Towards Poisoning of Deep Learning Algorithms with Back-gradient Optimization

Luis Muñoz-González
Imperial College
London, UK
l.munoz@imperial.ac.uk

Battista Biggio
Pluribus One
Dept. of Electrical and Electronic
Eng., University of Cagliari, Italy
battista.biggio@diee.unica.it

Ambra Demontis
Dept. of Electrical and Electronic
Eng., University of Cagliari, Italy
ambra.demontis@diee.unica.it

Andrea Paudice
Imperial College London, UK
a.paudice15@imperial.ac.uk

Vasin Wongrassamee
Imperial College London, UK
vasin.wongrassamee14@imperial.ac.uk

Emil C. Lupu
Imperial College London, UK
e.c.lupu@imperial.ac.uk

Fabio Roli
Pluribus One
Dept. of Electrical and Electronic
Eng., University of Cagliari, Italy
roli@diee.unica.it

ABSTRACT

A number of online services nowadays rely upon machine learning to extract valuable information from data collected in the wild. This exposes learning algorithms to the threat of data poisoning, i.e., a coordinate attack in which a fraction of the training data is controlled by the attacker and manipulated to subvert the learning process. To date, these attacks have been devised only against a limited class of binary learning algorithms, due to the inherent complexity of the gradient-based procedure used to optimize the poisoning points (a.k.a. adversarial training examples). In this work, we first extend the definition of poisoning attacks to multiclass problems. We then propose a novel poisoning algorithm based on the idea of back-gradient optimization, i.e., to compute the gradient of interest through automatic differentiation, while also reversing the learning procedure to drastically reduce the attack complexity. Compared to current poisoning strategies, our approach is able to target a wider class of learning algorithms, trained with gradient-based procedures, including neural networks and deep learning architectures. We empirically evaluate its effectiveness on several application examples, including spam filtering, malware detection, and handwritten digit recognition. We finally show that, similarly to adversarial test examples, adversarial training examples can also be transferred across different learning algorithms.

AISec'17, November 3, 2017, Dallas, TX, USA

ACM ISBN 978-1-4503-5202-4/17/11…$15.00
https://doi.org/10.1145/3128572.3140451

KEYWORDS

Adversarial Machine Learning; Training Data Poisoning; Adversarial Examples; Deep Learning.

1 INTRODUCTION

In recent years technology has become pervasive, enabling a rapid a disruptive change in the way society is organized. Our data is provided to third-party services which are supposed to facilitate and protect our daily work and activities. Most of these services leverage machine learning to extract valuable information from the overwhelming amount of input data received. Although this provides advantages to the users themselves, e.g., in terms of usability and functionality of such services, it is also clear that these services may be abused, providing great opportunities for cybercriminals to conduct novel, illicit, and highly-profitable activities. Being one of the main components behind such services makes machine learning an appealing target for attackers, who may gain a significant advantage by gaming the learning algorithm. Notably, machine learning itself can be the *weakest link* in the security chain, as its vulnerabilities can be exploited by the attacker to compromise the whole system infrastructure. To this end, she may inject malicious data to poison the learning process, or manipulate data at test time to evade detection.[1] These kinds of attack have been reported against anti-virus engines, anti-spam filters, and systems aimed to detect fake profiles or news in social networks – all problems involving a well-crafted deployment of machine learning algorithms [8, 16, 17, 19, 23, 25, 32, 34, 37–39]. Such attacks have fuelled a growing interest in the research area of *adversarial machine learning*, at the intersection of cybersecurity and machine learning. This recent research field aims at understanding the security properties of current learning algorithms, as well as at developing more secure ones [7, 16, 17].

[1] We refer to the attacker here as feminine due to the common interpretation as "Eve" or "Carol" in cryptography and security.

Among the different attack scenarios envisaged against machine learning, *poisoning attacks* are considered one of the most relevant and emerging security threats for data-driven technologies, i.e., technologies relying upon the collection of large amounts of data in the wild [17]. In a poisoning attack, the attacker is assumed to control a fraction of the training data used by the learning algorithm, with the goal of subverting the entire learning process, or facilitate subsequent system evasion [8, 19, 23, 25, 32, 39]. More practically, data poisoning is already a relevant threat in different application domains. For instance, some online services directly exploit users' feedback on their decisions to update the trained model. PDFRate[2] is an online malware detection tool that analyzes the submitted PDF files to reveal the presence of embedded malware [34]. After classification, it allows the user to provide feedback on its decision, i.e., to confirm or not the classification result. A malicious user may thus provide wrong feedback to gradually poison the system and compromise its performance over time. Notably, there is a more general underlying problem related to the collection of large data volumes with reliable labels. This is a well-known problem in malware detection, where malware samples are collected by means of compromised machines with known vulnerabilities (i.e., honeypots), or via other online services, like VirusTotal,[3] in which labelling errors are often reported.

Previous work has developed poisoning attacks against popular learning algorithms like Support Vector Machines (SVMs), LASSO, logistic and ridge regression, in different applications, like spam and malware detection [8, 19, 20, 23, 25, 32, 39]. The main technical difficulty in devising a poisoning attack is the computation of the poisoning samples, also recently referred to as *adversarial training examples* [20]. This requires solving a bilevel optimization problem in which the outer optimization amounts to maximizing the classification error on an untainted validation set, while the inner optimization corresponds to training the learning algorithm on the poisoned data [23]. Since solving this problem with black-box optimization is too computationally demanding, previous work has exploited gradient-based optimization, along with the idea of *implicit differentiation*. The latter consists of replacing the inner optimization problem with its stationarity (Karush-Kuhn-Tucker, KKT) conditions to derive an implicit equation for the gradient [8, 20, 23, 39]. This approach however can only be used against a limited class of learning algorithms, excluding neural networks and deep learning architectures, due to the inherent complexity of the procedure used to compute the required gradient. Another limitation is that, to date, previous work has only considered poisoning of two-class learning algorithms.

In this work, we overcome these limitations by first extending the threat model proposed in [1, 2, 7, 16] to account for multiclass poisoning attacks (Sect. 2). We then exploit a recent technique called *back-gradient optimization*, originally proposed for hyper-parameter optimization [3, 14, 22, 31], to implement a much more computationally-efficient poisoning attack. The underlying idea is to compute the gradient of interest through reverse-mode (automatic) differentiation (i.e., *back-propagation*), while reversing the underlying learning procedure to trace back the entire sequence of parameter updates performed during learning, without storing it. In fact, storing this sequence in memory would be infeasible for learning algorithms that optimize a large set of parameters across several iterations. Our poisoning algorithm only requires the learning algorithm to update its parameters during training in a *smooth* manner (e.g., through gradient descent), to correctly trace these changes backwards. Accordingly, compared to previously-proposed poisoning strategies, our approach is the first capable of targeting a wider class of learning algorithms, trainable with gradient-based procedures, like neural networks and deep learning architectures (Sect. 3).

Another important contribution of this work is to show how the performance of learning algorithms may be drastically compromised even by the presence of a small fraction of poisoning points in the training data, in the context of real-world applications like spam filtering, malware detection, and handwritten digit recognition (Sect. 4). We also investigate the *transferability* property of poisoning attacks, i.e., the extent to which attacks devised against a specific learning algorithm are effective against different ones. To our knowledge, this property has been investigated for evasion attacks (a.k.a. adversarial test examples), i.e., attacks aimed to evade a trained classifier at test time [6, 24, 27, 37], but never for poisoning attacks. We conclude our work by discussing related work (Sect. 5), the main limitations of our approach, and future research directions (Sect. 6).

2 THREAT MODEL

In this section, we summarize the framework originally proposed in [1, 2, 16] and subsequently extended in [7], which enables one to envision different attack scenarios against learning algorithms (including deep learning ones), and to craft the corresponding attack samples. Remarkably, these include attacks at training and at test time, usually referred to as poisoning and evasion attacks [6–8, 16, 23, 39] or, more recently, as adversarial (training and test) examples (when crafted against deep learning algorithms) [27, 28, 36].

The framework characterizes the attacker according to her goal, knowledge of the targeted system, and capability of manipulating the input data. Based on these assumptions, it allows one to define an optimal attack strategy as an optimization problem whose solution amounts to the construction of the attack samples, i.e., of the *adversarial examples*.

In this work, we extend this framework, originally developed for binary classification problems, to multiclass classification. While this generalization holds for evasion attacks too, we only detail here the main poisoning attack scenarios.

Notation. In a classification task, given the instance space $\mathcal{X}$ and the label space $\mathcal{Y}$, the learner aims to estimate the underlying (possibly noisy) latent function f that maps $\mathcal{X} \mapsto \mathcal{Y}$. Given a training set $\mathcal{D}_{\text{tr}} = \{\boldsymbol{x}_i, y_i\}_{i=1}^n$ with n i.i.d. samples drawn from the underlying probability distribution $p(\mathcal{X}, \mathcal{Y})$,[4] we can estimate f with a parametric or non-parametric model $\mathcal{M}$ trained by minimizing an objective function $\mathcal{L}(\mathcal{D}, \boldsymbol{w})$ (normally, a tractable estimate of

[2] http://pdfrate.com
[3] https://virustotal.com

[4] While normally the set notation $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$ does not admit duplicate entries, we admit our data sets to contain potentially duplicated points.

the generalization error), with respect to its parameters and/or hyperparameters $\boldsymbol{w}$.[5]

Thus, while $\mathcal{L}$ denotes the learner's objective function (possibly including regularization), we use $L(\mathcal{D}, \boldsymbol{w})$ to denote only the *loss* incurred when evaluating the learner parameterized by $\boldsymbol{w}$ on the samples in $\mathcal{D}$.

2.1 Attacker's Goal

The goal of the attack is determined in terms of the desired **security violation** and **attack specificity**. In multiclass classification, misclassifying a sample does not have a unique meaning, as there is more than one class different from the correct one. Accordingly, we extend the current framework by introducing the concept of **error specificity**. These three characteristics are detailed below.

Security Violation. This characteristic defines the high-level security violation caused by the attack, as normally done in security engineering. It can be: an *integrity* violation, if malicious activities evade detection without compromising normal system operation; an *availability* violation, if normal system functionality is compromised, e.g., by increasing the classification error; or a *privacy* violation, if the attacker obtains private information about the system, its users or data by reverse-engineering the learning algorithm.

Attack Specificity. This characteristic ranges from *targeted* to *indiscriminate*, respectively, if the attack aims to cause misclassification of a specific set of samples (to target a given system user or protected service), or of any sample (to target any system user or protected service).

Error Specificity. We introduce here this characteristic to disambiguate the notion of misclassification in multiclass problems. The error specificity can thus be: *specific*, if the attacker aims to have a sample misclassified as a specific class; or *generic*, if the attacker aims to have a sample misclassified as any of the classes different from the true class.[6]

2.2 Attacker's Knowledge

The attacker can have different levels of knowledge of the targeted system, including: ($k.i$) the training data $\mathcal{D}_{\text{tr}}$; ($k.ii$) the feature set $\mathcal{X}$; ($k.iii$) the learning algorithm $\mathcal{M}$, along with the objective function $\mathcal{L}$ minimized during training; and, possibly, ($k.iv$) its (trained) parameters $\boldsymbol{w}$. The attacker's knowledge can thus be characterized in terms of a space Θ that encodes the aforementioned assumptions ($k.i$)-($k.iv$) as $\boldsymbol{\theta} = (\mathcal{D}, \mathcal{X}, \mathcal{M}, \boldsymbol{w})$. Depending on the assumptions made on each of these components, one can envisage different attack scenarios. Typically, two main settings are considered, referred to as attacks with *perfect* and *limited* knowledge.

Perfect-Knowledge (PK) Attacks. In this case, the attacker is assumed to know everything about the targeted system. Although this setting may be not always representative of practical cases, it enables us to perform a worst-case evaluation of the security of learning algorithms under attack, highlighting the upper bounds on the performance degradation that may be incurred by the system under attack. In this case, we have $\boldsymbol{\theta}_{\text{PK}} = (\mathcal{D}, \mathcal{X}, \mathcal{M}, \boldsymbol{w})$.

Limited-Knowledge (LK) Attacks. Although LK attacks admit a wide range of possibilities, the attacker is typically assumed to know the feature representation $\mathcal{X}$ and the learning algorithm $\mathcal{M}$, but not the training data (for which surrogate data from similar sources can be collected). We refer to this case here as LK attacks with Surrogate Data (LK-SD), and denote it with $\boldsymbol{\theta}_{\text{LK–SD}} = (\hat{\mathcal{D}}, \mathcal{X}, \mathcal{M}, \hat{\boldsymbol{w}})$ (using the *hat* symbol to denote limited knowledge of a given component). Notably, in this case, as the attacker is only given a surrogate data set $\hat{\mathcal{D}}$, also the learner's parameters have to be estimated by the attacker, e.g., by optimizing $\mathcal{L}$ on $\hat{\mathcal{D}}$.

Similarly, we refer to the case in which the attacker knows the training data (e.g., if the learning algorithm is trained on publicly-available data), but not the learning algorithm (for which a surrogate learner can be trained on the available data) as LK attacks with Surrogate Learners (LK-SL). This scenario can be denoted with $\boldsymbol{\theta}_{\text{LK–SL}} = (\mathcal{D}, \mathcal{X}, \hat{\mathcal{M}}, \hat{\boldsymbol{w}})$, even though the parameter vector $\hat{\boldsymbol{w}}$ may belong to a different vector space than that of the targeted learner. Note that LK-SL attacks also include the case in which the attacker knows the learning algorithm, but she is not able to derive an optimal attack strategy against it (e.g., if the corresponding optimization problem is not tractable or difficult to solve), and thus uses a surrogate learning model to this end. Experiments on the *transferability* of attacks among learning algorithms, firstly demonstrated in [6] and then in subsequent work on deep learners [27], fall under this category of attacks.

2.3 Attacker's Capability

This characteristic is defined based on the **influence** that the attacker has on the input data, and on the presence of **data manipulation constraints**.

Attack Influence. In supervised learning, the attack influence can be causative, if the attacker can influence both training and test data, or exploratory, if the attacker can only manipulate test data. These settings are more commonly referred to as *poisoning* and *evasion* attacks [2, 6–8, 16, 23, 39].

Data Manipulation Constraints. Another aspect related to the attacker's capability is the presence of constraints on the manipulation of input data, which is however strongly dependent on the given practical scenario. For example, if the attacker aims to evade a malware classification system, she should manipulate the exploitation code embedded in the malware sample without compromising its intrusive functionality. In the case of poisoning, the labels assigned to the training samples are not typically under the control of the attacker. She should thus consider additional constraints while manipulating the poisoning samples to have them labelled as desired; for instance, a maximum amount of perturbation on the input data. This may also be important to craft poisoning samples which are more difficult to detect with data pre-filtering or outlier detection techniques, although their impact may be also reduced. Typically, these constraints can be nevertheless accounted for in the definition of the optimal attack strategy. In particular, we characterize them by assuming that an initial set of attack samples

[5] For instance, for kernelized SVMs, $\boldsymbol{w}$ may include the dual variables $\boldsymbol{\alpha}$, the bias b, and even the regularization parameter C. In this work, as in [8, 23, 39], we however consider only the optimization of the model parameters, and not of its hyperparameters.

[6] In [28], the authors defined *targeted* and *indiscriminate* attacks (at test time) depending on whether the attacker aims to cause *specific* or *generic* errors. Here we do not follow their naming convention, as it can cause confusion with the interpretation of *targeted* and *indiscriminate* attacks introduced in previous work [1, 2, 4, 7, 9, 10, 16, 39].

$\mathcal{D}_c$ is given, and that it is modified according to a space of possible modifications $\Phi(\mathcal{D}_c)$ (e.g., constraining the norm of the input perturbation on each poisoning sample).

2.4 Attack Strategy

Given the attacker's knowledge $\boldsymbol{\theta} \in \Theta$ and a set of manipulated attack samples $\mathcal{D}'_c \in \Phi(\mathcal{D}_c)$, the attacker's goal can be characterized in terms of an objective function $\mathcal{A}(\mathcal{D}'_c, \boldsymbol{\theta}) \in \mathbb{R}$ which evaluates how effective the attacks $\mathcal{D}'_c$ are. The optimal attack strategy can be thus given as:

$$\mathcal{D}_c^\star \in \underset{\mathcal{D}'_c \in \Phi(\mathcal{D}_c)}{\arg\max}\ \mathcal{A}(\mathcal{D}'_c, \boldsymbol{\theta}) \tag{1}$$

While this high-level formulation encompasses both evasion and poisoning attacks, in both binary and multiclass problems, in the remainder of this work we only focus on the definition of some poisoning attack scenarios.

2.5 Poisoning Attack Scenarios

We focus here on two poisoning attack scenarios of interest for multiclass problems, noting that other attack scenarios can be derived in a similar manner.

Error-Generic Poisoning Attacks. The most common scenario considered in previous work [8, 23, 39] considers poisoning two-class learning algorithms to cause a *denial of service*. This is an availability attack, and it could be targeted or indiscriminate, depending on whether it affects a specific system user or service, or any of them. In the multiclass case, it is thus natural to extend this scenario assuming that the attacker is not aiming to cause specific errors, but only *generic* misclassifications. As in [8, 23, 39], this poisoning attack (as any other poisoning attack) requires solving a bilevel optimization, where the inner problem is the learning problem. This can be made explicit by rewriting Eq. (1) as:

$$\mathcal{D}_c^\star \in \underset{\mathcal{D}'_c \in \Phi(\mathcal{D}_c)}{\arg\max} \quad \mathcal{A}(\mathcal{D}'_c, \boldsymbol{\theta}) = L(\hat{\mathcal{D}}_{\text{val}}, \hat{\boldsymbol{w}})\,, \tag{2}$$

$$\text{s.t.} \quad \hat{\boldsymbol{w}} \in \underset{\boldsymbol{w}' \in \mathcal{W}}{\arg\min}\ \mathcal{L}(\hat{\mathcal{D}}_{\text{tr}} \cup \mathcal{D}'_c, \boldsymbol{w}')\,, \tag{3}$$

where the surrogate data $\hat{\mathcal{D}}$ available to the attacker is divided into two disjoint sets $\hat{\mathcal{D}}_{\text{tr}}$ and $\hat{\mathcal{D}}_{\text{val}}$. The former, along with the poisoning points $\mathcal{D}'_c$ is used to learn the surrogate model, while the latter is used to evaluate the impact of the poisoning samples on untainted data, through the function $\mathcal{A}(\mathcal{D}'_c, \boldsymbol{\theta})$. In this case, the function $\mathcal{A}(\mathcal{D}'_c, \boldsymbol{\theta})$ is simply defined in terms of a loss function $L(\hat{\mathcal{D}}_{\text{val}}, \hat{\boldsymbol{w}})$ that evaluates the performance of the (poisoned) surrogate model on $\hat{\mathcal{D}}_{\text{val}}$. The dependency of $\mathcal{A}$ on $\mathcal{D}'_c$ is thus indirectly encoded through the parameters $\hat{\boldsymbol{w}}$ of the (poisoned) surrogate model.[7] Note that, since the learning algorithm (even if convex) may not exhibit a unique solution in the feasible set $\mathcal{W}$, the outer problem has to be evaluated using the exact solution $\hat{\boldsymbol{w}}$ found by the inner optimization. Worth remarking, this formulation encompasses all previously-proposed poisoning attacks against binary learners [8, 23, 39], provided that the loss function L is selected accordingly (e.g., using the hinge loss against SVMs [8]). In the multiclass case, one can use a multiclass loss function, like the log-loss with softmax activation, as done in our experiments.

Error-Specific Poisoning Attacks. Here, we assume that the attacker's goal is to cause specific misclassifications – a plausible scenario only for multiclass problems. This attack can cause an integrity or an availability violation, and it can also be targeted or indiscriminate, depending on the desired misclassifications. The poisoning problem remains that given by Eqs. (2)-(3), though the objective is defined as:

$$\mathcal{A}(\mathcal{D}'_c, \boldsymbol{\theta}) = -L(\hat{\mathcal{D}}'_{\text{val}}, \hat{\boldsymbol{w}})\,, \tag{4}$$

where $\hat{\mathcal{D}}'_{\text{val}}$ is a set that contains the same data as $\hat{\mathcal{D}}_{\text{val}}$, though with different labels, chosen by the attacker. These labels correspond to the desired misclassifications, and this is why there is a minus sign in front of L, i.e., the attacker effectively aims at *minimizing* the loss on her desired set of labels. Note that, to implement an integrity violation or a targeted attack, some of these labels may actually be the same as the true labels (such that normal system operation is not compromised, or only specific system users are affected).

3 POISONING ATTACKS WITH BACK-GRADIENT OPTIMIZATION

In this section, we first discuss how the bilevel optimization given by Eqs. (2)-(3) has been solved in previous work to develop gradient-based poisoning attacks [8, 20, 23, 39]. As we will see, these attacks can only be used against a limited class of learning algorithms, excluding neural networks and deep learning architectures, due to the inherent complexity of the procedure used to compute the required gradient. To overcome this limitation, we exploit a recent technique called *back-gradient optimization* [14, 22], which allows computing the gradient of interest in a more computationally-efficient and stabler manner. Notably, this enables us to devise the first poisoning attack able to target neural networks and deep learning architectures (without using any surrogate model).

Before delving into the technical details, we make the same assumptions made in previous work [8, 23, 39] to reduce the complexity of Problem (2)-(3): (*i*) we consider the optimization of one poisoning point at a time, denoted hereafter with $\boldsymbol{x}_c$; and (*ii*) we assume that its label y_c is initially chosen by the attacker, and kept fixed during the optimization. The poisoning problem can be thus simplified as:

$$\boldsymbol{x}_c^\star \in \underset{\boldsymbol{x}'_c \in \Phi(\{\boldsymbol{x}_c, y_c\})}{\arg\max} \quad \mathcal{A}(\{\boldsymbol{x}'_c, y_c\}, \boldsymbol{\theta}) = L(\hat{\mathcal{D}}_{\text{val}}, \hat{\boldsymbol{w}})\,, \tag{5}$$

$$\text{s.t.} \quad \hat{\boldsymbol{w}} \in \underset{\boldsymbol{w}' \in \mathcal{W}}{\arg\min}\ \mathcal{L}(\boldsymbol{x}'_c, \boldsymbol{w}')\,. \tag{6}$$

The function Φ imposes constraints on the manipulation of $\boldsymbol{x}_c$, e.g., upper and lower bounds on its manipulated values. These may also depend on y_c, e.g., to ensure that the poisoning sample is labelled as desired when updating the targeted classifier. Note also that, for notational simplicity, we only report $\boldsymbol{x}'_c$ as the first argument of $\mathcal{L}$ instead of $\hat{\mathcal{D}}_{\text{tr}} \cup \{\boldsymbol{x}'_c, y_c\}$.

Gradient-based Poisoning Attacks. We discuss here how Problem (5)-(6) has been solved in previous work [8, 20, 23, 39]. For some classes of loss functions L and learning objective functions $\mathcal{L}$, this problem can be indeed solved through *gradient ascent*. In

[7] Note that $\mathcal{A}$ can also be directly dependent on $\mathcal{D}'_c$, as in the case of nonparametric models; e.g., in kernelized SVMs, when the poisoning points are support vectors [8].

Algorithm 1 Poisoning Attack Algorithm

Input: $\hat{\mathcal{D}}_{\text{tr}}$, $\hat{\mathcal{D}}_{\text{val}}$, $\mathcal{L}$, L, the initial poisoning point $\boldsymbol{x}_c^{(0)}$, its label y_c, the learning rate η, a small positive constant ε.

1: $i \leftarrow 0$ (iteration counter)
2: **repeat**
3: $\hat{\boldsymbol{w}} \in \arg\min_{\boldsymbol{w}'} \mathcal{L}(\boldsymbol{x}_c^{(i)}, \boldsymbol{w}')$ (train learning algorithm)
4: $\boldsymbol{x}_c^{(i+1)} \leftarrow \Pi_\Phi\left(\boldsymbol{x}_c{}^{(i)} + \eta \nabla_{\boldsymbol{x}_c} \mathcal{A}(\{\boldsymbol{x}_c^{(i)}, y_c\})\right)$
5: $i \leftarrow i + 1$
6: **until** $\mathcal{A}(\{\boldsymbol{x}_c^{(i)}, y_c\}) - \mathcal{A}(\{\boldsymbol{x}_c^{(i-1)}, y_c\}) < \varepsilon$

Output: the final poisoning point $\boldsymbol{x}_c \leftarrow \boldsymbol{x}_c^{(i)}$

particular, provided that the loss function L is differentiable w.r.t. $\boldsymbol{w}$ and $\boldsymbol{x}_c$, we can compute the gradient $\nabla_{\boldsymbol{x}_c}\mathcal{A}$ using the chain rule:

$$\nabla_{\boldsymbol{x}_c}\mathcal{A} = \nabla_{\boldsymbol{x}_c} L + \frac{\partial \hat{\boldsymbol{w}}}{\partial \boldsymbol{x}_c}^{\top} \nabla_{\boldsymbol{w}} L\,, \tag{7}$$

where $L(\hat{\mathcal{D}}_{\text{val}}, \hat{\boldsymbol{w}})$ is evaluated on the parameters $\hat{\boldsymbol{w}}$ learned after training (including the poisoning point). The main difficulty here is computing $\frac{\partial \hat{\boldsymbol{w}}}{\partial \boldsymbol{x}_c}$, i.e., understanding how the solution of the learning algorithm varies w.r.t. the poisoning point. Under some regularity conditions, this can be done by replacing the inner learning problem with its stationarity (KKT) conditions. For example, this holds if the learning problem $\mathcal{L}$ is convex, which implies that all stationary points are global minima [31]. In fact, poisoning attacks have been developed so far only against learning algorithms with convex objectives [8, 20, 23, 39]. The trick here is to replace the inner optimization with the implicit function $\nabla_{\boldsymbol{w}}\mathcal{L}(\mathcal{D}_{\text{tr}} \cup \{\boldsymbol{x}_c, y_c\}, \hat{\boldsymbol{w}}) = \boldsymbol{0}$, corresponding to its KKT conditions. Then, assuming that it is differentiable w.r.t. $\boldsymbol{x}_c$, one yields the linear system $\nabla_{\boldsymbol{x}_c}\nabla_{\boldsymbol{w}}\mathcal{L} + \frac{\partial \hat{\boldsymbol{w}}}{\partial \boldsymbol{x}_c}^{\top} \nabla^2_{\boldsymbol{w}}\mathcal{L} = \boldsymbol{0}$. If $\nabla^2_{\boldsymbol{w}}\mathcal{L}$ is not singular, we can solve this system w.r.t. $\frac{\partial \hat{\boldsymbol{w}}}{\partial \boldsymbol{x}_c}$, and substitute its expression in Eq. (7), yielding:

$$\nabla_{\boldsymbol{x}_c}\mathcal{A} = \nabla_{\boldsymbol{x}_c} L - (\nabla_{\boldsymbol{x}_c}\nabla_{\boldsymbol{w}}\mathcal{L})(\nabla^2_{\boldsymbol{w}}\mathcal{L})^{-1}\nabla_{\boldsymbol{w}} L\,. \tag{8}$$

This gradient is then iteratively used to update the poisoning point through gradient ascent, as shown in Algorithm 1.[8] Recall that the projection operator Π_Φ is used to map the current poisoning point onto the feasible set Φ (cf. Eqs. 5-6).

This is the state-of-the-art approach used to implement current poisoning attacks [8, 20, 23, 39]. The problem here is that computing and inverting $\nabla^2_{\boldsymbol{w}}\mathcal{L}$ scales in time as $O(p^3)$ and in memory as $O(p^2)$, being p the cardinality of $\boldsymbol{w}$. Moreover, Eq. (8) requires solving one linear system per parameter. These aspects make it prohibitive to assess the effectiveness of poisoning attacks in a variety of practical settings.

To mitigate these issues, as suggested in [13, 14, 20, 22], one can apply conjugate gradient descent to solve a simpler linear system, obtained by a trivial re-organization of the terms in the second part of Eq. (8). In particular, one can set $(\nabla^2_{\boldsymbol{w}}\mathcal{L})\,\mathbf{v} = \nabla_{\boldsymbol{w}} L$, and compute $\nabla_{\mathbf{x}_c}\mathcal{A} = \nabla_{\boldsymbol{x}_c} L - \nabla_{\mathbf{x}_c}\nabla_{\boldsymbol{w}}\mathcal{L}\,\mathbf{v}$. The computation of the

[8] Note that Algorithm 1 can be exploited to optimize multiple poisoning points too. As in [39], the idea is to perform several passes over the set of poisoning samples, using Algorithm 1 to optimize each poisoning point at a time, while keeping the other points fixed. Line searches can also be exploited to reduce complexity.

Algorithm 2 Gradient Descent

Input: initial parameters $\boldsymbol{w}_0$, learning rate η, $\hat{\mathcal{D}}_{\text{tr}}$, $\mathcal{L}$.

1: **for** $t = 0, \ldots, T-1$ **do**
2: $\mathbf{g}_t = \nabla_{\boldsymbol{w}}\mathcal{L}(\hat{\mathcal{D}}_{\text{tr}}, \boldsymbol{w}_t)$
3: $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \eta\, \mathbf{g}_t$
4: **end for**

Output: trained parameters $\boldsymbol{w}_T$

Algorithm 3 Back-gradient Descent

Input: trained parameters $\boldsymbol{w}_T$, learning rate η, $\hat{\mathcal{D}}_{\text{tr}}$, $\hat{\mathcal{D}}_{\text{val}}$, poisoning point $\boldsymbol{x}'_c$, y_c, loss function L, learner's objective $\mathcal{L}$.
initialize $d\boldsymbol{x}_c \leftarrow \boldsymbol{0}$, $d\boldsymbol{w} \leftarrow \nabla_{\boldsymbol{w}} L(\hat{\mathcal{D}}_{\text{val}}, \boldsymbol{w}_T)$

1: **for** $t = T, \ldots, 1$ **do**
2: $d\boldsymbol{x}_c \leftarrow d\boldsymbol{x}'_c - \eta\, d\boldsymbol{w}\nabla_{\boldsymbol{x}_c}\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{x}'_c, \boldsymbol{w}_t)$
3: $d\boldsymbol{w} \leftarrow d\boldsymbol{w} - \eta\, d\boldsymbol{w}\nabla_{\boldsymbol{w}}\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{x}'_c, \boldsymbol{w}_t)$
4: $\mathbf{g}_{t-1} = \nabla_{\boldsymbol{w}_t}\mathcal{L}(\boldsymbol{x}'_c, \boldsymbol{w}_t)$
5: $\boldsymbol{w}_{t-1} = \boldsymbol{w}_t + \alpha \mathbf{g}_{t-1}$
6: **end for**

Output: $\nabla_{\boldsymbol{x}_c}\mathcal{A} = \nabla_{\boldsymbol{x}_c} L + d\boldsymbol{x}_c$

matrices $\nabla_{\boldsymbol{x}_c}\nabla_{\boldsymbol{w}}\mathcal{L}$ and $\nabla^2_{\boldsymbol{w}}\mathcal{L}$ can also be avoided using Hessian-vector products [30]:

$$(\nabla_{\boldsymbol{x}_c}\nabla_{\boldsymbol{w}}\mathcal{L})\,\mathbf{z} = \lim_{h\to 0}\frac{1}{h}\left(\nabla_{\boldsymbol{x}_c}\mathcal{L}\left(\boldsymbol{x}'_c, \hat{\boldsymbol{w}} + h\mathbf{z}\right) - \nabla_{\boldsymbol{x}_c}\mathcal{L}\left(\boldsymbol{x}'_c, \hat{\boldsymbol{w}}\right)\right)\,,$$

$$(\nabla_{\boldsymbol{w}}\nabla_{\boldsymbol{w}}\mathcal{L})\,\mathbf{z} = \lim_{h\to 0}\frac{1}{h}\left(\nabla_{\boldsymbol{w}}\mathcal{L}\left(\boldsymbol{x}'_c, \hat{\boldsymbol{w}} + h\mathbf{z}\right) - \nabla_{\boldsymbol{w}}\mathcal{L}\left(\boldsymbol{x}'_c, \hat{\boldsymbol{w}}\right)\right)\,.$$

Although this approach allows poisoning learning algorithms more efficiently w.r.t. previous work [8, 23, 39], it still requires the inner learning problem to be solved exactly. From a practical perspective, this means that the KKT conditions have to be met with satisfying numerical accuracy. However, as these problems are always solved to a finite accuracy, it may happen that the gradient $\nabla_{\boldsymbol{x}_c}\mathcal{A}$ is not sufficiently precise, especially if convergence thresholds are too loose [14, 22].

It is thus clear that such an approach can not be used, in practice, to poison learning algorithms like neural networks and deep learning architectures, as it may not only be difficult to derive proper stationarity conditions involving all parameters, but also as it may be too computationally demanding to train such learning algorithms with sufficient precision to correctly compute the gradient $\nabla_{\boldsymbol{x}_c}\mathcal{A}$.

Poisoning with Back-gradient Optimization. In this work, we overcome this limitation by exploiting *back-gradient optimization* [14, 22]. This technique has been first exploited in the context of energy-based models and hyperparameter optimization, to solve bilevel optimization problems similar to the poisoning problem discussed before. The underlying idea of this approach is to replace the inner optimization with a set of iterations performed by the learning algorithm to update the parameters $\boldsymbol{w}$, provided that such updates are *smooth*, as in the case of gradient-based learning algorithms. According to [14], this technique allows to compute the desired gradients in the outer problem using the parameters $\boldsymbol{w}_T$ obtained from an incomplete optimization of the inner problem (after T iterations).

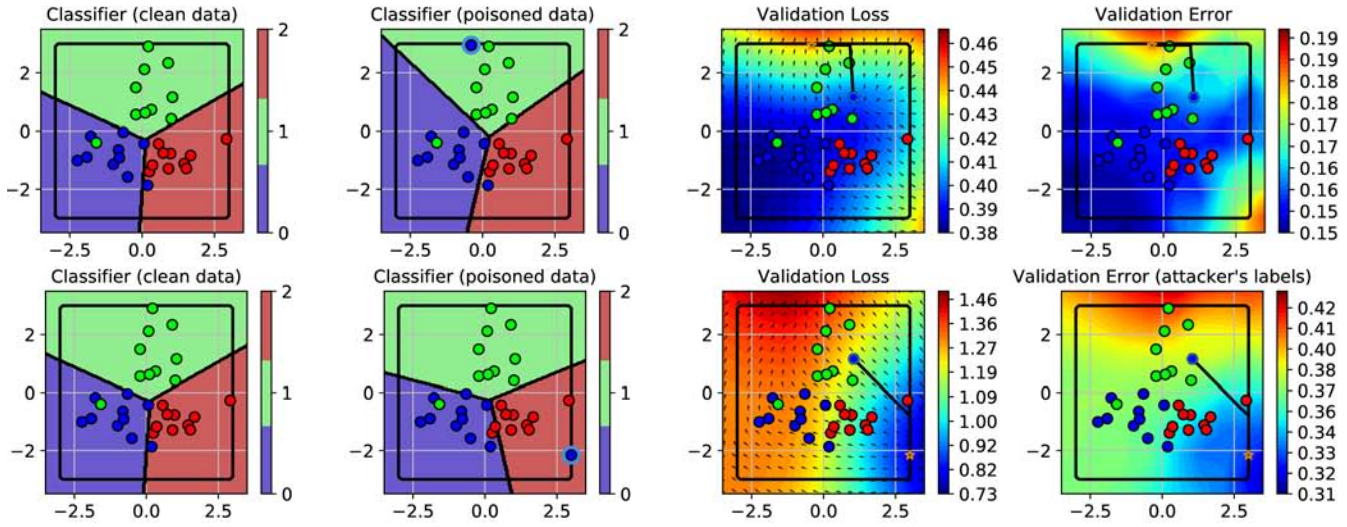

Figure 1: Error-generic (top row) and error-specific (bottom row) poisoning attacks on a three-class synthetic dataset, against a multiclass logistic classifier. In the error-specific case, the attacker aims to have red points misclassified as blue, while preserving the labels of the other points. We report the decision regions on the clean (first column) and on the poisoned (second column) data, in which we only add a poisoning point labelled as blue (highlighted with a blue circle). The validation loss $L(\hat{\mathcal{D}}_{\rm val}, \hat{\boldsymbol{w}})$ and $L(\hat{\mathcal{D}}'_{\rm val}, \hat{\boldsymbol{w}})$, respectively maximized in error-generic and minimized in error-specific attacks, is shown in colors, as a function of the attack point $\boldsymbol{x}_c$ (third column), along with the corresponding back-gradients (shown as arrows), and the path followed while optimizing $\boldsymbol{x}_c$. To show that the logistic loss used to estimate L provides a good approximation of the true error, we also report the validation error measured with the zero-one loss on the same data (fourth column).

This represent a significant computational improvement compared to traditional gradient-based approaches, since it only requires a reduced number of training iterations for the learning algorithm. This is especially important in large neural networks and deep learning algorithms, where the computational cost per iteration can be high. Then, assuming that the inner optimization runs for T iterations, the idea is to exploit reverse-mode differentiation, or *back-propagation*, to compute the gradient of the outer objective. However, using back-propagation in a naïve manner would not work for this class of problems, as it requires storing the whole set of parameter updates $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_T$ performed during training, along with the forward derivatives. These are indeed the elements required to compute the gradient of the outer objective with a *backward* pass (we refer the reader to [22] for more details). This process can be extremely memory-demanding if the learning algorithm runs for a large number of iterations T, and especially if the number of parameters $\boldsymbol{w}$ is large (as in deep networks). Therefore, to avoid storing the whole training trajectory $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_T$ and the required forward derivatives, Domke [14] and Maclaurin et al. [22] proposed to compute them directly during the backward pass, by *reversing* the steps followed by the learning algorithm to update them. Computing $\boldsymbol{w}_T, \ldots, \boldsymbol{w}_1$ in reverse order w.r.t. the forward step is clearly feasible only if the learning procedure can be exactly traced backwards. Nevertheless, this happens to be feasible for a large variety of gradient-based procedures, including gradient descent with fixed step size, and stochastic gradient descent with momentum.

In this work, we leverage back-gradient descent to compute $\nabla_{\boldsymbol{x}_c}\mathcal{A}$ (Algorithm 3) by reversing a standard gradient-descent procedure with fixed step size that runs for a truncated training of the learning algorithm to T iterations (Algorithm 2). Notably, lines 2-3 in Algorithm 3 can be efficiently computed with Hessian-vector products, as discussed before. We exploit this algorithm to compute the gradient $\nabla_{\boldsymbol{x}_c}\mathcal{A}$ in line 4 of our poisoning attack algorithm (Algorithm 1). In this case, line 3 of Algorithm 1 is replaced with the incomplete optimization of the learning algorithm, truncated to T iterations. Note finally that, as in [14, 22], the time complexity of our back-gradient descent is $O(T)$. This drastically reduces the complexity of the computation of the outer gradient, making it feasible to evaluate the effectiveness of poisoning attacks also against large neural networks and deep learning algorithms. Moreover, this outer gradient can be accurately estimated from a truncated optimization of the inner problem with a reduced number of iterations. This allows for a tractable computation of the poisoning points in Algorithm 1, since training the learning algorithm at each iteration can be prohibitive, especially for deep networks.

We conclude this section by noting that, in the case of error-specific poisoning attacks (Sect. 2.5), the outer objective in Problem (5)-(6) is $-L(\hat{\mathcal{D}}'_{val}, \hat{\boldsymbol{w}})$. This can be regarded as a minimization problem, and it thus suffices to modify line 4 in Algorithm 1 to update the poisoning point along the opposite direction. We clarify this in Fig. 1, where we also discuss the different effect of error-generic and error-specific poisoning attacks in a multiclass setting.

4 EXPERIMENTAL ANALYSIS

In this section, we first evaluate the effectiveness of the back-gradient poisoning attacks described in Sect. 3 on spam and malware detection tasks. In these cases, we also assess whether poisoning samples can be *transferred* across different learning algorithms. We then investigate the impact of error-generic and error-specific poisoning attacks in the well-known multiclass problem of handwritten digit recognition. In this case, we also report the first proof-of-concept adversarial training examples computed by poisoning a convolutional neural network in an *end-to-end* manner (i.e., not just using a surrogate model trained on the deep features, as in [20]).

4.1 Spam and Malware Detection

We consider here two distinct datasets, respectively representing a spam email classification problem (`Spambase`) and a malware detection task (`Ransomware`). The `Spambase` data [11] consists of a collection of 4, 601 emails, including 1, 813 spam emails. Each email is encoded as a feature vector consisting of 54 binary features, each denoting the presence or absence of a given word in the email. The `Ransomware` data [33] consists of 530 ransomware samples and 549 benign applications. Ransomware is a very recent kind of malware which encrypts the data on the infected machine, and requires the victim to pay a ransom to obtain the decryption key. This dataset has 400 binary features accounting for different sets of actions, API invocations, and modifications in the file system and registry keys during the execution of the software.

We consider the following leaning algorithms: (*i*) Multi-Layer Perceptrons (MLPs) with one hidden layer consisting of 10 neurons; (*ii*) Logistic Regression (LR); and (*iii*) Adaline (ADA). For MLPs, we have used hyperbolic tangent activation functions for the neurons in the hidden layer, and softmax activations in the output layer. Moreover, for MLPs and LR, we use the cross-entropy (or log-loss) as the loss function, while we use the mean squared error for ADA.

We assume here that the attacker aims to cause a denial of service, and thus runs a poisoning *availability* attack whose goal is simply to maximize the classification error. Accordingly, we run Algorithm 1 injecting up to 20 poisoning points in the training data. We initialize the poisoning points by cloning training points and flipping their label. We set the number of iterations T for obtaining stable back-gradients to 200, 100, and 80, respectively for MLPs, LR and ADA. We further consider two distinct settings: PK attacks, in which the attacker is assumed to have full knowledge of the attacked system (for a worst-case performance assessment); and LK-SL attacks, in which she knows everything except for the learning algorithm, and thus uses a surrogate learner $\hat{\mathcal{M}}$. This scenario, as discussed in Sect. 2.2, is useful to assess the *transferability* property of the attack samples. To the best of our knowledge, this has been demonstrated in [6, 27] for evasion attacks (i.e., adversarial *test* examples) but never for poisoning attacks (i.e., adversarial *training* examples). To this end, we optimize the poisoning samples using alternatively MLPs, LR or ADA as the surrogate learner, and then evaluate the impact of the corresponding attacks against the other two algorithms.

The experimental results, shown in Figs. 2-3, are averaged on 10 independent random data splits. In each split, we use 100 samples for training and 400 for validation, i.e., to respectively construct

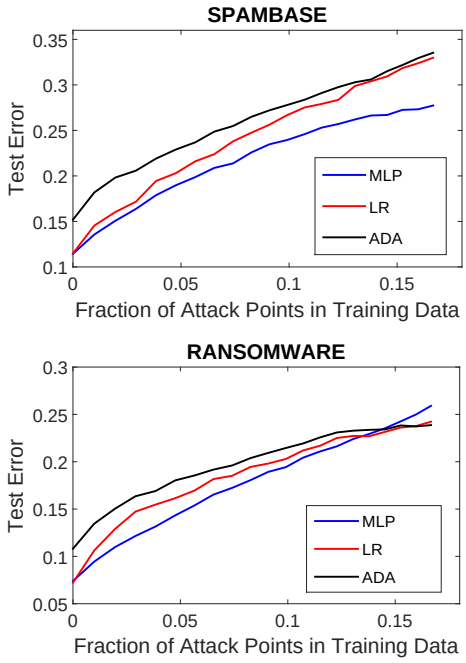

Figure 2: Results for PK poisoning attacks.

$\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{val}}$. Recall indeed that in both PK and LK-SL settings, the attacker has perfect knowledge of the training set used to learn the true (attacked) model, i.e., $\hat{\mathcal{D}}_{\text{tr}} = \mathcal{D}_{\text{tr}}$. The remaining samples are used for testing, i.e., to assess the classification error under poisoning.[9]

We can observe from Fig. 2 that PK poisoning attacks can significantly compromise the performance of all the considered classifiers. In particular, on `Spambase`, they cause the classification error of ADA and LR to increase up to 30% even if the attacker only controls 15% of the training data. Although the MLP is more resilient to poisoning than these linear classifiers, its classification error also increases significantly, up to 25%, which is not tolerable in several practical settings. The results for PK attacks on `Ransomware` are similar, although the MLP seems as vulnerable as ADA and LR in this case.

Transferability of Poisoning Samples. Regarding LK-SL poisoning attacks, we can observe from Fig. 3 that the attack points generated using a linear classifier (either ADA or LR) as the surrogate model have a very similar impact on the other linear classifier. In contrast, the poisoning points crafted with these linear algorithms have a lower impact against the MLP, although its performance is still noticeably affected. When the MLP is used as the surrogate model, instead, the performance degradation of the other algorithms is similar. However, the impact of these attacks is much lower. To summarize, our results show that the attack points can be effectively transferred across linear algorithms and also have a noticeable impact on (nonlinear) neural networks. In contrast,

[9] Note indeed that the validation error only provides a biased estimate of the true classification error, as it is used by the attacker to optimize the poisoning points [8].

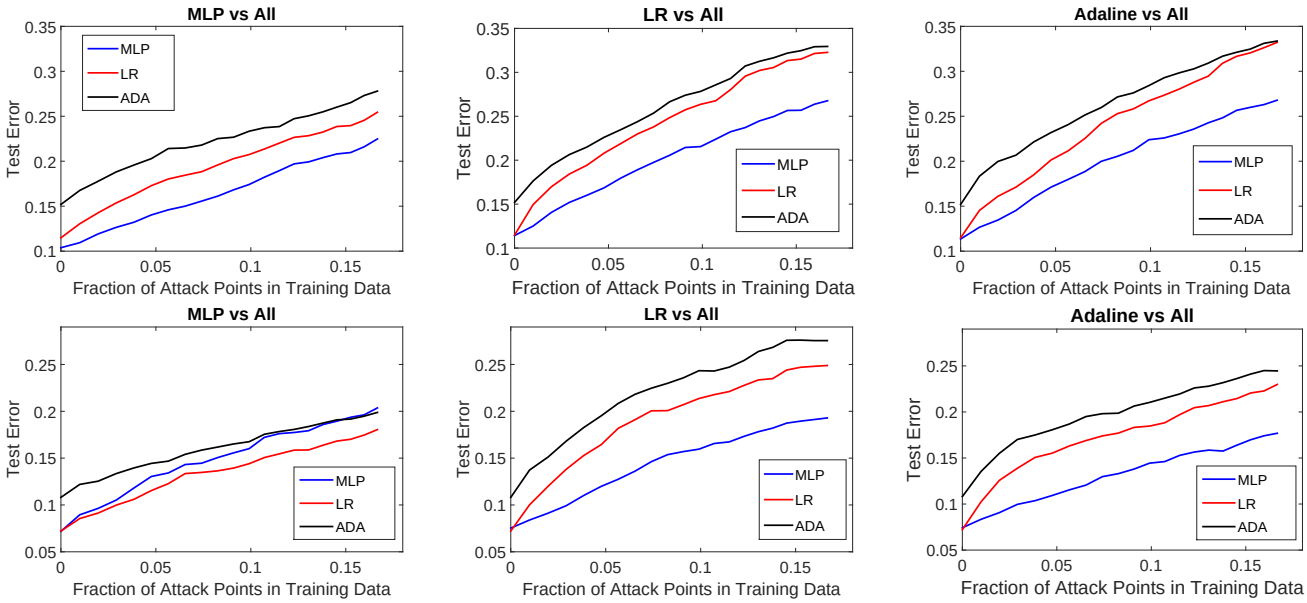

Figure 3: Results for LK-SL poisoning attacks (transferability of poisoning samples) on Spambase (top row) and Ransomware (bottom row).

transferring poisoning samples from nonlinear to linear models seems to be less effective.

4.2 Handwritten Digit Recognition

We consider here the problem of handwritten digit recognition, which involves 10 classes (each corresponding to a digit, from 0 to 9), using the MNIST data [21]. Each digit image consists of 28×28 = 784 pixels, ranging from 0 to 255 (images are in grayscale). We divide each pixel value by 255 and use it as a feature. We evaluate the effect of error-generic and error-specific poisoning strategies against a multiclass LR classifier using softmax activation and the log-loss as the loss function.

Error-generic attack. In this case, the attacker aims to maximize the classification error regardless of the resulting kinds of error, as described in Sect. 2.5. This is thus an *availability* attack, aimed to cause a denial of service. We generate 10 independent random splits using 1000 samples for training, 1000 for validation, and 8000 for testing. To compute the back-gradients $\nabla_{\boldsymbol{x}_c}\mathcal{A}$ required by our poisoning attack, we use $T = 60$ iterations. We initialize the poisoning points by cloning randomly-chosen training points and changing their label at random In addition, we compare our poisoning attack strategy here against a label-flip attack in which the attack points are drawn from the validation set and their labels are flipped at random. In both cases, we inject up to 60 attack points into the training set.

The results are shown in Fig. 4 (top row). Note first that our error-generic poisoning attack almost doubles the classification error in the absence of poisoning, with less than 6% of poisoning points. It is also much more effective than random label flips and, as expected, it causes a similar increase of the classification error over all classes (although some classes are easier to poison, like digit 5). This is even more evident from the difference between the confusion matrix obtained under 6% poisoning and that obtained in the absence of attack.

Error-specific attack. Here, we assume that the attacker aims to misclassify 8s as 3s, while not having any preference regarding the classification of the other digits. This can be thus regarded as an *availability* attack, targeted to cause the misclassification of a specific set of samples. We generate 10 independent random splits with 1000 training samples, 4000 samples for validation, and 5000 samples for testing. Recall that the goal of the attacker in this scenario is described by Eq. (4). In particular, she aims at minimizing $L(\hat{\mathcal{D}}'_{val}, \hat{\boldsymbol{w}})$, where the samples in the validation set $\hat{\mathcal{D}}'_{val}$ are re-labelled according to the attacker's goal. Here, the validation set thus only consists of digits of class 8 labelled as 3. We set $T = 60$ to compute the back-gradients used in our poisoning attack, and inject up to 40 poisoning points into the training set. We initialize the poisoning points by cloning randomly-chosen samples from the classes 3 and 8 in the training set, and flipping their label from 3 to 8, or vice-versa. We consider only these two classes here as they are the only two actively involved in the attack.

The results are shown in Fig. 4 (bottom row). We can observe that only the classification error rate for digit 8 is significantly affected, as expected. In particular, it is clear from the difference of the confusion matrix obtained under poisoning and the one obtained in the absence of attack that most of the 8s are misclassified as 3s. After adding less than 4% of poisoning points, in fact, the error rate for digit 8 increases approximately from 20% to 50%. Note that, as a side effect, the error rate of digit 3 also slightly increases, though not to a significant extent.

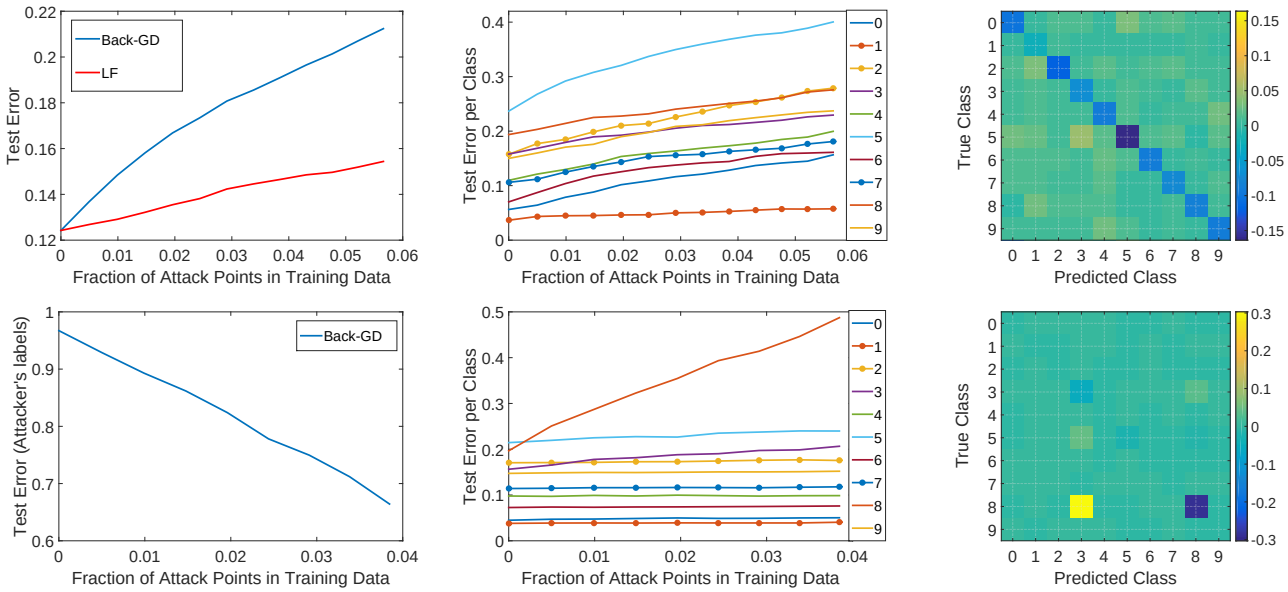

Figure 4: Error-generic (top row) and error-specific (bottom row) poisoning against multiclass LR on the MNIST data. In the first column, we report the test error (which, for error-specific poisoning attacks, is computed using the attacker's labels instead of the true labels, and so it decreases while approaching the attacker's goal). In the second column, we report the error per class, i.e., the probability of misclassifying a digit given that it belongs to the class reported in the legend. In the third column, we report the difference between the confusion matrix obtained under poisoning (after injecting the maximum number of poisoning samples) and that obtained in the absence of attack, to highlight how the errors affect each class.

Poisoning Deep Neural Networks. We finally report a proof-of-concept experiment to show the applicability of our attack algorithm to poison a deep network in an *end-to-end* manner, i.e., accounting for all weight updates in each layer (instead of using a surrogate model trained on a frozen deep feature representation [20]). To this end, we consider the convolutional neural network (CNN) proposed in [21] for classification of the MNIST digit data, which requires optimizing more than 450,000 parameters.[10] In this proof-of-concept attack, we inject 10 poisoning points into the training data, and repeat the experiment on 5 independent data splits, considering 1,000 samples for training, and 2,000 for validation and testing. For simplicity, we only consider the classes of digits 1, 5, and 6 in this case. We use Algorithm 1 to craft each single poisoning point, but, similarly to [39], we optimize them iteratively, making 2 passes over the whole set of poisoning samples. We also use the line search exploited in [39], instead of a fixed gradient step size, to reduce the attack complexity (i.e., the number of training updates to the deep network). Under this setting, however, we find that our attack points only slightly increase the classification error, though not significantly, while random label flips do not have any substantial effect. For comparison, we also attack a multiclass LR classifier under the same setting, yielding an increase of the error rate from 2% to 4.3% with poisoning attacks, and to only 2.1% with random label flips. This shows that, at least in this simple case, deep networks seem to be more resilient against (a very small fraction of) poisoning attacks (i.e., less than 1%). Some of the poisoning samples crafted against the CNN and the LR are shown in Figs. 5 and 6. We report the initial digit (and its true label y), its poisoned version (and its label y_c), and the difference between the two images, in absolute value (rescaled to visually appreciate the modified pixels). Notably, similarly to adversarial test examples, also poisoning samples against deep networks are visually indistinguishable from the initial image (as in [20]), while this is not the case when targeting the LR classifier. This might be due to the specific shape of the decision function learned by the deep network in the input space, as explained in the case of adversarial test examples [15, 36]. We however leave a more detailed investigation of this aspect to future work, along with a more systematic security evaluation of deep networks against poisoning attacks. We conclude this section with a simple *transferability* experiment, in which we use the poisoning samples crafted against the LR classifier to attack the CNN, and vice-versa. In the former case, the attack is totally ineffective, while in the latter case it has a similar effect to that of random label flips (as the minimal modifications to the CNN-poisoning digits are clearly irrelevant for the LR classifier).

[10] We use the implementation available at https://github.com/tflearn/tflearn/blob/master/examples/images/convnet_mnist.py.

5 RELATED WORK

Seminal work on the analysis of supervised learning in the presence of *omniscient* attackers that can compromise the training data has been presented in [12, 18]. While their results show the infeasibility of learning in such settings, their analysis reports an overly-pessimistic perspective on the problem. The first practical

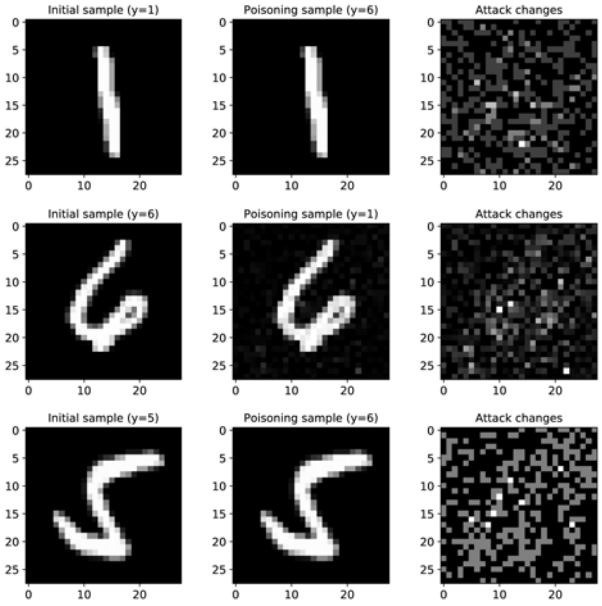

Figure 5: Poisoning samples targeting the CNN.

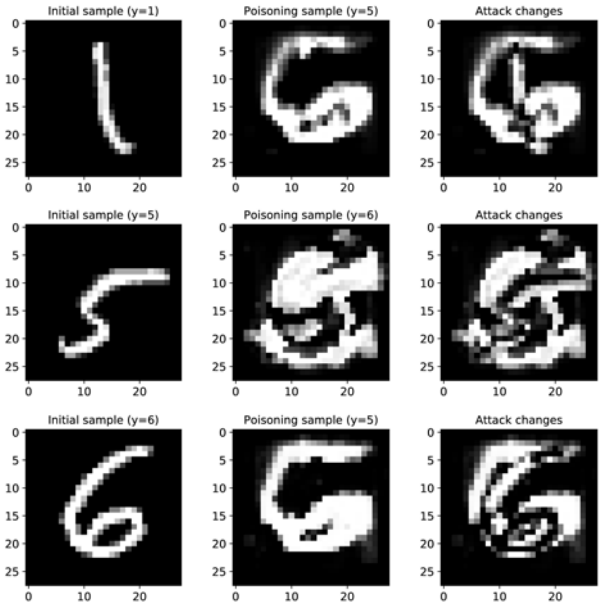

Figure 6: Poisoning samples targeting the LR.

poisoning attacks against two-class classification algorithms have been proposed in [19, 26], in the context of spam filtering and anomaly detection. However, such attacks do not easily generalize to different learning algorithms. More systematic attacks, based on the exploitation of KKT conditions to solve the bilevel problem corresponding to poisoning attacks have been subsequently proposed in [8, 20, 23, 39]. In particular, Biggio et al. [8] have been the first to demonstrate the vulnerability of SVMs to poisoning attacks. Following the same approach, Xiao et al. [39] have shown how to poison LASSO, ridge regression, and the elastic net. Finally, Mei and Zhu [23] has systematized such attacks under a unified framework to poison convex learning algorithms with Tikhonov regularizers, based on the concept of machine teaching [29, 40]. The fact that these techniques require full re-training of the learning algorithm at each iteration (to fulfil the KKT conditions up to a sufficient finite precision), along with the intrinsic complexity required to compute the corresponding gradients, makes them too computationally demanding for several practical settings. Furthermore, this limits their applicability to a wider class of learning algorithms, including those based on gradient descent and subsequent variants, like deep neural networks, as their optimization is often truncated prior to meeting the stationarity conditions with the precision required to compute the poisoning gradients effectively. Note also that, despite recent work [20] has provided a first proof of concept of the existence of *adversarial training examples* against deep networks, this has been shown on a binary classification task using a surrogate model (attacked with standard KKT-based poisoning). In particular, the authors have generated the poisoning samples by attacking a logistic classifier trained on the features extracted from the penultimate layer of the network (which have been kept fixed). Accordingly, to our knowledge, our work is thus the first to show how to poison a deep neural network in an *end-to-end* manner, considering all its parameters and layers, and without using any surrogate model. Notably, our work is also the first to show (in a more systematic way) that poisoning samples can be *transferred* across different learning algorithms, using *substitute* (a.k.a. *surrogate*) models, as similarly demonstrated for evasion attacks (i.e., adversarial test examples) in [6, 37] against SVMs and NNs, and subsequently in [27] against deep networks.

6 CONCLUSIONS, LIMITATIONS AND FUTURE WORK

Advances in machine learning have led to a massive use of data-driven technologies with emerging applications in many different fields, including cybersecurity, self-driving cars, data analytics, biometrics and industrial control systems. At the same time, the variability and sophistication of cyberattacks have tremendously increased, making machine learning systems an appealing target for cybercriminals [2, 16].

In this work, we have considered the threat of training data poisoning, i.e., an attack in which the training data is purposely manipulated to maximally degrade the classification performance of learning algorithms. While previous work has shown the effectiveness of such attacks against binary learners [8, 20, 23, 39], in this work we have been the first to consider poisoning attacks in multiclass classification settings. To this end, we have extended the commonly-used threat model proposed in [1, 2, 16] by introducing the concept of *error specificity*, to denote whether the attacker aims to cause specific misclassification errors (i.e., misclassifying samples as a specific class), or generic ones (i.e., misclassifying samples as any class different than the correct one).

Another important contribution of this work has been to overcome the limitations of state-of-the-art poisoning attacks, which require exploiting the stationarity (KKT) conditions of the attacked

learning algorithms to optimize the poisoning samples [8, 20, 23, 39]. As discussed throughout this work, this requirement, as well as the intrinsic complexity of such attacks, limits their application only to a reduced class of learning algorithms. In this work, we have overcome these limitations by proposing a novel poisoning algorithm based on back-gradient optimization [14, 22, 31]. Our approach can be applied to a wider class of learning algorithms, as it only requires the learning algorithm to update smoothly its parameters during training, without even necessarily fulfilling the optimality conditions with very high precision. Moreover, the gradients can be accurately estimated with the parameters obtained from an incomplete optimization of the learning algorithm truncated to a reduced number of iterations. This enables the efficient application of our attack strategy to large neural networks and deep learning architectures, as well as any other learning algorithm trained through gradient-based procedures. Our empirical evaluation on spam filtering, malware detection, and handwritten digit recognition has shown that neural networks can be significantly compromised even if the attacker only controls a small fraction of training points. We have also empirically shown that poisoning samples designed against one learning algorithm can be rather effective also in poisoning another algorithm, highlighting an interesting *transferability* property, as that shown for evasion attacks (a.k.a. adversarial test examples) [6, 27, 37].

The main limitation of this work is that we have not run an extensive evaluation of poisoning attacks against deep networks, to thoroughly assess their security to poisoning. Our preliminary experiments seem to show that they can be more resilient against this threat than other learning algorithms. This may be due to their higher capacity and number of parameters, which may allow the network to memorize the poisoning samples without affecting what has been correctly learned elsewhere. Despite this, it is also worth remarking that our attack has been designed to maximize the overall classification error (potentially on a subset of classes) and, thus, attacks with less ambitious goals (such as targeted attacks aimed at misclassifying only a small subset of samples, as in [20]) may still be more effective. Therefore, a more complete and systematic analysis remains to be performed, and we plan to more systematically investigate the effectiveness of our back-gradient poisoning attack against deep networks in the very near future. Besides the extension and evaluation of this poisoning attack strategy to different deep learning architectures and nonparametric models, further research avenues include: the investigation of the existence of *universal perturbations* (not dependent on the initial attack point) for poisoning samples against deep networks, similarly to the case of universal adversarial test examples [15, 24]; and the evaluation of defense mechanisms against poisoning attacks, through the exploitation of data sanitization and robust learning algorithms [5, 32, 35].

ACKNOWLEDGMENTS

This work has been supported in part by the UK EPSRC under grants EP/L022729/1 and EP/L016796/1.

REFERENCES

[1] Marco Barreno, Blaine Nelson, Anthony Joseph, and J. Tygar. 2010. The security of machine learning. *Machine Learning* 81 (2010), 121–148. Issue 2.

[2] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. 2006. Can machine learning be secure?. In *Proc. ACM Symp. Information, Computer and Comm. Sec. (ASIACCS '06)*. ACM, New York, NY, USA, 16–25.

[3] Y. Bengio. 2000. Gradient-based optimization of hyperparameters. *Neural Computation* 12, 8 (2000), 1889–1900.

[4] Battista Biggio, Samuel Rota Bulò, Ignazio Pillai, Michele Mura, Eyasu Zemene Mequanint, Marcello Pelillo, and Fabio Roli. 2014. Poisoning complete-linkage hierarchical clustering. In *Joint IAPR Int'l Workshop on Structural, Syntactic, and Statistical Pattern Recognition (Lecture Notes in Computer Science)*, P. Franti, G. Brown, M. Loog, F. Escolano, and M. Pelillo (Eds.), Vol. 8621. Springer Berlin Heidelberg, Joensuu, Finland, 42–52.

[5] Battista Biggio, Igino Corona, Giorgio Fumera, Giorgio Giacinto, and Fabio Roli. 2011. Bagging Classifiers for Fighting Poisoning Attacks in Adversarial Classification Tasks. In *10th International Workshop on Multiple Classifier Systems (MCS) (Lecture Notes in Computer Science)*, Carlo Sansone, Josef Kittler, and Fabio Roli (Eds.), Vol. 6713. Springer-Verlag, 350–359.

[6] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli. 2013. Evasion attacks against machine learning at test time. In *Machine Learning and Knowledge Discovery in Databases (ECML PKDD), Part III (LNCS)*, Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Železný (Eds.), Vol. 8190. Springer Berlin Heidelberg, 387–402.

[7] Battista Biggio, Giorgio Fumera, and Fabio Roli. 2014. Security Evaluation of Pattern Classifiers Under Attack. *IEEE Transactions on Knowledge and Data Engineering* 26, 4 (April 2014), 984–996.

[8] Battista Biggio, Blaine Nelson, and Pavel Laskov. 2012. Poisoning attacks against support vector machines, In 29th Int'l Conf. on Machine Learning, John Langford and Joelle Pineau (Eds.). *Int'l Conf. on Machine Learning (ICML)*, 1807–1814.

[9] Battista Biggio, Ignazio Pillai, Samuel Rota Bulò, Davide Ariu, Marcello Pelillo, and Fabio Roli. 2013. Is Data Clustering in Adversarial Settings Secure?. In *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security (AISec '13)*. ACM, New York, NY, USA, 87–98.

[10] Battista Biggio, Konrad Rieck, Davide Ariu, Christian Wressnegger, Igino Corona, Giorgio Giacinto, and Fabio Roli. 2014. Poisoning Behavioral Malware Clustering. In *2014 Workshop on Artificial Intelligent and Security (AISec '14)*. ACM, New York, NY, USA, 27–36.

[11] C. Blake and C.J. Merz. 1998. UCI Repository of machine learning databases. *http://www. ics. uci. edu/˜ mlearn/MLRepository. html* (1998).

[12] NaderH. Bshouty, Nadav Eiron, and Eyal Kushilevitz. 1999. PAC Learning with Nasty Noise. In *Algorithmic Learning Theory*, Osamu Watanabe and Takashi Yokomori (Eds.). Lecture Notes in Computer Science, Vol. 1720. Springer Berlin Heidelberg, 206–218. https://doi.org/10.1007/3-540-46769-6_17

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

[14] Justin Domke. 2012. Generic Methods for Optimization-Based Modeling. In *15th Int'l Conf. Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, Neil D. Lawrence and Mark Girolami (Eds.), Vol. 22. PMLR, La Palma, Canary Islands, 318–326.

[15] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations*.

[16] L. Huang, A. D. Joseph, B. Nelson, B. Rubinstein, and J. D. Tygar. 2011. Adversarial Machine Learning. In *4th ACM Workshop on Artificial Intelligence and Security (AISec 2011)*. Chicago, IL, USA, 43–57.

[17] Anthony D. Joseph, Pavel Laskov, Fabio Roli, J. Doug Tygar, and Blaine Nelson. 2013. Machine Learning Methods for Computer Security (Dagstuhl Perspectives Workshop 12371). *Dagstuhl Manifestos* 3, 1 (2013), 1–30.

[18] Michael Kearns and Ming Li. 1993. Learning in the presence of malicious errors. *SIAM J. Comput.* 22, 4 (1993), 807–837. https://doi.org/10.1137/0222052

[19] Marius Kloft and Pavel Laskov. 2012. Security Analysis of Online Centroid Anomaly Detection. *Journal of Machine Learning Research* 13 (2012), 3647–3690.

[20] P. W. Koh and P. Liang. 2017. Understanding Black-box Predictions via Influence Functions. In *International Conference on Machine Learning (ICML)*.

[21] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-Based Learning Applied to Document Recognition. In *Proceedings of the IEEE*, Vol. 86. 2278–2324.

[22] Dougal Maclaurin, David Duvenaud, and Ryan P. Adams. 2015. Gradient-based Hyperparameter Optimization Through Reversible Learning. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37 (ICML'15)*. JMLR.org, 2113–2122.

[23] Shike Mei and Xiaojin Zhu. 2015. Using Machine Teaching to Identify Optimal Training-Set Attacks on Machine Learners. In *29th AAAI Conf. Artificial Intelligence (AAAI '15)*.

[24] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. 2017. Universal adversarial perturbations. In *CVPR*.

[25] B. Nelson, M. Barreno, F.J. Chi, A.D. Joseph, B.I.P. Rubinstein, U. Saini, C.A. Sutton, J.D. Tygar, and K. Xia. 2008. Exploiting Machine Learning to Subvert your Spam Filter. *LEET* 8 (2008), 1–9.

[26] Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D. Joseph, Benjamin I. P. Rubinstein, Udam Saini, Charles Sutton, J. D. Tygar, and Kai Xia. 2008. Exploiting machine learning to subvert your spam filter. In *LEET'08: Proceedings*

of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats. USENIX Association, Berkeley, CA, USA, 1–9.

[27] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. 2017. Practical Black-Box Attacks Against Machine Learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (ASIA CCS '17)*. ACM, New York, NY, USA, 506–519.

[28] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. 2016. The Limitations of Deep Learning in Adversarial Settings. In *Proc. 1st IEEE European Symposium on Security and Privacy*. IEEE, 372–387.

[29] K.R. Patil, X. Zhu, L. Kopeć, and B.C. Love. 2014. Optimal teaching for limited-capacity human learners. In *Advances in Neural Information Processing Systems*. 2465–2473.

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

[31] F. Pedregosa. 2016. Hyperparameter optimization with approximate gradient. In *33rd International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.), Vol. 48. PMLR, New York, New York, USA, 737–746.

[32] Benjamin I.P. Rubinstein, Blaine Nelson, Ling Huang, Anthony D. Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and J. D. Tygar. 2009. ANTIDOTE: understanding and defending against poisoning of anomaly detectors. In *Proceedings of the 9th ACM SIGCOMM Internet Measurement Conference (IMC '09)*. ACM, New York, NY, USA, 1–14.

[33] D. Sgandurra, L. Muñoz-González, R. Mohsen, and E.C. Lupu. 2016. Automated Dynamic Analysis of Ransomware: Benefits, Limitations and use for Detection. *arXiv preprint arXiv:1609.03020* (2016).

[34] Charles Smutz and Angelos Stavrou. 2012. Malicious PDF Detection Using Metadata and Structural Features. In *Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC '12)*. ACM, New York, NY, USA, 239–248.

[35] J. Steinhardt, P. W. Koh, and P. Liang. 2017. Certified Defenses for Data Poisoning Attacks. *arXiv preprint arXiv:1706.03691* (2017).

[36] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *International Conference on Learning Representations*. http://arxiv.org/abs/1312.6199

[37] Nedim Šrndic and Pavel Laskov. 2014. Practical Evasion of a Learning-Based Classifier: A Case Study. In *Proc. 2014 IEEE Symp. Security and Privacy (SP '14)*. IEEE CS, Washington, DC, USA, 197–211.

[38] Gang Wang, Tianyi Wang, Haitao Zheng, and Ben Y. Zhao. 2014. Man vs. Machine: Practical Adversarial Detection of Malicious Crowdsourcing Workers. In *23rd USENIX Security Symposium (USENIX Security 14)*. USENIX Association, San Diego, CA.

[39] Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert, and Fabio Roli. 2015. Is Feature Selection Secure against Training Data Poisoning?. In *JMLR W&CP - Proc. 32nd Int'l Conf. Mach. Learning (ICML)*, Francis Bach and David Blei (Eds.), Vol. 37. 1689–1698.

[40] X. Zhu. 2013. Machine Teaching for Bayesian Learners in the Exponential Family. In *Advances in Neural Information Processing Systems*. 1905–1913.