Regularization Can Help Mitigate Poisoning Attacks... with the Right Hyperparameters

Javier Carnerero-Cano
Imperial College London

Luis Muñoz-González
Imperial College London

Phillippa Spencer
Defence Science and Technology Laboratory DSTL

Emil Constantin Lupu
Imperial College London

REGULARIZATION CAN HELP MITIGATE POISONING ATTACKS... WITH THE RIGHT HYPERPARAMETERS

Javier Carnerero-Cano[1], **Luis Muñoz-González**[1], **Phillippa Spencer**[2], **Emil C. Lupu**[1]
[1]Department of Computing, Imperial College London
180 Queen's Gate, SW7 2AZ, London, United Kingdom
[2]Cyber and Information Systems Division, Defence Science and Technology laboratory (Dstl)
Porton Down, Salisbury, United Kingdom
{j.cano, l.munoz, e.c.lupu}@imperial.ac.uk

ABSTRACT

Machine learning algorithms are vulnerable to poisoning attacks, where a fraction of the training data is manipulated to degrade the algorithms' performance. We show that current approaches, which typically assume that regularization hyperparameters remain constant, lead to an overly pessimistic view of the algorithms' robustness and of the impact of regularization. We propose a novel optimal attack formulation that considers the effect of the attack on the hyperparameters, modelling the attack as a *minimax bilevel optimization problem*. This allows to formulate optimal attacks, select hyperparameters and evaluate robustness under worst case conditions. We apply this formulation to logistic regression using L_2 regularization, empirically show the limitations of previous strategies and evidence the benefits of using L_2 regularization to dampen the effect of poisoning attacks.

1 INTRODUCTION

In many applications, Machine Learning (ML) systems rely for their training on data collected from *untrusted* sources, such as humans, sensors, or IoT devices that can be easily compromised. Spoofed data from such sources can be used to poison the learning algorithms and maximize their error in targeted or indiscriminate ways. The vulnerability of ML algorithms to poisoning attacks has been amply studied (Biggio et al., 2012; Mei & Zhu, 2015; Xiao et al., 2015; Koh & Liang, 2017; Muñoz-González et al., 2017; Steinhardt et al., 2017; Koh et al., 2018; Paudice et al., 2018; Shafahi et al., 2018; Zhang et al., 2018; Diakonikolas et al., 2019; Muñoz-González et al., 2019b; Zhu et al., 2019; Huang et al., 2020; Geiping et al., 2021; Levine & Feizi, 2021), as well as how adversaries can manipulate a fraction of the training data to subvert learning, decrease its overall performance or produce specific errors (Barreno et al., 2010; Huang et al., 2011; Muñoz-González et al., 2019a).

Several systematic poisoning attacks have been proposed to analyze, under worst-case conditions, different families of ML algorithms such as Support Vector Machines (SVMs) (Biggio et al., 2012), other linear classifiers (Xiao et al., 2015; Mei & Zhu, 2015; Koh et al., 2018), and neural networks (Koh & Liang, 2017; Muñoz-González et al., 2017; Huang et al., 2020). These attacks are typically formulated as a bilevel optimization problem where the attacker aims to maximize some malicious objective function (e.g. to maximize the error) by manipulating a fraction of the training data. At the same time, the defender aims to optimize a different objective function to learn the model's parameters, typically evaluated on the poisoned training set.

Some of these attacks target algorithms that have hyperparameters, but consider them constant regardless of the fraction of poisoning points. We show that this can provide a misleading analysis of the robustness of the algorithms, as the hyperparameters' values can change depending on the type and strength of the attack. We propose a novel and more general poisoning attack formulation to test ML algorithms that use hyperparameters in worst-case scenarios. In such cases, the attacker is aware of the dataset used, aims to maximize the overall error and the attack can be modelled as a *bilevel optimization* problem where the outer problem is a *minimax* that includes both the learning of the poisoning points and the hyperparameters and the inner problem involves learning the model's parameters. We have applied our formulation to test the robustness of Logistic Regression (LR) classifiers that use L_2 regularization. We have used *hypergradient* (i.e. the gradient in the outer problem (Maclaurin et al., 2015; Franceschi et al., 2017; 2018; Grazzi et al., 2020)) descent/ascent to solve the minimax bilevel optimization problem. Our experiments show that the results reported in (Xiao et al., 2015) provide an overly pessimistic view of the effect of regularization. When used with the right regularization hyperparameter, L_2 regularization helps partially mitigate the effect of poisoning attacks. We show that the value of the regularization hyperparameter (if optimized) can increase with the attack strength, i.e., the algorithm tries to compensate the negative effect of the attack by increasing the strength of the regularization term. We start by setting the problem, study the effect of regularization in a small illustrative example and then present our findings.

2 General Optimal Poisoning Attacks

In a classification task, given the input space $\mathcal{X} \subseteq \mathbb{R}^m$ and the label space $\mathcal{Y}$, the learner aims to estimate the mapping $f : \mathcal{X} \rightarrow \mathcal{Y}$. Given a training set $\mathcal{D}_{\text{tr}} = \{(\mathbf{x}_{\text{tr}_i}, y_{\text{tr}_i})\}_{i=1}^{n_{\text{tr}}}$ drawn from the underlying distribution $p(\mathcal{X}, \mathcal{Y})$, we can estimate f with a model $\mathcal{M}$ trained by minimizing a loss function $\mathcal{L}(\mathcal{D}_{\text{tr}}, \mathbf{\Lambda}, \mathbf{w})$ w.r.t. its parameters, $\mathbf{w} \in \mathbb{R}^d$, given the hyperparameters $\mathbf{\Lambda} \in \mathbb{R}^c$. Similarly to previous work (Foo et al., 2008; Xiao et al., 2015), we assume the learner has access to a small validation dataset $\mathcal{D}_{\text{val}} = \{(\mathbf{x}_{\text{val}_j}, y_{\text{val}_j})\}_{j=1}^{n_{\text{val}}}$ with n_{val} trusted points, representative of the underlying distribution. This validation set is held out for hyperparameter optimization, i.e. $\min_{\mathbf{\Lambda} \in \Phi(\mathbf{\Lambda})} \mathcal{L}(\mathcal{D}_{\text{val}}, \mathbf{w}^\star)$, where $\mathbf{w}^\star$ are the parameters learned and $\Phi(\cdot)$ is the feasible domain.

In data poisoning attacks the adversary injects a set of n_{p} malicious data points, $\mathcal{D}_{\text{p}} = \{(\mathbf{x}_{\text{p}_k}, y_{\text{p}_k})\}_{k=1}^{n_{\text{p}}}$, in the training set to maximize an objective function $\mathcal{A}$. We assume the attacker can arbitrarily manipulate all the features and the label of the injected points, provided that they remain within the feasible domain of valid data points. We further assume the attacker knows everything about the training data, the feature representation, the loss function, and the ML model. This allows us to analyze worst-case scenario attacks of different strength.

In this work we focus on indiscriminate attacks, where the attacker aims to increase the overall classification error. Therefore, the attacker's objective function $\mathcal{A}$ coincides with the defender's loss function $\mathcal{L}$ evaluated on $\mathcal{D}_{\text{val}}$ at the end of training, i.e. $\mathcal{L}(\mathcal{D}_{\text{val}}, \mathbf{w}^\star)$. Hence, the attacker aims to maximize the loss evaluated on the defender's validation set, which can be formulated as a bilevel optimization problem where the outer objective is the attacker's objective, which depends on the optimal parameters obtained from the inner (training) problem. Previous studies have neglected the effect of the regularization hyperparameter in the attacker's problem (Biggio et al., 2012; Xiao et al., 2015; Mei & Zhu, 2015; Muñoz-González et al., 2017; Koh et al., 2018) and thus incompletely model the outer problem as a maximization w.r.t. the poisoning points. Taking hyperparameters into account, we propose to formulate the outer objective as a minimax problem:

$$\begin{aligned} &\min_{\mathbf{\Lambda} \in \Phi(\mathbf{\Lambda})} \max_{\mathcal{D}_{\text{p}} \in \Phi(\mathcal{D}_{\text{p}})} \mathcal{L}(\mathcal{D}_{\text{val}}, \mathbf{w}^\star) \\ &\quad \text{s.t.} \quad \mathbf{w}^\star \in \operatorname*{arg\,min}_{\mathbf{w} \in \mathcal{W}} \mathcal{L}\left(\mathcal{D}'_{\text{tr}}, \mathbf{\Lambda}, \mathbf{w}\right), \end{aligned} \tag{1}$$

where $\mathcal{D}'_{\text{tr}} = \mathcal{D}_{\text{tr}} \cup \mathcal{D}_{\text{p}}$ is the poisoned dataset. In the more general formulation we propose in (1) it is clear that the poisoning points in $\mathcal{D}'_{\text{tr}}$ have an effect not only on the classifier's parameters, but also on its hyperparameters. This effect has previously been ignored.

Solving the bilevel problem in (1) is strongly NP-Hard (Bard, 2013) and the problem is, in general, non-convex. However, hypergradient-based approaches can be used to obtain (possibly) sub-optimal solutions, i.e. finding saddle points for the minimax problem in (1). To compute the hypergradients for the outer objective, we assume that the loss function, $\mathcal{L}$, is convex and its first and second derivatives are Lipschitz-continuous. Similarly to (Biggio et al., 2012; Mei & Zhu, 2015; Xiao et al., 2015; Muñoz-González et al., 2017) we assume that the label of the poisoning points is set a priori, so the attacker just needs to learn their features $\mathbf{X}_{\text{p}}$.

We use Reverse-Mode Differentiation (RMD) (Domke, 2012; Maclaurin et al., 2015; Franceschi et al., 2017; Muñoz-González et al., 2017; Franceschi et al., 2018; Grazzi et al., 2020) to estimate the hypergradients. RMD requires to first train for T epochs. Then, the hypergradients estimate is computed by reversing the steps followed by the learning algorithm. Compared to grid search, this approach reduces the computational cost, as the algorithm does not need to be trained completely and evaluated for each combination of hyperparameters. After computing the hypergradients, we use projected hypergradient descent/ascent to update the poisoning points and the hyperparameters:

$$\mathbf{X}_{\text{p}} \leftarrow \Pi_{\Phi(\mathcal{D}_{\text{p}})}\left(\mathbf{X}_{\text{p}} + \alpha\, \nabla_{\mathbf{X}_{\text{p}}}\mathcal{A}\right), \qquad \mathbf{\Lambda} \;\leftarrow \Pi_{\Phi(\mathbf{\Lambda})}\left(\mathbf{\Lambda} - \beta\, \nabla_{\mathbf{\Lambda}}\mathcal{A}\right), \tag{2}$$

where α and β are respectively the learning rates and $\Pi_{\Phi(\mathcal{D}_{\text{p}})}$ and $\Pi_{\Phi(\mathbf{\Lambda})}$ are the projection operators for the features of the poisoning points, $\mathbf{X}_{\text{p}}$, and the hyperparameters, $\mathbf{\Lambda}$.

3 Illustrating the mitigating effect of L_2 Regularization

L_2 (or Tikhonov) regularization is often used to increase the stability of ML algorithms (Xu et al., 2011). Using L_2 regularization we add a penalty term to the original loss function, which shrinks the norm of the model's parameters, so that $\mathcal{L}(\mathcal{D}_{\text{tr}}, \mathbf{\Lambda}, \mathbf{w}) = \mathcal{L}(\mathcal{D}_{\text{tr}}, \mathbf{w}) + \frac{e^{\lambda}}{2}\,||\mathbf{w}||_2^2$, where λ is the hyperparameter controlling the strength of the regularization term.

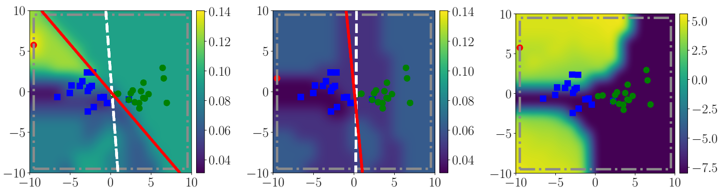


Figure 1: Effect of regularization on a synthetic example with green and blue classes. The red point is the poisoning point (labelled as green). The dashed-dotted grey box represent the attacker's constraints. (Left) Standard LR with no regularization. (Center) LR with L_2 regularization ($\lambda \approx 3$). (Right) Value of λ learned that minimizes the validation error as a function of the injected poisoning point.

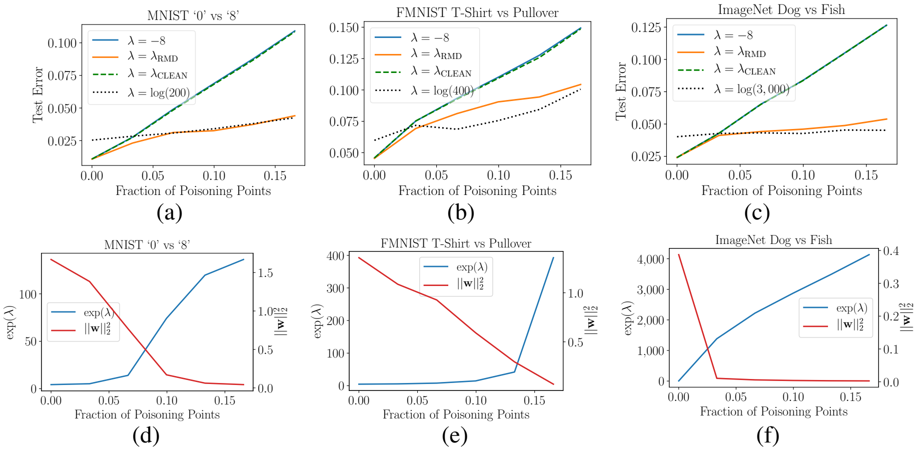
In Fig. 1 we illustrate the effect of regularization and the limitations of the approach in (Xiao et al., 2015) using a synthetic example with an LR classifier. Fig. 1(left) shows the effect of injecting a single poisoning point that maximizes the error (measured on a separate validation set) against a non-regularized LR classifier. The dashed-white line represents the decision boundary learned when training on the clean dataset, whereas the red line depicts the decision boundary when training on the poisoned dataset. We can observe that a single poisoning point can significantly alter the decision boundary. Fig. 1(center), shows a similar scenario, but training an LR classifier with L_2 regularization, setting a large value for λ ($\lambda \approx 3$). Here, we can observe that the effect of the poisoning is significantly reduced and the decision boundary shifts only slightly. In the background of Fig. 1(left) and Fig. 1(center) we represent the validation error of the LR trained on a poisoned dataset as a function of the location of the poisoning point. We can observe that, when there is no regularization (left) the error can significantly increase when we inject the poisoning point in certain regions. On the contrary, when regularization is applied (center), the colormap is more uniform, i.e. the algorithm is quite stable regardless of the position of the poisoning point.

Fig. 1(right) shows how the value of the regularization hyperparameter, λ, changes as a function of the poisoning point. The colormap in the background represents the value of λ that minimizes the error on the validation set. We can observe that λ changes significantly depending on the position of the poisoning point. Thus, λ is much bigger for the regions where the poisoning point can influence the classifier more (Fig. 1(left)). Thus, when the poisoning attack can have a very negative impact on the classifier's performance, the importance of the regularization term, controlled by λ, increases. It is thus clear that selecting the value of λ appropriately can have a significant impact on the classifier's robustness. Furthermore, when testing the robustness of L_2-regularized classifiers it is necessary to consider the interplay between the attack strength and the value of λ.

4 Experiments

In this section, we evaluate the effectiveness of the attack strategy in (1) against LR.[1] We use three different binary classification problems: MNIST ('0' vs '8') (LeCun et al., 1998), Fashion-MNIST (FMNIST) (*T-shirt* vs *pullover*) (Xiao et al., 2017), and ImageNet (*dog* vs *fish*) (Russakovsky et al.,

[1] The PyTorch implementation of the algorithms used for the experiments is available at https://github.com/javiccano/regularization-and-poisoning—secml.



(a) (b) (c)

(d) (e) (f)

Figure 2: Results for the optimal attack against LR: The first row represents the average test error on (a) MNIST, (b) FMNIST, and (c) ImageNet. The second row contains the plots for the average λ and $||\mathbf{w}||_2^2$ for (d) MNIST, (e) FMNIST, and (f) ImageNet.

2015) preprocessed as in (Koh & Liang, 2017) (we use the same Inception-v3 (Szegedy et al., 2016) features).[2] We assume the attacker knows the value of λ for the defender. To evaluate the effect of the regularization hyperparameter, we craft optimal poisoning attacks, setting the value of λ to different constant values: a very small regularization term, a very large one, and the value of λ optimized with 5-fold cross-validation, training the model on the clean dataset (λ_{CLEAN}). The latter is similar to the settings used in (Xiao et al., 2015). We compare these values with the value of λ_{RMD} calculated according to Eq. (1).

In Fig. 2(a)-(c) we can observe that for the small λ and λ_{CLEAN} the attacks are very effective and the test error increases significantly when compared to the algorithm's performance on the clean dataset. In contrast, for the largest λ the test error increases only slightly with the increasing fraction of poisoning points, showing a stable performance regardless of the attack strength. However, in the absence of an attack, the algorithm clearly *underfits* and the error is significantly higher compared to the other models. When the value of λ is learned (λ_{RMD}) by solving the problem in (1), the increase in the test error remains significantly smaller and the algorithm has a similar performance to the case where the value of λ is large as the fraction of poisoning points increases. In this sense, the error does not increase further by injecting more poisoning points. In the absence of an attack, the performance for λ_{RMD} is as good as in the case of λ_{CLEAN}. These results show that the attack and the methodology presented in (Xiao et al., 2015) provide an overly pessimistic view on the robustness of L_2 regularization to poisoning attacks and that, by appropriately selecting the value of λ, we can effectively reduce the impact of poisoning attacks. We can also observe that there is a trade-off between accuracy and robustness: setting a very large value for λ makes the algorithm more robust, but the performance on clean data is degraded. Our formulation allows to learn the value of λ most appropriate for the training set.

In Fig. 2(d)-(f) we show the value of λ learned and the norm of the model's parameters, $||\mathbf{w}||_2^2$, as a function of the fraction of poisoning points injected for the solution of problem (1) with RMD. We can observe that in all cases, the regularization hyperparameter increases as we increase the fraction of poisoning points. This confirms that the regularization term tries to compensate the effect of the malicious samples on the model's parameters. Thus, as expected, L_2 provides a natural mechanism to stabilize the model in the presence of attacks. For ImageNet, the order of magnitude of λ is higher, as there are more features in the model. In Fig. 2(d)-(f) we can also observe that, as expected from the properties of L_2 regularization, when λ increases, the norm of the parameters decreases.

5 Conclusions

We have shown that previous approaches to optimal poisoning attacks, where the model's hyperparameters are considered constant, provide a misleading view of the algorithms' robustness. We

[2] The details of the experimental setup and hardware used for the experiments can be found in the Appendix.

have evidenced that poisoning attacks can have a strong influence on the hyperparameters learned, and thus, their influence must be considered when assessing algorithm robustness. To achieve this, we have introduced a novel worst-case poisoning attack strategy to evaluate the robustness of ML classifiers that contain hyperparameters. The attack is formulated as a minimax bilevel optimization problem that can be solved with hypergradient-based techniques. We have shown that choosing a fixed a priori value of λ can be detrimental: if the value is too high it damages accuracy, if the value is too low it damages robustness. In this sense, in contrast to the results reported in (Xiao et al., 2015) (which uses a fixed regularization hyperparameter), when facing stronger attacks i.e., more poisoning points, the regularization hyperparameter can also increase to compensate the instability that the attacker tries to produce in the algorithm. It is therefore important for organizations training models from data that may be untrusted, to regularly adjust hyperparameters to adapt the performance and robustness of the algorithm to current threats.

ACKNOWLEDGMENTS

We gratefully acknowledge financial support for this work from the UK Defence Science and Technology Laboratory (Dstl); contract no: DSTLX-1000120987.

REFERENCES

J. F. Bard. *Practical Bilevel Optimization: Algorithms and Applications*, volume 30. Springer Science & Business Media, 2013.

M. Barreno, B. Nelson, A. D. Joseph, and J. D. Tygar. The security of machine learning. *Machine Learning*, 81(2):121–148, 2010.

B. Biggio, B. Nelson, and P. Laskov. Poisoning Attacks against Support Vector Machines. In *International Conference on Machine Learning*, pp. 1807–1814, 2012.

C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

I. Diakonikolas, G. Kamath, D. Kane, J. Li, J. Steinhardt, and A. Stewart. Sever: A Robust Meta-Algorithm for Stochastic Optimization. In *International Conference on Machine Learning*, pp. 1596–1606, 2019.

J. Domke. Generic Methods for Optimization-Based Modeling. In *Artificial Intelligence and Statistics*, pp. 318–326, 2012.

C.-S. Foo, C. B. Do, and A. Y. Ng. Efficient multiple hyperparameter learning for log-linear models. In *Advances in Neural Information Processing Systems*, pp. 377–384, 2008.

L. Franceschi, M. Donini, P. Frasconi, and M. Pontil. Forward and Reverse Gradient-Based Hyperparameter Optimization. In *International Conference on Machine Learning*, pp. 1165–1173, 2017.

L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil. Bilevel Programming for Hyperparameter Optimization and Meta-Learning. In *International Conference on Machine Learning*, pp. 1568–1577, 2018.

J. Geiping, L. Fowl, W. R. Huang, W. Czaja, G. Taylor, M. Moeller, and T. Goldstein. Witches' Brew: Industrial Scale Data Poisoning via Gradient Matching. In *International Conference on Learning Representations*, 2021.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

R. Grazzi, L. Franceschi, M. Pontil, and S. Salzo. On the Iteration Complexity of Hypergradient Computation. In *International Conference on Machine Learning*, pp. 3748–3758, 2020.

A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, volume 105. SIAM, 2008.

L. Huang, A. D. Joseph, B. Nelson, B. I. P. Rubinstein, and J. D. Tygar. Adversarial Machine Learning. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, pp. 43–58. ACM, 2011.

W. R. Huang, J. Geiping, L. Fowl, G. Taylor, and T. Goldstein. MetaPoison: Practical General-purpose Clean-label Data Poisoning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 12080–12091, 2020.

P. W. Koh and P. Liang. Understanding Black-box Predictions via Influence Functions. In *International Conference on Machine Learning*, pp. 1885–1894, 2017.

P. W. Koh, J. Steinhardt, and P. Liang. Stronger Data Poisoning Attacks Break Data Sanitization Defenses. *arXiv preprint arXiv:1811.00741*, 2018.

Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

A. Levine and S. Feizi. Deep Partition Aggregation: Provable Defense against General Poisoning Attacks. In *International Conference on Learning Representations*, 2021.

D. Maclaurin, D. Duvenaud, and R. Adams. Gradient-based Hyperparameter Optimization through Reversible Learning. In *International Conference on Machine Learning*, pp. 2113–2122, 2015.

S. Mei and X. Zhu. Using Machine Teaching to Identify Optimal Training-Set Attacks on Machine Learners. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pp. 2871–2877, 2015.

L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli. Towards Poisoning of Deep Learning Algorithms with Back-gradient Optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pp. 27–38. ACM, 2017.

L. Muñoz-González, J. Carnerero-Cano, K. T. Co, and E. C. Lupu. Challenges and Advances in Adversarial Machine Learning. *Resilience and Hybrid Threats: Security and Integrity for the Digital World*, 55:102, 2019a.

L. Muñoz-González, B. Pfitzner, M. Russo, J. Carnerero-Cano, and E. C. Lupu. Poisoning Attacks with Generative Adversarial Nets. *arXiv preprint arXiv:1906.07773*, 2019b.

A. Paudice, L. Muñoz-González, A. György, and E. C. Lupu. Detection of Adversarial Training Examples in Poisoning Attacks through Anomaly Detection. *arXiv preprint arXiv:1802.03041*, 2018.

B. A. Pearlmutter. Fast Exact Multiplication by the Hessian. *Neural Computation*, 6(1):147–160, 1994.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein. Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks. In *Advances in Neural Information Processing Systems*, pp. 6103–6113, 2018.

J. Steinhardt, P. W. W. Koh, and P. S. Liang. Certified Defenses for Data Poisoning Attacks. In *Advances in Neural Information Processing Systems*, pp. 3517–3529, 2017.

C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception Architecture for Computer Vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826, 2016.

H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli. Is Feature Selection Secure against Training Data Poisoning? In *International Conference on Machine Learning*, pp. 1689–1698, 2015.

H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

H. Xu, C. Caramanis, and S. Mannor. Sparse Algorithms Are Not Stable: A No-Free-Lunch Theorem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(1):187–193, 2011.

X. Zhang, X. Zhu, and S. Wright. Training Set Debugging Using Trusted Items. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

C. Zhu, W. R. Huang, H. Li, G. Taylor, C. Studer, and T. Goldstein. Transferable Clean-Label Poisoning Attacks on Deep Neural Nets. In *International Conference on Machine Learning*, pp. 7614–7623, 2019.

A Hessian-Vector Products

Let $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^m$, $\mathbf{y} \in \mathcal{Y} \subseteq \mathbb{R}^n$, $\mathbf{v} \in \mathbb{R}^m$, and $f(\mathbf{x},\mathbf{y}) \in \mathbb{R}$. If the second partial derivatives of $f(\mathbf{x},\mathbf{y})$ are continuous *almost everywhere*[3] in $\mathcal{X}$ and $\mathcal{Y}$ (mild condition), the Hessian-vector products $\left(\nabla^2_{\mathbf{x}} f(\mathbf{x},\mathbf{y})\right)\mathbf{v}$ and $\left(\nabla_{\mathbf{y}}\nabla_{\mathbf{x}} f(\mathbf{x},\mathbf{y})\right)^{\mathsf{T}}\mathbf{v}$ can be computed exactly and efficiently by using the following identities (Pearlmutter, 1994):

$$\begin{aligned} \left(\nabla^2_{\mathbf{x}} f(\mathbf{x},\mathbf{y})\right)\mathbf{v} &= \nabla_{\mathbf{x}}\left(\mathbf{v}^{\mathsf{T}}\nabla_{\mathbf{x}} f(\mathbf{x},\mathbf{y})\right), \\ \left(\nabla_{\mathbf{y}}\nabla_{\mathbf{x}} f(\mathbf{x},\mathbf{y})\right)^{\mathsf{T}}\mathbf{v} &= \nabla_{\mathbf{y}}\left(\mathbf{v}^{\mathsf{T}}\nabla_{\mathbf{x}} f(\mathbf{x},\mathbf{y})\right). \end{aligned} \tag{3}$$

The upper and lower expressions scale as $\mathcal{O}(m)$ and $\mathcal{O}(\max(m,n))$, respectively, both in time and space. Analogous expressions can be obtained when $\mathbf{v} \in \mathbb{R}^n$. An elegant aspect of this technique is that, for machine learning models optimized with gradient-based methods, the equations for exactly evaluating the Hessian-vector products emulate closely those for standard forward and backward propagation. Hence, the application of existing automatic differentiation frameworks to compute this product is typically straightforward (Pearlmutter, 1994; Bishop, 2006).

B Reverse-Mode Differentiation Algorithm

As described in (Franceschi et al., 2017), we can think of the training algorithm as a discrete-time dynamical system, described by a set of states $\mathbf{s}^{(t)} \in \mathbb{R}^{d_s}$, with $t = 0, \ldots, T$, where each state depends on model's parameters, the accumulated gradients and/or the velocities. In this paper, we focus on stochastic gradient descent, so that $\mathbf{s}^{(t)} = \mathbf{w}^{(t)}$. Then, from a reduced number of training iterations, T, we can estimate the hypergradients from the values of the parameters collected in the set of states. Depending on the order to compute the operations we can differentiate two approaches to estimate the hypergradients: Reverse-Mode (RMD) and Forward-Mode Differentiation (FMD) (Griewank & Walther, 2008; Franceschi et al., 2017). In the first case, RMD requires first to train the learning algorithm for T epochs, and then, to compute $\mathbf{w}^{(0)}$ to $\mathbf{w}^{(T)}$. Then, the hypergradients estimate is computed by reversing the steps followed by the learning algorithm from $\mathbf{w}^{(T)}$ down to $\mathbf{w}^{(0)}$. On the other hand, FMD computes the estimate of the hypergradients as the algorithm is trained, i.e. from $\mathbf{w}^{(0)}$ to $\mathbf{w}^{(T)}$ (i.e. the estimates can be computed in parallel with the training procedure).

To estimate the hypergradients, RMD requires to compute a forward and a backward pass through the set of states. In contrast, FMD just needs to do the forward computation. However, the scalability of FMD depends heavily on the number of hyperparameters compared to RMD. Then, for problems where the number of hyperparameters is large, as is the case for the poisoning attacks we introduced in the paper, RMD is computationally more efficient to estimate the hypergradients. For this reason, we used RMD in our experiments.

Here we include the Reverse-Mode Differentiation (RMD) algorithm (Alg. 1) (Domke, 2012; Maclaurin et al., 2015; Franceschi et al., 2017; Muñoz-González et al., 2017; Franceschi et al., 2018;

[3] A property that holds almost everywhere holds throughout all space except on a set of *measure zero*. Intuitively, a set of measure zero occupies a negligible volume in the measured space (Goodfellow et al., 2016).

Algorithm 1 Reverse-Mode Differentiation

```
Input: $\mathcal{M}$, $\mathcal{A}$, $\mathcal{L}$, $\mathcal{D}_{\text{val}}$, $\mathcal{D}'^{(\tau)}_{\text{tr}}$, $\mathbf{\Lambda}^{(\tau)}$, $\mathbf{w}^{(0)}$, $T$, $\eta$
Output: $\nabla_{\mathbf{X}_\text{p}} \mathcal{A}\left(\mathbf{w}^{(T)}\right)$
for $t = 0$ to $T - 1$ do
    $\mathbf{g} \leftarrow \nabla_{\mathbf{w}} \mathcal{L}\left(\mathbf{w}^{(t)}\right)$
    $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \mathbf{g}$                // stochastic gradient descent
end for
$d\mathbf{X}_\text{p}^{(T)} \leftarrow \mathbf{0}$
$d\mathbf{w}^{(T)} \leftarrow \nabla_{\mathbf{w}} \mathcal{A}\left(\mathbf{w}^{(T)}\right)$
for $t = T - 1$ down to $0$ do
    $d\mathbf{g}_\text{w} \leftarrow \left(\nabla^2_{\mathbf{w}} \mathcal{L}\left(\mathbf{w}^{(t)}\right)\right) d\mathbf{w}^{(t+1)}$        // Hessian-vector product (Sect. A)
    $d\mathbf{g}_{\text{X}_\text{p}} \leftarrow \left(\nabla_{\mathbf{X}_\text{p}} \nabla_{\mathbf{w}} \mathcal{L}\left(\mathbf{w}^{(t)}\right)\right)^\mathsf{T} d\mathbf{w}^{(t+1)}$  // Hessian-vector product (Sect. A)
    $d\mathbf{w}^{(t)} \leftarrow d\mathbf{w}^{(t+1)} - \eta d\mathbf{g}_\text{w}$
    $d\mathbf{X}_\text{p}^{(t)} \leftarrow d\mathbf{X}_\text{p}^{(t+1)} - \eta d\mathbf{g}_{\text{X}_\text{p}}$
end for
$\nabla_{\mathbf{X}_\text{p}} \mathcal{A}\left(\mathbf{w}^{(T)}\right) \leftarrow d\mathbf{X}_\text{p}^{(0)}$
```

Grazzi et al., 2020), which we use to compute the hypergradient estimate at the outer level problem (both for the features of the poisoning points, $\mathbf{X}_\text{p}$, and the hyperparameters, $\mathbf{\Lambda}$). We make use of a similar notation to the algorithms presented in (Maclaurin et al., 2015; Muñoz-González et al., 2017). The resulting algorithm for the case of $\mathbf{\Lambda}$ is analogous.

On the other hand, the choice of the number of epochs for the inner problem, T, depends on the model and the training dataset. Low values of T could lead to low-quality approximations for the hypergradient. As T increases, the solution of RMD approaches the exact (true) hypergradient, but at the risk of overfitting the outer objective in the bilevel optimization problem (Franceschi et al., 2018).

C Experimental Settings

In the following specifications, $T_{\mathcal{D}_\text{p}}$ denotes the number of hyperiterations at the outer problem when the poisoning points are learned and λ is fixed, T_λ when the training set is clean and λ is learned, and T_mul when $\mathbf{X}_\text{p}$ and λ are learned in a coordinate manner. Finally, η_tr is the learning rate when testing the attack.

C.1 Synthetic Example

For the synthetic experiment shown in Sect. 4 of the paper, we sample the attacker's data from two bivariate Gaussian distributions, $\mathcal{N}(\boldsymbol{\mu}_0, \mathbf{\Sigma}_0)$ and $\mathcal{N}(\boldsymbol{\mu}_1, \mathbf{\Sigma}_1)$, with mean vectors $\boldsymbol{\mu}_i$ and covariance matrices $\mathbf{\Sigma}_i$ for each class, $i = 0, 1$:

$$\boldsymbol{\mu}_0 = \begin{bmatrix} -3.0 \\ 0.0 \end{bmatrix}, \qquad \mathbf{\Sigma}_0 = \begin{bmatrix} 2.5 & 0.0 \\ 0.0 & 1.5 \end{bmatrix},$$

$$\boldsymbol{\mu}_1 = \begin{bmatrix} 3.0 \\ 0.0 \end{bmatrix}, \qquad \mathbf{\Sigma}_1 = \begin{bmatrix} 2.5 & 0.0 \\ 0.0 & 1.5 \end{bmatrix}.$$

The attacker uses 32 points (16 per class) for training and 64 (32 per class) for validation, and one poisoning point cloned from the validation set (in the example of the paper, cloned from the set labelled as blue), whose label is flipped. This poisoning point is concatenated into the training set and the features of this point are optimized with RMD. In order to poison the LR classifier, we use a learning rate and a number of hyperiterations for the poisoning point $\alpha = 0.4$, $T_{\mathcal{D}_\text{p}} = 50$; the feasible domain $\Phi(\mathcal{D}_\text{p}) \in [-9.5, 9.5]^2$; the learning rate and number of epochs for the inner problem $\eta = 0.2$, $T = 500$; and when evaluating the attack, $\eta_\text{tr} = 0.2$, batch size $= 32$ (full batch), and number of epochs $= 100$. When we apply regularization, we fix $\lambda = \log(20) \approx 3$.

Name	# Training Samples	# Validation Samples	# Test Samples	# Features
MNIST (‘0’ vs ‘8’)	512	171	$1,954$	784
FMNIST (*T-shirt* vs *pullover*)	512	171	$2,000$	784
ImageNet (*dog* vs *fish*)	512	171	600	$2,048$

Table 1: Characteristics of the datasets used in the experiments.

Name	$T_{\mathcal{D}_\text{p}}$	T_λ	T_mul	α	β
MNIST (‘0’ vs ‘8’) (LR)	100	50	100	0.99	0.80
FMNIST (*T-shirt* vs *pullover*) (LR)	100	50	100	0.90	0.30
ImageNet (*dog* vs *fish*) (LR)	100	50	200	0.90	0.40

Name	$\Phi(\mathcal{D}_\text{p})$	$\Phi(\lambda)$	η	T
MNIST (‘0’ vs ‘8’) (LR)	$[0.0, 1.0]^{784}$	$[-8, \log(200)]$	0.10	150
FMNIST (*T-shirt* vs *pullover*) (LR)	$[0.0, 1.0]^{784}$	$[-8, \log(400)]$	0.08	200
ImageNet (*dog* vs *fish*) (LR)	$[-0.5, 0.5]^{2,048}$	$[-8, \log(20,000)]$	0.05	200

Table 2: Experimental settings for the poisoning attack.

Name	η_tr	Batch Size	Number of Epochs
MNIST (‘0’ vs ‘8’) (LR)	10^{-2}	64	200
FMNIST (*T-shirt* vs *pullover*) (LR)	10^{-2}	64	200
ImageNet (*dog* vs *fish*) (LR)	10^{-3}	64	300

Table 3: Experimental settings for training the models.

To plot the colormap that shows the value of the regularization hyperparameter learned (Fig. 1(right) of the paper) as a function of the poisoning point injected in the training set, the values of λ explored for each possible poisoning point are in the range $[-8, 6]$. Then, the optimal value of λ is chosen such that it minimizes the error of the model, trained on each combination of the poisoning point (concatenated into the training set) and λ in the grid, and evaluated on the validation set.

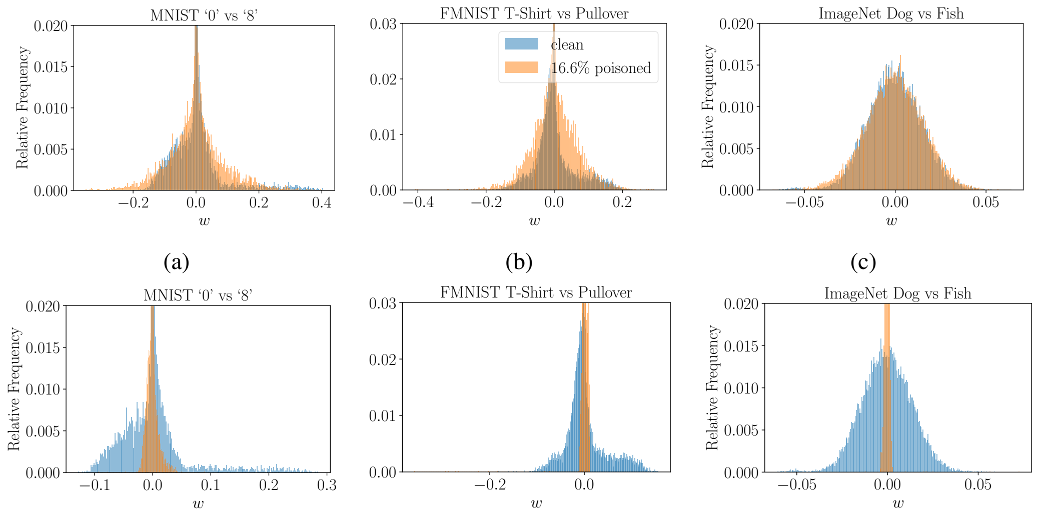
C.2 MNIST, FMNIST and ImageNet

In our experiments, all the results for MNIST ‘0’ vs ‘8’ (LeCun et al., 1998), Fashion-MNIST (FMNIST) *T-shirt* vs *pullover* (Xiao et al., 2017), and ImageNet *dog* vs *fish* (Russakovsky et al., 2015) (preprocessed as in (Koh & Liang, 2017)) are the average of 10 repetitions with different random data splits for training and validation, whereas the test set is fixed. For all the attacks we measure the average test error for different attack strengths, where the number of poisoning points is in the range between 0 and 85. To reduce the computational cost, the size of the batch of poisoning points that are simultaneously optimized is 17 for all the datasets. This way, we simulate six different ratios of poisoning ranging from 0% to 16.6%.

The details of each dataset are included in Table 1. All the datasets are balanced. Moreover, both MNIST and FMNIST sets are normalized to be in the range $[0, 1]^{784}$, whereas for the ImageNet sets we use the same Inception-v3 (Szegedy et al., 2016) features as in (Koh & Liang, 2017), and normalized them with respect to their training mean and standard deviation.

For all the experiments, we make use of stochastic gradient descent both to update the parameters in the forward pass of RMD[4] (full batch training), and to train the model when testing the attack

[4] To refine the solutions obtained, when the poisoning points are learned and λ is fixed, and when the training set is clean and λ is learned, if the optimization algorithm gets stuck in a poor local optimum we restart the bilevel optimization procedure. In the case of the poisoning points, we reinitialize them with different values uniformly sampled without duplicates from the validation set. For the regularization hyperparameters, let $\lambda^{(\tau)}$ denote their latest value: We reinitialize them with values uniformly sampled from the range $\left[\lambda^{(\tau)} - 0.5, \lambda^{(\tau)} + 0.5\right]$. It is however noteworthy that these reinitializations were not crucial to obtain the results shown in the paper. The exploration of warm-restart techniques in the case of minimax bilevel problems is left for future work.



(a) (b) (c)

(d) (e) (f)

Figure 3: Histograms (in terms of relative frequency) of the LR's parameters. For a better visualisation, the upper part of some subfigures has been omitted. The first row represents the case where no regularisation is applied: (a) MNIST, (b) FMNIST, and (c) ImageNet. The second row shows the case where λ is learned with RMD: (d) MNIST, (e) FMNIST, and (f) ImageNet.

(mini-batch training). The details of the attack settings are shown in Table 2, whereas the ones for training are in Table 3. Additionally, λ_{CLEAN} is optimized with 5-fold cross-validation—training the model on the clean dataset, as in (Xiao et al., 2015). For all the datasets the ranges of values of λ explored to compute λ_{CLEAN} is $[-8, 1]$ (no better performance was observed for larger values of λ). On the other hand, to accelerate the optimization of λ_{RMD}, when the training set is clean these hyperparameters are warm-started with a value $\lambda = \log(5) \approx 1.61$.

C.3 Details of the Hardware Used

All the experiments are run on 2×11 GB NVIDIA GeForce® GTX 1080 Ti GPUs. The RAM memory is 64 GB (4×16 GB) Corsair VENGEANCE DDR4 3000 MHz. The processor (CPU) is Intel® Core™ i7 Quad Core Processor i7-7700k (4.2 GHz) 8 MB Cache.

D Histograms of the Models' Parameters

In Fig. 3 we represent the histogram of the values of the parameters, w, of the LR classifier on MNIST, FMNIST and ImageNet, when the training sets are clean (blue bars) and when a 16.6% of each training set is replaced by poisoning points (orange bars). To appreciate better the distribution of the parameters, we omit the upper part of some plots. The first row depicts the cases when no regularisation is applied, and the second row shows the case where λ is learned using RMD. We can clearly appreciate the effect of the regularisation: For all the datasets, the range of values of the parameters (blue bars of the second row) is narrowed down, and when the attacker injects poisoning points, this forces the model to compress more these values (orange bars of the second row) close to 0, as the value of λ increases. This leads to a more stable model under possible malicious manipulations of the training data.