)$ is kept fixed and the classifiers $h_i^{'}$ for the new tasks are trained i.e. $\min_{w_i^{'}} \; L_{\text{test}}(u, w_i^{'}) \;\; i = 1, \cdots, N$ where $N$ is the total number of tasks for testing.

**Training-data poisoning.** Training-data poisoning refers to the setting in which an adversary can modify the training data so that the model trained on the altered data performs poorly/differently compared to one trained on the unaltered data. Attacker adds one or more 'poisoned' examples $u = \{u_1, \cdots, u_M\}$ to the original training data $X = \{x_1, \cdots, x_N\}$ i.e., $X' = X \bigcup u$ with arbitrary labels. Additionally, to evade detection, attacker can generate poisoned images starting from an existing clean images (called base image) with a bound on the maximum perturbation allowed. Let the validation set consist of target images that an attacker would like to misclassify, let $L_{\text{poison}}(u, w) := \frac{1}{N} \sum_{(x_i, y_i) \in X' \times Y'} l(h(x_i; u, w), y_i)$ be the loss on the poisoned training data and $\epsilon$ be the bound on the maximum perturbation allowed for poisoned points. Then the problem of generating such poisoning points can be formulated as

$$\min_u \; L_{\text{val}}(u, w) \;\; \text{s.t.} \;\; \|x_{\text{base}} - u\|_2 < \epsilon \;\; \text{and} \;\; w = \arg\min_w \; L_{\text{poison}}(u, w), \tag{5}$$

**Challenges of deep bilevel optimization.** General bilevel optimization problems cannot be solved using simultaneous optimization of the upper- and lower-level cost and are in fact, known to be NP-hard even in cases with linear upper-level and quadratic lower-level functions [2]. Moreover, recent deep learning models, with millions of variables, only permit the use of first-order methods such as gradient descent. However, for bilevel problems, even the first-order methods are difficult to apply since they require computation of the inverse Hessian–gradient product to get the exact hypergradient (see Sec. 2.1). Since direct inversion of the Hessian is impractical even for moderate-sized problems, previous approaches approximate the exact hypergradient using forward/reverse-mode differentiation [18, 9, 29] or approximate inversion by solving a linear system [7, 24, 25]. However, these approaches have high space and time complexities which is problematic especially in deep learning settings.

**Contributions.** We propose an algorithm (Penalty) based on the classical penalty function (Alg. 1) for solving large-scale bilevel optimization problems which can also handle upper-level constraints. We prove convergence of the method under mild conditions (Theorem 2) and show that it computes the exact hypergradient asymptotically (Lemma 3). We present complexity analysis of our algorithm to show that it has linear time and constant space complexity (Table 1), making it superior to forward/reverse-mode differentiation and similar to the approximate inversion based methods. Smaller space and time complexity of our approach enables us to effectively solve large-scale bilevel problems involving deep neural networks (Table 5 in Appendix F). We apply Penalty to several machine learning problems such as data denoising, few-shot learning, and training-data poisoning. In addition to being able to solve constrained problems, our method also performs competitively to the state-of-the-art methods on simpler problems (with convex lower-level cost) and significantly outperforms other methods on complex problems (with non-convex lower-level cost), in terms of accuracy (Sec. 3), runtime (Table 4 and Fig. 5 in Appendix E) and convergence speed (Fig. 4 in Appendix D) demonstrating that it is an effective solver for various bilevel problems.

The rest of the paper is organized as follows. We present and analyze the main algorithm in Sec. 2, perform comprehensive experiments in Sec. 3, and conclude in Sec. 4. The proofs, experimental settings and additional results are presented in the appendix. All codes are available at `https://github.com/jihunhamm/bilevel-penalty`.

## 2 Inversion-Free Penalty Method

In this work we assume that upper- and lower-level costs $f$ and $g$ are twice continuously differentiable and the constraint function $h$ is continuously differentiable in both $u$ and $v$. We use $\nabla_u f$ and $\nabla_v f$ to denote gradient vectors, $\nabla^2_{uv} f$ for the matrix $\left[\frac{\partial^2 f}{\partial u_i \partial v_j}\right]$, and $\nabla^2_{vv} f$ for the Hessian matrix $\left[\frac{\partial^2 f}{\partial v_i \partial v_j}\right]$. Following previous works we assume that the lower-level solution $v^*(u) := \arg\min_v \ g(u,v)$ is unique for all $u$ and that $\nabla^2_{vv} g$ is invertible everywhere.

### 2.1 Background

**Hypergradient for bilevel optimization.** If we can express the solution to the lower-level problem $v^*(u) := \arg\min_v g(u,v)$ explicitly., e.g., in a closed form, then the bilevel problem can be reduced to a single-level problem $\min_u \ f(u, v^*(u))$. Using a gradient-based approach on this single-level problem we can compute the total derivative $\frac{df}{du}(u, v^*(u))$, also called the hypergradient. Then by the chain rule, we get $\frac{df}{du} = \nabla_u f + \frac{dv}{du} \cdot \nabla_v f$ at $(u, v^*(u))$. Even if $v^*(u)$ cannot be found explicitly, we can still compute $\frac{dv}{du}$ using the implicit function theorem. As $\nabla_v g = 0$ at $v = v^*(u)$ and $\nabla^2_{vv} g$ is invertible we get $du \cdot \nabla^2_{uv} g + dv \cdot \nabla^2_{vv} g = 0$, and $\frac{dv}{du} = -\nabla^2_{uv} g (\nabla^2_{vv} g)^{-1}$, so the hypergradient is

$$\frac{df}{du} = \nabla_u f + \frac{dv}{du} \nabla_v f = \nabla_u f - \nabla^2_{uv} g (\nabla^2_{vv} g)^{-1} \nabla_v f \ \text{ at } \ (u, v^*(u)) \tag{6}$$

Existing approaches [7, 18, 24, 9, 29, 25] can be viewed as implicit methods of approximating the hypergradient, with distinct trade-offs in efficiency and complexity.

**Classical penalty function approach.** A bilevel problem can be considered as a constrained optimization problem since the lower-level optimality $v^*(u) = \arg\min_v g(u,v)$ is a constraint along with possible additional constraints in the upper- and lower-levels. For simple problems without additional constraints, we can replace the lower-level problem by its necessary condition for optimality resulting in the following problem $\min_{u,v} f(u,v), \ \text{s.t.} \ \nabla_v g(u,v) = 0$. The penalty function method is a well-known approach for solving such constrained optimization problems (see [5] for a review). It has been previously applied to bilevel problems under strict assumptions with only high-level descriptions of the algorithm presented [1, 12]. The penalty function that is minimized has the form $\tilde{f}(u,v;\gamma) := f(u,v) + \frac{\gamma}{2}\|\nabla_v g(u,v)\|^2$ which is the sum of the original cost $f$ and the penalty term for lower-level optimality. Assuming $(\hat{u}_k, \hat{v}_k)$ is the minimizer of the penalty function $\tilde{f}(u,v;\gamma)$ for a given $\gamma = \gamma_k$: $(\hat{u}_k, \hat{v}_k) = \arg\min_{u,v} \tilde{f}(u,v;\gamma_k)$, then the following result holds.

**Theorem 1** (Simplified Theorem 8.3.1 of [3])**.** *Assume $f$ and $g$ are convex in $v$ for any fixed $u$. Let $\{\gamma_k\}$ be any positive ($\gamma_k > 0$) and divergent ($\gamma_k \to \infty$) sequence. If $\{(\hat{u}_k, \hat{v}_k)\}$ is the corresponding sequence of optimal solutions of $\arg\min_{u,v} \tilde{f}(u,v;\gamma_k)$, then the sequence $\{(\hat{u}_k, \hat{v}_k)\}$ has limit points any one of which is a solution to $\min_{u,v} \ f(u,v), \ \text{s.t.} \ v = \arg\min_v g(u,v)$.*

### 2.2 Our approach

Theorem 1 presents a strong result, however its not very practical, specially for deep learning settings, since the minimizer $(\hat{u}_k, \hat{v}_k)$ cannot be computed exactly for each $\gamma_k$ and $f$ and $g$ need not be convex in $v$ for any $u$. In this work we present a new method for solving bilevel problems, which can also handle upper-level constraints[1], which has not been explored by existing bilevel solvers in machine learning. Concretely, we deal with

$$\min_{u,v} \ \ f(u,v), \ \ \text{s.t.} \ \ h(u,v) = 0 \ \ \text{and} \ \ v^*(u) = \arg\min_v g(u,v). \tag{7}$$

Inequality constraints $h(u,v) \leq 0$ can also be handled by using a slack variable $s$ and an equality constraint $h(u,v) + s^2 = 0$. Using the assumption of unique lower-level solution for each $u$ we can convert the bilevel problem in Eq. (7) into the following single level constrained problem:

$$\min_{u,v} \ \ f(u,v), \ \ \text{s.t.} \ \ h(u,v) = 0 \ \ \text{and} \ \ \nabla_v g = 0. \tag{8}$$

This problem can then be solved using the penalty function method as follows

$$(\hat{u}_k, \hat{v}_k) = \arg\min_{u,v} \left[ \tilde{f}(u,v;\gamma_k) := f(u,v) + \frac{\gamma_k}{2}(\|h(u,v)\|^2 + \|\nabla_v g(u,v)\|^2) \right], \tag{9}$$

To guarantee convergence in practical deep learning settings, we allow non-convexity of $f$ and $\epsilon_k$-optimal (instead of exact) solution to Eq. (9) at each $k$. We proof the convergence of our method to a KKT point of Eq. (8) assuming that linear independence constraint qualification (LICQ) is satisfied at the optimum, i.e., linear independence of the gradients of the constraints ($h$ and $\nabla_v g$).

**Theorem 2.** *Suppose $\{\epsilon_k\}$ is a positive ($\epsilon_k > 0$) and convergent ($\epsilon_k \to 0$) sequence, $\{\gamma_k\}$ is a positive ($\gamma_k > 0$), non-decreasing ($\gamma_1 \leq \gamma_2 \leq \cdots$), and divergent ($\gamma_k \to \infty$) sequence. Let $\{(u_k, v_k)\}$ be the sequence of approximate solutions to Eq. (9) with tolerance $(\nabla_u \tilde{f}(u_k, v_k))^2 + (\nabla_v \tilde{f}(u_k, v_k))^2 \leq \epsilon_k^2$ for all $k = 0, 1, \cdots$ and LICQ is satisfied at the optimum. Then any limit point of $\{(u_k, v_k)\}$ satisfies the KKT conditions of the problem in Eq. (8).*

Alg. 1 describes our method where we minimize the penalty function in Eq. (9), alternatively over $v$ and $u$ which is just a single-level problem.

For unconstrained problems ($h \equiv 0$), Lemma 3 below shows that the approximate gradient direction $\nabla_u \tilde{f}$, computed from Alg. 1 becomes the exact hypergradient Eq. (6) asymptotically.

**Lemma 3.** *Assume $h \equiv 0$. Given $u$, let $\hat{v}$ be $\hat{v} := \arg\min_v \tilde{f}(u_k, v_k; \gamma)$ from Eq. (9). Then, $\nabla_u \tilde{f}(u, \hat{v}; \gamma) = \frac{df}{du}(u, \hat{v})$ as in Eq. (6).*

Thus if we find the minimizer $\hat{v}$ of the penalty function for given $u$ and $\gamma$, Alg. 1 computes the exact hypergradient for unconstrained problems Eq. (6) at $(u, \hat{v})$. Furthermore, under the conditions of Theorem 1, $\hat{v}(u) \to v^*(u)$ as $\gamma \to \infty$ and we get the exact hypergradient asymptotically.

**Algorithm 1** Penalty method (Penalty)

Input: $K, T, \{\sigma_k\}, \{\rho_{k,t}\}, \gamma_0, \epsilon_0, c_\gamma$(=1.1), $c_\epsilon$(=0.9)
Output: $(u_K, v_T)$
Initialize $u_0, v_0$ randomly
Begin
  **for** $k = 0, \cdots, K$-1 **do**
    **while** $\|\nabla_u \tilde{f}\|^2 + \|\nabla_v \tilde{f}\|^2 > \epsilon_k^2$ **do**
      **for** $t = 0, \cdots, T$-1 **do**
        $v_{t+1} \leftarrow v_t - \rho_{k,t} \nabla_v \tilde{f}$ (from Eq. (9))
      **end for**
      $u_{k+1} \leftarrow u_k - \sigma_k \nabla_u \tilde{f}$ (from Eq. (9))
    **end while**
    $\gamma_{k+1} \leftarrow c_\gamma \gamma_k, \ \epsilon_{k+1} \leftarrow c_\epsilon \epsilon_k$
  **end for**

**Comparison with other methods.** Previous methods for solving bilevel optimization problems in machine learning include forward/reverse-mode differentiation (FMD/RMD) [18, 9, 29] and approximate hypergradient computation by solving a linear system (ApproxGrad) [7, 24, 25] (See Appendix B for a brief summary). These methods have not been shown to handle constrained problems whereas our method can handle upper-level constraints. Moreover, for problems with multiple lower-level solutions, Penalty converges to the optimistic case solution without modification (Appendix C.3) where as convergence of other methods is unknown.

For unconstrained problems, we show the trade-offs of the different methods for computing the hypergradient in Table 1. We see that as $T$ (total number of $v$-updates per one hypergradient computation) increases, FMD and RMD become impractical due to $O(UVT)$ time complexity and

[1] Problems with lower-level constraints are not common in machine learning and are left for future work.

Table 1: Complexity analysis of various bilevel methods (Appendix B) on unconstrained problems. $U$ is the size of $u$, $V$ is the size of $v$, and $T$ is the number of $v$-updates per one hypergradient computation. $P$, $p$ and $q$ are variables of size $U \times V$, $U \times 1$, and $V \times 1$ used to compute the hypergradient. We use gradient descent as the process for FMD and RMD. Hessian-vector product has $O(V)$ complexity [23].

| Method | $v$-update | Intermediate updates | Time | Space |
|---|---|---|---|---|
| FMD | $v \leftarrow v - \rho \nabla_v g$ | $P \leftarrow P(I$ - $\rho \nabla^2_{vv} g) - \rho \nabla^2_{uv} g$ | $O(UVT)$ | $O(UV)$ |
| RMD | $v \leftarrow v - \rho \nabla_v g$ | $p \leftarrow p - \rho \nabla^2_{uv} g \cdot q$<br>$q \leftarrow q - \rho \nabla^2_{vv} g \cdot q$ | $O(VT)$ | $O(U + VT)$ |
| ApproxGrad | $v \leftarrow v - \rho \nabla_v g$ | $q \leftarrow q - \rho \nabla^2_{vv} g [\nabla^2_{vv} g \cdot q - \nabla_v f]$ | $O(VT)$ | $O(U$+$V)$ |
| Penalty | $v \leftarrow v - \rho [\nabla_v f$ + $\gamma \nabla^2_{vv} g \nabla_v g]$ | Not required | $O(VT)$ | $O(U$+$V)$ |

$O(U + VT)$ space complexity, respectively, whereas ApproxGrad and Penalty, have the same linear time complexity and constant space complexity, which is a big advantage over FMD and RMD. However, complexity analysis does not show the quality of hypergradient approximation of each method. In Sec. 3 we show that Penalty has better convergence properties than all the other methods on synthetic and real problems. We present a detailed comparison between ApproxGrad and Penalty since they have the same complexities and show that Penalty is superior or comparable to ApproxGrad in performance, run-time and convergence speed (Figs. 4, 5 in Appendix D).

**Improvements.** Some of the assumptions such as unique lower-level solution may not hold in practice. Here we discuss several techniques to address these and improve Alg. 1 further. The first problem is related to non-convexity of the lower-level cost $g$, creating the problem that the local minimum of $\|\nabla_v g\|$ can be either a minimum or a maximum of $g$. To address this we modify the $v$-update for Eq. (9) by adding a 'regularization' term $\lambda_k g$ to the cost Thus, the minimization over $v$ becomes $\min_v(\tilde{f} + \lambda_k g)$. This only affects the optimization in the beginning; as $\lambda_k \to 0$ the final solution remains unaffected with or without regularization. The second problem is that the tolerance $\nabla_{(u,v)} \tilde{f}(u_k, v_k; \gamma_k) \leq \epsilon_k$ may not be satisfied in a limited time and the optimization may terminate before $\gamma_k$ becomes large enough. The method of multipliers and augmented Lagrangian [4] can help the penalty method to find a solution with a finite $\gamma_k$. Thus we add the term $\nabla_v g^T \nu$ to the penalty function (Eq. (9)) to get $\min_{u,v}(\tilde{f} + \nabla_v g^T \nu)$ and use the method of multiplier to update $\nu$. In summary, we use the following update rules. $u_{k+1} \leftarrow u_k - \rho \nabla_u (\tilde{f} + \nabla_v g^T \nu_k)$, $v_{k+1} \leftarrow v_k - \sigma \nabla_v (\tilde{f} + \nabla_v g^T \nu_k + \lambda_k g)$, $\nu_{k+1} \leftarrow \nu_k + \gamma_k \nabla_v g$. Practically, the improvement due to these changes was only moderate and problem-dependent (see Appendix C for details.)

# 3 Applications

In this section, we evaluate the performance of the proposed penalty method (Penalty) on various machine learning problems discussed in the introduction. Since previously proposed bilevel methods in machine learning dealt only with unconstrained problems, we evaluate Penalty on unconstrained problems Sec. 3.1-Sec. 3.3 and show it's effectiveness in solving constrained problems in Sec. 3.4.

## 3.1 Validation with synthetic problems

We start by comparing Penalty against gradient descent (GD), reverse-mode differentiation (RMD), and approximate hypergradient method (ApproxGrad) on synthetic examples. We omit the comparison with forward-mode differentiation (FMD) because of its impractical time complexity for larger problems (Table 1). GD refers to the alternating minimization: $u \leftarrow u - \rho \nabla_u f$, $v \leftarrow v - \sigma \nabla_v g$. For RMD, we used the version with gradient descent as the lower-level process. For ApproxGrad experiments, we used Adam for the updates as well as solving the linear system. Using simple quadratic surfaces for $f$ and $g$, we compare all the algorithms by observing their convergence as a function of the number of upper-level iterations for different number of lower-level updates ($T$). We measure the convergence of these methods using the Euclidean distance of the current iterate $(u, v)$ from the closest optimal solution $(u^*, v^*)$ . Since the synthetic examples are not learning problems, we can only measure the distance of the iterates to an optimal solution ($\|(u, v) - (u^*, v^*)\|_2^2$). Fig. 1 shows the performance of two 10-dimensional examples described in the caption (see Appendix F.1). As one would expect, increasing the number $T$ of $v$-updates makes all the algorithms, except GD,

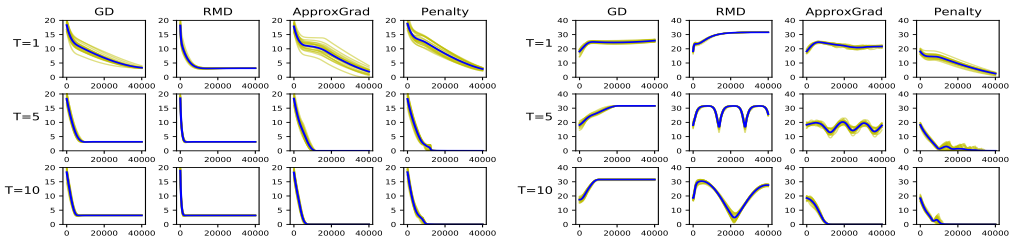



Figure 1: Convergence of GD, RMD, ApproxGrad, and Penalty for two bilevel problems vs epochs (x-axis) $f(u,v) = \|u\|^2 + \|v\|^2, g(u,v) = \|1\text{-}u\text{-}v\|^2$ (LEFT), $f(u,v) = \|v\|^2 - \|u-v\|^2, g(u,v) = \|u\text{-}v\|^2$ (RIGHT) . The mean curve (blue) is superimposed on 20 independent trials (yellow).

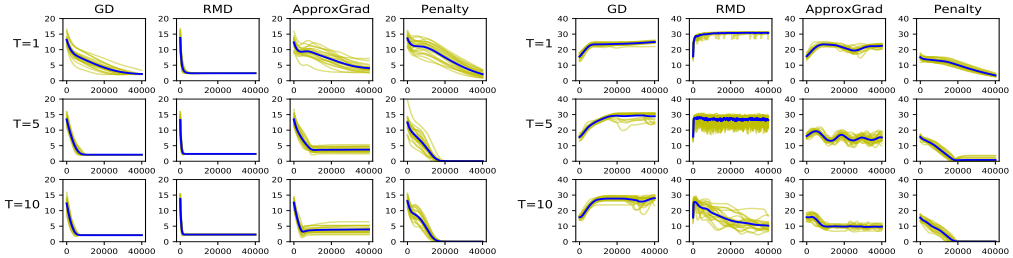



Figure 2: Convergence of GD, RMD, ApproxGrad, and Penalty for two bilevel problems where $A^TA$ is a rank-deficient random matrix $f(u,v)=\|u\|^2 + \|v\|^2, g(u,v)=(1\text{-}u\text{-}v)^TA^TA(1\text{-}u\text{-}v)$ (LEFT) and $f(u,v)=\|v\|^2\text{-}(u\text{-}v)^TA^TA(u\text{-}v), g(u,v)=(u\text{-}v)^TA^TA(u\text{-}v)$ (RIGHT), vs epochs (x-axis). The mean curve (blue) is superimposed on 20 independent trials (yellow).

better since doing more lower-level iterations makes the hypergradient estimation more accurate (Eq. (6)) but it also increases the run time of the methods. However, even for these examples, only Penalty and ApproxGrad converge to the optimal solution and GD and RMD converge to non-solution points. Moreover, from Fig. 1(b), we see that Penalty converges even with $T$=1 while ApproxGrad requires at least $T$=10 and RMD needs an even higher $T$ to converge, showing that our method approximates the hypergradient accurately even with smaller $T$. This directly translates to smaller run-time for our method since the run-time is directly proportional to $T$ (Table. 1).

In Fig. 2 we show examples similar to Fig. 1 but with ill-conditioned or singular Hessian $\nabla^2_{vv}g$ for the lower-level problem. The ill-conditioning poses difficulty for the methods since the implicit function theorem requires the invertibility of the Hessian at the solution point. Compared to Fig. 1, Fig. 2 shows that only Penalty converges to the true solution despite the fact that we add regularization $\nabla^2_{vv}g + \lambda I$ in ApproxGrad to improve the ill-conditioning when solving the linear systems by minimization. Additionally, we report the wall clock times for different methods on the four examples tested here in Table 4 in the Appendix. We can see that as we increase the number of lower-level iterations all methods get slower but Penalty is still faster than both RMD and ApproxGrad. Although, Penalty is slower than GD but as shown in Fig. 1 and Fig. 2, GD does not converge to optima for most of the synthetic examples.

### 3.2 Data denoising by importance learning

We evaluate the performance of Penalty for learning a classifier from a dataset with corrupted labels (training data). We pose the problem as an importance learning problem presented in Eq. (3). We evaluate the performance of the classifier learned by Penalty, with 20 lower-level updates, against the following classifiers: **Oracle**: classifier trained on the portion of training data with clean labels and the validation data, **Val-only**: classifier trained only on the validation data, **Train+Val**: classifier trained on the entire training and validation data, **ApproxGrad**: classifier trained with our implementation of ApproxGrad, with 20 lower-level and 20 linear system updates. We test the performance on MNIST, CIFAR10 and SVHN datasets with validation set sizes of 1000, 10000 and 1000 points respectively. We used convolutional neural networks (architectures described in

Appendix F.2) at the lower-level for this task. Table 2 summarizes our results for this problem and shows that Penalty outperforms Val-only, Train+Val and ApproxGrad by significant margins and in fact performs very close to the Oracle classifier (which is the ideal classifier), even for high noise levels. This demonstrates that Penalty is extremely effective in solving bilevel problems involving several million variables (see Table 5 in Appendix) and shows its effectiveness at handling non-convex problems. Along with improvement in terms of accuracy over other bilevel methods like ApproxGrad, Penalty also gives better run-time per upper-level iteration and higher convergence speed leading to a decrease in the overall wall-clock time of the experiments (Fig. 4(a), 5(a), in Appendix D, E).

Table 2: Test accuracy (%) of the classifier learnt from datasets with noisy labels after data denoising using using importance learning. (Mean $\pm$ s.d. of 5 runs)

| Dataset (Noise%) | Oracle | Val-Only | Train+Val | Bilevel Approaches | |
|---|---|---|---|---|---|
| | | | | ApproxGrad | Penalty |
| MNIST (25) | 99.3±0.1 | 90.5±0.3 | 83.9±1.3 | 98.11±0.08 | **98.89**±0.04 |
| MNIST (50) | 99.3±0.1 | 90.5±0.3 | 60.8±2.5 | 97.27±0.15 | **97.51**±0.07 |
| CIFAR10 (25) | 82.9±1.1 | 70.3±1.8 | 79.1±0.8 | 71.59±0.87 | **79.67**±1.01 |
| CIFAR10 (50) | 80.7±1.2 | 70.3±1.8 | 72.2±1.8 | 68.08±0.83 | **79.03**±1.19 |
| SVHN (25) | 91.1±0.5 | 70.6±1.5 | 71.6±1.4 | 80.05±1.37 | **88.12**±0.16 |
| SVHN (50) | 89.8±0.6 | 70.6±1.5 | 47.9±1.3 | 74.18±1.05 | **85.21**±0.34 |

Next, evaluate Penalty against the recent method [27] which uses a meta-learning based approach and assigns weights to examples based on their gradient directions. We used the same setting as their uniform flip experiment with 36% label noise on CIFAR10 dataset and Wide ResNet 28-10 (WRN-28-10) model (see Appendix F.2). Using $T$=1 for Penalty and 1000 validation points, we get an accuracy of 87.41 $\pm$ 0.26 (mean $\pm$ s.d. of 5 trials) higher than 86.92 $\pm$ 0.19 reported by them. Due to the enormous size of WRN-28-10, we do not use larger values of $T$, but expect it to only improve the results based on the Fig. 5(a) in Appendix E. We also compared Penalty against a RMD-based method [9], using the same setting as their Sec. 5.1, on a subset of MNIST data corrupted with 50% label noise and softmax regression as the model (see Appendix F.2). The accuracy of the classifier trained on a subset of the data with points having importance values greater than 0.9 (as computed by Penalty with $T = 20$) along with the validation set is 90.77% better than 90.09% reported by the RMD-based method.

### 3.3 Few-shot learning

Next, we evaluate the performance of Penalty on the task of learning a common representation for the few-shot learning problem. We use the formulation presented in Eq. (4) and use Omniglot [14] and Mini-ImageNet [33] datasets for our experiments. Following the protocol proposed by [33] for $N$-way $K$-shot classification, we generate meta-training and meta-testing datasets. Each meta-set is built using images from disjoint classes. For Omniglot, our meta-training set comprises of images from the first 1200 classes and the remaining 423 classes are used in the meta-testing dataset. We also augment the meta-datasets with three different rotations (90, 180 and 270 degrees) of the images as used by [28]. For the experiments with Mini-Imagenet, we used the split of 64 classes in meta-training, 16 classes in meta-validation and 20 classes in meta-testing as used by [26].

Each meta-batch of the meta-training and meta-testing dataset comprises of a number of tasks which is called the meta-batch-size. Each task in the meta-batch consists of a training set with $K$ images and a testing set consists of 15 images from $N$ classes. We train Penalty using a meta-batch-size of 30 for 5 way and 15 for 20 way classification for Omniglot and with a meta-batch-size of 2 for Mini-ImageNet experiments. The training sets of the meta-train-batch are used to train the lower-level problem and the test sets are used as validation sets for the upper-level problem in Eq. (4). The final accuracy is reported using the meta-test-set, for which we fix the common representation learnt during meta-training. We then train the classifiers at the lower-level for 100 steps using the training sets from the meta-test-batch and evaluate the performance of each task on the associated test set from the meta-test-batch. Average performance of Penalty and ApproxGrad over 600 tasks is reported in Table 3. Penalty outperforms other bilevel methods namely the ApproxGrad (trained with 20 lower-level iterations and 20 updates for the linear system) and the RMD-based method [10] on Mini-Imagenet and is comparable to them on Omniglot. We demonstrate convergence speed of Penalty in comparison to ApproxGrad (Fig. 4(b) in Appendix D) and the trade-off between using higher $T$ and time for the two methods (Fig. 5(b) in Appendix E) and show that Penalty converges much faster than ApproxGrad. In comparison to non-bilevel approaches that used models of the similar size as ours, Penalty is comparable to most approaches and is only slightly worse than [20] which makes use of temporal convolutions and soft attention. We used convolutional neural networks and a residual network for learning the common task representation (upper-level) for Omniglot and Mini-ImageNet, respectively and use logistic regression to learn task specific classifiers (lower-level).

Table 3: Few-shot classification accuracy (%) on Omniglot and Mini-ImageNet. We report mean±s.d. for Omniglot and 95% confidence intervals for Mini-Imagenet over five trials. Results for learning a common representation using Penalty, ApproxGrad and RMD[10] are averaged over 600 randomly-sampled tasks from the meta-test set. Results for previous methods using similar models are also reported

| | | | | | | Learning a common representation | | |
|---|---|---|---|---|---|---|---|---|
| | MAML[8] | iMAML[25] | Proto Net[31] | Rel Net[32] | SNAIL[20] | RMD | ApproxGrad | Penalty |
| Omniglot | | | | | | | | |
| 5-way 1-shot | 98.7 | **99.50**±0.26 | 98.8 | **99.6**±0.2 | 99.1 | **98.6** | 97.75±0.06 | 97.83±0.35 |
| 5-way 5-shot | **99.9** | 99.74 ± 0.11 | 99.7 | **99.8**±0.1 | 99.8 | **99.5** | **99.51**±0.05 | **99.45**±0.05 |
| 20-way 1-shot | 95.8 | 96.18 ± 0.36 | 96.0 | **97.6**±0.2 | **97.6** | **95.5** | 94.69±0.22 | 94.06±0.17 |
| 20-way 5-shot | 98.9 | 99.14 ± 0.10 | 98.9 | 99.1±0.1 | **99.4** | 98.4 | **98.46**±0.08 | **98.47**±0.08 |
| Mini-Imagenet | | | | | | | | |
| 5-way 1-shot | 48.70±1.75 | 49.30 ± 1.88 | 49.42±0.78 | 50.44±0.82 | **55.71**±0.99 | 50.54±0.85 | 43.74±1.75 | **53.17**±0.96 |
| 5-way 5-shot | 63.11±0.92 | - | 68.20±0.66 | 65.32±0.82 | **68.88**±0.92 | 64.53±0.68 | 65.56±0.67 | **67.74**±0.71 |

### 3.4 Training-data poisoning

We evaluate Penalty on the clean label data poisoning attack problem. We use the setting presented in [30] which adds a single poison point to misclassify a particular target image from the test set. Unlike the original approach [30], we use a bilevel formulation along with a constraint on the maximum perturbation. We use the dog vs. fish dataset and InceptionV3 network as the representation map. We choose a target image $t$ and a base image $b$ from the test set such that the representation of the base is closest to that of the target but has a different label. The poison point is initialized from the base image. We solve the bilevel problem in Eq. (5) with an additional feature collision term ($\|r(t) - r(u)\|_2^2$) from [30] which is shown to be helpful in practice. $r(\cdot)$ is the 2048-dimensional representation map. The lower-level problem trains a softmax on top of this representation. (The full problem is in Eq. (10) of Appendix F.4.1.) We evaluate the attack success by retraining the softmax on the clean dataset augmented with the poison point. Attack is considered successful if the target point is misclassified after retraining. Choosing each correctly classified point from the test set as the target we search for the smallest $\epsilon \in \{1, 2, ..., 16\}$ that when used as the upper bound for the perturbation of the poison image, leads to misclassification of that target. For comparison we use the Alg.1 from [30] with a modification of constraining the perturbation at every step by projecting it on a ball of radius $\epsilon$. Similar to Penalty, the smallest $\epsilon$ that causes misclassification is recorded for each target. (See Appendix F.4 for details). Fig. 3 shows that Penalty achieves higher attack success with the same amount of distortion, or in another view, achieves the same attack success with much less distortion. We ascribe this to the benefits of using the bilevel formulation as opposed to the non-bilevel formulation [30]. Poison points generated by Penalty are also shown in Fig. 7(a) in Appendix F.4. Lastly, we also test Penalty on a simpler data poisoning problem without upper-level constraint for the purpose of comparison with RMD and ApproxGrad which have not been shown to handle constrained problems. In Appendix F.4.2 we show that Penalty outperforms RMD and is comparable to ApproxGrad.

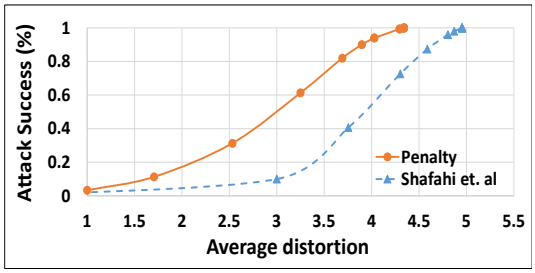



Figure 3: Clean label poisoning attack

## 4 Conclusion

A wide range of interesting machine learning problems can be expressed as bilevel optimization problems, and new applications are still being discovered. So far, the difficulty of solving bilevel optimization has limited its wide-spread use for solving large problems involving deep models. In this paper we presented an efficient algorithm based on penalty function which is simple and has theoretical and practical advantages over existing methods. Compared to previous methods we demonstrated our method's ability to handle constraints, achieve competitive performance on problems with convex lower-level costs and get significant improvements on problems with non-convex lower-level costs in terms of accuracy and convergence speed, highlighting its effectiveness in deep learning settings. In future works, we plan to tackle other challenges in bilevel optimization such as handling problems with non-unique lower-level solutions.

## Broader Impact

The research presented in the paper proposes a new method for solving bilevel optimization problems that appear in the field of machine learning. The work in this paper can be considered on the level of basic research in the field of machine learning. Researchers and practitioners working on machine learning problems which require solving a bilevel problem will benefit the most from this work.

## References


[1] Eitaro Aiyoshi and Kiyotaka Shimizu. A solution method for the static constrained stackelberg problem via penalty method. *IEEE Transactions on Automatic Control*, 29(12):1111–1114, 1984.

[2] Jonathan F Bard. Some properties of the bilevel programming problem. *Journal of optimization theory and applications*, 68(2):371–378, 1991.

[3] Jonathan F Bard. *Practical bilevel optimization: algorithms and applications*, volume 30. Springer Science & Business Media, 2013.

[4] Dimitri P Bertsekas. On penalty and multiplier methods for constrained minimization. *SIAM Journal on Control and Optimization*, 14(2):216–235, 1976.

[5] Dimitri P Bertsekas. Nonlinear programming. *Journal of the Operational Research Society*, 48(3):334–334, 1997.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[7] Justin Domke. Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, pages 318–326, 2012.

[8] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135, 2017.

[9] Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Forward and reverse gradient-based hyperparameter optimization. In *International Conference on Machine Learning*, pages 1165–1173, 2017.

[10] 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*, pages 1563–1572, 2018.

[11] Jihun Hamm and Yung-Kyun Noh. K-beam minimax: Efficient optimization for deep adversarial learning. *International Conference on Machine Learning (ICML)*, 2018.

[12] Yo Ishizuka and Eitaro Aiyoshi. Double penalty method for bilevel optimization problems. *Annals of Operations Research*, 34(1):73–88, 1992.

[13] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, pages 1885–1894, 2017.

[14] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[15] Tongliang Liu and Dacheng Tao. Classification with noisy labels by importance reweighting. *IEEE Transactions on pattern analysis and machine intelligence*, 38(3):447–461, 2016.

[16] Jelena Luketina, Mathias Berglund, Klaus Greff, and Tapani Raiko. Scalable gradient-based tuning of continuous regularization hyperparameters. In *International Conference on Machine Learning*, pages 2952–2960, 2016.

[17] Chunjie Luo, Jianfeng Zhan, Xiaohe Xue, Lei Wang, Rui Ren, and Qiang Yang. Cosine normalization: Using cosine similarity instead of dot product in neural networks. In *International Conference on Artificial Neural Networks*, pages 382–391. Springer, 2018.

[18] Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122, 2015.

[19] Shike Mei and Xiaojin Zhu. Using machine teaching to identify optimal training-set attacks on machine learners. In *AAAI*, pages 2871–2877, 2015.

[20] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. *arXiv preprint arXiv:1707.03141*, 2017.

[21] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 27–38. ACM, 2017.

[22] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

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

[24] Fabian Pedregosa. Hyperparameter optimization with approximate gradient. In *International conference on machine learning*, pages 737–746, 2016.

[25] Aravind Rajeswaran, Chelsea Finn, Sham M Kakade, and Sergey Levine. Meta-learning with implicit gradients. In *Advances in Neural Information Processing Systems*, pages 113–124, 2019.

[26] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. *International Conference on Learning Representations (ICLR)*, 2017.

[27] Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. In *ICML*, 2018.

[28] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. One-shot learning with memory-augmented neural networks. *arXiv preprint arXiv:1605.06065*, 2016.

[29] Amirreza Shaban, Ching-An Cheng, Nathan Hatch, and Byron Boots. Truncated back-propagation for bilevel optimization. *arXiv preprint arXiv:1810.10667*, 2018.

[30] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *Advances in Neural Information Processing Systems*, pages 6103–6113, 2018.

[31] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4080–4090, 2017.

[32] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1199–1208, 2018.

[33] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pages 3630–3638, 2016.

[34] Heinrich von Stackelberg. *Market structure and equilibrium*. Springer Science & Business Media, 2010.

[35] Xiyu Yu, Tongliang Liu, Mingming Gong, Kun Zhang, and Dacheng Tao. Transfer learning with label noise. *arXiv preprint arXiv:1707.09724*, 2017.

# Appendix

We provide missing proofs in Appendix A, a review of other methods of hypergradient computation in Appendix B, discuss modifications to improve the Alg. 1 in Appendix C, and the experiment details and additional results in Appendix F.

## A Proofs

**Theorem 2.** *Suppose $\{\epsilon_k\}$ is a positive ($\epsilon_k > 0$) and convergent ($\epsilon_k \to 0$) sequence, $\{\gamma_k\}$ is a positive ($\gamma_k > 0$), non-decreasing ($\gamma_1 \leq \gamma_2 \leq \cdots$), and divergent ($\gamma_k \to \infty$) sequence. Let $\{(u_k, v_k)\}$ be the sequence of approximate solutions to Eq. (9) with tolerance $(\nabla_u \tilde{f}(u_k, v_k))^2 + (\nabla_v \tilde{f}(u_k, v_k))^2 \leq \epsilon_k^2$ for all $k = 0, 1, \cdots$ and LICQ is satisfied at the optimum. Then any limit point of $\{(u_k, v_k)\}$ satisfies the KKT conditions of the problem in Eq. (8).*

*Proof.* The proof follows the standard proof for penalty function methods, e.g., [22]. Let $w := (u, v)$ refer to the pair, and let $\overline{w} := (\overline{u}, \overline{v})$ be any limit point of the sequence $\{w_k := (u_k, v_k)\}$, and

$$\tilde{g} := \begin{pmatrix} h(u, v) \\ \nabla_v g(u, v) \end{pmatrix}$$

then there is a subsequence $\mathcal{K}$ such that $\lim_{k \in \mathcal{K}} w_k = \overline{w}$. From the tolerance condition

$$\|\nabla_w \tilde{f}(w_k; \gamma_k)\| = \|\nabla_w f(w_k) + \gamma_k J_w^T(\tilde{g}(w_k))\tilde{g}(w_k)\| \leq \epsilon_k$$

we have

$$\|J_w^T(\tilde{g}(w_k))\tilde{g}(w_k)\| \leq \frac{1}{\gamma_k}[\|\nabla_w f(w_k)\| + \epsilon_k]$$

Take the limit with respect to the subsequence $\mathcal{K}$ on both sides to get $J_w^T(\tilde{g}(\overline{w}))\tilde{g}(\overline{w}) = 0$. Assuming linear independence constraint quantification (LICQ) we have that the columns of the $J_w^T \tilde{g} = \begin{pmatrix} J_u^T h & \nabla_{uv}^2 g \\ J_v^T h & \nabla_{vv}^2 g \end{pmatrix}$ are linearly independent. Therefore $\tilde{g}(\overline{w}) = 0$, which is the primary feasibility condition for Eq. (8). Furthermore, let $\mu_k := -\gamma_k \tilde{g}(w_k)$, then by definition,

$$\nabla_w \tilde{f}(w_k; \gamma_k) = \nabla_w f(w_k) - J_w^T(\tilde{g}(w_k))\mu_k$$

We can write

$$\left[J_w(\tilde{g}(w_k))J_w^T(\tilde{g}(w_k))\right]\mu_k = J_w(\tilde{g}(w_k))\left[\nabla_w f(w_k) - \nabla_w \tilde{f}(w_k; \gamma_k)\right]$$

The corresponding limit $\overline{\mu}$ can be found by taking the limit of the subsequence $\mathcal{K}$

$$\overline{\mu} := \lim_{k \in \mathcal{K}} \mu_k = \left[J_w(\tilde{g}(\overline{w}))J_w^T(\tilde{g}(\overline{w}))\right]^{-1} J_w(\tilde{g}(w_k))\nabla_w f(\overline{w})$$

Since $\lim_{k \in \mathcal{K}} \nabla_w \tilde{f}(w_k; \gamma_k) = 0$ from the condition $\epsilon_k \to 0$, we get

$$\nabla_w f(\overline{w}) - J_w^T(\tilde{g}(\overline{w}))\overline{\mu} = 0$$

at the limit $\overline{w}$, which is the stationarity condition of Eq. (8). Together with the feasibility condition $\tilde{g}(\overline{w}) = 0$, the two KKT conditions of Eq. (8) are satisfied at the limit point. □

**Lemma 3.** *Assume $h \equiv 0$. Given $u$, let $\hat{v}$ be $\hat{v} := \arg\min_v \tilde{f}(u_k, v_k; \gamma)$ from Eq. (9). Then, $\nabla_u \tilde{f}(u, \hat{v}; \gamma) = \frac{df}{du}(u, \hat{v})$ as in Eq. (6).*

*Proof.* At the minimum $\hat{v}$ the gradient $\nabla_v \tilde{f}$ vanishes, that is $\nabla_v f + \gamma \nabla_{vv}^2 g \nabla_v g = 0$. Equivalently, $\nabla_v g = -\gamma^{-1}(\nabla_{vv}^2 g)^{-1}\nabla_v f$. Then,

$$\nabla_u \tilde{f}(\hat{v}) = \nabla_u f(\hat{v}) + \gamma \nabla_{uv}^2 g(\hat{v})\nabla_v g(\hat{v}) = \nabla_u f(\hat{v}) - \nabla_{uv}^2 g(\hat{v})\nabla_{vv}^2 g^{-1}(\hat{v})\nabla_v f(\hat{v}),$$

where $\gamma$ disappears, which is the hypergradient $\frac{df}{du}(u, \hat{v})$ as in Eq. (6). □

That is, if we find the minimum $\hat{v}$ of the penalty function for given $u$ and $\gamma$, we get the hypergradient Eq. (6) at $(u, \hat{v})$. Furthermore, under the conditions of Theorem 1, $\hat{v}(u) \to v^*(u)$ as $\gamma \to \infty$ (see Lemma 8.3.1 of [3]), and we get the exact hypergradient asymptotically.

## B Review of other bilevel optimization methods for unconstrained problems

Several methods have been proposed to solve bilevel optimization problems appearing in machine learning, including forward/reverse-mode differentiation [18, 9] and approximate gradient [7, 24] described briefly here.

**Forward-mode (FMD) and Reverse-mode differentiation (RMD).** Domke [7], Maclaurin et al.[18], Franceschi et al. [9], and Shaban et al. [29] studied forward and reverse-mode differentiation to solve the minimization problem $\min_u f(u, v)$ where the lower-level variable $v$ follows a dynamical system $v_{t+1} = \Phi_{t+1}(v_t; u), \;\; t = 0, 1, 2, \cdots, T-1$. This setting is more general than that of a bilevel problem. However, a stable dynamical system is one that converges to a steady state and thus, the process $\Phi_{t+1}(\cdot)$ can be considered as minimizing an energy or a potential function.

Define $A_{t+1} := \nabla_v \Phi_{t+1}(v_t)$ and $B_{t+1} := \nabla_u \Phi_{t+1}(v_t)$, then the hypergradient Eq. (6) can be computed by

$$\frac{df}{du} = \nabla_u f(u, v_T) + \sum_{t=0}^{T} B_t A_{t+1} \times \cdots \times A_T \nabla_v f(u, v_T)$$

When the lower-level process is one step of gradient descent on a cost function $g$, that is,

$$\Phi_{t+1}(v_t; u) = v_t - \rho \nabla_v g(u, v_t)$$

we get

$$A_{t+1} = I - \rho \nabla^2_{vv} g(u, v_t), \;\; B_{t+1} = -\rho \nabla^2_{uv} g(u, v_t).$$

$A_t$ is of dimension $V \times V$ and $B_t$ is of dimension $V \times U$. The sequences $\{A_t\}$ and $\{B_t\}$ can be computed in forward or reverse mode.
For **reverse-mode differentiation**, first compute

$$v_{t+1} = \Phi_{t+1}(v_t), \;\; t = 0, 1, \cdots, T\text{-}1,$$

then compute

$$q_T \leftarrow \nabla_v f(u, v_T), \; p_T \leftarrow \nabla_u f(u, v_T)$$
$$p_{t-1} \leftarrow p_t + B_t q_t, \; q_{t-1} \leftarrow A_t q_t, \;\; t = T, T\text{-}1, \cdots, 1.$$

Time and space Complexity for computing $p_t$ is $O(V)$ since the Jacobian vector product can be computed in $O(V)$ time and space. The final hypergradient for RMD is $\frac{df}{du} = p_0$.
Hence the final time complexity for RMD is $O(VT)$ and space complexity is $O(U + VT)$.

For **forward-mode differentiation**, simultaneously compute $v_t$, $A_t$, $B_t$ and

$$P_0 \leftarrow 0, \quad P_{t+1} \leftarrow P_t A_{t+1} + B_{t+1}, \quad t = 0, 1, \cdots, T\text{-}1.$$

Time complexity for computing $P_t$ is $O(UV)$ since $P_t A_{t+1}$ can be computed using $U$ Hessian vector products each needing $O(V)$ and $B_{t+1}$ also needs $O(UV)$ using unit vectors $e_i$ for $i = 1...U$. The space complexity for each $P_t$ is $O(UV)$. The final hypergradient for FMD is

$$\frac{df}{du} = \nabla_u f(u, v_T) + P_T \nabla_v f(v_T).$$

Hence the final time complexity for FMD is $O(UVT)$ and space complexity is $O(U + UV) = O(UV)$.

**Approximate hypergradient (ApproxGrad).** Since computing the inverse of the Hessian $(\nabla^2_{vv} g)^{-1}$ directly is difficult even for moderately-sized neural networks, Domke [7] proposed to find an approximate solution to $q = (\nabla^2_{vv} g)^{-1} \nabla_v f$ by solving the linear system of equations $\nabla^2_{vv} g \cdot q \approx \nabla_v f$. This can be done by solving

$$\min_q \; \|\nabla^2_{vv} g \cdot q - \nabla_v f\|$$

using gradient descent, conjugate gradient descent or any other iterative solver. Note that the minimization requires evaluation of the Hessian-vector product, which can be done in linear time [23]. Hence the time complexity of the method is $O(VT)$ and space complexity is $O(U + V)$ since we only need to store single copy of $u$ and $v$ same as Penalty. The asymptotic convergence with approximate solutions was shown by [24].

## C Improvements to Algorithm 1

Here we discuss the details of the modifications to Alg. 1 presented in the main text which can be added to improve the performance of the algorithm in practice.

### C.1 Improving local convexity by regularization

One of the common assumptions of this and previous works is that $\nabla^2_{vv} g$ is invertible and locally positive definite. Neither invertibility nor positive definiteness hold in general for bilevel problems, involving deep neural networks, and this causes difficulties in the optimization. Note that if $g$ is non-convex in $v$, minimizing the penalty term $\|\nabla_v g\|$ does not necessarily lower the cost $g$ but instead just moves the variable towards a stationary point – which is a known problem even for the Newton's method. Thus we propose the following modification to the $v$-update:

$$\min_v \quad \left[\tilde{f} + \lambda_k g\right]$$

keeping the $u$-update intact. To see how this affects the optimization, note that $v$-update becomes

$$v \leftarrow v - \rho \left[\nabla_v f + \gamma_k \nabla^2_{vv} g \nabla_v g + \lambda_k \nabla_v g\right]$$

After $v$ converges to a stationary point, we get $\nabla_v g = -(\gamma_k \nabla^2_{vv} g + \lambda_k I)^{-1} \nabla_v f$, and after plugging this into $u$-update, we get

$$u \leftarrow u - \sigma \left[\nabla_u f - \nabla^2_{uv} g \left(\nabla^2_{vv} g + \frac{\lambda_k}{\gamma_k} I\right)^{-1} \nabla_v f\right]$$

that is, the Hessian inverse $\nabla^2_{vv} g^{-1}$ is replaced by a regularized version $(\nabla^2_{vv} g + \frac{\lambda_k}{\gamma_k} I)^{-1}$ to improve the positive definiteness of the Hessian. With a decreasing or constant sequence $\{\lambda_k\}$ such that $\lambda_k / \gamma_k \to 0$ the regularization does not change to solution.

### C.2 Convergence with finite $\gamma_k$

The penalty function method is intuitive and easy to implement, but the sequence $\{(\hat{u}_k, \hat{v}_k)\}$ is guaranteed to converge to an optimal solution only in the limit with $\gamma \to \infty$, which may not be achieved in practice in a limited time. It is known that the penalty method can be improved by introducing an additional term into the function, which is called the augmented Lagrangian (penalty) method [4]:

$$\min_{u,v} \quad \left[\tilde{f} + \nabla_v g^T \nu\right].$$

This new term $\nabla_v g^T \nu$ allows convergence to the optimal solution $(u^*, v^*)$ even when $\gamma_k$ is finite. Furthermore, using the update rule $\nu \leftarrow \nu + \gamma \nabla_v g$, called the method of multipliers, it is known that $\nu$ converges to the true Lagrange multiplier of this problem corresponding to the equality constraints $\nabla_v g = 0$.

### C.3 Non-unique lower-level solution

Most existing methods have assumed that the lower-level solution $\arg\min_v g(u, v)$ is unique for all $u$. Regularization from the previous section, can improve the ill-conditioning of the Hessian $\nabla^2_{vv} g$ but it does not address the case of multiple disconnected global minima of $g$. With multiple lower-level solutions $Z(u) = \{v \mid v = \arg\min g(u, v)\}$, there is an ambiguity in defining the upper-level problem. If we assume that $v \in Z(u)$ is chosen adversarially (or pessimistically), then the upper-level problem should be defined as

$$\min_u \max_{v \in Z(u)} f(u, v).$$

If $v \in Z(u)$ is chosen cooperatively (or optimistically), then the upper-level problem should be defined as

$$\min_u \min_{v \in Z(u)} f(u, v),$$

**Algorithm 2** Modified Alg. 1 with regularization and augmented Lagrangian

Input: $K, T, \{\sigma_k\}, \{\rho_{k,t}\}, \gamma_0, \epsilon_0, \lambda_0, \nu_0, c_\gamma$(=1.1),
$c_\epsilon$(=0.9), $c_\lambda$(=0.9)
Output: $(u_K, v_T)$
Initialize $u_0, v_0$ randomly
Begin
  **for** $k = 0, \cdots, K$-1 **do**
    **while** $\|\nabla_u \tilde{f}_1\|^2 + \|\nabla_v \tilde{f}_2\|^2 > \epsilon_k^2$ **do**
      **for** $t = 0, \cdots, T$-1 **do**
        $v_{t+1} \leftarrow v_t - \rho_{k,t} \nabla_v \tilde{f}_2(u_k, v_t)$ (from Appendix C.4)
      **end for**
      $u_{k+1} \leftarrow u_k - \sigma_k \nabla_u \tilde{f}_1(u_k, v_T)$ (from Appendix C.4)
    **end while**
    $\gamma_{k+1} \leftarrow c_\gamma \gamma_k$
    $\epsilon_{k+1} \leftarrow c_\epsilon \epsilon_k$
    $\lambda_{k+1} \leftarrow c_\lambda \lambda_k$
    $\nu_{k+1} \leftarrow \nu_k + \gamma_k \nabla_v g$
  **end for**

and the results can be quite different between these two cases. Note that the proposed penalty function method is naturally solving the optimistic case, as Alg. 1 is solving the problem of $\min_{u,v} \tilde{f}(u, v)$ by alternating gradient descent. However, with a gradient-based method, we cannot hope to find all disconnected multiple solutions. In a related problem of min-max optimization, which is a special case of bilevel optimization, an algorithm for handling non-unique solutions was proposed recently [11]. This idea of keeping multiple candidate solution may be applicable to bilevel problems too and further analysis of the non-unique lower-level problem is left as future work.

### C.4 Modified algorithm

Here we present the modified algorithm which incorporates regularization (Appendix. C.1) and augmented Lagrangian (Appendix. C.2) as discussed previously. The augmented Lagrangian term $\nabla_v g^T \nu$ applies to both $u$- and $v$-update, but the regularization term $\lambda g$ applies to only the $v$-update as its purpose is to improve the ill-conditioning of $\nabla^2_{vv} g$ during $v$-update. The modified penalized functions $\tilde{f}_1$ for $u$-update and $\tilde{f}_2$ for $v$-update are

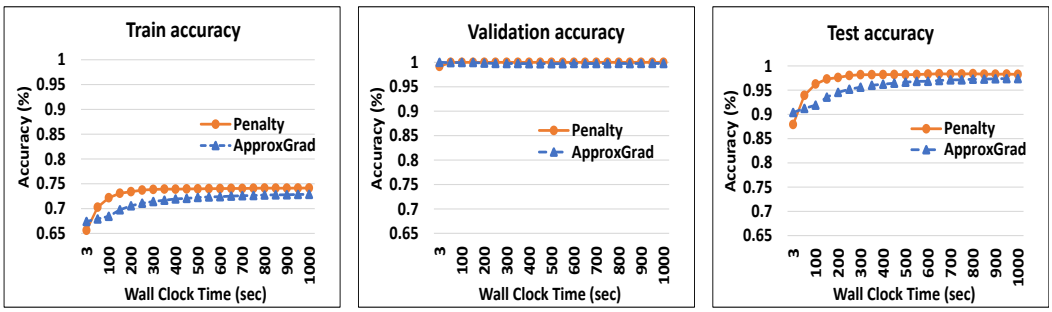
$$\tilde{f}_1(u, v; \gamma, \nu) := \tilde{f} + \nabla_v g^T \nu$$
$$\tilde{f}_2(u, v; \gamma, \lambda, \nu) := \tilde{f} + \nabla_v g^T \nu + \lambda g$$

The new algorithm (Alg. 2) is similar to Alg. 1 with additional steps for updating $\lambda_k$ and $\nu_k$.

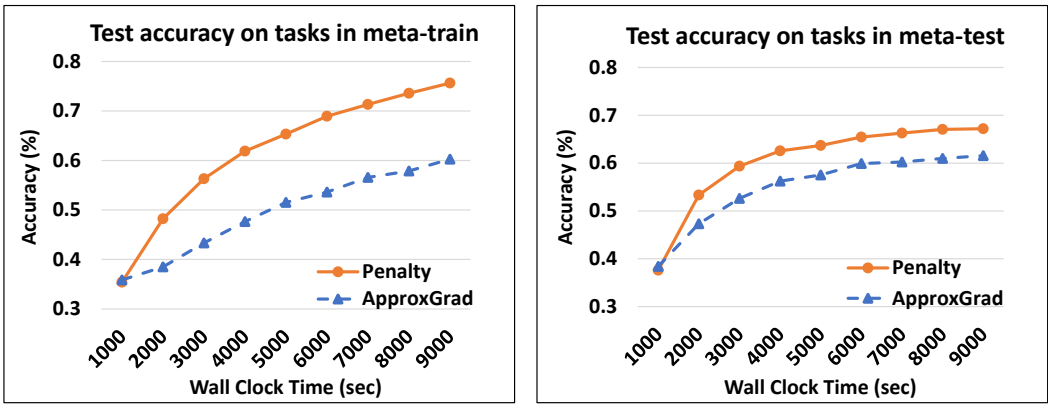
## D Comparison of accuracy vs wall clock time

Here, we compare the accuracy and wall clock time for Penalty and ApproxGrad (Fig. 4) on real problems to demonstrate that the Penalty method converges faster than ApproxGrad for data denoising and few shot learning problem. Training accuracy is indicative of the optimality of the lower-level and validation/meta-train-test accuracy shows how well the upper-level problem is being solved during bilevel training. For data denoising the test accuracy is a measure of how well the classifier trained with found importance values performs on the test set. For few-shot learning the meta-test-test accuracy is a measure of how well the common representation generalize to new unseen tasks after training the classifier on meta-test-train set.

We see that for the importance learning problem Penalty quickly reaches a training accuracy of roughly 75% which is desirable since the dataset contains 25% noise. High test and validation accuracy are indicative that the importance being found for the training data is indeed helping to learn a good classifier. To report the performance for few-shot learning we train the softmax on top of the common representation using the data from the meta-train-train and meta-test-train and then evaluate their performance on meta-train-test and meta-test-test. The result shown in (Fig. 4)




(a) Data denoising for MNIST with 25% label noise and 1000 validation points. Figures show the performance of the classifier learnt using the importance re-weighted dataset during bilevel training. Train accuracy being close to 75% and val accuracy being 100% is an indication that bilevel problem is correctly solved. We see Penalty reaches this point faster than ApproxGrad signifying higher convergence speed. High test accuracy indicates the importance values are enabling good performance on the test set.



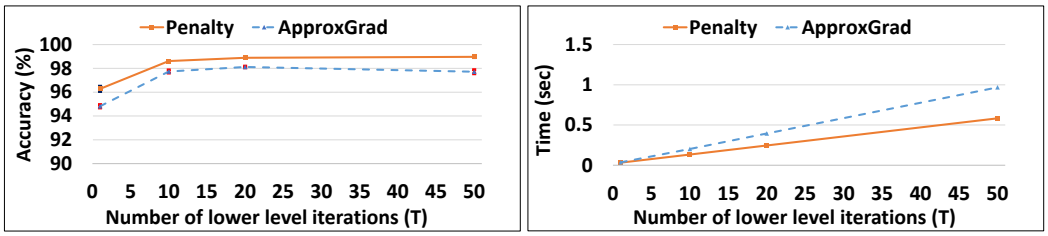

(b) 5-way 5-shot learning on Miniimagenet. Figures show the average performance of the softmax trained using the train sets of meta-train and meta-test sets, respectively, on top of the common representation which is learnt during bilevel training. The accuracy on the test set of meta-train indicates how well the upper-level problem is being solved. The performance on test sets of meta-test show how well the common representation generalizes to new tasks not observed during training.

Figure 4: Comparison of accuracy and wall clock time (convergence speed) during bilevel training of Penalty and ApproxGrad on data denoising problem (Sec. 3.2) and few-shot learning problem (Sec. 3.3).

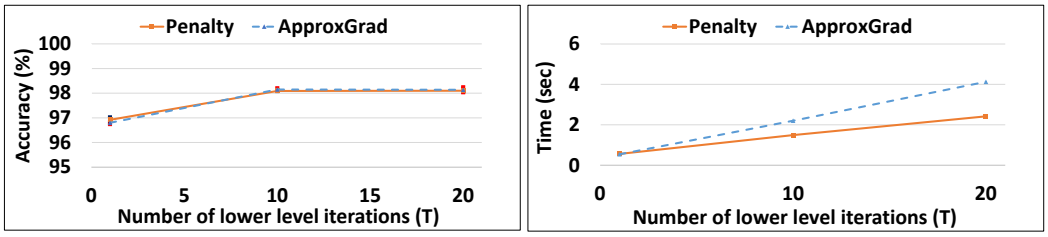
is an average over 600 tasks. The performance on the meta-train-test is indicative of how well the upper-level problem is being solved. Accuracy on meta-test-test indicates the performance of the the learnt representation new tasks which have not been shown during training.

## E Impact of T on accuracy and run-time

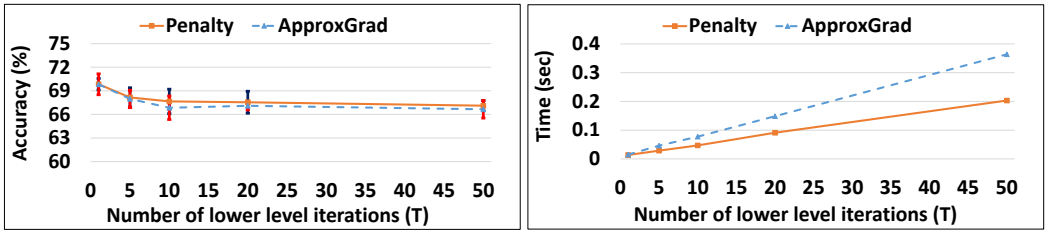
Here, we compare the accuracy and time for Penalty and ApproxGrad (Fig. 5 and Table 4) as we vary the number of lower-level iterations $T$ for different experiments. Intuitively, a larger $T$ corresponds to a more accurate approximation of the hypergradient and therefore improves the results for both the methods. But this improvement comes with a significant increase in the time. Moreover, Fig. 5 shows that relative improvement after $T = 20$ is small in comparison to the increased run-time for Penalty and specially for ApproxGrad. Based on these results we used $T = 20$ for all our experiments on real data for both the methods. The figure also shows that even though Penalty and ApproxGrad have the same linear time complexity (Table 1), Penalty is about twice as fast ApproxGrad in wall-clock time on real experiments.




(a) Importance learning




(b) Few-shot learning




(c) Untargeted data poisoning

Figure 5: Comparison of the final accuracy for different number of lower-level iterations $T$ and wall-clock time required for single upper-level iteration for different values of $T$ for Penalty and ApproxGrad (with $T$ updates for the linear system) on data denoising problem (Sec. 3.2 with 25% noise on MNIST) and few-shot learning problem (Sec. 3.3 with 20 way 5 shot classification on Omniglot) and untargeted data poisoning (Appendix. F.4.2 with 60 poisoned points on MNIST) .

Table 4: Mean wall-clock time (sec) for 10,000 upper-level iterations for synthetic experiments. Boldface is the smallest among RMD, ApproxGrad, and Penalty. (Mean $\pm$ s.d. of 10 runs)

| Example 1 | GD | RMD | ApproxGrad | Penalty | Example 2 | GD | RMD | ApproxGrad | Penalty |
|---|---|---|---|---|---|---|---|---|---|
| T=1 | 7.4±0.3 | **15.0±0.1** | 17.4±0.2 | 17.2±0.1 | T=1 | 7.7±0.1 | 18.5±0.1 | **17.2±0.3** | 17.4±0.2 |
| T=5 | 14.3±0.1 | 51.4±0.3 | 39.3±2.3 | **34.3±0.3** | T=5 | 17.3±0.1 | 62.7±0.1 | 37.9±0.1 | **35.0±0.2** |
| T=10 | 23.2±0.1 | 95.4±0.2 | 60.9±0.3 | **57.0±1.0** | T=10 | 22.4±2.6 | 115.0±0.4 | 64.2±0.3 | **52.7±1.4** |
| Example 3 | GD | RMD | ApproxGrad | Penalty | Example 4 | GD | RMD | ApproxGrad | Penalty |
| T=1 | 8.2±0.2 | **18.8±0.1** | 19.8±0.1 | 19.1±0.1 | T=1 | 7.9±0.1 | **19.5±0.1** | 20.4±0.0 | 19.6±0.1 |
| T=5 | 17.4±0.1 | 72.4±0.1 | 47.1±0.4 | **38.6±0.4** | T=5 | 16.9±0.2 | 72.8±0.5 | 48.4±0.6 | **40.2±0.1** |
| T=10 | 28.7±0.6 | 125.0±9.3 | 80.6±0.3 | **62.7±0.1** | T=10 | 28.3±0.2 | 138.0±0.2 | 81.2±1.6 | **58.0±4.3** |

# F Details of experiments

All codes are written in Python using Tensorflow/Keras, and were run on Intel CORE i9-7920X CPU with 128 GB of RAM and dual NVIDIA TITAN RTX. Implementation and hyperparameters of the algorithms are experiment-dependent and described separately below.

## F.1 Synthetic problems

In this experiment, four simple bilevel problems with known optimal solutions are used to check the convergence of different algorithms. The two problems in Fig. 1 are

$$\min_{u,v} \|u\|^2 + \|v\|^2, \text{ s.t. } v = \arg\min_{v} \|1 - u - v\|^2,$$

Table 5: Upper- and lower-level variable sizes for different experiments

| Experiment | Dataset | Upper-level variable | Lower-level variable |
|---|---|---|---|
| Data denoising | MNIST | 59K | 1.4M |
| | CIFAR10 (Alexnet) | 40K | 1.2M |
| | CIFAR10 (WRN-28-10) | 44K | **36M** |
| | SVHN | 72K | 1.3M |
| Few-shot learning | Omniglot | 111K | 39K |
| | Mini-Imagenet | **3.8M** | 5K |
| Data poisoning | MNIST (Augment 60 poison points) | 47K | 8K |
| | ImageNet (Clean label attack) | 268K | 4K |

and

$$\min_{u,v} \|v\|^2 - \|u-v\|^2, \text{ s.t. } v = \arg\min_{v} \|u-v\|^2,$$

where $u = [u_1, \cdots, u_{10}]^T, \quad |u_i| \leq 5$ and $v = [v_1, \cdots, v_{10}]^T, \quad |v_i| \leq 5$. The optimal solutions are $u_i = v_i = 0.5, \quad i = 1, \cdots, 10$ for the former and $u_i = v_i = 0, \quad i = 1, \cdots, 10$ for the latter. Since there are unique solutions, convergence is measured by the Euclidean distance $\sqrt{\|u-u^*\|^2 + \|v-v^*\|^2}$ of the current iterate $(u, v)$ and the optimal solution $(u^*, v^*)$.

The two problems in Fig. 2 are

$$\min_{u,v} \|u\|^2 + \|v\|^2, \text{s.t.} v = \arg\min_{v} (1-u-v)^T A^T A (1-u-v)$$

and

$$\min_{u,v} \|v\|^2 - (u-v)^T A^T A (u-v), \text{ s.t.} v = \arg\min_{v} (u-v)^T A^T A (u-v),$$

where $A$ is a $5 \times 10$ real matrix such that $A^T A$ is rank-deficient, and the domains are the same as before. These problems are ill-conditioned versions of the previous two problems and are more challenging. The optimal solutions to these two example problems are not unique. For the former, the solutions are $u = 0.5 + p$ and $v = 0.5 + p$ for any vector $p \in \text{Null}(A)$. For the latter, $u = p$ and $v = 0$ for any vector $p \in \text{Null}(A)$. Since they are non-unique, convergence is measured by the residual distance $\sqrt{\|P(u-0.5)\|^2 + \|P(v-0.5)\|^2}$ for the former and $\sqrt{\|Pu\|^2 + \|v\|^2}$ for the latter, where $P = A^T(AA^T)^{-1}A$ is the orthogonal projection to the row-space of $A$.

Algorithms used in this experiment are GD, RMD, ApproxGrad, and Penalty. Adam optimizer is used for minimization everywhere except RMD which uses gradient descent for a simpler implementation. The learning rates common to all algorithms are $\sigma_0 = 10^{-3}$ for $u$-update and $\rho_0 = 10^{-4}$ for $v$- and $p$-updates. For Penalty, the values $\gamma_0 = 1$, $\lambda_0 = 10$, and $\epsilon_0 = 1$ are used. For each problem and algorithm, 20 independent trials are performed with random initial locations $(u_0, v_0)$ sampled uniformly in the domain, and random entries of $A$ sampled from independent Gaussian distributions. We test with $T = 1, 5, 10$. Each run was stopped after $K = 40000$ iterations of $u$-updates.

### F.2 Data denoising by importance learning

Following the formulation for data denoising presented in Eq. (3), we associate an importance value (denoted by $u_i$) with each point in the training data. Our goal is to find the correct values for these $u_i$'s such that the noisy points are given a lower importance values and clean points are given a higher importance values. In our experiments, we allow the importance values to be between 0 and 1. We use the change of variable technique to achieve this. We set $u_i' = 0.5(tanh(u_i) + 1)$ and since $-1 \leq tanh(u_i) \leq 1$, $u_i'$ is automatically scaled between 0 and 1. We use a warm start for the bilevel methods (Penalty and ApproxGrad) by pre-training the network using the validation set and initializing the importance values with the predicted output probability from the pre-trained network. We see an advantage in convergence speed of the bilevel methods with this pre-training. Below we describe the network architectures used for our experiments.

For the experiments on the MNIST dataset, our network consists of a convolution layer with kernel size of 5x5, 64 filters and ReLU activation, followed by a max pooling layer of size 2x2 and a dropout layer with drop rate of 0.25. This is followed by another convolution layer with 5x5 kernel, 128 filters and ReLU activation followed by similar max pooling and dropout layers. Then we have 2 fully connected layers with ReLU activation of size 512 and 256 respectively, each followed by a dropout layer with a drop rate of 0.5. Lastly, we have a softmax layer with 10 classes. We used the Adam optimizer with a learning rate of 0.00001, batch size of 200 and 100 epochs to report the accuracy of Oracle, Val-Only and Train+Val classifiers. For bilevel training using Penalty we used $K = 100$, $T = 20$, $\sigma_0$=3, $\rho_0$=0.00001, $\gamma_0$=0.01, $\epsilon_0$=0.01, $\lambda_0$=0.01, $\nu_0$=0.000001 as per Alg. 2.

For the experiments on the CIFAR10 dataset, our network consists of 3 convolution blocks with filter sizes of 48, 96, and 192. Each convolution block consists of two convolution layers, each with kernel size of 3x3 and ReLU activation. This is followed by a max pooling layer of size 2x2 and a drop out layer with drop rate of 0.25. After these 3 blocks we have 2 dense layers with ReLU activation of size 512 and 256 respectively, each followed by a dropout layer with rate 0.5. Finally we have a softmax layer with 10 classes. This is optimized with the Adam optimizer using a learning rate of 0.001 for 200 epochs with batch size of 200 to report the accuracy of Oracle, Val-Only and Train+Val classifiers. For this experiment we used data augmentation during our training. For the bilevel training using Penalty we used $K = 200$, $T = 20$, $\sigma_0$=3, $\rho_0$=0.00001, $\gamma_0$=0.01, $\epsilon_0$=0.01, $\lambda_0$=0.01, $\nu_0$=0.0001 with mini-batches of size 200. We also use data augmentation for bilevel training.

For the experiments on the SVHN dataset, our network consists of 3 blocks each with 2 convolution layers with kernel size of 3x3 and ReLU activation followed by a max pooling and drop out layer (drop rate = 0.3). The two convolution layers of the first block has 32 filters, second block has 64 filters and the last block has 128 filters. This is followed by a dense layer of size 512 with ReLU activation and dropout layer with drop rate = 0.3. Finally we have a softmax layer with 10 classes. This is optimized with the Adam optimizer and learning rate of 0.001 for 100 epochs to report results of of Oracle, Val-Only and Train+Val classifiers. The bilevel training uses $K = 100$ and $T = 20$, $\sigma_0$=3, $\rho_0$=0.00001, $\gamma_0$=0.01, $\epsilon_0$=0.01, $\lambda_0$=0.01, $\nu_0$=0.0 with batch-size of 200. The test accuracy of these models when trained on the entire training data without any label corruption are 99.5% for MNIST, 86.2% for CIFAR10 and 91.23% for SVHN. For all the experiments with ApproxGrad, we used 20 updates for the lower-level and 20 updates for the linear system and did same number of epochs as for Penalty (i.e. 100 for MNIST and SVHN and 200 for CIFAR), with a mini-batch-size 200.

#### F.2.1 Comparison with [9]

For comparison of Penalty against the RMD-based method presented in [9], we used their setting from Sec. 5.1, which is a smaller version of this data denoising task. For this, we choose a sample of 5000 training, 5000 validation and 10000 test points from MNIST and randomly corrupted labels of 50% of the training points and used softmax regression in the lower-level of the bilevel formulation (Eq. (3)). The accuracy of the classifier trained on a subset of the dataset comprising only of points with importance values greater than 0.9 (as computed by Penalty) along with the validation set is 90.77%. This is better than the accuracy obtained by Val-only (90.54%), Train+Val (86.25%) and the RMD-based method (90.09%) used by [9] and is close to the accuracy achieved by Oracle classifier (91.06%). The bilevel training uses $K = 100$ and $T = 20$, $\sigma_0$=3, $\rho_0$=0.00001, $\gamma_0$=0.01, $\epsilon_0$=0.01, $\lambda_0$=0.01, $\nu_0$=0.0 with batch-size of 200

#### F.2.2 Comparison with [27]

To demonstrate the effectiveness of penalty in solving the importance learning problem with bigger models, we compared its performance against the recent method proposed by [27], which uses a meta-learning approach to find the weights for each example in the noisy training set based on their gradient directions. We use the same setting as their uniform flip experiment with 36% label noise on CIFAR dataset. We also use our own implementation of the Wide Resnet 28-10 (WRN-28-10) which achieves roughly 93% accuracy without any label noise. For comparison, we used the validation set of 1000 points and training set of 44000 points with labels of 36% points corrupted, same as used by [27]. We use Penalty with $T = 1$ since using larger $T$ was not possible due to extremely high computational needs. However, using larger value $T$ is expected to improve the results further based on Fig. 5(a). Different from other experiments in this section we did not use the arc tangent conversion

to restrict importance values between 0 and 1 but instead just normalize the importance values in a batch, similar to the method used by [27], for proper comparison. We used $K = 200$ and $T = 1$, $\sigma_0$=3, $\rho_0$=0.0001, $\gamma_0$=1, $\epsilon_0$=1, $\lambda_0$=10, $\nu_0$=0.0 with batch-size of 75 and used data augmentation during training. We achieve an accuracy of 87.41 $\pm$ 0.26.

### F.3 Few-shot learning

For these experiments, we used the Omniglot [14] dataset consisting of 20 instances (size 28 $\times$ 28) of 1623 characters from 50 different alphabets and the Mini-ImageNet [33] dataset consisting of 60000 images (size 84 $\times$ 84) from 100 different classes of the ImageNet [6] dataset. For the experiments on the Omniglot dataset we used a network with 4 convolution layers to learn the common representation for the tasks. The first three layers of the network have 64 filters, batch normalization, ReLU activation and a 2 $\times$ 2 max-pooling. The final layer is same as the previous ones with the exception that it does not have any activation function. The final representation size is 64. For the Mini-ImageNet experiments we used a residual network with 4 residual blocks consisting of 64, 96, 128 and 256 filters followed by a 1 $\times$ 1 convolution block with 2048 filters, average pooling and finally a 1 $\times$ 1 convolution block with 384 filters. Each residual block consists of 3 blocks of 1 $\times$ 1 convolution, batch normalization, leaky ReLU with leak = 0.1, before the residual connection and is followed by dropout with rate = 0.9. The last convolution block does not have any activation function. The final representation size is 384. Similar architectures have been used by [10] in their work with a difference that we don't use any activation function in the last layers of the representation in our experiments. For both the datasets, the lower-level problem is a softmax regression with a difference that we normalize the dot product of the input representation and the weights with the $l_2$-norm of the weights and the $l_2$-norm of the input representation, similar to the cosine normalization proposed by [17]. For $N$ way classification, the dimension of the weights in the lower-level are 64 $\times$ $N$ for Omniglot and 384 $\times$ $N$ for Mini-ImageNet. For our Omniglot experiments we use a meta-batch-size 30 for 5-way and 20-way classification and a meta-batch-size of 2 for 5-way classification with Mini-ImageNet. We use $T = 20$ iterations for the lower-level in all experiments and ran them for $K$=10000. The hyper-parameters used for Penalty are $\sigma_0$=0.001, $\rho_0$=0.001, $\gamma_0$=0.01, $\epsilon_0$=0.01, $\lambda_0$=0.01, $\nu_0$=0.0001.

### F.4 Training-data poisoning

Here we discuss the details of the clean label data poisoning attack experiment from the main text along with the simple data poisoning attack problem which does not involve any constraint on the amount of perturbation needed to cause poisoning.

#### F.4.1 Clean label data poisoning attack

We solve the following problem for clean label poisoning:

$$\min_{u} \; L_{\mathrm{t}}(u, w) + \|r(t) - r(u)\| \;\; \text{s.t.} \;\; \|x_{\mathrm{base}} - u\|_2 \leq \epsilon \;\; \text{and} \;\; w = \arg\min_{w} \; L_{\mathrm{poison}}(u, w), \quad (10)$$

We use the dog vs. fish image dataset as used by [13], consisting of 900 training and 300 testing examples from each of the two classes. The size of the images in the dataset is 299 $\times$ 299 with pixel values scaled between -1 and 1. Following the setting in Sec. 5.2 of [13], we use the InceptionV3 model with weights pre-trained on ImageNet. We train a dense layer on top of these pre-trained features using the RMSProp optimizer and a learning rate of 0.001 optimized for 1000 epochs before starting bilevel training. Test accuracy obtained with training on clean training data is 98.33. We repeat the exact same procedure as training during evaluation and train the dense layer with training data augmented with poisoned point. For solving the Eq. (10) with Penalty we converted the inequality constraint to an equality constraint by adding a non-negative slack variable. Penalty is optimized with $K = 200$, $T = 10$, $\sigma_0$=0.01, $\rho_0$ = 0.001, $\gamma_0$=1, $\epsilon_0$=1, $\lambda_0$=1.

The experiment shown in Fig. 3 is done on the correctly classified instances from the test set. For a fair comparison with Alg. 1 in [30] we choose the same target and base instance for both the algorithms and generate the poison points. We modify Alg. 1 of [30] in order to constrain the amount of perturbation it adds to the base image to generate the poison point. We achieve this by projecting the perturbation back on to the $l_2$ ball of radius $\epsilon$ whenever it exceeds. This is a standard trick used by several methods which generate adversarial examples for test time attacks. We use $\beta = 0.1,\ \lambda = 0.01$

Table 6: Test accuracy (%) of untargeted poisoning attack (LEFT) and success rate (%) of targeted attack (RIGHT), using MNIST and logistic regression (Mean $\pm$ s.d. of 5 runs).

| | Untargeted Attacks (**lower** accuracy is better) | | | | Targeted Attacks (**higher** accuracy is better) | | | |
|---|---|---|---|---|---|---|---|---|
| Poisoned points | Label flipping | RMD [21] | ApproxGrad | Penalty | Label flipping | RMD [21] | ApproxGrad | Penalty |
| 1% | 86.71±0.32 | 85 | **82.09**±0.84 | **83.29**±0.43 | 7.76±1.07 | 10 | **18.84**±1.90 | **17.40**±3.00 |
| 2% | 86.23± 0.98 | 83 | **77.5**4±0.57 | **78.14**±0.53 | 12.08±2.13 | 15 | **39.64**±3.72 | **41.64**±4.43 |
| 3% | 85.17±0.96 | 82 | **74.41**±1.14 | **75.14**±1.09 | 18.36±1.23 | 25 | **52.76**±2.69 | **51.40**±2.72 |
| 4% | 84.93±0.55 | 81 | **71.88**±0.40 | **72.70**±0.46 | 24.41±2.05 | 35 | **60.01**±1.61 | **61.16**±1.34 |
| 5% | 84.39±1.06 | 80 | **68.69**±0.86 | **69.48**±1.93 | 30.41±4.24 | - | **65.61**±4.01 | **65.52**±2.85 |
| 6% | 84.64±0.69 | 79 | **66.91**±0.89 | **67.59**±1.17 | 32.88±3.47 | - | **71.48**±4.24 | **70.01**±2.95 |

for Alg. 1 of [30] and run it for 2000 epochs in this experiment. For both the algorithms we aim to find the smallest $\epsilon$ that causes misclassification. We incrementally search for the $\epsilon \in \{1, 2, ..., 16\}$ and record the minimum one that misclassifies the particular target. These are then used to report the average distortion in Fig. 3.

### F.4.2 Simple data poisoning attack

We solve the following problem for simple data poisoning:

$$\max_{u} \; L_{\text{val}}(u, w) \;\; \text{s.t.} \;\; w = \arg\min_{w} \; L_{\text{poison}}(u, w), \tag{11}$$

Here, we evaluate Penalty on the task of generating poisoned training data, such that models trained on this data, perform poorly/differently as compared to the models trained on the clean data. We use the same setting as Sec. 4.2 of [21] and test both untargeted and targeted data poisoning on MNIST using data augmentation technique. We assume regularized logistic regression will be the classifier used for training. The poisoned points obtained after solving Eq. (11) by various methods are added to the clean training set and the performance of a new classifier trained on this data is used to report the results in Table 6. For untargeted attack, our aim is to generally lower the performance of the classifier on the clean test set. For this experiment, we select a random subset of 1000 training, 1000 validation and 8000 testing points from MNIST and initialize the poisoning points with random instances from the training set but assign them incorrect random labels. We use these poisoned points along with clean training data to train logistic regression, in the lower-level problem of Eq. (11). For targeted attacks, we aim to misclassify images of eights as threes. For this, we selected a balanced subset (each of the 10 classes are represented equally in the subset) of 1000 training, 4000 validation and 5000 testing points from the MNIST dataset. Then we select images of class 8 from the validation set and label them as 3 and use only these images for the upper-level problem in Eq. (11) with a difference that now we want to minimize the error in the upper level instead of maximizing. To evaluate the performance we selected images of 8 from the test set and labeled them as 3 and report the performance on this modified subset of the original test set in targeted attack section of Table 6. For this experiment the poisoned points are initialized with images of classes 3 and 8 from the training set, with flipped labels. This is because images of threes and eights are the only ones involved in the poisoning. We compare the performance of Penalty against the performance reported using RMD in [21] and ApproxGrad. For ApproxGrad, we used 20 lower-level and 20 linear system updates to report the results in the Table 6. We see that Penalty significantly outperforms the RMD based method and performs similar to ApproxGrad. However, in terms of wall clock time Penalty has a advantage over ApproxGrad (see Fig. 5(c) in Appendix E). We also compared the methods against a label flipping baseline where we select poisoned points from the validation sets and change their labels (randomly for untargeted attacks and mislabel threes as 8 and eights as 3 for targeted attacks). All bilevel methods are able to beat this baseline showing that solving the bilevel problem generates better poisoning points. Examples of the poisoned points for untargeted and targeted attacks generated by Penalty are shown in Figs. 7(b) and 7(c).

For this experiment, we used $l_2$-regularized logistic regression implemented as a single layer neural network with the cross entropy loss and a weight regularization term with a coefficient of 0.05. The model is trained for 10000 epochs using the Adam optimizer with learning rate of 0.001 for training with and without poisoned data. We pre-train the lower-level with clean training data for 5000 epochs with the Adam optimizer and learning rate 0.001 before starting bilevel training. For untargeted attacks, we optimized Penalty with $K = 5000$, $T = 20$, $\sigma_0$=0.1, $\rho_0$ = 0.001, $\gamma_0$=10, $\epsilon_0$=1, $\lambda_0$=100,

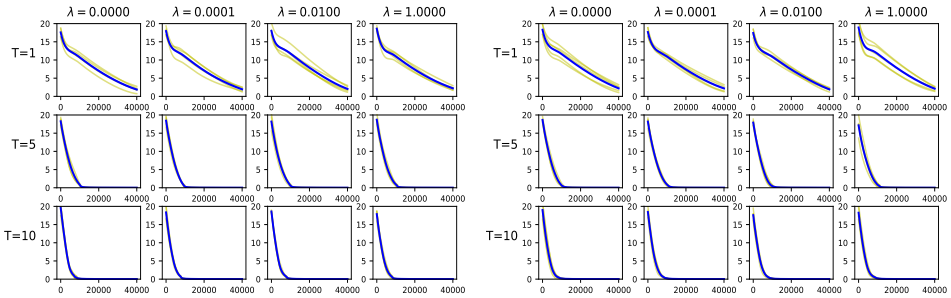



Figure 6: Penalty method for $T$=1,5,10 and $\lambda_0 = 0, 10^{-4}, 10^{-2}, 1$ for Example 1 of Sec.3.1. Top: with $\nu$. Bottom: without $\nu$. Averaged over 5 trials.

Table 7: Effect of using different initial values for various hyper-parameters with Penalty on untargeted data poisoning attacks, Appendix F.4.2 (**lower** accuracy is better) with 60 poisoning points (Mean $\pm$ s.d. of 5 runs with $T$ = 20 (lower-level iterations)). We used the parameters corresponding to the bold values for the results reported in Table 6.

| Hyper-parameters | Different initial values of various hyperparameters | | | |
|---|---|---|---|---|
| $\lambda_0$ | $\lambda_0$ = 0 | $\lambda_0$ = 1 | $\lambda_0$ = 10 | $\lambda_0$ = 100 |
| | 67.87±1.35 | 68.21±1.78 | 68.18±1.04 | **67.59**±1.17 |
| $\nu$ | with $\nu$ | | without $\nu$ | |
| | **67.59**±1.17 | | 68.82±0.75 | |
| $\gamma_0$ | $\gamma_0$ = 1 | $\gamma_0$ = 10 | | $\gamma_0$ = 100 |
| | 73.38±4.98 | **67.59**±1.17 | | 71.96±3.56 |

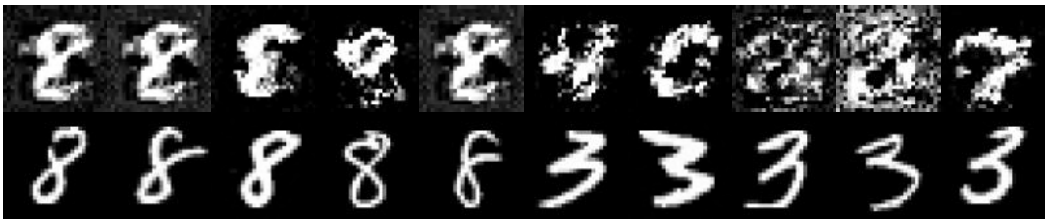
$\nu_0$=0.0. The test accuracy of this model trained on clean data is 87%. For targeted attack, Penalty is optimized with $K = 5000$, $T = 20$, $\sigma_0$=0.1, $\rho_0$ = 0.001, $\gamma_0$=10, $\epsilon_0$=1, $\lambda_0$=1, $\nu_0$=0.0.

### F.5 Impact of various hyperparameters and terms

Here we evaluate the impact of different initial values for the hyperparameters and impact of different terms added in the modified algorithm (Algorithm 2). In particular, we examine the effect of using different initial values of $\lambda_0$ for synthetic experiments and $\lambda_0, \gamma_0$ for untargeted data poisoning with 60 points and also test the effect of having the $\lambda_k g$ and $\nabla_v g^T \nu$ (Fig. 6 and Table 7). Based on the results we find that initial value of the regularization parameter $\lambda_0$ does not influence the results too much and absence of $\lambda_k g$ ($\lambda_k$ = 0) also does not change the results too much. We also don't see significant gains from using the augmented Lagrangian term and method of multipliers on these simple problems. However, the initial value of the parameter $\gamma_0$ does influence the results since starting from very large $\gamma_0$ makes the algorithm focus only on satisfying the necessary condition at the lower level ignoring the $f$ where as with small $\gamma_0$ it can take a large number of iterations for the penalty term to have influence. Apart from these, we also tested the effects of the rate of tolerance decrease ($c_\epsilon$) and penalty increase ($c_\gamma$), and initial value for $\epsilon_0$. Within certain ranges, the results do not change much.

(a) Clean label poisoning attack on dog-fish dataset. The top and middle rows show the target and base instances from the test set and the last row shows the poisoned instances obtained from Penalty. Notice that poisoned images (bottom row) are visually indistinguishable from the base images (middle row) and can evade visual detection.

(b) Untargeted data poisoning attack on MNIST. Top row shows the learned poisoned image using Penalty, starting from the images in the bottom row as initial poisoned images. The column number represents the fixed label of the image, i.e. the label of the images in first column is digit 0, second column is digit 1, etc.



(c) Targeted data poisoning attack on MNIST. Top row shows the learned poisoned images using Penalty, starting from the images in the bottom row as initial poisoned images. Images in the first 5 columns have the fixed label of digit 3, and in the next 5 columns are images with the fixed label of digit 8.

Figure 7: Poisoning points for clean label and simple data poisoning attacks