CERTIFIED ROBUSTNESS TO DATA POISONING IN GRADIENT-BASED TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern machine learning pipelines leverage large amounts of public data, making it infeasible to guarantee data quality and leaving models open to poisoning and backdoor attacks. Provably bounding model behavior under such attacks remains an open problem. In this work, we address this challenge by developing the first framework providing provable guarantees on the behavior of models trained with potentially manipulated data without modifying the model or learning algorithm. In particular, our framework certifies robustness against untargeted and targeted poisoning, as well as backdoor attacks, for bounded and unbounded manipulations of the training inputs and labels. Our method leverages convex relaxations to over-approximate the set of all possible parameter updates for a given poisoning threat model, allowing us to bound the set of all reachable parameters for any gradient-based learning algorithm. Given this set of parameters, we provide bounds on worst-case behavior, including model performance and backdoor success rate. We demonstrate our approach on multiple real-world datasets from applications including energy consumption, medical imaging, and autonomous driving.

1 INTRODUCTION

To achieve state-of-the-art performance, modern machine learning pipelines involve pre-training on massive, uncurated datasets; subsequently, models are fine-tuned with task-specific data to maximize downstream performance (Han et al., 2021). Unfortunately, the datasets used in both steps are potentially untrustworthy and of such scale that rigorous quality checks become impractical.

Data Poisoning. Yet, adversarial manipulation, i.e., *poisoning attacks*, affecting even a small proportion of data used for either pre-training or fine-tuning can lead to catastrophic model failures (Carlini et al., 2023). For instance, Yang et al. (2017) show how popular recommender systems on sites such as YouTube, Ebay, and Yelp can be easily manipulated by poisoning. Likewise, Zhu et al. (2019) show that poisoning even 1% of training data can lead models to misclassify targeted examples, and Han et al. (2022) use poisoning to selectively trigger backdoor vulnerabilities in lane detection systems to force critical errors.

Poisoning Defenses. Despite the gravity of the failure modes induced by poisoning attacks, counter-measures are generally attack-specific and only defend against known attack methods (Tian et al., 2022). The result of attack-specific defenses is an effective "arms race" between attackers trying to circumvent the latest defenses and counter-measures being developed for the new attacks. In effect, even best practices, i.e., using the latest defenses, provide no guarantees of protection against poisoning attacks. To date, relatively few approaches have sought provable guarantees against poisoning attacks. These methods are often limited in scope, e.g., applying only to linear models (Rosenfeld et al., 2020; Steinhardt et al., 2017) or only providing approximate guarantees for a limited set of poisoning settings (Rosenfeld et al., 2020; Xie et al., 2022). Other approaches partition datasets into hundreds or thousands of disjoint shards and then aggregate predictions such that the effects of poisoning is provably limited (Levine & Feizi, 2020; Wang et al., 2022). In contrast, our goal in this work is not to produce a robust learning algorithm, but to efficiently analyze the sensitivity of (un-modified) algorithms. Further discussion of related works is provided in Appendix A.

This Work: General Certificates of Poisoning Robustness. We present an approach for computing sound and general certificates of robustness to poisoning attacks for any model trained with first-order optimization methods, e.g., stochastic gradient descent or Adam (Kingma & Ba, 2014). The proposed strategy begins by treating various poisoning attacks as constraints over an adversary's perturbation 'budget' in input and label spaces. Following the comprehensive taxonomy by Tian et al. (2022), we view the objective of each poisoning attack as an optimization problem. We consider three objectives: (i) untargeted attacks: reducing model test performance to cause denial-of-service, (ii) targeted attacks: compromising model performance on certain types of inputs, and (iii) backdoor attacks: leaving the model performance stable, but introducing a trigger pattern that causes errors at deployment time. Our approach then leverages convex relaxations of both the training problem and the constraint sets defining the threat model to compute *a sound (but incomplete) certificate that bounds the impact of the poisoning attack.*

Paper Outline. The paper is organized as follows. We first provide a general framework for poisoning attacks, describing how our formulation captures the settings studied in prior works (related works are reviewed in Appendix A). We then present *abstract gradient training* (AGT), our technique for over-approximating the effect of a given poisoning attack and discuss implementation details, including a novel, explicit formulation of CROWN-like bounds (Zhang et al., 2018) on the weight gradients. We conclude with extensive ablation experiments on datasets from household energy consumption, medical image classification, and autonomous vehicle driving. In summary, this paper makes the following key contributions:

- A framework, including a novel bound propagation strategy, for computing sound bounds on the influence of a poisoning adversary on any model trained with gradient-based methods.
- Based on the above, a series of formal proofs that allow us to bound the effect of poisoning attacks that seek to corrupt the system with targeted, untargeted, or backdoor attacks.
- An extensive empirical evaluation demonstrating the effectiveness of our approach.

2 Preliminaries: Poisoning Attacks

We denote a machine learning model as a parametric function f with parameters θ, feature space $x \in \mathbb{R}^{n_{\text{in}}}$, and label space $y \in \mathcal{Y}$. The label space $\mathcal{Y}$ may be discrete (e.g. classification) or continuous (e.g. regression). We operate in the supervised learning setting with a dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$ where we index the dataset such that $\mathcal{D}_x^{(i)}$ is the i^{th} feature vector and $\mathcal{D}_y^{(i)}$ is the i^{th} label. We denote the parameter initialization θ' and a training algorithm M as $\theta = M(f, \theta', \mathcal{D})$, i.e., given a model, initialization, and data, the training function M returns a "trained" parameterization θ. Finally, we assume the loss function is computed element-wise from the dataset, denoted as $\mathcal{L}(f(x^{(i)}), y^{(i)})$.

Given this abstraction of model training, we now turn to developing an abstraction of the data poisoning attacks, defining their capabilities to adversarially manipulate the training input. To complete the threat model, we formulate adversary goals, i.e., what the adversaries seek to accomplish with their manipulation as optimization problems. Typical threat models additionally specify the adversary's system knowledge; however, as we aim to upper bound a worst-case adversary, we assume unrestricted access to all training information including model architecture and initialization, data, data ordering, hyper-parameters, etc.

2.1 Poisoning Attack Capabilities

This section defines the capabilities of the poisoning attack adversaries that we seek to certify against. We consider two distinct threat models: (1) ℓ_p-norm bounded adversaries, which are commonly assumed in backdoor attack models (Saha et al., 2020; Weber et al., 2023); (2) unbounded adversaries, which are more general and have the capability to inject arbitrary data into the training set. In both cases, we consider adversaries able to modify both features and labels simultaneously.

Bounded Attacks. Under a bounded attack setting, we allow an adversary to perturb a subset of the training data in both the feature and label space, where the magnitude of the perturbation is bounded in a given norm. In feature space, we allow for an adversary to modify at most n datapoints by a distance of up to ϵ in the ℓ_p-norm. Similarly, we define the label-space poisoning capability as modifying at most m labels by magnitude at most ν in an ℓ_q-norm. Note that label-space poisoning encompasses both classification and (multivariate) regression settings. Common label-flipping attacks can be considered under this attack model by setting $q = 0$. Formally, given a dataset $\mathcal{D}$ and an adversary $\langle n, \epsilon, p, m, \nu, q \rangle$, the set of potentially poisoned datasets is defined as

$$\mathcal{T}^{n,\epsilon,p}_{m,\nu,q}(\mathcal{D}) := \left\{ \widetilde{\mathcal{D}} \;\middle|\; \begin{array}{l} \|\mathcal{D}^{(i)}_x - \widetilde{\mathcal{D}}^{(i)}_x\|_p \leq \epsilon \; \forall i \in I, \quad \mathcal{D}^{(i')}_x = \widetilde{\mathcal{D}}^{(i')}_x \; \forall i' \notin I, \quad \forall I \in \mathcal{S}_n \\ \|\mathcal{D}^{(j)}_y - \widetilde{\mathcal{D}}^{(j)}_y\|_q \leq \nu \; \forall j \in J, \quad \mathcal{D}^{(j')}_y = \widetilde{\mathcal{D}}^{(j')}_y \; \forall j' \notin J, \quad \forall J \in \mathcal{S}_m \end{array} \right\} \tag{1}$$

where $\mathcal{S}_n$ is the set of all subsets of integers less than N with cardinality at most n. The index sets I and J refer to the data-points that have been poisoned in the feature and label spaces, respectively. Note that in a paired modification setting, adversaries must choose a set of inputs and modify both their features and labels, which corresponds to setting $I = J$ in the above. Our more general setting allows for adversaries to modify features and labels of different inputs.

Unbounded Attacks. It may not always be realistic to assume that the effect of a poisoning adversary is bounded. A more powerful adversary may be able to inject arbitrary data points into the training data set, for example by exploiting the collection of user data. In this unbounded attack setting, we consider only a 'paired' modification setting, where an adversary can substitute both the features and labels of any n data-points. Specifically, for a dataset $\mathcal{D}$, the set of potentially poisoned datasets is

$$\mathcal{T}^n(\mathcal{D}) := \left\{ \widetilde{\mathcal{D}} = (\mathcal{D} \setminus \mathcal{D}^r) \cup \mathcal{D}^a \;\middle|\; |\mathcal{D}^a| \leq n, |\mathcal{D}^r| \leq n, \mathcal{D}^a \subset \mathcal{D} \right\} \tag{2}$$

where $|\cdot|$ is the cardinality operator, $\mathcal{D}^a$ is a set of arbitrary added data-points, and $\mathcal{D}^r$ is the set of corresponding data-points removed from the original dataset. To simplify the exposition below, we interchangeably refer to $\mathcal{T}^n$ as either the poisoning adversary or the set of poisoned datasets. We use $\mathcal{T}$ to refer to cases where either the bounded or unbounded adversaries may be applied.

2.2 Poisoning Attack Goals

Here, we cover the different goals poisoning adversaries may pursue and briefly outline what it means to certify that a training algorithm is robust against such an adversary.

Untargeted Poisoning. Untargeted attacks aim to prevent training convergence, leading to an unusable model and denial of service (Tian et al., 2022). Given the training dataset $\{(x^{(i)}, y^{(i)})\}_{i=1}^k$, the adversary's objective is thus:

$$\max_{\mathcal{D}' \in \mathcal{T}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}\big(f^{M(f,\theta',\mathcal{D}')}(x^{(i)}), y^{(i)}\big) \tag{3}$$

We can certify robustness to this kind of attack using a sound upper bound on the solution of (3).

Targeted Poisoning. Targeted poisoning is more task-specific and is typically evaluated over the test dataset. Rather than simply attempting to increase the loss, the adversary seeks to make model predictions fall outside a 'safe' set of outputs $S(x^{(i)}, y^{(i)})$ (e.g., the set of predictions matching the ground truth). The safe set can be more specific however, i.e., mistaking a lane marking for a person is safe, but not vice versa. The adversary's objective is given by:

$$\max_{\mathcal{D}' \in \mathcal{T}} \frac{1}{k} \sum_{i=1}^{k} \mathbb{1}\big(f^{M(f,\theta',\mathcal{D}')}(x^{(i)}) \notin S(x^{(i)}, y^{(i)})\big) \tag{4}$$

As before, a sound upper bound on (4) bounds the success rate of any targeted poisoning attacker. Note that with $k = 1$ we recover the pointwise certificate setting studied by Rosenfeld et al. (2020). This setting also covers 'unlearnable examples', such as the attacks considered by Huang et al. (2021).

Backdoor Poisoning. Backdoor attacks deviate from the above attacks by assuming that test-time data can be altered, via a so-called trigger manipulation. However, backdoor attacks typically aim to leave the model performance on 'clean' data unchanged. By assuming that the trigger manipulation(s) are bounded to a set $V(x)$ (e.g., an ℓ_∞ ball around the input), one can formulate the backdoor attack's goal as producing predictions outside a safe set $S(x^{(i)}, y^{(i)})$ (defined as before) for manipulated inputs:

$$\max_{\mathcal{D}' \in \mathcal{T}} \frac{1}{k} \sum_{i=1}^{k} \mathbb{1}\big(\exists x^\star \in V(x^{(i)}) \; s.t. \; f^{M(f,\theta',\mathcal{D}')}(x^\star) \notin S(x^{(i)}, y^{(i)})\big) \tag{5}$$

Any sound upper bound to the above is a sound bound on the success rate of any backdoor attacker.

3 METHODOLOGY

This section develops a novel approach for certifying robustness to poisoning attacks starting by assuming we have access to the set of all reachable trained models in the form of intervals over model parameters. We first show how these intervals can be used to certify robustness to the above attacks. We then formulate a general algorithm for bounding the effect of adversaries on model parameters, producing an interval containing all reachable training parameters. We then instantiate it using a novel formulation of CROWN-style bounds which can soundly compute the quantities required within.

3.1 PARAMETER-SPACE CERTIFICATES OF POISONING ROBUSTNESS

The key concept behind our framework is to bound the parameters obtained via the training function $M(f, \theta', \mathcal{D})$ given $\mathcal{T}(\mathcal{D})$. Before detailing the method, we first formalize our definition of parameter-space bounds and how they can be translated into formal, provable guarantees on poisoning robustness.

Definition 1. ***(Valid parameter-space bounds)*** *An interval over parameters $[\theta^L, \theta^U]$ such that $\forall i,\ \theta_i^L \leq \theta_i^U$ is a valid parameter-space bound on a poisoning adversary, $\mathcal{T}(\mathcal{D})$, if $\forall i$:*

$$\theta_i^L \leq \min_{\mathcal{D}' \in \mathcal{T}(\mathcal{D})} M(f, \theta', \mathcal{D}')_i \leq M(f, \theta', \mathcal{D})_i \leq \max_{\mathcal{D}' \in \mathcal{T}(\mathcal{D})} M(f, \theta', \mathcal{D}')_i \leq \theta_i^U \tag{6}$$

Intuitively, Definition 1 allows us to measure the poisoning adversary's influence in parameter space. From such bounds, we can then derive guarantees on the poisoning robustness against any of the aforementioned attack vectors.

Theorem 3.1. *Given valid parameter bounds $[\theta^L, \theta^U]$ for an adversary $\mathcal{T}(\mathcal{D})$, one can compute a sound upper bound (i.e., certificate) on any poisoning objective by optimization over the parameter space, rather than dataset space:*

$$\max_{\mathcal{D}' \in \mathcal{T}} J\big(f^{M(f,\theta',\mathcal{D}')}(x)\big) \leq \max_{\theta^\star \in [\theta^L, \theta^U]} J\big(f^{\theta^\star}(x)\big)$$

where J is one of the objective functions from (3)–(5)*. Full expressions are provided in Appendix I.1.*

The advantage of Theorem 3.1 is that each of these upper-bounds can be computed directly using bounds from works studying certification of adversarial robustness of probabilistic models (Adams et al., 2023; Wicker et al., 2020; 2023).

3.2 ABSTRACT GRADIENT TRAINING FOR VALID PARAMETER SPACE BOUNDS

In this section, we provide an intuition and high-level framework for computing parameter bounds that respect Definition 1. We call this framework *abstract gradient training* (AGT). Our framework is applicable to any training function M based on first-order optimization, e.g., stochastic gradient descent or Adam. To keep our exposition intuitive, we choose to focus on SGD, written as:

$$\theta \leftarrow \theta - \alpha \frac{1}{|\mathcal{B}|} \sum_{(x,y) \in \mathcal{B}} \nabla_\theta \mathcal{L}\big(f^\theta(x), y\big) \tag{7}$$

where $\mathcal{B} \subseteq \mathcal{D}$ is the sampled batch at the current iteration. The function $M(f, \theta', \mathcal{D})$, in the simplest case, iteratively applies the update (7) for a fixed, finite number of iterations starting from $\theta^{(0)} = \theta'$. Therefore, to bound the effect of a poisoning attack, we can iteratively apply bounds on update (7). In Algorithm 1, we present a general framework for computing parameter-space bounds on the output of SGD given a poisoning adversary $\mathcal{T}$. In particular, we highlight that Algorithm 1 first computes the standard stochastic gradient descent update (lines 4-5), and then computes a bound on the set of all possible descent directions that could be taken at this iteration. Note that the gradient clipping operation, highlighted in purple, is optional for a bounded adversary, and is only required for the unbounded adversary to limit the maximum contribution of any added data-points. This bound on the descent direction is then soundly combined with the existing parameter space bound to obtain an updated reachable parameter interval $[\theta^L, \theta^U]$. Since the reachable parameter interval is maintained over every iteration of the algorithm, we have the following theorem (which we prove in Appendix I.2):

Algorithm 1: ABSTRACT GRADIENT TRAINING FOR POISONING CERTIFICATION

input : f - ML model, θ' - param. initialisation, $\mathcal{D}$ - nominal dataset, E - epochs, α - learning rate, $\mathcal{T}$ - allowable dataset perturbations (poisoning adversary), κ - optional clipping parameter.
output: θ - nominal SGD parameter, $[\theta^L, \theta^U]$ - reachable parameter interval.

$\theta \leftarrow \theta'$; $[\theta^L, \theta^U] \leftarrow [\theta', \theta']$
for *E-many epochs* **do**
  for *each batch $\mathcal{B} \subset \mathcal{D}$* **do**
    `/* Compute the (optionally clipped) SGD update. */`
    $\Delta\theta \leftarrow \frac{1}{|\mathcal{B}|} \sum_{(x,y)\in\mathcal{B}} \mathrm{Clip}_\kappa \left[\nabla_\theta \mathcal{L}\left(f^\theta(x), y\right)\right]$
    $\theta \leftarrow \theta - \alpha\Delta\theta$
    `/* Define the set of descent directions possible under` $\mathcal{T}$. `*/`
    $\Delta\Theta \leftarrow \left\{ \frac{1}{|\widetilde{\mathcal{B}}|} \sum_{(\tilde{x},\tilde{y})\in\widetilde{\mathcal{B}}} \mathrm{Clip}_\kappa \left[\nabla_{\theta'} \mathcal{L}\left(f^{\theta'}(\tilde{x}), \tilde{y}\right)\right] \mid \widetilde{\mathcal{B}} \in \mathcal{T}(\mathcal{B}), \theta' \in \left[\theta^L, \theta^U\right] \right\}$
    `/* Bound the set of descent directions possible under` $\mathcal{T}$. `*/`
    Compute $\Delta\theta^L$, $\Delta\theta^U$ s.t. $\Delta\theta^L \preceq \Delta\theta \preceq \Delta\theta^U$ for all $\Delta\theta \in \Delta\Theta$
    `/* Update the reachable parameter interval under` $\mathcal{T}$. `*/`
    $\left[\theta^L, \theta^U\right] \leftarrow \left[\theta^L - \alpha\Delta\theta^U, \theta^U - \alpha\Delta\theta^L\right]$
return θ, $[\theta^L, \theta^U]$

Theorem 3.2. *Algorithm 1 returns valid parameter-space bounds on a $T^{m,\nu,q}_{n,\epsilon,p}(\mathcal{D})$ poisoning adversary for a stochastic gradient descent training procedure $M(f, \theta', \mathcal{D})$.*

Bounding the Descent Direction. The main complexity in Algorithm 1 is in bounding the set $\Delta\Theta$, which is the set of all possible descent directions at the given iteration under $\mathcal{T}$. In particular, $\widetilde{\mathcal{B}} \in \mathcal{T}(\mathcal{B})$ represents the effect of the adversary's perturbations on the *current* batch, while the reachable parameter interval $\theta' \in \left[\theta^L, \theta^U\right]$ represents the worst-case effect of adversarial manipulations to all *previously seen* batches. Exactly computing the set $\Delta\Theta$ is not computationally tractable, so we instead seek over-approximate element-wise bounds that can be computed efficiently within the training loop. We present a procedure for computing bounds for the bounded poisoning adversary in the following theorem. The analogous theorem for the unbounded adversary can be found in Appendix B.

Theorem 3.3 (Bounding the descent direction for a bounded adversary)**.** *Given a nominal batch $\mathcal{B} = \left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^b$ of size b, a parameter set $\left[\theta^L, \theta^U\right]$, and a bounded adversary $\langle n, \epsilon, p, m, \nu, q\rangle$, the SGD parameter update $\Delta\theta = \frac{1}{b}\sum_{\widetilde{\mathcal{B}}} \nabla_\theta \mathcal{L}\left(f^\theta\left(\tilde{x}^{(i)}\right), \tilde{y}^{(i)}\right)$ is bounded element-wise by*

$$\Delta\theta^L = \frac{1}{b}\left(\underset{m+n}{\mathrm{SEMin}}\left\{\tilde{\delta}_L^{(i)} - \delta_L^{(i)}\right\}_{i=1}^b + \sum_{i=1}^b \delta_L^{(i)}\right), \quad \Delta\theta^U = \frac{1}{b}\left(\underset{m+n}{\mathrm{SEMax}}\left\{\tilde{\delta}_U^{(i)} - \delta_U^{(i)}\right\}_{i=1}^b + \sum_{i=1}^b \delta_U^{(i)}\right)$$

for any $\widetilde{\mathcal{B}} \in T^{n,\epsilon,p}_{m,\nu,q}(\mathcal{B})$ and $\theta \in [\theta^L, \theta^U]$. The terms $\delta_L^{(i)}, \delta_U^{(i)}$ are sound bounds that account for the effect of previous adversarial manipulations. Likewise, the terms $\tilde{\delta}_L^{(i)}, \tilde{\delta}_U^{(i)}$ are bounds on the worst-case adversarial manipulations of the i-th data-point in the current batch, i.e.

$$\delta_L^{(i)} \preceq \delta \preceq \delta_U^{(i)} \quad \forall \delta \in \left\{\nabla_{\theta'}\mathcal{L}\left(f^{\theta'}(x^{(i)}), y^{(i)}\right) \mid \theta' \in [\theta^L, \theta^U]\right\}, \tag{8}$$

$$\tilde{\delta}_L^{(i)} \preceq \tilde{\delta} \preceq \tilde{\delta}_U^{(i)} \quad \forall \tilde{\delta} \in \left\{\nabla_{\theta'}\mathcal{L}\left(f^{\theta'}(\tilde{x}), \tilde{y}\right) \mid \theta' \in [\theta^L, \theta^U], \|x^{(i)} - \tilde{x}\|_p \le \epsilon, \|y^{(i)} - \tilde{y}\|_q \le \nu\right\}. \tag{9}$$

The operations SEMax_a and SEMin_a correspond to taking the sum of the element-wise top/bottom-a elements over each index of the input vectors. The update rule in Theorem 3.3 accounts for the effect of poisoning in previous batches by taking the lower and upper bounds on the gradient ($\delta_L^{(i)}, \delta_U^{(i)}$) for all θ reachable at the current iteration. Then, we bound the effect of adversarial manipulation in the current batch by taking the $n + m$ points that have the worst-case gradient bounds $\tilde{\delta}_L^{(i)}, \tilde{\delta}_U^{(i)}$ under poisoning. We assume that $m + n \le b$. If $m + n > b$, we take SEMin/Max with respect to

$\min(b, m+n)$ instead. Since we wish to soundly over-approximate this operation for all parameters, we perform this bounding operation independently over each index of the parameter vector. This is certainly a loose approximation, as the $n+m$ points that maximize the gradient at a particular index will likely not maximize the gradient of other indices. However, this relaxation allows us to efficiently compute and propagate interval enclosures between successive iterations of AGT. A further relaxation is taken when computing gradient bounds $\delta_L^{(i)}, \delta_U^{(i)}$, since they are computed independently for each sample i and the min/max for different samples may be attained for different $\theta' \in \left[\theta^L, \theta^U\right]$. Performing this computation independently for each i, however, allows for efficient bound-propagation that we describe in more detail in subsequent sections.

For the bounded adversary, gradient clipping is not a requirement and we can directly bound the un-modified SGD training procedure. However, it can still be desirable to add gradient clipping even in the bounded adversary case, as it can improve the tightness of our descent direction bounds and subsequently improve the guarantees afforded by AGT with little cost to training performance.

3.3 Computation of Sound Gradient Bounds

This section presents a novel algorithm for computing bounds on the following optimization problem

$$\min_{x^\star, y^\star, \theta^\star} \left\{ \nabla_{\theta'} \mathcal{L}\left(f^{\theta'}\left(\tilde{x}\right), \tilde{y}\right) \mid \theta' \in [\theta^L, \theta^U], \|x^{(i)} - \tilde{x}\|_p \leq \epsilon, \|y^{(i)} - \tilde{y}\|_q \leq \nu \right\}. \tag{10}$$

Computing the exact solution to this problem is, in general, a non-convex and NP-hard optimization problem. However, we require only over-approximate solutions; while these can introduce (significant) over-approximation of the reachable parameter set, they will always maintain soundness. Future work could investigate exact solutions, e.g., via mixed-integer programming (Huchette et al., 2023; Tsay et al., 2021). Owing to their tractability, our discussion focuses on the novel linear bound propagation techniques we develop for abstract gradient training. Noting that problems of the form (8) can be recovered by setting $\nu = \epsilon = 0$, we focus solely on this more general case in this section.

Neural networks. While the algorithm presented in Section 3.2 is general to any machine learning model trained via stochastic gradient descent, we focus our discussion on neural network models for the remained of the paper. We first define a neural network model $f^\theta : \mathbb{R}^{n_{\text{in}}} \to \mathbb{R}^{n_{\text{out}}}$ with K layers and parameters $\theta = \left\{(W^{(i)}, b^{(i)})\right\}_{i=1}^K$ as:

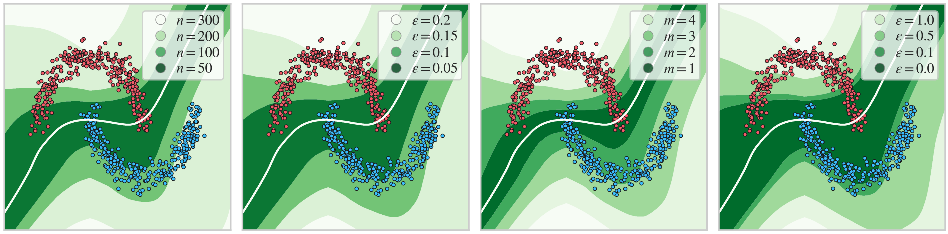
$$\hat{z}^{(k)} = W^{(k)} z^{(k-1)} + b^{(k)}, \quad z^{(k)} = \sigma\left(\hat{z}^{(k)}\right)$$

where $z^{(0)} = x$, $f^\theta(x) = \hat{z}^{(K)}$, and σ is the activation function, which we take to be ReLU.

Solving problems of the form $\min\{\cdot \mid \|x - x^\star\|_p \leq \epsilon\}$ for neural networks has been well-studied in the context of adversarial robustness certification. However, optimizing over inputs, labels and parameters, e.g., $\min\left\{\cdot \mid \theta^\star \in [\theta^L, \theta^U], \|x - x^\star\|_p \leq \epsilon, \|y - y^\star\|_q \leq \nu\right\}$ is much less well-studied, and to-date similar problems have appeared primarily in the certification of probabilistic neural networks (Wicker et al., 2020).

Interval Arithmetic. For ease of exposition, we will represent interval matrices with bold symbols i.e., $\boldsymbol{A} := [A_L, A_U\,] \subset \mathbb{R}^{n_1 \times n_2}$ and interval matrix multiplication as $\otimes$, meaning $AB \in \boldsymbol{A} \otimes \boldsymbol{B}$ for all $A \in \boldsymbol{A}$ and $B \in \boldsymbol{B}$. Additionally, we define $\odot$ and $\oplus$ as element-wise interval matrix multiplication and addition, respectively. This implies $A \circ B \in \boldsymbol{A} \odot \boldsymbol{B}$ (where $\circ$ is the element-wise product) and $A + B \in \boldsymbol{A} \oplus \boldsymbol{B}$ for all $A \in \boldsymbol{A}$ and $B \in \boldsymbol{B}$, which can be computed using standard interval arithmetic techniques. We denote interval vectors as $\boldsymbol{a} := [a_L, a_U]$ with analogous operations. More details of interval arithmetic operations over matrices and vectors can be found in Appendix C.

Forward Pass Bounds Mirroring developments in robustness certification of neural networks, we provide a novel, explicit extension of the CROWN algorithm (Zhang et al., 2018) to account for interval-bounded weights. The standard CROWN algorithm bounds the outputs of the m-th layer of a neural network by back-propagating linear bounds over each intermediate activation function to the input layer. We extend this framework to interval parameters, where the weights and biases involved in these linear relaxations are themselves intervals. We note that linear bound propagation with interval parameters has been studied previously in the context of floating-point sound certification (Singh et al., 2019). In the interest of space, we present only the upper bound of our extended CROWN algorithm here, with the full version presented in Appendix D.



(a) Feature poisoning ($\varepsilon = 0.01, m = 0$) (b) Feature poisoning ($n = 10, m = 0$) (c) Label poisoning ($n = 0, \varepsilon = 0$) (d) Label + feature poisoning ($n = m = 1$)

Figure 1: Bounds on a classification threshold trained on the halfmoons dataset for a bounded adversary that can perturb up to n data-points by up to ϵ in the $p = \infty$ norm; in label space, the adversary may flip up to m labels (corresponding to $\gamma = 1, q = 0$). The white line shows the decision boundary of the nominal classification model. The coloured regions show the areas for which we *cannot* certify robustness for the given adversary strength.

Proposition 1 (Explicit upper bounds of neural network f with interval parameters)**.** *Given an m-layer neural network function $f : \mathbb{R}^{n_{in}} \to \mathbb{R}^{n_{out}}$ whose unknown parameters lie in the intervals $b^{(k)} \in \boldsymbol{b}^{(k)}$ and $W^{(k)} \in \boldsymbol{W}^{(k)}$ for $k = 1, \dots, m$, there exist an explicit function*

$$f_j^U\left(x, \Lambda^{(0:m)}, \Delta^{(1:m)}, b^{(1:m)}\right) = \Lambda_{j,:}^{(0)} x + \sum_{k=1}^{m} \Lambda_{j,:}^{(k)} \left(b^{(k)} + \Delta_{:,j}^{(k)}\right) \tag{11}$$

such that $\forall x \in \boldsymbol{x}$

$$f_j(x) \le \max\left\{ f_j^U\left(x, \Lambda^{(0:m)}, \Delta^{(1:m)}, b^{(1:m)}\right) \mid \Lambda^{(k)} \in \boldsymbol{\Lambda}^{(k)}, b^k \in \boldsymbol{b}^{(k)} \right\} \tag{12}$$

where $\boldsymbol{x}$ is a closed input domain and $\Lambda^{(0:m)}, \Delta^{(1:m)}$ are the equivalent weights and biases of the linear upper bounds, respectively. The bias term $\Delta^{(1:m)}$ is explicitly computed based on the linear bounds on the activation functions. The weight $\Lambda^{(0:m)}$ lies in an interval $\boldsymbol{\Lambda}^{(0:m)}$ which is computed in an analogous way to standard, non-interval CROWN (see Appendix D for further details).

Given the upper bound function $f_j^U(\cdot)$ defined above and intervals over all the relevant variables, we can compute the following closed-form global upper bound:

$$\gamma_j^U = \max\left\{ \boldsymbol{\Lambda}_{j,:}^{(0)} \otimes \boldsymbol{x} \oplus \sum_{k=1}^{m} \boldsymbol{\Lambda}_{j,:}^{(k)} \otimes \left[\boldsymbol{b}^{(k)} \oplus \Delta_{:,j}^{(k)}\right] \right\}$$

where $\max$ is performed element-wise and returns the upper bound of the interval enclosure. Then, we have $f_j(x) \le \gamma_j^U$ for all $x \in \boldsymbol{x}$, $b^{(k)} \in \boldsymbol{b}^{(k)}$ and $W^{(k)} \in \boldsymbol{W}^{(k)}$. The equivalent procedure for computing the global lower bound $f_j(x) \ge \gamma_j^L$ can be found in Appendix D.

Backward Pass Bounds. Given bounds on the forward pass of the neural network, we can bound the backward pass (the gradients) of the model. We do this by extending the interval arithmetic based approach of Wicker et al. (2022) (which bounds derivatives of the form $\partial\mathcal{L}/\partial z^{(k)}$) to additionally bound the derivatives w.r.t. the parameters. Details of this computation can be found in Appendix F.

3.4 Algorithm Analysis and Discussion

Figure 1 visualizes the resulting worst-case decision boundaries for a simple binary classifier consisting of a neural network with a hidden layer of 128 neurons. In this classification setting, label poisoning results in looser bounds than feature space poisoning, with $m = 5$ producing bounds of approximately the same width as $n = 50$. This is due to the relatively large interval introduced by a label flipping attack $y^{(i)} \in \{0, 1\}$, compared to an interval of width ϵ introduced in a feature-space attack. We also emphasise that Algorithm 1 assumes at most m, n poisoned points *per batch*, rather than per dataset. In regression settings, label poisoning is relatively weaker than feature poisoning for a given strength $\epsilon = \nu$, since the feature-space interval propagates through both the forward and backward training passes, while the label only participates in the backward pass. This effect is particularly pronounced in deep networks, since interval/CROWN bounds tend to weaken exponentially with depth (Mao et al., 2023; Sosnin & Tsay, 2024).

Comparison to Interval Bound Propagation. The CROWN algorithm in § 3.3 is not strictly tighter than interval bound propagation (IBP). Specifically, the non-associativity of double-interval matrix multiplication leads to significantly different interval sizes depending on the order in which the multiplications are performed: IBP performs interval matrix multiplications in a 'forwards' ordering, while CROWN uses a 'backwards' ordering. Empirically, we observe that CROWN tends to be tighter for deeper networks, while IBP may outperform CROWN for smaller networks. In our numerical experiments, we compute both CROWN and IBP bounds and take the element-wise tightest bound.

Combined Forward and Backward Pass Bounds. The CROWN algorithm can be applied to any composition of functions that can be upper- and lower-bounded by linear equations. Therefore, it is possible to consider both the forwards and backwards passes in a single combined CROWN pass for many loss functions. However, linear bounds on the gradient of the loss function tend to be relatively loose, e.g., linear bounds on the softmax function may be orders-of-magnitude looser than constant $[0, 1]$ bounds (Wei et al., 2023). As a result, we found that the tightest bounds were obtained using IBP/CROWN on the forward pass and IBP on the backward pass.

Computational Complexity. The computational complexity of Algorithm 1 depends on the method used to bound on the gradients. In the simplest case, IBP can be used to compute bounds on the gradients in $4\times$ the cost of a standard forward and backward pass (see Appendix C). Likewise, our CROWN bounds admit a cost of at most 4 times the cost of the original CROWN algorithm. For an m layer network with n neurons per layer and n outputs, the time complexity of the original CROWN algorithm is $\mathcal{O}(m^2n^3)$ (Zhang et al., 2018). We further note that the SEMin / SEMax operations required by Theorems B.1 and 3.3 can be computed in $\mathcal{O}(b)$ for each index and in practice can be efficiently parallelized using GPU-based implementations. In summary, Abstract Gradient Training using IBP has time complexity equivalent to standard neural network training ($\mathcal{O}(bmn^2)$ for each batch of size b), but with our tighter, CROWN-based, bounds the complexity is $\mathcal{O}(bm^2n^3)$ per batch.

Limitations. While Algorithm 1 is able to obtain valid-parameter space bounds for any gradient-based training algorithm, the tightness of these bounds depends on the exact architecture, hyperparameters and training procedure used. In particular, bound-propagation between successive iterations of the algorithm assumes the worst-case poisoning at *each parameter index*, which may not be achievable by realistic poisoning attacks. Therefore, obtaining non-vacuous guarantees with our algorithm often requires training with larger batch-sizes and/or for fewer epochs than is typical. Additionally, certain loss functions, such as multi-class cross entropy, have particularly loose interval relaxations. Therefore, AGT obtains relatively weaker guarantees for multi-class problems when compared to regression or binary classification settings. We hope that tighter bound-propagation approaches, such as those based on more expressive abstract domains, may overcome this limitation in future works.

Computing Certificates of Poisoning Robustness. Algorithm 1 returns valid parameter-space bounds $[\theta_L, \theta_U]$ for a given poisoning adversary. To provide certificates of poisoning robustness for a specific query at a point x, we first bound the model output $f^\theta(x) \forall \theta \in [\theta_L, \theta_U]$ using the bound-propagation procedure described above. In classification settings, the robustness of the prediction can then be certified by checking if the lower bound on the output logit for the target class is greater than the upper bounds of all other classes (i.e. $[f_j^\theta(x)]_L \geq [f_i^\theta(x)]_U \forall i \neq j$). If this condition is satisfied, then the model always predicts class j at the point x for all parameters within our parameter-space bounds, and thus this prediction is certifiably robust to poisoning. Details on computing bounds on other poisoning adversary objectives (3), (4), and (5) can be found in Appendix G.

4 Experiments

Computational experiments were performed using a Python implementation of Algorithm 1. Bounds on accuracy/error are computed by bounding the respective optimization problem from Theorem 4.1 using IBP/CROWN. To investigate the tightness of AGT, we also compare our bounds with heuristic poisoning attacks in Appendix H. The experimental set-up and datasets are described in Appendix J.

UCI Regression (Household Power Consumption). We first consider a relatively simple regression model for the household electric power consumption ('houseelectric') dataset from the UCI repository (Hebrail & Berard, 2012) with fully connected neural networks and MSE as loss function. Figure 2 (top) shows the progression of the nominal and worst/best-case MSE (computed for the test set) for a 1×50 neural network and various parameterizations of poisoning attacks. As expected, we observe that increasing each of n, m, ϵ, and ν results in looser performance bounds. We note that the settings of $m = 10000$ and $n = 10000$ correspond 100% of the data (batchsize $b = 10000$) being poisoned.

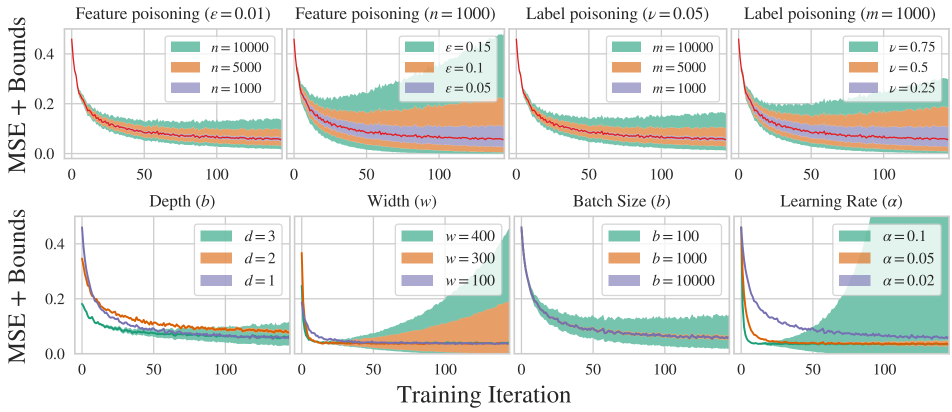


Figure 2: Mean squared error bounds on the UCI-houseelectric dataset. Top: Effect of adversary strength. Bottom: Effect of model/training hyperparameters (with $n = 100, \epsilon = 0.01, p = \infty$). Where not stated, $d = 1, w = 50, b = 10000$, and $\alpha = 0.02$.

Figure 2 (bottom) shows the progression of bounds on the MSE (computed for the test set) over the training procedure for a fixed poisoning attack ($n = 100, \epsilon = 0.01$) and various hyperparameters of the regression model. In general, we observe that increasing model size (width or depth) results in looser performance guarantees. As expected, increasing the batch size improves our bounds, as the number of potentially poisoned samples n remains fixed and their worst-case effect is 'diluted'. Increasing the learning rate accelerates both the model training and the deterioration of the bounds.

MNIST Digit Recognition. We consider a label-flipping attack ($q = 0, \nu = 1$) on the MNIST dataset. In label-only poisoning settings, it is common to use unsupervised learning approaches on the (assumed clean) features prior to training a classification model (e.g. SS-DPA (Levine & Feizi, 2020)). Therefore, we first project the data into a 32-dimensional feature space using PCA and then train a linear classifier using AGT to obtain a certified classification model. Figure 3 illustrates the certifiable accuracy using this methodology. Compared to regression or binary classification settings, AGT provides limited guarantees for multi-class problems; we hope that tighter bound propagation techniques will overcome this limitation in the future. Stronger guarantees can be obtained by increasing the gradient clipping parameter κ, at the cost of decreased model utility.

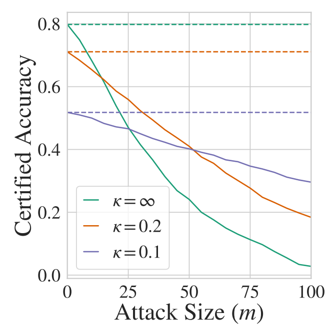


Figure 3: Certified accuracy on the MNIST dataset under a label-flipping attack.

MedMNIST Image Classification. Next, we consider fine-tuning a classifier trained on the retinal OCT dataset (OCTMNIST) (Yang et al., 2021), which contains four classes—one normal and three abnormal. The dataset is unbalanced, and we consider the simpler normal vs abnormal binary classification setting. We consider the 'small' architecture from Gowal et al. (2018), comprising two convolutional layers of width 16 and 32 and a dense layer of 100 nodes, and the following fine-tuning scenario: the model is first pre-trained without the rarest class (Drusen) using the robust training procedure from Wicker et al. (2022), so that the resulting model is robust to feature perturbations during fine-tuning. We then assume Drusen samples may be poisoned and add them as a new abnormal class to fine-tune the dense layer, with a mix of 50% Drusen samples ($b = 6000$ with 3000 Drusen) per batch.

In general, this fine-tuning step improves accuracy on the new class (Drusen) from approximately 0.5 to over 0.8. Nevertheless, Figure 4 shows how increasing the amount of potential poisoning worsens the bound on prediction accuracy. With feature-only poisoning, a poisoning attack greater than $\epsilon = 0.02$ over $n \approx 500$ samples produces bounds worse than the prediction accuracy of the original pre-trained model. With an unbounded label and feature poisoning adversary, the bounds are weaker, as expected. Higher certified accuracy can be obtained by increasing the clipping parameter κ, at the

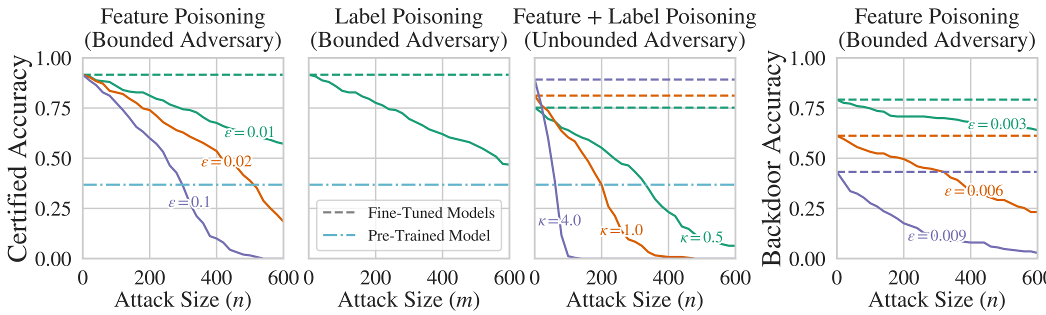


Figure 4: Certified accuracy (left) and backdoor accuracy (right) for a binary classifier fine-tuned on the Drusen class of OCTMNIST for an attack size up to 10% poisoned data per batch ($b = 6000, p = \infty, q = 0, \nu = 1$). Dashed lines show the nominal accuracy of each fine-tuned model.

Figure 5: Left: Fine-tuning PilotNet on unseen data with a bounded label poisoning attack ($q = \infty$). Right: Steering angle prediction bounds after fine-tuning ($m = 300, q = \infty, \nu = 0.01$).

cost of nominal model accuracy (Figure 4, center right). With label-only poisoning, the certificates are relatively stronger, as the training procedure requires approximately $m \geq 600$ poisoned samples for the prediction accuracy bound to reach the original pre-trained model's accuracy. The setting of $m = 600$ corresponds to 20% of the Drusen data per batch being mis-labeled as healthy.

Finally, we consider a backdoor attack setting where the ϵ used at training and inference times is the same. The model is highly susceptible to adversarial perturbations at inference time even without data poisoning, requiring only $\epsilon = 0.009$ to reduce the certified backdoor accuracy to $< 50\%$. As the strength of the (bounded) adversary increases, the accuracy that we are able to certify decreases. We note that tighter verification algorithms can be applied at inference time to obtain stronger guarantees.

Fine-Tuning PilotNet. Finally, we fine-tune a model that predicts steering angles for autonomous driving given an input image (Bojarski et al., 2016). The model contains convolutional layers of 24, 36, 48, and 64 filters, followed by fully connected layers of 100, 50, and 10 nodes. The fine-tuning setting is similar to above: first, we pre-train the model on videos 2–6 of the Udacity self-driving car dataset (`github.com/udacity/self-driving-car/tree/master`). We then fine-tune the dense layers on video 1 (challenging lighting conditions) assuming potential label poisoning.

Figure 5 shows the bounds on mean squared error for the video 1 data and visualizes how the bounds translate to the predicted steering angle. We again see that fine-tuning improves accuracy on the new data, but also that the MSE bounds deteriorate as the number of potentially poisoned samples increases (Figure 5, left). The rate of deterioration depends strongly on poisoning strength ν.

5 Conclusions

We proposed a mathematical framework for computing sound parameter-space bounds on the influence of a poisoning attack for gradient-based training. Our framework defines generic constraint sets to represent general poisoning attacks and propagates them through the forward and backward passes of model training. Based on the resulting parameter-space bounds, we provided rigorous bounds on the effects of various poisoning attacks. Finally, we demonstrated our proposed approach to be effective on tasks including autonomous driving and the classification of medical images.

REFERENCES

Steven Adams, Andrea Patane, Morteza Lahijanian, and Luca Laurenti. Bnn-dp: robustness certification of bayesian neural networks via dynamic programming. In *International Conference on Machine Learning*, pp. 133–151. PMLR, 2023.

Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2154–2156, 2018.

Battista Biggio, Konrad Rieck, Davide Ariu, Christian Wressnegger, Igino Corona, Giorgio Giacinto, and Fabio Roli. Poisoning behavioral malware clustering. In *Proceedings of the 2014 workshop on artificial intelligent and security workshop*, pp. 27–36, 2014.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

Elena Botoeva, Panagiotis Kouvaros, Jan Kronqvist, Alessio Lomuscio, and Ruth Misener. Efficient verification of relu-based neural networks via dependency analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3291–3299, 2020.

Rudy R Bunel, Ilker Turkaslan, Philip Torr, Pushmeet Kohli, and Pawan K Mudigonda. A unified view of piecewise linear neural network verification. *Advances in Neural Information Processing Systems*, 31, 2018.

Nicholas Carlini, Matthew Jagielski, Christopher A Choquette-Choo, Daniel Paleka, Will Pearce, Hyrum Anderson, Andreas Terzis, Kurt Thomas, and Florian Tramèr. Poisoning web-scale training datasets is practical. *arXiv preprint arXiv:2302.10149*, 2023.

Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE symposium on security and privacy (SP)*, pp. 3–18. IEEE, 2018.

Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

Xingshuo Han, Guowen Xu, Yuan Zhou, Xuehuan Yang, Jiwei Li, and Tianwei Zhang. Physical backdoor attacks to lane detection systems in autonomous driving. In *Proceedings of the 30th ACM International Conference on Multimedia*, pp. 2957–2968, 2022.

Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, et al. Pre-trained models: Past, present and future. *AI Open*, 2:225–250, 2021.

Georges Hebrail and Alice Berard. Individual Household Electric Power Consumption. UCI Machine Learning Repository, 2012. DOI: https://doi.org/10.24432/C58K54.

Sanghyun Hong, Varun Chandrasekaran, Yiğitcan Kaya, Tudor Dumitraş, and Nicolas Papernot. On the effectiveness of mitigating data poisoning attacks with gradient shaping. *arXiv preprint arXiv:2002.11497*, 2020.

Hanxun Huang, Xingjun Ma, Sarah Monazam Erfani, James Bailey, and Yisen Wang. Unlearnable examples: Making personal data unexploitable. *arXiv preprint arXiv:2101.04898*, 2021.

Joey Huchette, Gonzalo Muñoz, Thiago Serra, and Calvin Tsay. When deep learning meets polyhedral theory: A survey. *arXiv preprint arXiv:2305.00241*, 2023.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*, pp. 97–117. Springer, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alexander Levine and Soheil Feizi. Deep partition aggregation: Provable defense against general poisoning attacks. *arXiv preprint arXiv:2006.14768*, 2020.

Suyi Li, Yong Cheng, Wei Wang, Yang Liu, and Tianjian Chen. Learning to detect malicious clients for robust federated learning. *arXiv preprint arXiv:2002.00211*, 2020.

Yuhao Mao, Mark Niklas Müller, Marc Fischer, and Martin T. Vechev. Understanding certified training with interval bound propagation. *CoRR*, abs/2306.10426, 2023. doi: 10.48550/ARXIV.2306.10426. URL `https://doi.org/10.48550/arXiv.2306.10426`.

Mark Niklas Müller, Franziska Eckert, Marc Fischer, and Martin Vechev. Certified training: Small boxes are all you need. *arXiv preprint arXiv:2210.04871*, 2022.

Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pp. 27–38, 2017.

James Newsome, Brad Karp, and Dawn Song. Paragraph: Thwarting signature learning by training maliciously. In *Recent Advances in Intrusion Detection: 9th International Symposium, RAID 2006 Hamburg, Germany, September 20-22, 2006 Proceedings 9*, pp. 81–105. Springer, 2006.

Hong Diep Nguyen. Efficient implementation of interval matrix multiplication. In *Applied Parallel and Scientific Computing: 10th International Conference, PARA 2010, Reykjavík, Iceland, June 6-9, 2010, Revised Selected Papers, Part II 10*, pp. 179–188. Springer, 2012.

Keivan Rezaei, Kiarash Banihashem, Atoosa Chegini, and Soheil Feizi. Run-off election: Improved provable defense against data poisoning attacks. In *International Conference on Machine Learning*, pp. 29030–29050. PMLR, 2023.

Elan Rosenfeld, Ezra Winston, Pradeep Ravikumar, and Zico Kolter. Certified robustness to label-flipping attacks via randomized smoothing. In *International Conference on Machine Learning*, pp. 8230–8241. PMLR, 2020.

Siegfried M Rump. Fast and parallel interval arithmetic. *BIT Numerical Mathematics*, 39:534–554, 1999.

Aniruddha Saha, Akshayvarun Subramanya, and Hamed Pirsiavash. Hidden trigger backdoor attacks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 11957–11965, 2020.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL), jan 2019. doi: 10.1145/3290354.

Philip Sosnin and Calvin Tsay. Scaling mixed-integer programming for certification of neural network controllers using bounds tightening. *arXiv preprint arXiv:2403.17874*, 2024.

Jacob Steinhardt, Pang Wei W Koh, and Percy S Liang. Certified defenses for data poisoning attacks. *Advances in neural information processing systems*, 30, 2017.

Zhiyi Tian, Lei Cui, Jie Liang, and Shui Yu. A comprehensive survey on poisoning attacks and countermeasures in machine learning. *ACM Computing Surveys*, 55(8):1–35, 2022.

Calvin Tsay, Jan Kronqvist, Alexander Thebelt, and Ruth Misener. Partition-based formulations for mixed-integer optimization of trained relu neural networks. *Advances in neural information processing systems*, 34:3068–3080, 2021.

Wenxiao Wang, Alexander J Levine, and Soheil Feizi. Improved certified defenses against data poisoning with (deterministic) finite aggregation. In *International Conference on Machine Learning*, pp. 22769–22783. PMLR, 2022.

Maurice Weber, Xiaojun Xu, Bojan Karlaš, Ce Zhang, and Bo Li. Rab: Provable robustness against backdoor attacks. In *2023 IEEE Symposium on Security and Privacy (SP)*, pp. 1311–1328. IEEE, 2023.

Dennis Wei, Haoze Wu, Min Wu, Pin-Yu Chen, Clark Barrett, and Eitan Farchi. Convex bounds on the softmax function with applications to robustness verification, 2023.

Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. Feature-guided black-box safety testing of deep neural networks. In *Tools and Algorithms for the Construction and Analysis of Systems: 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I 24*, pp. 408–426. Springer, 2018.

Matthew Wicker, Luca Laurenti, Andrea Patane, and Marta Kwiatkowska. Probabilistic safety for bayesian neural networks. In *Conference on uncertainty in artificial intelligence*, pp. 1198–1207. PMLR, 2020.

Matthew Wicker, Juyeon Heo, Luca Costabello, and Adrian Weller. Robust explanation constraints for neural networks. *arXiv preprint arXiv:2212.08507*, 2022.

Matthew Wicker, Andrea Patane, Luca Laurenti, and Marta Kwiatkowska. Adversarial robustness certification for bayesian neural networks. *arXiv preprint arXiv:2306.13614*, 2023.

Chulin Xie, Yunhui Long, Pin-Yu Chen, and Bo Li. Uncovering the connection between differential privacy and certified robustness of federated learning against poisoning attacks. *arXiv preprint arXiv:2209.04030*, 2022.

Guolei Yang, Neil Zhenqiang Gong, and Ying Cai. Fake co-visitation injection attacks to recommender systems. In *NDSS*, 2017.

Jiancheng Yang, Rui Shi, and Bingbing Ni. MedMNIST classification decathlon: A lightweight AutoML benchmark for medical image analysis. In *IEEE 18th International Symposium on Biomedical Imaging (ISBI)*, pp. 191–195, 2021.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient Neural Network Robustness Certification with General Activation Functions. November 2018. doi: 10.48550/arXiv.1811.00866. URL `http://arxiv.org/abs/1811.00866`. arXiv:1811.00866 [cs, stat].

Chen Zhu, W Ronny Huang, Hengduo Li, Gavin Taylor, Christoph Studer, and Tom Goldstein. Transferable clean-label poisoning attacks on deep neural nets. In *International conference on machine learning*, pp. 7614–7623. PMLR, 2019.

A Related Works

Data Poisoning. Poisoning attacks have existed for nearly two decades and are a serious security concern (Biggio & Roli, 2018; Biggio et al., 2014; Newsome et al., 2006). In Muñoz-González et al. (2017) the authors formulate a general gradient-based attack that generates poisoned samples that corrupt model performance when introduced into the dataset (now termed, untargeted attack). Backdoor attacks manipulate a small proportion of the data such that, when a specific pattern is seen at test-time, the model returns a specific, erroneous prediction Chen et al. (2017); Gu et al. (2017); Han et al. (2022); Zhu et al. (2019). Popular defenses are attack specific, e.g., generating datasets using known attack strategies to classify and reject potentially poisoned inputs (Li et al., 2020). Alternative strategies apply noise or clipping to mitigate certain attacks (Hong et al., 2020).

Poisoning Defenses. General defenses to poisoning attacks seek to provide upper-bounds on the effectiveness of *any* attack strategy. In this area, Steinhardt et al. (2017) provide such upper-bounds for linear models trained with gradient descent. Rosenfeld et al. (2020) present a statistical upper-bound on the effectiveness of ℓ_2 perturbations on training labels for linear models using randomized smoothing. Xie et al. (2022) observe that differential privacy, which usually covers addition or removal of data points, can also provide statistical guarantees in some limited poisoning settings.

Certified Poisoning Robustness Relative to inference-time adversarial robustness (also referred to as evasion attacks), less attention has been devoted to provable guarantees against data poisoning adversaries. Existing methods for deterministic certification of robustness to poisoning adversaries involve design of a learning process with careful partitioning and ensembling such that the resulting model has poisoning robustness guarantees (Levine & Feizi, 2020; Wang et al., 2022; Rezaei et al., 2023). We refer to these methods as "aggregation" methods. In contrast, our approach is a method for analysis and certification of standard, unmodified machine learning algorithms. Aggregation approaches have been shown to offer strong guarantees against poisoning adversaries albeit at a substantial computational cost including: storing and training thousands of models on (potentially disjoint) subsets of the dataset and the requirement to evaluate each of the potentially thousands of models for each prediction; additionally, these methods require that one have potentially thousands of times more data than is necessary for training a single classifier. By designing algorithms to be robust to poisoning adversaries aggregation based approaches are able to scale to larger models than are considered in this work (Levine & Feizi, 2020; Wang et al., 2022; Rezaei et al., 2023). Yet, the computational cost of our approach, in the simplest case, is only four times that of standard training and inference and we do not require that one has access to enough data to train multiple well-performing models. Furthermore, our approach enables reasoning about backdoor attacks, where these partitioning approaches cannot. We finally highlight that the method presented in this paper is orthogonal/complementary to the partitioning approach, and thus future works may be able to combine the two effectively.

Certified Adversarial Robustness. Sound algorithms (i.e., no false positives) for upper-bounding the effectiveness of inference-time adversaries are well-studied for trained models (Gehr et al., 2018) and training models for robustness (Gowal et al., 2018; Müller et al., 2022). These approaches typically utilize ideas from formal methods (Katz et al., 2017; Wicker et al., 2018) or optimization (Botoeva et al., 2020; Bunel et al., 2018; Huchette et al., 2023). Most related to this work are strategies that consider intervals over both model inputs and parameters (Wicker et al., 2020), as well as some preliminary work on robust explanations that bound the input gradients of a model Wicker et al. (2022). Despite these methodological relationships, none of these methods directly apply to the general training setting studied here.

B Bounding the Descent Direction for an Unbounded Adversary

Theorem B.1 (Bounding the descent direction for an unbounded adversary)**.** *Given a nominal batch $\mathcal{B} = \left\{ \left(x^{(i)}, y^{(i)}\right) \right\}_{i=1}^{b}$ with batchsize b, a parameter set $\left[\theta^L, \theta^U\right]$, and a clipping level κ, the clipped*

SGD parameter update $\Delta\theta = \frac{1}{b}\sum_{\widetilde{\mathcal{B}}} \mathrm{Clip}_\kappa\left[\nabla_\theta \mathcal{L}\left(f^\theta\left(\tilde{x}^{(i)}\right), \tilde{y}^{(i)}\right)\right]$ *is bounded element-wise by*

$$\Delta\theta^L = \frac{1}{b}\left(\underset{b-n}{\mathrm{SEMin}}\left\{\delta_L^{(i)}\right\}_{i=1}^b - n\kappa\mathbf{1}_d\right), \quad \Delta\theta^U = \frac{1}{b}\left(\underset{b-n}{\mathrm{SEMax}}\left\{\delta_U^{(i)}\right\}_{i=1}^b + n\kappa\mathbf{1}_d\right) \tag{13}$$

for any poisoned batch $\widetilde{\mathcal{B}}$ *derived from* $\mathcal{B}$ *by substituting up to* n *data-points with poisoned data and any* $\theta \in [\theta^L, \theta^U]$. *The terms* $\delta_L^{(i)}, \delta_U^{(i)}$ *are sound bounds that account for the worst-case effect of additions/removals in any previous iterations. That is, they bound the gradient given any parameter* $\theta^\star \in [\theta^L, \theta^U]$ *in the reachable set, i.e. for all* $i = 1, \ldots, b$, *we have* $\delta_L^{(i)} \preceq \delta^{(i)} \preceq \delta_U^{(i)}$ *for any*

$$\delta^{(i)} \in \left\{\mathrm{Clip}_\kappa\left[\nabla_{\theta'}\mathcal{L}\left(f^{\theta'}(x^{(i)}), y^{(i)}\right)\right] \mid \theta' \in \left[\theta^L, \theta^U\right]\right\}. \tag{14}$$

The operations SEMax_a and SEMin_a correspond to taking the sum of the element-wise top/bottom-a elements over each index of the input vectors. Therefore, the update step in (13) corresponds to substituting the n elements with the *largest / smallest* gradients (by taking the sum of only the min / max $b-n$ gradients) with the *minimum / maximum* possible gradient updates ($-\kappa, \kappa$, respectively, due to the clipping operation). Since we wish to soundly over-approximate this operation for all parameters, we perform this bounding operation independently over each index of the parameter vector. This is certainly a loose approximation, as the n points that maximize the gradient at a particular index will likely not maximize the gradient of other indices. Note that without clipping, the min / max effect of adding arbitrary data points into the training data is unbounded and we cannot compute any guarantees.

C Interval Matrix Arithmetic

In this appendix, we provide a basic introduction to interval matrix arithmetic, which forms the basic building block of our CROWN-style bounds. We denote intervals over matrices as $\boldsymbol{A} := [A_L, A_U] \subseteq \mathbb{R}^{n\times m}$ such that for all $A \in \boldsymbol{A}$, $A_L \leq A \leq A_U$.

Definition 2 (Interval Matrix Arithmetic)**.** *Let* $\boldsymbol{A} = [A_L, A_U]$ *and* $\boldsymbol{B} = [B_L, B_U]$ *be intervals over matrices. Let* $\oplus$, $\otimes$, $\odot$ *represent interval matrix addition, matrix multiplication and elementwise multiplication, such that*

$$\begin{aligned} A + B &\in [\boldsymbol{A} \oplus \boldsymbol{B}] \quad \forall A \in \boldsymbol{A}, B \in \boldsymbol{B}, \\ A \times B &\in [\boldsymbol{A} \otimes \boldsymbol{B}] \quad \forall A \in \boldsymbol{A}, B \in \boldsymbol{B}, \\ A \circ B &\in [\boldsymbol{A} \odot \boldsymbol{B}] \quad \forall A \in \boldsymbol{A}, B \in \boldsymbol{B}. \end{aligned}$$

These operations can be computed using standard interval arithmetic techniques in at most $4\times$ the cost of a standard matrix operation. For example, interval matrix multiplication can be computed using the following procedure.

Definition 3 (Interval Matrix Multiplication)**.** *Given element-wise intervals over matrices* $[A_L, A_U]$ *where* $A_L, A_U \in \mathbb{R}^{n\times m}$ *and* $[B_L, B_U]$ *where* $B_L, B_U \in \mathbb{R}^{m\times k}$, *define the matrices* $A_\mu = (A_U + A_L)/2$ *and* $A_r = (A_U - A_L)/2$. *Allow* B_μ *and* B_r *to be defined analogously, then computing using Rump's algorithm (Rump, 1999),*

$$\begin{aligned} C_L &= A_\mu B_\mu - |A_\mu| B_r - A_r|B_\mu| - A_r B_r \\ C_U &= A_\mu B_\mu + |A_\mu| B_r + A_r|B_\mu| + A_r B_r, \end{aligned}$$

we have that $C_{Li,j} \leq [A'B']_{i,j} \leq C_{Ui,j}\ \forall A' \in [A_L, A_U],\ B' \in [B_L, B_U]$. *Nguyen (2012) showed that the above bounds have a worst-case overestimation factor of 1.5.*

Interval arithmetic is commonly applied as a basic verification or adversarial training technique by propagating intervals through the intermediate layers of a neural network (Gowal et al., 2018).

D CROWN with Interval Parameters

In this section, we present our full extension of the CROWN algorithm (Zhang et al., 2018) for neural networks with interval parameters. The standard CROWN algorithm bounds the outputs of the

m-th layer of a neural network by back-propagating linear bounds over each intermediate activation function to the input layer. We extend this framework to interval parameters, where the weights and biases involved in these linear relaxations are themselves intervals. We note that linear bound propagation with interval parameters has been studied previously in the context of floating-point sound certification (Singh et al., 2019). Here, we present an explicit instantiation of the CROWN algorithm for interval parameters, which we recall from Section 3.3.

Proposition 1 (Explicit output bounds of neural network f with interval parameters)**.** *Given an m-layer neural network function $f : \mathbb{R}^{n_{in}} \rightarrow \mathbb{R}^{n_{out}}$ whose unknown parameters lie in the intervals $b^{(k)} \in \boldsymbol{b}^{(k)}$ and $W^{(k)} \in \boldsymbol{W}^{(k)}$ for $k = 1, \ldots, m$, there exist two explicit functions*

$$f_j^L\left(x, \Omega^{(0:m)}, \Theta^{(1:m)}, b^{(1:m)}\right) = \Omega_{j,:}^{(0)} x + \sum_{k=1}^{m} \Omega_{j,:}^{(k)} \left(b^{(k)} + \Theta_{:,j}^{(k)}\right) \tag{15}$$

$$f_j^U\left(x, \Lambda^{(0:m)}, \Delta^{(1:m)}, b^{(1:m)}\right) = \Lambda_{j,:}^{(0)} x + \sum_{k=1}^{m} \Lambda_{j,:}^{(k)} \left(b^{(k)} + \Delta_{:,j}^{(k)}\right) \tag{16}$$

such that $\forall x \in \boldsymbol{x}$

$$f_j(x) \geq \min\left\{f_j^L\left(x, \Omega^{(0:m)}, \Theta^{(1:m)}, b^{(1:m)}\right) \mid \Omega^{(k)} \in \boldsymbol{\Omega}^{(k)}, b^k \in \boldsymbol{b}^{(k)}\right\}$$
$$f_j(x) \leq \max\left\{f_j^U\left(x, \Lambda^{(0:m)}, \Delta^{(1:m)}, b^{(1:m)}\right) \mid \Lambda^{(k)} \in \boldsymbol{\Lambda}^{(k)}, b^k \in \boldsymbol{b}^{(k)}\right\}$$

where $\boldsymbol{x}$ is a closed input domain and $\Lambda^{(0:m)}, \Delta^{(1:m)}, \Omega^{(0:m)}, \Theta^{(1:m)}$ are the equivalent weights and biases of the upper and lower linear bounds, respectively. The bias terms $\Delta^{(1:m)}, \Theta^{(1:m)}$ are explicitly computed based on the linear bounds on the activation functions. The weights $\Lambda^{(0:m)}, \Omega^{(0:m)}$ lie in intervals $\boldsymbol{\Lambda}^{(0:m)}, \boldsymbol{\Omega}^{(0:m)}$ which are computed in an analogous way to standard (non-interval) CROWN.

Computing Equivalent Weights and Biases. Our instantiation of the CROWN algorithm in Proposition 1 relies on the computation of the equivalent bias terms $\Delta^{(1:m)}, \Theta^{(1:m)}$ and interval enclosures over the equivalent weights $\Omega^{(0:m)}, \Lambda^{(0:m)}$. This proceeds similarly to the standard CROWN algorithm but now accounting for intervals over the parameters $b^{(1:m)}, W^{(1:m)}$ of the network. All interval operations are as described in Appendix C.

The standard CROWN algorithm bounds the outputs of the m-th layer of a neural network by back-propagating linear bounds over each intermediate activation function to the input layer. In the case of interval parameters, the sign of a particular weight may be ambiguous (when the interval spans zero), making it impossible to determine which linear bound to back-propagate. In such cases, we propagate a concrete bound for that neuron instead of its linear bounds.

When bounding the m-th layer of a neural network, we assume that we have pre-activation bounds $\hat{z}^{(k)} \in \left[l^{(k)}, u^{(k)}\right]$ on all previous layers on the network. Given such bounds, it is possible to form linear bounds on any non-linear activation function in the network. For the r-th neuron in k-th layer with activation function $\sigma(z)$, we define two linear functions

$$h_{L,r}^{(k)}(z) = \alpha_{L,r}^{(k)}\left(z + \beta_{L,r}^{(k)}\right), \quad h_{U,r}^{(k)}(z) = \alpha_{U,r}^{(k)}\left(z + \beta_{U,r}^{(k)}\right)$$

such that $h_{L,r}^{(k)}(z) \leq \sigma(z) \leq h_{U,r}^{(k)}(z)\ \forall z \in \left[l_r^{(k)}, u_r^{(k)}\right]$. The coefficients $\alpha_{U,r}^{(k)}, \alpha_{L,r}^{(k)}, \beta_{U,r}^{(k)}, \beta_{L,r}^{(k)} \in \mathbb{R}$ are readily computed for many common activation functions (Zhang et al., 2018).

Given the pre-activation and activation function bounds, the interval enclosures over the weights $\Omega^{(0:m)}, \Lambda^{(0:m)}$ are computed via a back-propagation procedure. The back-propagation is initialised with $\mathbf{\Omega}^{(m)} = \mathbf{\Lambda}^{(m)} = [I^{n_m}, I^{n_m}]$ and proceeds as follows:

$$\mathbf{\Lambda}^{(k-1)} = \left(\mathbf{\Lambda}^{(k)} \otimes \mathbf{W}^{(k)}\right) \odot \lambda^{(k-1)}, \quad \lambda_{j,i}^{(k)} = \begin{cases} \alpha_{U,i}^{(k)} & \text{if } k \neq 0,\ 0 \leq \left[\mathbf{\Lambda}^{(k+1)} \otimes \mathbf{W}^{(k+1)}\right]_{j,i} \\ \alpha_{L,i}^{(k)} & \text{if } k \neq 0,\ 0 \geq \left[\mathbf{\Lambda}^{(k+1)} \otimes \mathbf{W}^{(k+1)}\right]_{j,i} \\ 0 & \text{if } k \neq 0,\ 0 \in \left[\mathbf{\Lambda}^{(k+1)} \otimes \mathbf{W}^{(k+1)}\right]_{j,i} \\ 1 & \text{if } k = 0. \end{cases}$$

$$\mathbf{\Omega}^{(k-1)} = \left(\mathbf{\Omega}^{(k)} \otimes \mathbf{W}^{(k)}\right) \odot \omega^{(k-1)}, \quad \omega_{j,i}^{(k)} = \begin{cases} \alpha_{L,i}^{(k)} & \text{if } k \neq 0,\ 0 \leq \left[\mathbf{\Omega}^{(k+1)} \otimes \mathbf{W}^{(k+1)}\right]_{j,i} \\ \alpha_{U,i}^{(k)} & \text{if } k \neq 0,\ 0 \geq \left[\mathbf{\Omega}^{(k+1)} \otimes \mathbf{W}^{(k+1)}\right]_{j,i} \\ 0 & \text{if } k \neq 0,\ 0 \in \left[\mathbf{\Omega}^{(k+1)} \otimes \mathbf{W}^{(k+1)}\right]_{j,i} \\ 1 & \text{if } k = 0. \end{cases}$$

where we use $0 \leq [\cdot]$ and $0 \geq [\cdot]$ to denote that an interval is strictly positive or negative, respectively.

Finally, the bias terms $\Delta^{(k)}, \Theta^{(k)}$ for all $k < m$ can be computed as

$$\Delta_{i,j}^{(k)} = \begin{cases} \beta_{U,i}^{(k)} & \text{if } 0 \leq \left[\mathbf{\Lambda}^{(k+1)} \otimes \mathbf{W}^{(k+1)}\right]_{j,i} \\ \beta_{L,i}^{(k)} & \text{if } 0 \geq \left[\mathbf{\Lambda}^{(k+1)} \otimes \mathbf{W}^{(k+1)}\right]_{j,i} \\ u^{(k)} & \text{if } 0 \in \left[\mathbf{\Lambda}^{(k+1)} \otimes \mathbf{W}^{(k+1)}\right]_{j,i} \end{cases}, \Theta_{i,j}^{(k)} = \begin{cases} \beta_{L,i}^{(k)} & \text{if } 0 \leq \left[\mathbf{\Omega}^{(k+1)} \otimes \mathbf{W}^{(k+1)}\right]_{j,i} \\ \beta_{U,i}^{(k)} & \text{if } 0 \geq \left[\mathbf{\Omega}^{(k+1)} \otimes \mathbf{W}^{(k+1)}\right]_{j,i} \\ l^{(k)} & \text{if } 0 \in \left[\mathbf{\Omega}^{(k+1)} \otimes \mathbf{W}^{(k+1)}\right]_{j,i} \end{cases}$$

with the m-th bias terms given by $\Theta_{i,j}^{(m)} = \Delta_{i,j}^{(m)} = 0$.

Closed-Form Global Bounds. Given the two functions $f_j^L(\cdot), f_j^U(\cdot)$ as defined above and intervals over all the relevant variables, we can compute the following closed-form global bounds:

$$\gamma_j^L = \min \left\{ \mathbf{\Omega}_{j,:}^{(0)} \otimes \boldsymbol{x} \oplus \sum_{k=1}^{m} \mathbf{\Omega}_{j,:}^{(k)} \otimes \left[\boldsymbol{b}^{(k)} \oplus \Theta_{:,j}^{(k)} \right] \right\}$$

$$\gamma_j^U = \max \left\{ \mathbf{\Lambda}_{j,:}^{(0)} \otimes \boldsymbol{x} \oplus \sum_{k=1}^{m} \mathbf{\Lambda}_{j,:}^{(k)} \otimes \left[\boldsymbol{b}^{(k)} \oplus \Delta_{:,j}^{(k)} \right] \right\}$$

where $\min$ / $\max$ are performed element-wise and return the lower / upper bounds of each interval enclosure. Then, we have $\gamma_j^L \leq f_j(x) \leq \gamma_j^U$ for all $x \in \boldsymbol{x}$, $b^{(k)} \in \boldsymbol{b}^{(k)}$ and $W^{(k)} \in \boldsymbol{W}^{(k)}$, which suffices to bound the output of the neural network as required to further bound the gradient of the network.

E Bounds on Loss Function Gradients

In this section we present the computation of bounds on the first partial derivative of the loss function required for Algorithm 1. In particular, we consider bounding the following optimization problem via interval arithmetic:

$$\min \& \max \left\{ \partial \mathcal{L}\left(y^{\star}, y'\right) / \partial y^{\star} \mid y^{\star} \in \left[y^L, y^U\right], \|y' - y^t\|_q \leq \nu \right\}$$

for some loss function $\mathcal{L}$ where $[y^L, y^U]$ are bounds on the logits of the model (obtained via the bound-propagation procedure), y^t is the true label and y' is the poisoned label.

Mean Squared Error Loss Taking $\mathcal{L}\left(y^{\star}, y'\right) = \|y^{\star} - y'\|_2^2$ to be the squared error and considering the $q = \infty$ norm, the required bounds are given by:

$$\partial l^L = 2\left(y^L - y^t - \nu\right)$$

$$\partial l^U = 2\left(y^U - y^t + \nu\right)$$

The loss itself can be upper-bounded by $l^U = \max\{(y^L - y^t)^2, (y^U - y^t)^2\}$ and lower bounded by

$$l^L = \begin{cases} 0 & \text{if } y^t \in [y^L, y^U] \\ \min\{(y^L - y^t)^2, (y^U - y^t)^2\} & \text{otherwise} \end{cases} \tag{17}$$

Cross Entropy Loss To bound the gradient of the cross entropy loss, we first bound the output probabilities $p_i = [\sum_j \exp\left(y^\star_j - y^\star_i\right)]^{-1}$ obtained by passing the logits through the softmax function:

$$p^L_i = \left[\sum_j \exp\left(y^U_j - y^L_i\right)\right]^{-1}, \quad p^U_i = \left[\sum_j \exp\left(y^L_j - y^U_i\right)\right]^{-1}$$

The categorical cross entropy loss and its first partial derivative are given by

$$\mathcal{L}\left(y^\star, y'\right) = -\sum_i y^t_i \log p_i, \quad \frac{\partial \mathcal{L}\left(y^\star, y'\right)}{\partial y^\star} = p - y^t$$

where y^t is a one-hot encoding of the true label. Considering label flipping attacks ($q = 0, \nu = 1$), we can bound the partial derivative by

$$\left[\partial l^L\right]_i = p^L_i - 1, \quad \left[\partial l^U\right]_i = p^U_i - 0$$

In the case of targeted label flipping attacks (e.g. only applying label flipping attacks to / from specific classes), stronger bounds can be obtained by considering the $0 - 1$ bounds only on the indices y^t_i affected by the attack. The cross entropy loss itself is bounded by $l^L = -\sum_i y^t_i \log p^U_i, l^U = -\sum_i y^t_i \log p^L_i$.

F Backwards Pass Bounds

Given bounds on the forward pass of the neural network, we now turn to bounding the objective of our original problem (10), replacing the forward pass constraints with their bounds computed using our CROWN algorithm,

$$\min \& \max \left\{ \frac{\partial}{\partial \theta^\star} \left(\mathcal{L}\left(\hat{z}^{(K)}, y^\star\right)\right) \mid \theta^\star \in \left[\theta^L, \theta^U\right], \hat{z}^{(k)} \in \left[\hat{z}^{(k)}_L, \hat{z}^{(k)}_U\right], \|y - y^\star\|_q \leq \nu \right\}. \quad (18)$$

We extend the interval arithmetic based approach of Wicker et al. (2022), which bounds derivatives of the form $\partial \mathcal{L} / \partial z^{(k)}$, to additionally compute bounds on the derivatives w.r.t. the parameters. First, we back-propagate intervals over $y^\star$ (the label) and $\hat{z}^{(K)}$ (the logits) to compute an interval over $\boldsymbol{\partial \mathcal{L} / \partial \hat{z}^{(K)}}$, the gradient of the loss w.r.t. the logits of the network. The procedure for computing this interval is described in Appendix E for a selection of loss functions. We then use interval bound propagation to back-propagate this interval through the network to compute intervals over all gradients:

$$\frac{\boldsymbol{\partial \mathcal{L}}}{\boldsymbol{\partial z^{(k-1)}}} = \left(\boldsymbol{W^{(k)}}\right)^\top \otimes \frac{\boldsymbol{\partial \mathcal{L}}}{\boldsymbol{\partial \hat{z}^{(k)}}}, \quad \frac{\boldsymbol{\partial \mathcal{L}}}{\boldsymbol{\partial \hat{z}^{(k)}}} = \left[H\left(\hat{z}^{(k)}_L\right), H\left(\hat{z}^{(k)}_U\right)\right] \odot \frac{\boldsymbol{\partial \mathcal{L}}}{\boldsymbol{\partial z^{(k)}}}$$
$$\frac{\boldsymbol{\partial \mathcal{L}}}{\boldsymbol{\partial W^{(k)}}} = \frac{\boldsymbol{\partial \mathcal{L}}}{\boldsymbol{\partial \hat{z}^{(k)}}} \otimes \left[\left(z^{(k-1)}_L\right)^\top, \left(z^{(k-1)}_U\right)^\top\right], \quad \frac{\boldsymbol{\partial \mathcal{L}}}{\boldsymbol{\partial b^{(k)}}} = \frac{\boldsymbol{\partial \mathcal{L}}}{\boldsymbol{\partial \hat{z}^{(k)}}}$$

where $H(\cdot)$ is the Heaviside function, and $\circ$ is the element-wise product. The resulting intervals are valid bounds the objective of our original problem (10) and its relaxation (18). That is, the gradients of the network lie within these intervals for all $W^{(k)} \in \boldsymbol{W^{(k)}}$, $b^{(k)} \in \boldsymbol{b^{(k)}}$, $\|x - x^\star\|_p \leq \epsilon$, and $\|y - y^\star\|_q \leq \nu$.

G Computing Bounds on Poisoning Objectives

In this section, we describe a procedure for computing bounds on each of the poisoning adversary's objectives. Given any objective J and a test set $\{(x^{(i)}, y^{(i)})\}_{i=1}^k$, the poisoning adversary's objective can be relaxed by taking each test sample independently, i.e.

$$\max_{\theta^\star \in [\theta^L, \theta^U]} \frac{1}{k} \sum_{i=1}^{k} J\left(\theta^\star, x^{(i)}, y^{(i)}\right) \leq \frac{1}{k} \sum_{i=1}^{k} \max_{\theta^\star \in [\theta^L, \theta^U]} J\left(\theta^\star, x^{(i)}, y^{(i)}\right). \quad (19)$$

Thus to bound the original poisoning objectives (3), (4), and (5), it suffices to compute bounds on the required quantity for each test sample independently.

Denial of Service. Computing bounds on the optimization problem

$$\max_{\theta^\star \in [\theta^L, \theta^U]} \mathcal{L}\big(f^{\theta^\star}(x^{(i)}), y^{(i)}\big) \tag{20}$$

for the cross-entropy and mean-squared-error losses is described in Section E.

Certified Prediction and Backdoor Robustness. Computing an bounds on

$$\max_{\theta^\star \in [\theta^L, \theta^U]} \mathbb{1}\big(f^{\theta^\star}(x^{(i)}) \notin S\big) \tag{21}$$

corresponds to checking if $f^{\theta^\star}(x^{(i)})$ lies within the safe set S for all $\theta^\star \in [\theta^L, \theta^U]$. As before, we first compute bounds f^L, f^U on $f^{\theta^\star}(x^{(i)})$ using our CROWN-based bounds. Given these bounds and assuming a multi-class classification setting, the predictions *not* reachable by any model within $[\theta^L, \theta^U]$ are those whose logit upper bounds lie below the logit lower bound of any other class. That is, the set of possible predictions S' is given by

$$S' = \left\{ i \text{ s.t. } \nexists j : f_i^U \leq f_j^L \right\}. \tag{22}$$

If $S' \subseteq S$, then $\max_{\theta^\star} \mathbb{1}\big(f^{\theta^\star}(x^{(i)}) \notin S\big) = 0$.

Backdoor attack robustness is computed in an analogous way, with the only difference being the logit bounds f^L, f^U being computed over all $x \in V(x)$ and $\theta^\star \in [\theta^L, \theta^U]$. This case is also computed via our CROWN-based bound propagation.

H Comparison with Empirical Attacks

In this section, we compare the tightness of our bounds with simple heuristic poisoning attacks for both the UCI-houseelectric and OCT-MNIST datasets.

H.1 Visualising Attacks in Parameter Space (UCI-houseelectric)

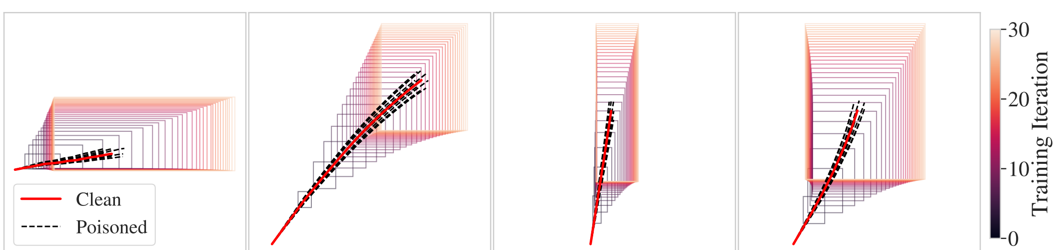


Figure 6: Training trajectory for selected parameters under parameter-targeted feature poisoning with an adversary of $\epsilon = 0.02, n = 2000, p = \infty$. The coloured boxes show the bounds $[\theta_L, \theta_U]$ obtained at each training iteration using AGT.

First, we investigate the tightness of our bounds in parameter space via a feature poisoning attack. The attack's objective is to maximize a given scalar function of the parameters, which we label $f^{\text{targ}}(\theta)$. We then take the following poisoning procedure at each training iteration:

1. Randomly sample a subset of n samples from the current training batch.
2. For each selected sample $x^{(i)}$, compute a poison $v^{(i)}$ such that $x^{(i)} + v^{(i)}$ maximizes the gradient $\partial f^{\text{targ}} / \partial x$ subject to $\|v^{(i)}\|_p \leq \epsilon$.
3. Add the noise to each of the n sampled points to produce the poisoned dataset.

The noise in step 2 is obtained via projected gradient descent (PGD). To visualise the effect of our attack in parameter space, we plot the trajectory taken by two randomly selected parameters θ_i, θ_j from the network. We then run our poisoning attack on a collection of poisoning objectives f^{targ}, such as $\theta_i + \theta_j, \theta_i, -\theta_j$, etc. The effect of our poisoning attack is to perturb the training trajectory in the direction to maximize the given objective.

Figure 6 shows the result of this poisoning procedure for a random selection of parameter training trajectories. We can see that the poisoned trajectories (in black) lie close the clean poisoned trajectory, while our bounds represent an over-approximation of all the possible training trajectories.

H.2 Feature-Space Collision Attack (UCI-houseelectric)

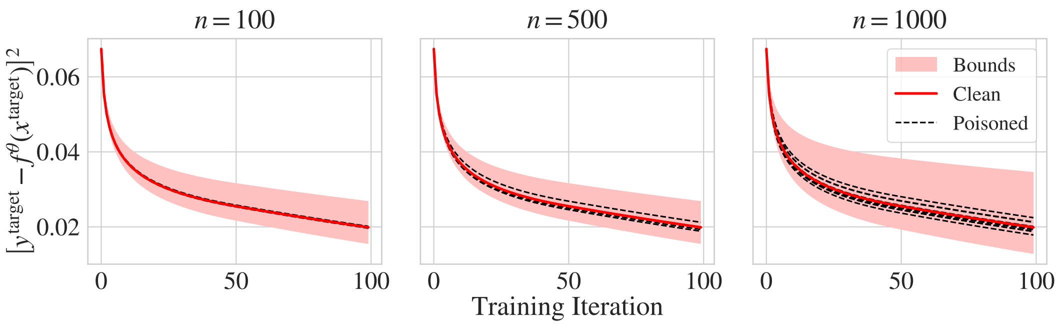


Figure 7: Mean squared error on the target point $(x^{\text{target}}, y^{\text{target}})$ in the UCI-houseelectric dataset. Black lines show loss trajectories under the randomized feature-collision poisoning attack.

We now consider an unbounded attack setting where the adversary's goal is to prevent the model from learning a particular training example $(x^{\text{target}}, y^{\text{target}})$. We again consider a simple randomized attack setting, where the adversary first selects a subset of n samples from each training batch. The adversary then replaces the features of each of the n samples with x^{target}, and assigns each one a randomly generated label. In this way, the adversary aims to obscure the true target label and prevent the model from learning the pair $(x^{\text{target}}, y^{\text{target}})$.

Figure 7 shows the loss of the model on the target point at each training iteration. To investigate the tightness of our loss lower bound, we also consider the case where the adversary replaces all of the n sampled instances from the batch with the true $(x^{\text{target}}, y^{\text{target}})$, thus over-representing the sample within the batch and causing the model to fit the target point faster. The bounds (in red) are obtained from AGT with an unbounded adversary ($\kappa = 0.05$). We can see that although our bounds are not tight to any of the attacks considered, they remain sound for all the poisoned training trajectories.

H.3 Randomized Label Flipping Attack (OCT-MNIST)

Here, we present the results of a label-flipping attack conducted on the OCT-MNIST dataset. Following the approach described in Section 4, we begin by pre-training a binary classification model to distinguish between two diseased classes and a healthy class. Next, we fine-tune the model's final dense layers on the 'Drusen' class, which we assume to be potentially compromised, using a training set composed of 50% clean data and 50% Drusen data, with 3000 samples from each category in each batch. Given that the Drusen class is a minority, we simulate a scenario where a random subset of the Drusen data is incorrectly labeled as the 'healthy' class. Figure 8 displays the model's accuracy when trained on the poisoned dataset. We can see that training on the mis-labelled data results in a significant decrease in model accuracy, though the poisoned accuracy remains within the bounds of certified by AGT.

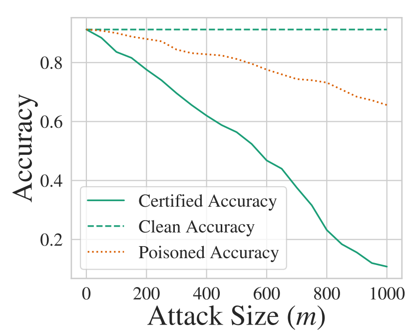


Figure 8: Accuracy on the OCT-MNIST dataset under a random label flipping attack on the Drusen class.

I Proofs

I.1 Proof of Theorem 3.1 (Bounding Adversary Goals via Parameter Space Bounds)

We begin the proof by writing out the form of the function we wish to optimize, J, for each attack setting considered. Below the right hand side of the inequality is taken to be the function J, and each inequality is the statement we would like to prove.

For denial of service our bound becomes:

$$\max_{\mathcal{D}' \in \mathcal{T}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}\big(f^{M(f,\theta',\mathcal{D}')}(x^{(i)}), y^{(i)}\big) \leq \max_{\theta^\star \in [\theta^L, \theta^U]} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}\big(f^{\theta^\star}(x^{(i)}), y^{(i)}\big)$$

For certified prediction poisoning robustness our bound becomes:

$$\max_{\mathcal{D}' \in \mathcal{T}} \frac{1}{k} \sum_{i=1}^{k} \mathbb{1}\big(f^{M(f,\theta',\mathcal{D}')}(x^{(i)}) \notin S\big) \leq \max_{\theta^\star \in [\theta^L, \theta^U]} \frac{1}{k} \sum_{i=1}^{k} \mathbb{1}\big(f^{\theta^\star}(x^{(i)}) \notin S\big)$$

And for backdoor attacks our bound becomes:

$$\max_{\mathcal{D}' \in \mathcal{T}} \frac{1}{k} \sum_{i=1}^{k} \mathbb{1}\big(\exists x^\star \in V(x^{(i)}) \; s.t. \; f^{M(f,\theta',\mathcal{D}')}(x^\star) \notin S\big)$$
$$\leq \max_{\theta^\star \in [\theta^L, \theta^U]} \frac{1}{k} \sum_{i=1}^{k} \mathbb{1}\big(\exists x^\star \in V(x^{(i)}) \; s.t. \; f^{\theta^\star}(x^\star) \notin S\big)$$

Proof: Without loss of generality, take the function we wish to optimize to be denoted simply by J. By definition, there exists a parameter, $\theta^\circ = M(f, \theta', \mathcal{D}')$ resulting from a particular dataset $\mathcal{D}' \in \mathcal{T}(\mathcal{D})$ such that θ° provides a (potentially non-unique) optimal solution to the optimization problem we wish to bound, i.e., the left hand side of the inequalities above. Given a valid parameter space bound $[\theta^L, \theta^U]$ satisfying Equation 6, we have that necessarily, $\theta^\circ \in [\theta^L, \theta^U]$. Therefore, the result of optimizing over $[\theta^L, \theta^U]$ can provide at a minimum the bound realized by θ°; however, due to approximation, this bound might not be tight, so optimizing over $[\theta^L, \theta^U]$ provides an upper-bound, thus proving the inequalities above. □

I.2 Proof of Theorem 3.2 (Algorithm Correctness)

Here we provide a proof of correctness for our algorithm (i.e., proof of Theorem 3.2) as well as a detailed discussion of the operations therein.

First, we recall the definition of valid parameter space bounds (Equation 6 in the main text):

$$\theta_i^L \leq \min_{\mathcal{D}' \in \mathcal{T}(\mathcal{D})} M(f, \theta', \mathcal{D}')_i \leq M(f, \theta', \mathcal{D})_i \leq \max_{\mathcal{D}' \in \mathcal{T}(\mathcal{D})} M(f, \theta', \mathcal{D}')_i \leq \theta_i^U$$

As well as the iterative equations for stochastic gradient descent:

$$\theta \leftarrow \theta - \alpha \Delta\theta, \qquad \Delta\theta \leftarrow \frac{1}{|\mathcal{B}|} \sum_{(x,y) \in \mathcal{B}} \nabla_\theta \mathcal{L}\left(f^\theta(x), y\right)$$

For ease of notation, we assume a fixed data ordering (one may always take the element-wise maximums/minimums over the entire dataset rather than each batch to relax this assumption).

Now, we proceed to prove by induction that Algorithm 1 maintains valid parameter space bounds on each step of gradient descent. We start with the base case of $\theta^L = \theta^U = \theta'$ according to line 1, which are valid parameter-space bounds. Our inductive hypothesis is that, given valid parameter space bounds satisfying Definition 1, each iteration of Algorithm 1 (lines 4–8) produces a new θ^L and θ^U that satisfy also Definition 1.

First, we observe that lines 4–5 simply compute the normal forward pass. Second, we note that lines 6–7 compute valid bounds on the descent direction for all possible poisoning attacks within $\mathcal{T}(\mathcal{D})$. In other words, the inequality $\Delta\theta^L \leq \Delta\theta \leq \Delta\theta^U$ holds element-wise for any possible batch $\tilde{\mathcal{B}} \in \mathcal{T}(\mathcal{D})$. Combining this largest and smallest possible update with the smallest and largest previous parameters yields the following bounds:

$$\theta^L - \alpha\Delta\theta^U \leq \theta - \alpha\Delta\theta \leq \theta^U - \alpha\Delta\theta^L$$

which, by definition, constitute valid parameter-space bounds and, given that these bounds are exactly those in Algorithm 1, we have that Algorithm 1 provides valid parameter space bounds as desired. □

I.3 Proof of Theorem B.1 (Descent Direction Bound for Unbounded Adversaries)

The nominal clipped descent direction for a parameter θ is the averaged, clipped gradient over a training batch $\mathcal{B}$, defined as

$$\Delta\theta = \frac{1}{b}\sum_{i=1}^{b} \text{Clip}_\kappa\left[\delta^{(i)}\right]$$

where each gradient term is given by $\delta^{(i)} = \nabla_\theta \mathcal{L}\left(f^\theta\left(x^{(i)}\right), y^{(i)}\right)$. Our goal is to bound this descent direction for the case when (up to) n points are removed or added to the training data, for any $\theta \in [\theta_L, \theta_U]$. We begin by bounding the descent direction for a fixed, scalar θ, then generalize to all $\theta \in [\theta_L, \theta_U]$ and to the multi-dimensional case (i.e., multiple parameters). We present only the upper bounds here; analogous results apply for lower bounds.

Bounding the descent direction for a fixed, scalar θ. Consider the effect of removing up to n data points from batch $\mathcal{B}$. Without loss of generality, assume the gradient terms are sorted in descending order, i.e., $\delta^{(1)} \geq \delta^{(2)} \geq \cdots \geq \delta^{(b)}$. Then, the average clipped gradient over all points can be bounded above by the average over the largest $b - n$ terms:

$$\Delta\theta = \frac{1}{b}\sum_{i=1}^{b} \text{Clip}_\kappa\left[\delta^{(i)}\right] \leq \frac{1}{b-n}\sum_{i=1}^{b-n} \text{Clip}_\kappa\left[\delta^{(i)}\right]$$

This bound corresponds to removing the n points with the smallest gradients.

Next, consider adding n arbitrary points to the training batch. Since each added point contributes at most κ due to clipping, the descent direction with up to n removals and n additions is bounded by

$$\frac{1}{b}\sum_{i=1}^{b} \text{Clip}_\kappa\left[\delta^{(i)}\right] \leq \frac{1}{b-n}\sum_{i=1}^{b-n} \text{Clip}_\kappa\left[\delta^{(i)}\right] \leq \frac{1}{b}\left(n\kappa + \sum_{i=1}^{b} \text{Clip}_\kappa\left[\delta^{(i)}\right]\right)$$

where the bound now accounts for replacing the n smallest gradient terms with the maximum possible value of κ from the added samples.

Bounding the effect of a variable parameter interval. We extend this bound to any $\theta \in [\theta_L, \theta_U]$. Assume the existence of upper bounds $\delta_U^{(i)}$ on the clipped gradients for each data point over the interval, such that

$$\delta_U^{(i)} \geq \text{Clip}_\kappa\left[\nabla_{\theta'}\mathcal{L}\left(f^{\theta'}(x^{(i)}), y^{(i)}\right)\right] \quad \forall\, \theta' \in [\theta_L, \theta_U].$$

Then, using these upper bounds, we further bound $\Delta\theta$ as

$$\Delta\theta \leq \frac{1}{b}\left(n\kappa + \sum_{i=1}^{b} \text{Clip}_\kappa\left[\delta_U^{(i)}\right]\right)$$

where, as before, we assume $\delta_U^{(i)}$ are indexed in descending order.

Extending to the multi-dimensional case. To generalize to the multi-dimensional case, we apply the above bound component-wise. Since gradients are not necessarily ordered for each parameter component, we introduce the SEMax_n operator, which selects and sums the largest n terms at each index. This yields the following bound on the descent direction:

$$\Delta\theta \leq \frac{1}{b}\left(\underset{b-n}{\text{SEMax}}\left\{\delta_U^{(i)}\right\}_{i=1}^{b} + n\kappa\mathbf{1}_d\right)$$

which holds for any $\theta \in [\theta_L, \theta_U]$ and up to n removed and replaced points. □

We have established the upper bound on the descent direction. The corresponding lower bound can be derived by reversing the inequalities and substituting SEMax with the analagous minimization operator, SEMin.

I.4 Proof of Theorem 3.3 (Descent Direction Bound for Bounded Adversaries)

The nominal descent direction for a parameter θ is the averaged gradient over a training batch $\mathcal{B}$, defined as

$$\Delta\theta = \frac{1}{b}\sum_{i=1}^{b} \delta^{(i)}$$

where each gradient term is given by $\delta^{(i)} = \nabla_\theta \mathcal{L}\left(f^\theta\left(x^{(i)}\right), y^{(i)}\right)$. Our goal is to upper bound this descent direction when up to n points are poisoned in the feature space and up to m points are poisoned in the label space. The bound is additionally computed with respect to any $\theta \in [\theta_L, \theta_U]$. We again begin by bounding the descent direction for a fixed θ, then generalize to all $\theta \in [\theta_L, \theta_U]$. We present only the upper bounds here, though corresponding results for the lower bound can be shown by reversing the inequalities and replacing SEMax with SEMin.

Bounding the descent direction for a fixed θ. Consider the effect of poisoning either the features or the labels of a data point. For a given data point, an adversary may choose to poison its features, its labels, or both. In total, at most $n+m$ points may be influenced by the poisoning adversary, which corresponds to choosing a disjoint sets for label and feature poisoning. We assume that $m+n \leq b$, otherwise take at most $\min(m+n, b)$ points to be poisoned.

Assume that we have access to sound gradient upper bounds

$$\delta^{(i)} \leq \tilde{\delta}_U^{(i)} \quad \forall \delta^{(i)} \in \left\{\nabla_{\theta'}\mathcal{L}\left(f^{\theta'}\left(\tilde{x}\right), \tilde{y}\right) \mid \|x^{(i)} - \tilde{x}\|_p \leq \epsilon, \|y^{(i)} - \tilde{y}\|_q \leq \nu\right\}.$$

where the inequalities are interpreted element-wise. Here, $\tilde{\delta}_U^{(i)}$ corresponds to an upper bound on the maximum possible gradient achievable at the data-point $(x^{(i)}, y^{(i)})$ through poisoning.

The adversary's maximum possible impact on the descent direction at any point i is given by $\tilde{\delta}_U^{(i)} - \delta^{(i)}$. To maximise an upper bound on $\Delta\theta$, we consider the $n+m$ points with the largest possible adversarial contributions. Therefore, we obtain

$$\Delta\theta = \frac{1}{b}\sum_{i=1}^{b}\delta^{(i)} \leq \frac{1}{b}\left(\underset{m+n}{\operatorname{SEMax}}\left\{\tilde{\delta}_U^{(i)} - \delta^{(i)}\right\}_{i=1}^{b} + \sum_{i=1}^{b}\delta^{(i)}\right),$$

where the SEMax operation corresponds to taking the sum of the largest $n+m$ elements of its argument at each element. This bound captures the maximum increase in $\Delta\theta$ that an adversary can induce by poisoning up to $m+n$ data points.

Bounding the effect of a variable parameter interval. Now, we wish to compute a bound on $\Delta\theta$ for any $\theta \in [\theta_L, \theta_U]$. To achieve this, we extend our previous gradient bounds to account for the interval over our parameters. Specifically, we define upper bounds on the nominal and adversarially perturbed gradients that hold across the entire parameter interval:

$$\begin{aligned}
\delta &\leq \delta_U^{(i)} \quad \forall \delta \in \left\{\nabla_{\theta'}\mathcal{L}\left(f^{\theta'}(x^{(i)}), y^{(i)}\right) \mid \theta' \in [\theta^L, \theta^U]\right\}, \\
\tilde{\delta} &\leq \tilde{\delta}_U^{(i)} \quad \forall \tilde{\delta} \in \left\{\nabla_{\theta'}\mathcal{L}\left(f^{\theta'}\left(\tilde{x}\right), \tilde{y}\right) \mid \theta' \in [\theta^L, \theta^U], \|x^{(i)} - \tilde{x}\|_p \leq \epsilon, \|y^{(i)} - \tilde{y}\|_q \leq \nu\right\}.
\end{aligned}$$

Thus, the descent direction is upper bounded by

$$\Delta\theta \leq \Delta\theta^U = \frac{1}{b}\left(\underset{m+n}{\operatorname{SEMax}}\left\{\tilde{\delta}_U^{(i)} - \delta_U^{(i)}\right\}_{i=1}^{b} + \sum_{i=1}^{b}\delta_U^{(i)}\right)$$

for all $\theta \in [\theta_L, \theta_U]$, where the appropriate bounds with respect to the parameter interval have been substituted in.

I.5 Proof of Proposition 1 (CROWN bounds)

To prove Proposition 1, we rely on the following result which we reproduce from Zhang et al. (2018):

Theorem I.1 (Explicit output bounds of a neural network f (Zhang et al., 2018))**.** *Given an m-layer neural network function $f:\mathbb{R}^{n_0}\to\mathbb{R}^{n_m}$, there exists two explicit functions $f_j^L:\mathbb{R}^{n_0}\to\mathbb{R}$ and $f_j^U:\mathbb{R}^{n_0}\to\mathbb{R}$ such that $\forall j\in[n_m], \forall x\in\mathbb{B}_p(x_0,\epsilon)$, the inequality $f_j^L(x)\le f_j(x)\le f_j^U(x)$ holds true, where*

$$f_j^U(x)=\Lambda_{j,:}^{(0)}x+\sum_{k=1}^{m}\Lambda_{j,:}^{(k)}\left(b^{(k)}+\Delta_{:,j}^{(k)}\right),\quad \Lambda_{j,:}^{(k-1)}=\begin{cases}e_j^\top & \text{if } k=m+1;\\ \left(\Lambda_{j,:}^{(k)}W^{(k)}\right)\circ\lambda_{j,:}^{(k-1)} & \text{if } k\in[m].\end{cases}$$

$$f_j^L(x)=\Omega_{j,:}^{(0)}x+\sum_{k=1}^{m}\Omega_{j,:}^{(k)}\left(b^{(k)}+\Theta_{:,j}^{(k)}\right),\quad \Omega_{j,:}^{(k-1)}=\begin{cases}e_j^\top & \text{if } k=m+1;\\ \left(\Omega_{j,:}^{(k)}W^{(k)}\right)\circ\omega_{j,:}^{(k-1)} & \text{if } k\in[m]\end{cases}$$

and $\forall i\in[n_k]$, we define four matrices $\lambda^{(k)},\omega^{(k)},\Delta^{(k)},\Theta^{(k)}\in\mathbb{R}^{n_m\times n_k}$:

$$\lambda_{j,i}^{(k)}=\begin{cases}\alpha_{U,i}^{(k)} & \text{if } k\ne 0, \Lambda_{j,:}^{(k+1)}W_{:,i}^{(k+1)}\ge 0;\\ \alpha_{L,i}^{(k)} & \text{if } k\ne 0, \Lambda_{j,:}^{(k+1)}W_{:,i}^{(k+1)}< 0;\\ 1 & \text{if } k=0.\end{cases}\qquad \omega_{j,i}^{(k)}=\begin{cases}\alpha_{L,i}^{(k)} & \text{if } k\ne 0, \Omega_{j,:}^{(k+1)}W_{:,i}^{(k+1)}\ge 0;\\ \alpha_{U,i}^{(k)} & \text{if } k\ne 0, \Omega_{j,:}^{(k+1)}W_{:,i}^{(k+1)}< 0;\\ 1 & \text{if } k=0.\end{cases}$$

$$\Delta_{i,j}^{(k)}=\begin{cases}\beta_{U,i}^{(k)} & \text{if } k\ne m, \Lambda_{j,:}^{(k+1)}W_{:,i}^{(k+1)}\ge 0;\\ \beta_{L,i}^{(k)} & \text{if } k\ne m, \Lambda_{j,:}^{(k+1)}W_{:,i}^{(k+1)}< 0;\\ 0 & \text{if } k=m.\end{cases}\qquad \Theta_{i,j}^{(k)}=\begin{cases}\beta_{L,i}^{(k)} & \text{if } k\ne m, \Omega_{j,:}^{(k+1)}W_{:,i}^{(k+1)}\ge 0;\\ \beta_{U,i}^{(k)} & \text{if } k\ne m, \Omega_{j,:}^{(k+1)}W_{:,i}^{(k+1)}< 0;\\ 0 & \text{if } k=m.\end{cases}$$

and $\circ$ is the Hadamard product and $e_j\in\mathbb{R}^{n_m}$ is a standard unit vector at j th coordinate.

The terms $\alpha_{L,i}^{(k)},\alpha_{U,i}^{(k)},\beta_{L,i}^{(k)}$, and $\beta_{U,i}^{(k)}$ represent the coefficients of linear bounds on the activation functions, that is for the r-th neuron in k-th layer with activation function $\sigma(x)$, there exist two linear functions

$$h_{L,r}^{(k)}(x)=\alpha_{L,r}^{(k)}\left(x+\beta_{L,r}^{(k)}\right),\quad h_{U,r}^{(k)}(x)=\alpha_{U,r}^{(k)}\left(x+\beta_{U,r}^{(k)}\right)$$

such that $h_{L,r}^{(k)}(x)\le\sigma(x)\le h_{U,r}^{(k)}(x)\ \forall x\in\left[l_r^{(k)},u_r^{(k)}\right]$. The terms $\left[l_r^{(k)},u_r^{(k)}\right]$ are assumed to be sound bounds on all previous neurons in the network. We first note that, for any neuron r in the k-th layer, the linear bounds may be replaced with concrete bounds by substituting $\alpha_{L,r}^{(k)}=\alpha_{U,r}^{(k)}=0$ and $\beta_{L,r}^{(k)}=l_r^{(k)},\beta_{U,r}^{(k)}=u_r^{(k)}$. Let S^L,S^U be index sets of tuples (i,k) indicating whether the lower and upper bounds (respectively) of the i-th neuron in the k-th layer should be concretized in this way. Then, the equivalent weights and biases take the following form:

$$\lambda_{j,i}^{(k)}=\begin{cases}\alpha_{U,i}^{(k)} & \text{if } (i,k)\notin S^U, k\ne 0, \Lambda_{j,:}^{(k+1)}W_{:,i}^{(k+1)}\ge 0;\\ \alpha_{L,i}^{(k)} & \text{if } (i,k)\notin S^U, k\ne 0, \Lambda_{j,:}^{(k+1)}W_{:,i}^{(k+1)}< 0;\\ 1 & \text{if } (i,k)\notin S^U, k=0;\\ 0 & \text{if } (i,k)\in S^U.\end{cases}$$

$$\omega_{j,i}^{(k)}=\begin{cases}\alpha_{L,i}^{(k)} & \text{if } (i,k)\notin S^L, k\ne 0, \Omega_{j,:}^{(k+1)}W_{:,i}^{(k+1)}\ge 0;\\ \alpha_{U,i}^{(k)} & \text{if } (i,k)\notin S^L, k\ne 0, \Omega_{j,:}^{(k+1)}W_{:,i}^{(k+1)}< 0;\\ 1 & \text{if } (i,k)\notin S^L, k=0;\\ 0 & \text{if } (i,k)\in S^L.\end{cases}$$

$$\Delta_{i,j}^{(k)}=\begin{cases}\beta_{U,i}^{(k)} & \text{if } (i,k)\notin S^U, k\ne m, \Lambda_{j,:}^{(k+1)}W_{:,i}^{(k+1)}\ge 0;\\ \beta_{L,i}^{(k)} & \text{if } (i,k)\notin S^U, k\ne m, \Lambda_{j,:}^{(k+1)}W_{:,i}^{(k+1)}< 0;\\ 0 & \text{if } (i,k)\notin S^U, k=m;\\ u^{(k)} & \text{if} (i,k)\in S^U.\end{cases}$$

$$\Theta_{i,j}^{(k)}=\begin{cases}\beta_{L,i}^{(k)} & \text{if } (i,k)\notin S^L, k\ne m, \Omega_{j,:}^{(k+1)}W_{:,i}^{(k+1)}\ge 0;\\ \beta_{U,i}^{(k)} & \text{if } (i,k)\notin S^L, k\ne m, \Omega_{j,:}^{(k+1)}W_{:,i}^{(k+1)}< 0;\\ 0 & \text{if } (i,k)\notin S^L, k=m;\\ l^{(k)} & \text{if} (i,k)\in S^L.\end{cases}$$

This is exactly the form described in Appendix D, where S^L and S^U are chosen to be the sets of neurons whose equivalent coefficient interval spans zero. Without this modification, the equivalent weights and biases of such neurons in the original formulation would be undefined. We now have bounds on the output of the neural network for which all operations are well-defined in interval arithmetic.

Replacing all operations in the computation of the equivalent weight terms by their interval arithmetic counterparts, we can compute sound, though over-approximated, intervals over $\Lambda^{(0:m)}$ and $\Omega^{(0:m)}$ which satisfy

$$\Lambda^{(k)} \in \boldsymbol{\Lambda^{(k)}} \quad \forall\, W^{(1:m)} \in \boldsymbol{W}^{(1:m)}, b^{(1:m)} \in \boldsymbol{b}^{(1:m)},$$
$$\Omega^{(k)} \in \boldsymbol{\Omega^{(k)}} \quad \forall\, W^{(1:m)} \in \boldsymbol{W}^{(1:m)}, b^{(1:m)} \in \boldsymbol{b}^{(1:m)}.$$

This is trivially true by the definitions of the interval arithmetic operations as given in Appendix C.

Turning to the upper bound (though analagous arguments hold for the lower bound), we have

$$f_j^U\left(x, \Lambda^{(0:m)}, \Delta^{(1:m)}, b^{(1:m)}\right) = \Lambda^{(0)}_{j,:} x + \sum_{k=1}^{m} \Lambda^{(k)}_{j,:}\left(b^{(k)} + \Delta^{(k)}_{:,j}\right)$$

where $\Lambda^{(0:m)}$ are functions of the weights and biases of the network and $\Delta^{(1:m)}$ are constants that depend on the bounds on the intermediate layers of the network. Thus, given parameter intervals $\boldsymbol{b}^{(k)}, \boldsymbol{W}^{(k)}$, the following result holds

$$\begin{aligned} f_j^U\left(x, \Lambda^{(0:m)}, \Delta^{(1:m)}, b^{(1:m)}\right) &\leq \max\left\{ f_j^U\left(x, \Lambda^{(0:m)}, \Delta^{(1:m)}, b^{(1:m)}\right) \;\middle|\; \begin{array}{l} W^{(1:m)} \in \boldsymbol{W}^{(1:m)} \\ b^{(1:m)} \in \boldsymbol{b}^{(1:m)} \end{array} \right\} \\ &\leq \max\left\{ f_j^U\left(x, \Lambda^{(0:m)}, \Delta^{(1:m)}, b^{(1:m)}\right) \;\middle|\; \begin{array}{l} \Lambda^{(0:m)} \in \boldsymbol{\Lambda}^{(0:m)} \\ b^{(1:m)} \in \boldsymbol{b}^{(1:m)} \end{array} \right\} \end{aligned}$$

for any set of intervals $\boldsymbol{\Lambda}^{(0:m)}$ that satisfy $\{\Lambda^{(k)} \mid W^{(1:m)} \in \boldsymbol{W}^{(1:m)}\} \subseteq \boldsymbol{\Lambda}^{(k)}$. Since our intervals $\boldsymbol{\Lambda}^{(0:m)}$ computed via interval arithmetic satisfy this property, we have that any valid bound on this maximization problem constitutes a bound on the output of the neural network $f_j(x)$ for any $W^{(1:m)} \in \boldsymbol{W}^{(1:m)}$ and $b^{(1:m)} \in \boldsymbol{b}^{(1:m)}$.

J Experimental Set-up and Additional Results

This section details the datasets and hyper-parameters used for the experiments detailed in Section 4. All experiments were run on a server equipped with 2x AMD EPYC 9334 CPUs and 2x NVIDIA L40 GPUs using an implementation of Algorithm 1 written in Python using Pytorch.

Table 1 shows a run-time comparison of our implementation of Algorithm 1 with (un-certified) training in Pytorch. We observe that training using Abstract Gradient Training typically incurs a modest additional cost per iteration when compared to standard training.

	Time per iteration (seconds)	
Dataset	**Abstract gradient training**	**Un-certified training**
UCI House-electric	0.25	0.12
MNIST (inc. PCA projection)	1.6	1.1
OCT-MNIST	0.96	0.10
Udacity Self-Driving	53	42

Table 1: Comparison of the run-time of AGT and standard model training in Pytorch.

Table 2 details the datasets along with the number of epochs, learning rate (α), decay rate (η) and batch size (b) used for each. We note that a standard learning rate decay of the form ($\alpha_n = \alpha/(1+\eta n)$ was applied during training. In the case of fine-tuning both OCT-MNIST and PilotNet, each batch consisted of a mix 70% 'clean' data previously seen during pre-training and 30% new, potentially poisoned, fine-tuning data.

Dataset	#Samples	#Features	#Epochs	α	η	b
UCI House-electric	2049280	11	1	0.02	0.2	10000
MNIST	60000	784	3	5.0	1.0	60000
OCT-MNIST	97477	784	2	0.05	5.0	6000
Udacity Self-Driving	31573	39600	2	0.25	10.0	10000

Table 2: Datasets and Hyperparameter Settings

J.1 Comparison with Deep Partition Aggregation

As discussed in Appendix A, existing approaches seeking certifiable robustness to data poisoning rely on partitioning the dataset and training large ensemble models. Figure 9 compares the guarantees provided by Abstract Gradient Training to those of a popular ensemble method, (Self-Supervised)-Deep Partition Aggregation (SS-DPA) (Levine & Feizi, 2020).

SS-DPA demonstrates both higher nominal and certified accuracies than AGT on the MNIST dataset. However, the ensemble approach is specifically designed to be robust to data poisoning attacks. On the other hand, AGT focuses on certifying an *existing* training algorithm by analyzing the sensitivity of standard training pipelines to data poisoning.

Additionally, our experiments show that AGT incurs a run-time cost of approximately 2-4 times standard training. In contrast, SS-DPA requires training an ensemble of thousands of classifiers, demanding significant data and computational costs. Finally, we highlight once again that AGT is complementary to ensemble approaches, and future works may seek to combine the two.

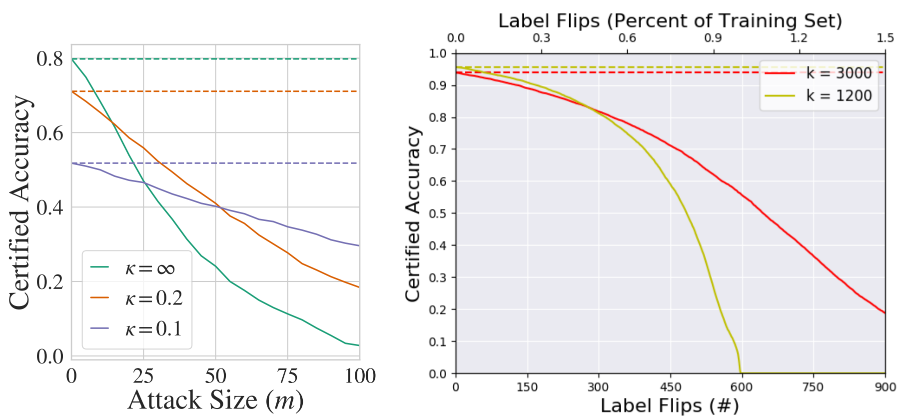


Figure 9: Comparison of Certified Accuracy on the MNIST dataset under a label flipping attack. Left: Certified Accuracy using AGT for a variety of clipping levels κ. Right: Certified Accuracy using SS-DPA for ensembles of size $k = 3000$ and $k = 1200$. Figure reproduced from Levine & Feizi (2020).