# Appendix

We provide the missing proofs in Appendix A, review other methods for solving bilevel problems in Appendix B. We discuss the modifications to improve the Alg. 1 in Appendix C, show additional experiment results in Appendix D and conclude by providing experiment details in Appendix E.

## Appendix 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 ($\gamma_1 \le \gamma_2 \le \cdots$), and divergent ($\gamma_k \to \infty$) sequence. Let $\{(u_k, v_k)\}$ be the sequence of approximate solutions to Eq. (10) with tolerance $(\nabla_u \tilde{f}(u_k, v_k))^2 + (\nabla_v \tilde{f}(u_k, v_k))^2 \le \epsilon_k^2$ 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., [Nocedal and Wright (2006)]. 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

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

we have

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

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 $\hat{v} := \arg\min_v \tilde{f}(u,v;\gamma)$ from Eq. (10). Then, $\nabla_u \tilde{f}(u,\hat{v};\gamma) = \frac{df}{du}(u,\hat{v})$ as in Eq. (9).*

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

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

where $\gamma$ disappears, which is the hypergradient $\frac{df}{du}(u,\hat{v})$ as in Eq. (9). □

That is, if we find the minimum $\hat{v}$ of the penalty function for given $u$ and $\gamma$, we get the hypergradient Eq. (9) 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 [Bard (2013)]), and we get the exact hypergradient asymptotically.

## Appendix 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 [Maclaurin et al. (2015); Franceschi et al. (2017)] and approximate gradient [Domke (2012); Pedregosa (2016)] described briefly here.

**Forward-mode (FMD) and Reverse-mode differentiation (RMD).** Domke [Domke (2012)], Maclaurin et al.[Maclaurin et al. (2015)], Franceschi et al. [Franceschi et al. (2017)], and Shaban et al. [Shaban et al. (2018)] 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. (9) 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

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

Time and space Complexity for computing $p_t$ is $O(c)$ since the Jacobian vector product can be computed in $O(c)$ time and space. The final hypergradient for RMD is $\frac{df}{du} = p_0$. Hence the final time complexity for RMD is $O(cT)$ 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 $O(Uc)$ since $P_t A_{t+1}$ can be computed using $U$ Hessian vector products each needing $O(c)$ and $B_{t+1}$ also needs $O(Uc)$ time for 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 $O(cUT)$ 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 [Domke (2012)] 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 [Pearlmutter (1994)]. Hence the time complexity of the method is $O(cT)$ and space complexity is $O(U + V)$ since we only need to store a single copy of $u$ and $v$ same as Penalty. The asymptotic convergence with approximate solutions was shown by [Pedregosa (2016)]. In our experiments we use $T$ steps to solve the linear system.

## Appendix 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 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 [Bertsekas (1976)]:

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

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

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

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 [Hamm and Noh (2018)]. This idea of keeping multiple candidate solutions 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 $\nu_k$.

## Appendix D. Additional experiments

### D.1. Additional comparison of Penalty on the data denoising problem

#### D.1.1. Comparison with [Franceschi et al. (2017)]

For comparison of Penalty against the RMD-based method presented in [Franceschi et al. (2017)], 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

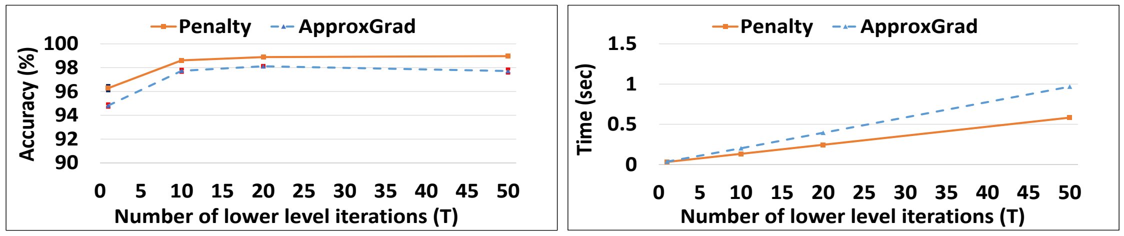
**Algorithm 2** Modified Alg. 1 with regularization and augmented Lagrangian

```
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
        $v_{t+1} \leftarrow v_t - \rho_{k,t} \nabla_v \tilde{f}_2(u_k, v_t)$ (Appendix C.4)
      end for
      $u_{k+1} \leftarrow u_k - \sigma_k \nabla_u \tilde{f}_1(u_k, v_T)$ (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
```

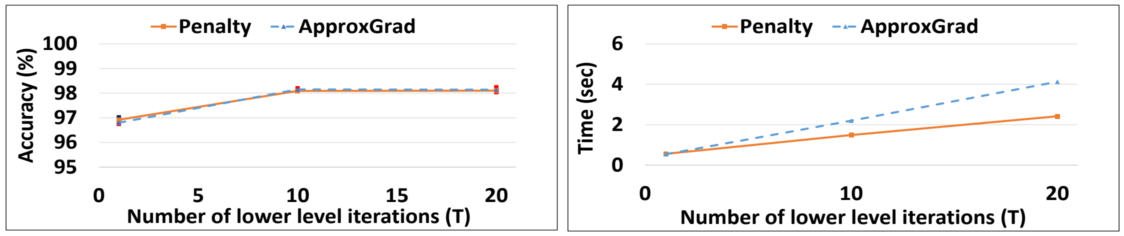
Table 4: Mean wall-clock time (sec) for 10,000 upper-level iterations for synthetic experiments. Boldface is the smallest among RMD, ApproxGrad, and Penalty. (Mean ± s.d. of 10 runs)

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

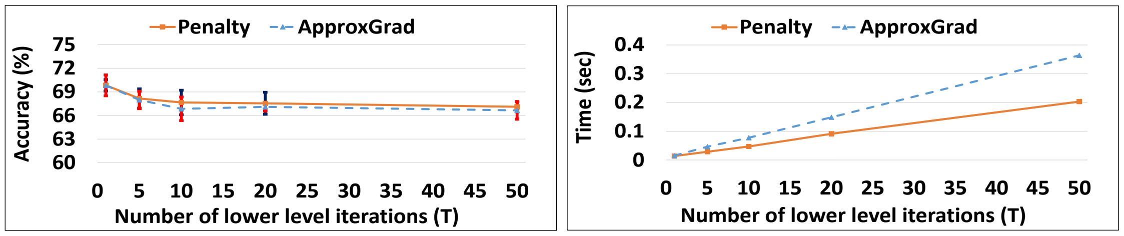
than the accuracy obtained by Val-only (90.54%), Train+Val (86.25%) and the RMD-based method (90.09%) used by [Franceschi et al. (2017)] and is close to the accuracy achieved by the Oracle classifier (91.06%). The bilevel training uses $K = 100$ and $T = 20$, $\sigma_0$=3, $\rho_0$=0.00001, $\gamma_0$=0.01, $\epsilon_0$=0.01, $\lambda_0$=0.01, $\nu_0$=0.0 with batch-size of 200



$(a)$ Importance learning



$(b)$ Few-shot learning



$(c)$ Untargeted data poisoning

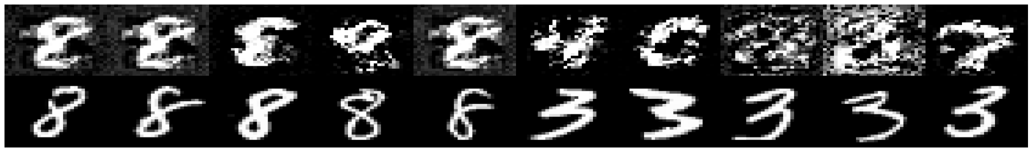
Figure 5: Comparison of the final accuracy for different number of lower-level iterations $T$ and wall-clock time required for single upper-level iteration for different values of $T$ for Penalty and ApproxGrad (with $T$ updates for the linear system) on data denoising problem (Sec. 3.2 with 25% noise on MNIST) and few-shot learning problem (Sec. 3.3 with 20 way 5 shot classification on Omniglot) and untargeted data poisoning (Appendix. D.2 with 60 poisoned points on MNIST) .

### D.1.2. Comparison with [Ren et al. (2018)]

To demonstrate the effectiveness of the penalty in solving the importance learning problem with bigger models, we compared its performance against the recent method proposed by [Ren et al. (2018)], 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 the CIFAR10 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 [Ren et al. (2018)]. We use Penalty with $T = 1$ since using larger $T$ was not possible due to extremely high computational needs. However, using a 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 arctangent conversion to restrict importance values between 0 and 1 but instead just normalize the important values in a batch, similar to the method used by [Ren

($a$) Clean label poisoning attack on dogfish 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. The 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 the first column is digit 0, the second column is digit 1, etc.



($c$) Targeted data poisoning attack on MNIST. The 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 6: Poisoning points for clean label and simple data poisoning attacks

et al. (2018)], 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.

## D.2. Simple data poisoning attack

Here we discuss a simple data poisoning attack problem that does not involve any constraint on the amount of perturbation on the poisoned data. We solve the following bilevel problem

$$\max_u \; L_{\text{val}}(u, w^*) \;\; \text{s.t.} \;\; w^* = \arg\min_w \; L_{\text{poison}}(u, w), \tag{11}$$

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 [Muñoz-González et al. (2017)] and test both untargeted and targeted data poisoning on MNIST using the data augmentation technique. We assume regularized logistic regression will be the classifier used for training.

Table 5: Test accuracy (%) of untargeted poisoning attack (TOP) and success rate (%) of targeted attack (BOTTOM), using MNIST (Mean ± s.d. of 5 runs). Results for RMD are from [Muñoz-González et al. (2017)].

| | Untargeted Attacks (**lower** accuracy is better) | | | | Targeted Attacks (**higher** accuracy is better) | | | |
|---|---|---|---|---|---|---|---|---|
| Poisoned points | Label flipping | RMD | ApproxGrad | Penalty | Label flipping | RMD | 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 | **77.54**±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 |

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 5. For untargeted attacks, 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 the targeted attack section of Table 5. 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 [Muñoz-González et al. (2017)] and ApproxGrad. For ApproxGrad, we used 20 lower-level and 20 linear system updates to report the results in Table 5. We see that Penalty significantly outperforms the RMD based method and performs similar to ApproxGrad. However, in terms of wall-clock time Penalty has an advantage over ApproxGrad (see Fig. 5(c) in Appendix D.3). 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 Fig. 6. 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, $\nu_0$=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.

### D.3. 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 methods. But this improvement comes with a significant increase in time. Moreover, Fig. 5 shows that relative improvement after $T = 20$ is small in comparison to the increased run-time for Penalty and especially for ApproxGrad. Based on these results we used $T = 20$ for all our experiments on real data for both 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.

### D.4. Impact of various hyperparameters and terms

Here we evaluate the impact of different initial values for the hyperparameters and the 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. 7 and Table 6). Based on the results we find that the initial value of the regularization parameter $\lambda_0$ does not influence the results too much and the 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 $f$ whereas 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.

## Appendix E. Details of the 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.

Table 6: Effect of using different initial values for various hyper-parameters with Penalty on untargeted data poisoning attacks, Appendix D.2 (**lower** accuracy is better) with 60 poisoning points (Mean ± 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 5.

| 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 |

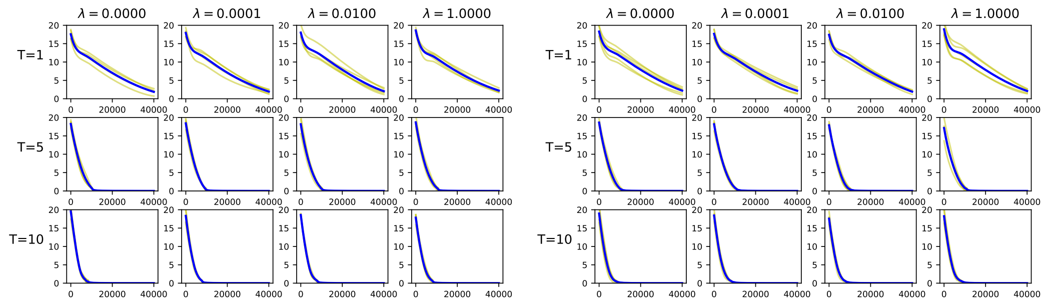


Figure 7: Penalty method for $T$=1,5,10 and $\lambda_0 = 0, 10^{-4}, 10^{-2}, 1$ for Example 1 of Sec.3.1. Top: with $\nu$. Bottom: without $\nu$. Averaged over 5 trials.

## E.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

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

and

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

where $u = [u_1, \cdots, u_{10}]^T$, $|u_i| \leq 5$ and $v = [v_1, \cdots, v_{10}]^T$, $|v_i| \leq 5$. The optimal solutions are $u_i = v_i = 0.5$, $i = 1, \cdots, 10$ for the former and $u_i = v_i = 0$, $i = 1, \cdots, 10$ 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

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

Table 7: 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

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

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 $A$.

The 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.

## E.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 lower importance values and clean points are given 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 $-1 \le tanh(u_i) \le 1$, $u_i'$ 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 the 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 a kernel size of 5x5, 64 filters, and ReLU activation, followed by a max-pooling layer of size 2x2 and a dropout layer with a drop rate of 0.25. This is followed by another convolution layer with a 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 sizes 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 a kernel size of 3x3 and ReLU activation. This is followed by a max-pooling layer of size 2x2 and a drop-out layer with a drop rate of 0.25. After these 3 blocks, we have 2 dense layers with ReLU activation of sizes 512 and 256 respectively, each followed by a dropout layer with a rate of 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 a 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 a 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 have 32 filters, the 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 a dropout layer with a 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 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, is 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 the same number of epochs as for Penalty (i.e. 100 for MNIST and SVHN and 200 for CIFAR), with a mini-batch-size 200.

## E.3. Few-shot learning

For these experiments, we used the Omniglot [Lake et al. (2015)] dataset consisting of 20 instances (size $28 \times 28$) of 1623 characters from 50 different alphabets and the Mini-ImageNet [Vinyals et al. (2016)] dataset consisting of 60000 images (size $84 \times 84$) from 100 different classes of the ImageNet [Deng et al. (2009)] 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 the 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 itefranceschi2018bilevel in their work with the 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 [Luo et al. (2018)]. 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.

## E.4. Clean label data poisoning attack

We solve the following problem for clean label poisoning:

$$\min_u \; L_{\mathrm{t}}(u, w^*) + \|r(t) - r(u)\| \;\text{ s.t. }\; \|x_{\mathrm{base}} - u\|_2 \leq \epsilon \text{ and } w^* = \arg\min_w \; L_{\mathrm{poison}}(u, w), \quad (12)$$

We use the dog vs. fish image dataset as used by [Koh and Liang (2017)], 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 [Koh and Liang (2017)], 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 same procedure as training during evaluation and train the dense layer with training data augmented with a poisoned point. For solving the Eq. (12) 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 [Shafahi et al. (2018)] we choose the same target and base instance for both the algorithms and generate the poison points. We modify Alg. 1 of [Shafahi et al. (2018)] 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 onto 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 $\beta = 0.1$, $\lambda = 0.01$ for Alg. 1 of [Shafahi et al. (2018)] 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.