FullCert: Deterministic End-to-End Certification for Training and Inference of Neural Networks

Tobias Lorenz[1,2], Marta Kwiatkowska[2], and Mario Fritz[1]

[1] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
[2] Department of Computer Science, University of Oxford, Oxford, UK

tobias.lorenz@cispa.de, marta.kwiatkowska@cs.ox.ac.uk, fritz@cispa.de

Abstract. Modern machine learning models are sensitive to the manipulation of both the training data (poisoning attacks) and inference data (adversarial examples). Recognizing this issue, the community has developed many empirical defenses against both attacks and, more recently, certification methods with provable guarantees against inference-time attacks. However, such guarantees are still largely lacking for training-time attacks. In this work, we present FullCert, the first end-to-end certifier with sound, deterministic bounds, which proves robustness against both training-time and inference-time attacks. We first bound all possible perturbations an adversary can make to the training data under the considered threat model. Using these constraints, we bound the perturbations' influence on the model's parameters. Finally, we bound the impact of these parameter changes on the model's prediction, resulting in joint robustness guarantees against poisoning *and* adversarial examples. To facilitate this novel certification paradigm, we combine our theoretical work with a new open-source library BoundFlow, which enables model training on bounded datasets. We experimentally demonstrate FullCert's feasibility on two datasets.

1 Introduction

Deep neural networks have shown impressive performance across many tasks, especially computer vision [8] and natural language [4] tasks. However, current research into the robustness of these models also reveals that their heavy reliance on data makes deep learning models susceptible to both training-time and inference-time data attacks [22]. Training-time attacks, e.g., data-poisoning [26, 37], manipulate the *training* data to change the model and, therefore, its predictions. Inference-time evasion attacks, e.g., adversarial examples [12, 28], change the *test* data to alter the prediction. To achieve reliable, trustworthy machine learning models, it is crucial to quantify and limit an attacker's influence on the models and to develop worst-case guarantees on the model's performance. Robustness guarantees will only become more critical as we increase the model's scale and train them on massive datasets scraped from the internet [4, 21].

Certified defenses against evasion attacks, i.e., attacks to change the behavior of a pre-trained model, have made significant progress over the last years,

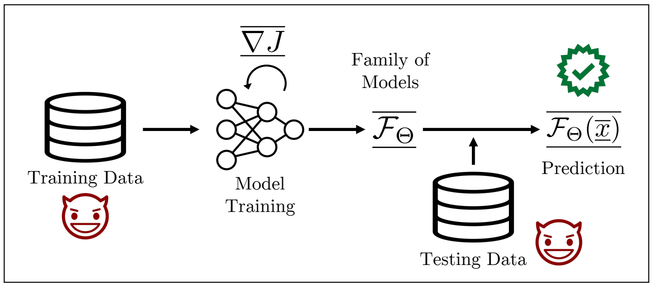


Fig. 1. Overview of FullCert. During training, we bound the effects that perturbation of the training data can have on the model. At inference, we combine these bounds with bounds for perturbation of the test data. The result is a certified prediction against training *and* test time attacks.

yielding solutions that can guarantee the robustness of medium-sized models [5, 9]. In contrast, for training-time poisoning defenses, current solutions still largely rely on heuristics without any rigorous guarantees and can therefore be circumvented by adaptive attacks [17]. First works on certified defenses against poisoning attacks with probabilistic guarantees [15, 24, 30, 36] show that the problem is much harder, requiring further investigation and new solutions.

Multiple dimensions make training-time certification more difficult than inference-time certification. While inference only depends on a single data point, training depends on a large set of training data. This gives the attacker a much higher degree of freedom, as they can influence multiple data points concurrently. A second aspect is the much deeper computational graph. While inference consists of a single forward pass, training consists of multiple iterations of forward and backward passes for each data point and, usually, for multiple epochs. Since certifying the robustness of a single forward pass is already NP-complete [16], certifying the robustness of the entire training procedure becomes even more challenging. Current approaches give only probabilistic guarantees and severely limit the attacker's influence to a few data samples [14, 24] or single pixels [30].

We propose FullCert – the first end-to-end certifier with deterministic worst-case guarantees against both training-time poisoning and inference-time evasion attacks. FullCert consists of three elements: (1) a formal problem definition of deterministic end-to-end certification, (2) a deterministic, sound certification formulation, and (3) an instantiation and implementation using interval bounds based on our new BoundFlow library. Our approach is based on abstract interpretation, which uses reachability analysis to train a family of infinitely many models that could have resulted from all possible poisonings within some bound of the training data. We then perform inference on this family of models, considering all possible input perturbations. This allows us to determine whether the *combination* of poisoning and evasion attack could have affected the final prediction and establish a robustness certificate if we can guarantee correctness. We implement our certifier in an open-source software package BoundFlow based on PyTorch and experimentally validate it on different tasks. We summarize our contributions as follows: 1. A formal definition of the deterministic end–to-end neural network certification problem 2. FullCert, the first deterministic certifier against both data-poisoning and evasion attacks 3. A formal convergence analysis of our method 4. An open-source software package BoundFlow 5. An experimental evaluation of FullCert. Our implementation is available at https://github.com/t-lorenz/FullCert.

2 Related Work

The work most closely related to our method stems from two general lines: first approaches to probabilistic robustness certification of training-time data-poisoning attacks and sound, deterministic certifiers against evasion attacks.

Training-Time Certification. The most closely related work to our approach is Wang et al. [30], which proposes a defense against backdoor attacks via Randomized Smoothing. By smoothing over training subsets, they are able to extend randomized smoothing to network training. Using a subset of 100 digits from two MNIST classes, converted into black-and-white values, the method can guarantee invariance to the modification of 2 pixels across all training images and the test image. In contrast, our guarantees are deterministic and hold with certainty, and we consider imperceptible perturbations to *all* pixels.

Rosenfeld et al. [24] also use Randomized Smoothing for guarantees against label-flipping attacks, and Weber et al. [33] extend Randomized Smoothing to defend against backdoor attacks with probabilistic guarantees to ℓ_2-norm perturbations.

A second line of work derives (probabilistic) guarantees against training-time attacks through bagging. Jia et al. [14] demonstrate that the data sub-sampling of vanilla bagging strategies show intrinsic robustness against poisoning attacks. Jia et al. [15] show that this property natively holds for nearest neighbor classifiers, even without ensembles. Wang et al. [32] build upon the bagging approach and improve the robustness guarantees based on an improved sampling strategy. Levine and Feizi [19] propose a deterministic version of bagging, which partitions the dataset based on a deterministic hash function. Zhang et al. [36] extend this approach to consider backdoor attacks with triggers. Recent work also considers different threat models, including temporal aspects [31] and dynamic attacks [2].

The main differences between this line of work and our approach are the (mostly) probabilistic versus our deterministic guarantees and the limitation to training-time attacks, whereas ours enables end-to-end certification with inference-time attacks, and different threat model assumptions (ℓ_0/ℓ_2 vs. ℓ_∞). We include a detailed comparison between the two approaches in section 5.

Inference-Time Certification. There is a long list of work on bound-based test- and inference-time certification [1, 10, 20, 27, 34, 35]. The main difference is their trade-off between the precision of the certificate versus the scalability of the method to larger network sizes. Most closely related to our work is Interval

Bound Propagation (IBP) [13], which, like our approach, uses intervals with upper and lower real-valued bounds.

Bound-Based Robust Training. Many inference-time certification approaches also exploit their bounds during training to achieve more robust models [13, 20, 35], sometimes referred to as "robust training" or "certified training" in the literature, which may cause confusion with our method. The goal of using bounds for training is to improve the model's robustness against *inference-time* attacks and to mitigate over-approximations. In contrast to our work, they do not provide any defense or guarantees against *training-time* attacks.

3 End-to-End Neural Network Certification

We present our method FullCert, the first deterministic end-to-end neural network certifier against training-time and inference-time attacks. In section 3.1, we formally define the end-to-end certification problem. We then present our general approach based on reachability analysis (via abstract interpretation) in section 3.2, independent of the concrete bounds. Finally, we introduce our instantiation using interval bounds in section 3.3 and conclude with a description of our implementation in section 3.4.

3.1 Problem Definition of End-to-End Certification

The goal of FullCert is to certify the robustness of a deep learning model against perturbations on the training and inference data. Intuitively, we view the combination of the model training and then its prediction on a single data point as a single function. This function takes a training dataset D_{train} consisting of N pairs of inputs and labels, as well as a single test sample x as input. We can then analyze the influence of changing a combination of D_{train} and x on the model's prediction. If we can guarantee that no changes would alter the prediction, we can certify the model's robustness.

Certified Training. More formally, we train a model $f_\theta : \mathcal{X} \mapsto \mathcal{Y}$ on a dataset D by optimizing its parameters θ using a training algorithm A, e.g., SGD:

$$\theta = A(D). \tag{1}$$

An adversary can influence the training set with a poisoning attack, which we model by considering a family of datasets $\mathcal{D}$, which include all bounded perturbations that the adversary could cause to the training data. One way to define $\mathcal{D}$ is through a similarity metric d, which measures the similarity of two datasets:

$$\mathcal{D} := \{D \mid d(D, D_{\text{train}}) \leq \epsilon\}. \tag{2}$$

We use the element-wise ℓ_∞-norm with small ϵ to restrict the adversary to imperceptible changes in analogy to evasion attacks.

Given $\mathcal{D}$ as possible inputs to A, we get a family of models

$$\mathcal{F}_\Theta := \{f_\theta \mid \theta = A(D), D \in \mathcal{D}\} \tag{3}$$

which contains all possible models that could result from the poisoning attacks. Assuming no inference-time attacks after deployment, we can certify the model robustness at x if

$$\forall f_\theta \in \mathcal{F}_\Theta : f_\theta(x) = y. \tag{4}$$

End-to-End Certification. For end-to-end certification, we also need to consider perturbations to x. We adopt the common formulation of adversarial examples [12], and define the bounded perturbation set $\mathcal{S} := \{x' \mid d'(x, x') \leq \epsilon'\}$. d' measures the distance between the original and perturbed data point, for which we use the ℓ_∞-norm again. We can then extend eq. (4) to certify end-to-end robustness against both training and inference time attacks:

$$\forall f_\theta \in \mathcal{F}_\Theta, x' \in \mathcal{S} : f_\theta(x') = y. \tag{5}$$

3.2 FullCert: End-to-End Certification of Neural Networks by Abstract Interpretation

Solving eq. (5) precisely is infeasible, as even the test-time certification problem for a single classifier, that is, $\forall x' \in \mathcal{S} : f_\theta(x') = y$, is NP-complete [16]. We, therefore, propose a solution inspired by work on test-time certification, which uses reachability analysis (based on abstract interpretation [6]) to compute a solution. The key is to only allow over-approximations in order to ensure sound certificates. In other words, if we guarantee robustness (i.e., the certifier outputs that eq. (5) holds), we want to be sure this guarantee is correct. However, we allow the certifier to not compute a guarantee in some cases, even though the underlying model is actually robust, to make a solution feasible.

To perform this reachability analysis, we view the training of the model on a dataset D and the following inference on a single datapoint x as a single function, which takes D and x as inputs and computes the prediction $f(x)$ as output. This allows us to define the potential perturbations on D and x as a precondition, propagate it through the unrolled training and inference, and finally check the postcondition of correct classification.

The first step in this process is to build the sets $\mathcal{D}$ and $\mathcal{S}$, which contain all possible perturbations permitted by the attack model. The concrete shape of these sets depends on the threat model, which in our case is an interval (also called hyperrectangle, orthotope, or box) with real-valued upper and lower bounds.

Once $\mathcal{D}$ and $\mathcal{S}$ are defined, we compute all outputs that could result from any combination of elements from those input sets. Since both sets contain infinitely many elements, we cannot test all combinations. We therefore unroll the training and subsequent inference into a series of function invocations $g_1 \circ g_2 \circ \cdots \circ g_n$, where each g_i represents a part of a forward or backward pass. For example, the layers of the forward pass, the loss function, or the gradient computations of the backward pass. We can then define abstract versions of g_i, labeled G_i, which compute the effect of g_i on an entire set of inputs.

More formally, given a function $g_i : \mathcal{U} \to \mathcal{V}$, we define an abstract version for sets $G_i : \mathbb{U} \to \mathbb{V}$, where $U \in \mathbb{U}, U \subset \mathcal{U}$, such that

$$\forall u \in U, U \in \mathbb{U} : g_i(u) \in G_i(U). \tag{6}$$

This property ensures that $G_i(U)$ contains all reachable points of $g_i(u)$, which makes the method sound. The opposite does not necessarily hold, i.e., we allow points in $G_i(U)$ that are not reachable by $g_i(u)$. This over-approximation makes the computation feasible, as it is not always possible to efficiently compute an exact $G_i(U)$. The next paragraphs describe training and inference in detail.

Forward Pass. For each training sample and its ground-truth label $(x, y) \in D_{\text{train}}$, the first step is to build the set $\mathcal{X}_0 = \{x' \mid d(x', x) \leq \epsilon\}$. We then execute the forward pass on $\mathcal{X}_0$ by building and abstract operation $L^{(i)}$ for each layer operation $l^{(i)}$ in $f_\theta = l^{(1)} \circ \cdots \circ l^{(k)}$, as detailed in eq. (6). After the last layer, we get a set $\mathcal{X}_k = L^{(k)}(\mathcal{X}_{k-1})$ with possible outputs. We compute the set of losses $\mathcal{O}$ on $\mathcal{X}_k$ using an abstract version of the loss function via the same principle.

Backward Pass. The backward pass uses the chain rule to compute the gradients wrt each parameter. Since all layer inputs and outputs are sets, we have to create abstract versions of the gradient functions for each layer. This means the gradient with respect to each parameter is a set of values as well. This set represents the influence an attacker has on the gradient by perturbing the training data. With sets as gradients, the parameter update also turns into a set operation:

$$\Theta_{t+1} = \Theta_t - \lambda_t \nabla_\theta J(\Theta_t, X, y), \tag{7}$$

with objective function J and learning rate λ_t. This leads to a set of possible parameters Θ, representing all models that could result from training with perturbations. This yields a multiplication of two sets, the parameters and the inputs, in all linear and convolution layers, which is an additional challenge compared to inference-time-only certification without bounds on the parameters.

The final result after this training is a family of models $\mathcal{F}_\Theta$, represented by a model architecture parameterized with an infinite set of parameters. This set of parameters contains all possible parameters the attacker could impose through data poisoning, in addition to elements due to over-approximation.

Inference. Using these sets of parameters resulting from training, we compute the final certificate during inference for a test input x. The principle is the same as a forward pass during training: build the set $\mathcal{X}$, and then propagate it through the abstract operations to obtain a set of outputs $\mathcal{X}_k$. If all of these possible outputs are classified correctly, we can conclude that no attack could have changed the prediction.

This certification scheme can be used both as online certificates, guaranteeing a robust prediction for a given sample, and as offline certificates, quantifying the robustness of a model over a test set.

3.3 FullCert Based on Interval Abstractions

The general solution introduced in section 3.2 is independent of the concrete choice of how to represent the sets and abstract functions. The literature for inference-time certification has used different convex models, from fast intervals to more complex polytopes. The choice of relaxation controls the trade-off between computational complexity and the precision of the over-approximations.

For this work, we use intervals (or hyperrectangles/boxes) for our relaxations. This abstraction has two main advantages: (1) interval bounds are fast to compute compared to other relaxations, which is crucial due to the large depth of the computational graph for model training, and (2) it is easy to implement the multiplication of two intervals, which is not straightforward for more general polytopes. We introduce abstract interval-versions of the operations required for neural network training below, and prove their soundness in appendix A[3].

Notation. We represent each set of values, e.g., $\mathcal{X}$ and Θ, as an n-dimensional interval with upper and lower real bounds. We denote an interval as $\overline{\underline{A}} = [\underline{A}, \overline{A}] \in \mathbb{I}^n$, with lower bound $\underline{A} \in \mathbb{R}^m$ and upper bound $\overline{A} \in \mathbb{R}^m$. Equivalently, we can define the same interval via its center and radius: $\overline{\underline{A}} = \{m_A, r_A\}$, where $m_A = \frac{\underline{A}+\overline{A}}{2}$ is the center and $r_A = \frac{\overline{A}-\underline{A}}{2}$ the radius. We use these interchangeably.

Linear Layer. For linear and convolution layers, we use Rump's algorithm [7, 25] for matrix multiplication based on the center and radius of the interval:

$$\begin{aligned} &\overline{\underline{C}} = \overline{\underline{A}} * \overline{\underline{B}}, \overline{\underline{A}} \in \mathbb{I}^{m\times n}, \overline{\underline{B}} \in \mathbb{I}^{n\times p}, \overline{\underline{C}} \in \mathbb{I}^{m\times p} \\ &m_C = m_A * m_B,\ r_C = (|m_A| + r_A) * r_B + r_A * |m_B|. \end{aligned} \tag{8}$$

We can compute bounds for linear layers with weights $\overline{\underline{W}}$, biases $\overline{\underline{B}}$, and inputs $\overline{\underline{X}}$, as well as the derivatives:

$$\overline{\underline{Y}} = \overline{\underline{WX}} + \overline{\underline{B}}, \qquad \overline{\underline{\frac{\partial Y}{\partial X}}} = \overline{\underline{W}} \quad \overline{\underline{\frac{\partial Y}{\partial W}}} = \overline{\underline{X}} \quad \overline{\underline{\frac{\partial Y}{\partial B}}} = \mathbf{1}. \tag{9}$$

Activation Functions. We can compute the upper and lower bound for any monotonic function by evaluating only the interval bounds (proof in appendix A.2):

$$G(\overline{\underline{X}}) = [\min\{g(\underline{X}), g(\overline{X})\}, \max\{g(\underline{X}), g(\overline{X})\}]. \tag{10}$$

In particular, the ReLU function and its derivative can be bound as:

$$\mathrm{ReLU}(\overline{\underline{X}}) = [\mathrm{ReLU}(\underline{X}), \mathrm{ReLU}(\overline{X})], \qquad \mathrm{ReLU}'(\overline{\underline{X}}) = [\mathrm{ReLU}'(\underline{X}), \mathrm{ReLU}'(\overline{X})].$$

[3] The appendix is available at https://arxiv.org/abs/2406.11522

Cross-Entropy Loss. To compute the gradients for the backward pass, we first need an interval version of the cross-entropy (CE) loss function. CE is defined as $J(p, y) = -\sum_{i=1}^{m} y_i \log(p_i)$ on the class probabilities p_i. A naive solution could use the upper and lower bounds of the softmax function followed by the logarithm. However, this is numerically unstable and, therefore, the softmax and logarithm are typically computed together [11]. We take this into account and define numerically stable bounds for the logsoftmax function:

$$\begin{aligned} \operatorname{logsoftmax}(\underline{\overline{z}}) = \Bigg[&(\underline{z}_c - a) - \log\Bigg(\sum_{i \neq c}^{m} \exp(\overline{z}_i - a) + \exp(\underline{z}_c - a)\Bigg), \\ &(\overline{z}_c - b) - \log\Bigg(\sum_{i \neq c}^{m} \exp(\underline{z}_i - b) + \exp(\overline{z}_c - b)\Bigg)\Bigg] \\ &a = \max\{\underline{z}_c, \overline{z}_{i \neq c}\},\ b = \max\{\overline{z}_c, \underline{z}_{i \neq c}\}. \end{aligned} \tag{11}$$

Refer to appendix A.3 for a detailed derivation, proofs of the soundness and tightness of these bounds, and numerically stable bounds for the CE derivative.

Binary Cross-Entropy. The BCE loss with sigmoid function $\sigma(z)$ is

$$J(p, y) = -(y \log(p) + (1 - y) \log(1 - p)),\ p = \sigma(z). \tag{12}$$

It has similar stability issues as CE with softmax. We define its bounds as

$$\begin{aligned} J(\underline{\overline{z}}, y) = \big[&\underline{z} - \underline{z}y + a + \log(\exp(-a) + \exp(-\underline{z} - a)), \\ &\overline{z} - \overline{z}y + b + \log(\exp(-b) + \exp(-\overline{z} - b))\big] \\ &a = \max(-\underline{z}, 0),\ b = \max(-\overline{z}, 0) \end{aligned} \tag{13}$$

and defer the derivative and proofs to appendix A.4.

3.4 Implementation

Existing bound-based certifiers only consider inference-time certification. They rely on standard deep learning libraries for training, which use tensors as their basic data type and cannot be easily extended to support bounded parameters. We, therefore, develop a new open-source software package for end-to-end certification. The primary goals are (1) a correct implementation of our method, (2) easy extensibility, (3) high performance through GPU support, and (4) easy integration with existing deep-learning frameworks.

We choose PyTorch's [23] low-level vector operations as our basis, which allows for performance-optimized operations. However, we cannot re-use PyTorch's pre-defined layer operations or automatic differentiation system, as it assumes tensor inputs. We, therefore, create our own open-source library BoundFlow, which implements layers, models, and gradient operations for bound-based model training, including GPU support, for many platforms. It supports basic layer operations for interval bounds, which can be extended to more complex operations and bounds as required.

4 Convergence Analysis

Training with FullCert can be thought of as executing SGD on infinitely many datasets concurrently. We analyze the convergence of FullCert towards an optimum to understand under which conditions certified training can work and where the challenges lie. Since proving the convergence of SGD for general, non-convex problems is not possible, we consider a simplified, convex setting and full-batch gradient descent, which is well understood [3] and gives insights into the algorithm's behavior.

In principle, two factors can cause the training to diverge. (1) The perturbations of the training set itself, i.e., the perturbations could include a dataset for which gradient descent no longer converges. (2) The over-approximations, which we allow to make the solution feasible, could prevent the algorithm from converging in some cases. We consider both factors in turn.

Perturbations. To analyze the influence of perturbations, we consider a hypothetical, exact certifier, which does not allow any over-approximations. That is, $\forall v \in G_i(U)\ \exists u \in U : g_i(u) = v$. Then we can show that the certified training A_{exact} on the family of perturbed datasets $\mathcal{D}$ converges exactly when the base gradient descent (GD) algorithm A_{GD} converges for each perturbed dataset:

$$\begin{aligned} \Theta_{\text{exact}} = A_{\text{exact}}(\mathcal{D}) \quad \forall \theta \in \Theta_{\text{exact}} : \exists D \in \mathcal{D} \mid \theta = A_{\text{GD}}(D) \\ A_{\text{GD}} \text{ converges } \forall D \in \mathcal{D} \Leftrightarrow A_{\text{exact}}(\mathcal{D}) \text{ converges.} \end{aligned} \tag{14}$$

This convergence property directly follows from the definition of exact certifiers, which implies that all parameter configurations $\theta \in \Theta_{\text{exact}}$ are reachable. Therefore, there has to be a perturbed dataset $D \in \mathcal{D}$ that produces the same θ using A_{GD}. The convergence property is desirable, as it implies a guarantee that the attacker could not have changed the convergence if certified training converges.

Over-Approximations. Practical certifiers cannot compute Θ_{exact} and therefore typically approximate it with $\Theta_{\text{relaxed}} \supset \Theta_{\text{exact}}$, which allows new values that could lead to divergence. We analyze this with a proof sketch following Bottou [3], which we generalize to families of datasets and sets of parameters.

As mentioned above, this proof only holds with some assumptions: (1) non-zero gradients everywhere except for the optimum (i.e., a single global optimum without saddle points), (2) Lipschitzness (i.e., bounded gradients), and (3) a decreasing, non-zero learning rate.

The core idea of Bottou [3] is to define a Lyapunov sequence $h_t = (\theta_t - \theta^*)^2$, a sequence of positive numbers whose value measures the distance to the target at step t. We can then show that the algorithm converges by showing that this sequence converges. We generalize this sequence to intervals as $h_t = (\Theta_t - \Theta^*)^2$.

Using the definition of the gradient update step (eq. (7)), we get

$$h_{t+1} - h_t = \underbrace{-2\lambda_t(\Theta_t - \Theta^*)\nabla_\Theta J(\Theta_t)}_{\text{distance to optimum}} + \underbrace{\lambda_t^2(\nabla_\Theta J(\Theta_t))^2}_{\text{discrete dynamics}}. \tag{15}$$

The second term is a consequence of the discrete nature of the system and is bounded by assumptions on the gradients and learning rate (see Bottou [3]). The first term relates to the distance to the optimum and should be negative for the training to converge. While this is easy to show for single θ_t and gradients, it does not always hold for intervals. For the term to be negative, $(\Theta_t - \Theta^*)\nabla_\Theta J(\Theta_t)$ has to be positive. This is the case exactly when $\Theta_t \cap \Theta^* = \emptyset$ due to the properties of interval multiplication. The algorithm starts diverging as soon as the optimum is contained within the parameter interval.

The second challenge lies in the gradient update step, which causes the radius of Θ_t to always grow. This is a consequence of subtraction in interval arithmetic:

$$\underline{\overline{c}} = \underline{\overline{a}} - \underline{\overline{b}} = [\underline{a} - \overline{b}, \overline{a} - \underline{b}] \Rightarrow r_c = r_a + r_b, \quad \Rightarrow r_{\Theta_{t+1}} = r_{\Theta_t} + \lambda r_{\nabla_\theta L(\Theta_t)}. \tag{16}$$

We draw two conclusions from this. (1) The fewer steps are required, the more precise the solution will be. This is partially due to fewer over-approximations but also a natural consequence of the fact that, the less we train, the smaller the influence of an attacker. (2) A small step size is beneficial, as the parameters slowly converge towards the optimum without including it within the parameter set prematurely.

5 Experiments

To complement our theoretical insights, we perform a series of experiments to demonstrate the feasibility of FullCert. We evaluate FullCert qualitatively and quantitatively on two different datasets. Appendix B contains additional experiments, e.g., on the influence of hyper-parameters and model architectures.

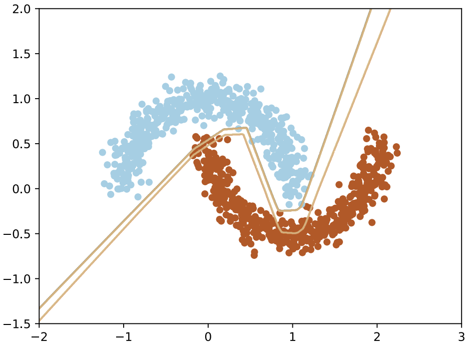


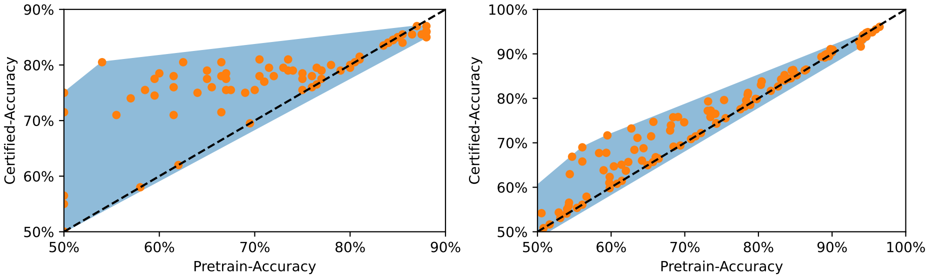
Fig. 2. Barriers to the decision boundaries for all models that could have resulted from poisoning the Two-Moons dataset. Our bounds on the parameters guarantee that all points outside these barriers are robustly classified.

Experimental Setup. We use our BoundFlow software package to implement end-to-end certified training and inference. For evaluation, we use two datasets: Two-Moons and MNIST 1/7 [18]. Two-Moons is a two-dimensional dataset with two classes of points configured in interleaving half circles. It is well-suited for analyzing the behavior of training algorithms and allows easy visualization. To test the behavior on more complex data, we follow prior work [30] and train a model on MNIST 1/7, the MNIST subset with only digits 1 and 7. We report *certified accuracy*, the percentage of test samples for which we can guarantee correct prediction. That means the model predicts the correct label, and eq. (5) holds. We set the perturbation radii to $\epsilon = \epsilon' = 10^{-3}$ for Two-Moons and $\epsilon = \epsilon' = 10^{-4}$ for MNIST. Training details are listed in app. B, and our implementation is available at https://github.com/t-lorenz/FullCert.

End-to-End Certification. These experiments demonstrate FullCert for SGD training. The plethora of successful poisoning attacks shows that SGD training is unstable and easy to influence, which makes certification challenging.

To illustrate FullCert, we perform a qualitative evaluation on Two-Moons. Figure 2 visualizes the dataset's two classes and the barriers to the decision boundaries of the model family trained with FullCert. FullCert guarantees that the decision boundaries of all models lie between the two barriers, and therefore guarantees that all points outside these barriers are robustly classified. An adversary could potentially have influenced the classification of points in between.

For a quantitative evaluation, we run experiments on Two-Moons and MNIST 1/7 datasets. We start certified training from different starting points to verify the results from section 4. Using a small subset of 100 unperturbed samples, we gradually pretrain models to different starting accuracies.



(a) Two-Moons (b) MNIST 1/7

Fig. 3. Certified accuracy for different initial accuracies on Two-Moons for $\epsilon = 10^{-3}$ and MNIST 1/7 for $\epsilon = 10^{-4}$. Each dot represents a separate model, with the convex hull in blue. The closer the initialization after pretraining is to the final operating point, the higher the final certified accuracy. Blue convex hulls are for visualization purposes.

Figure 3 shows the certified accuracy for both datasets for different starting points. The diagonal in the center represents the baseline certified accuracy with only pretraining. Points above the diagonal show improved certified accuracy, demonstrating benefits of our training scheme.

The results show that certified training is already effective without any pretraining (left end of the x-axis with 50% accuracy). On Two-Moons, it achieves 75% certified accuracy when training from scratch, and 60% on MNIST 1/7. The closer the model gets to the optimum during pretraining, the higher the final certified accuracy, slowly converging with the pretrained accuracy once the starting point reaches the final optimum.

Comparison to Related Approaches. In contrast to certification for inference, training-time certification is still in its early stages. In fact, our submission addresses deterministic end-to-end certification for the first time and is the first work to bring bound-based certification to training-time guarantees.

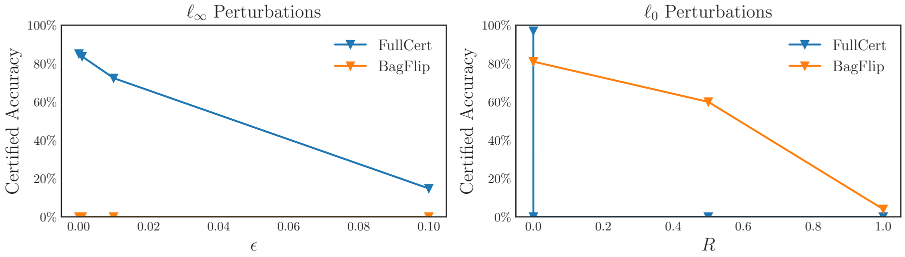


Fig. 4. Comparison between FullCert and BagFlip. Left: Certified Accuracy for different ϵ on Two-Moons. The threat model allows perturbations of up to ϵ for each feature. Right: Certified Accuracy for different R as a percentage of MNIST 1/7 images with one flipped feature or label (FL1 perturbation) as reported in BagFlip.

As discussed in section 2, existing work on training-time certification is based on bagging and randomized smoothing. These are fundamentally different certification techniques with different assumptions on the threat model, which make a direct comparison to our approach impossible. To illustrate this difference, we compare FullCert to BagFlip [36]. (1) BagFlip guarantees robustness against ℓ_0-"norm" perturbations, which means robustness against flipping a small percentage of features or labels of the training or test data. In contrast, FullCert guarantees robustness against ℓ_∞-norm perturbations, which means small changes to all features of the training and test data. Ideally, machine learning models should be robust against both. Figure 4 illustrates that each method succeeds in its perturbation model, but does not transfer to the other. (2) FullCert computes deterministic guarantees, which always hold. In contrast, BagFlip's guarantees only hold with high probability, which means a small chance remains that the guarantee is invalid. (3) BagFlip requires training and evaluation of 1000 models for the same task. In contrast, FullCert only trains a single model with bound propagation. (4) FullCert evaluates the robustness of a normal model as if it was trained regularly, while BagFlip changes the underlying model into an ensemble, which results in different predictions. (5) BagFlip only works on discretized features, while FullCert can handle both discretized and standard, real-valued features. All of these differences make it impossible to compare the two methods directly. We believe there is merit in exploring both techniques for training-time certification, analogous to how the community explores both approaches for test-time certification concurrently.

6 Discussion & Limitations

End-to-end certification of networks is a crucial yet difficult problem. We propose FullCert, the first deterministic certifier, which also extends its guarantees to model training. As any method addressing challenging problems, FullCert comes with some limitations.

One challenge common to all certification methods is limited scalability to larger models. This is due to the high computational complexity of the certification problem. Even a single inference pass is already NP-complete [16], which is exacerbated for training-time certification with multiple forward and backward passes. We address this using fast interval relaxations, which make the computation feasible at the cost of over-approximations. These over-approximations are currently the main limiting factor of our approach, leading to small certified radii. They could be addressed with more precise relaxations in future work. However, even in its current form, FullCert can be used to assess the stability of training against perturbations in smaller-scale settings.

The theoretical guarantees and bounds of FullCert are sound, as shown in section 3. Due to the underlying PyTorch framework not supporting sound floating-point arithmetic, the implementation may suffer from numerical imprecision. We believe that the effects are negligible in practice, which is supported by prior work with the same limitation [1, 34, 35].

Finally, the algorithm and threat model we consider are challenging. The work on poisoning attacks shows that training is very sensitive to perturbations [29]. The threat model gives the adversary control over all inputs, which is a worst-case assumption. The training instability and the adversary's high degree of freedom limit current guarantees to small perturbation radii compared to test-time certificates. We see great potential for tighter guarantees using more robust training algorithms and limiting the adversary's control to a subset of the data.

7 Conclusion

This work addresses the challenging problem of end-to-end robustness certificates against training-time and inference-time attacks. We formally define the certification problem and propose FullCert, the first certifier that can jointly compute sound, deterministic bounds for both types of attacks. The theoretical analysis shows the soundness and convergence behavior of this method. Our new open-source library BoundFlow allows the implementation and experimental evaluation of FullCert, demonstrating its feasibility.

Acknowledgments

This work was partially funded by ELSA - European Lighthouse on Secure and Safe AI funded by the European Union under grant agreement number 101070617, as well as the German Federal Ministry of Education and Research (BMBF) under the grant AIgenCY (16KIS2012), and Medizininformatik-Plattform "Privatsphären-schützende Analytik in der Medizin" (PrivateAIM), grant number 01ZZ2316G. MK received funding from the ERC under the European Union's Horizon 2020 research and innovation programme (FUN2MODEL, grant agreement number 834115).

Bibliography

[1] Boopathy, A., Weng, T.W., Chen, P.Y., Liu, S., Daniel, L.: Cnn-cert: An efficient framework for certifying robustness of convolutional neural networks. In: AAAI Conference on Artificial Intelligence (AAAI) (2019)

[2] Bose, A., Udell, M., Lessard, L., Fazel, M., Dvijotham, K.D.: Certifying robustness to adaptive data poisoning. In: ICML Workshop: Foundations of Reinforcement Learning and Control–Connections and Perspectives (2024)

[3] Bottou, L.: Online learning and stochastic approximations. On-Line Learning in Neural Networks (1998)

[4] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. In: Advances in neural information processing systems (NeurIPS) (2020)

[5] Carlini, N., Tramer, F., Dvijotham, K.D., Rice, L., Sun, M., Kolter, J.Z.: (certified!!) adversarial robustness for free! In: International Conference on Learning Representations (ICLR) (2023)

[6] Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Symposium on Principles of programming languages (POPL) (1977)

[7] Diep, N.H.: Efficient implementation of interval matrix multiplication. In: International Conference on Applied Parallel and Scientific Computing (2010)

[8] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N.: An image is worth 16x16 words: Transformers for image recognition at scale. In: International Conference on Learning Representations (ICLR) (2021)

[9] Ferrari, C., Mueller, M.N., Jovanović, N., Vechev, M.: Complete verification via multi-neuron relaxation guided branch-and-bound. In: International Conference on Learning Representations (ICLR) (2022)

[10] Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: Ai2: Safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy (S&P) (2018)

[11] Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. MIT press (2016)

[12] Goodfellow, I., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: International Conference on Learning Representations (ICLR) (2015)

[13] Gowal, S., Dvijotham, K., Stanforth, R., Bunel, R., Qin, C., Uesato, J., Arandjelovic, R., Mann, T., Kohli, P.: On the effectiveness of interval bound propagation for training verifiably robust models. arXiv preprint arXiv:1810.12715 (2018)

[14] Jia, J., Cao, X., Gong, N.Z.: Intrinsic certified robustness of bagging against data poisoning attacks. In: AAAI Conference on Artificial Intelligence (AAAI) (2021)

[15] Jia, J., Liu, Y., Cao, X., Gong, N.Z.: Certified robustness of nearest neighbors against data poisoning and backdoor attacks. In: AAAI Conference on Artificial Intelligence (AAAI) (2022)

[16] Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient smt solver for verifying deep neural networks. In: International Conference on Computer Aided Verification (CAV) (2017)

[17] Koh, P.W., Steinhardt, J., Liang, P.: Stronger data poisoning attacks break data sanitization defenses. Machine Learning (2022)

[18] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE (1998)

[19] Levine, A., Feizi, S.: Deep partition aggregation: Provable defenses against general poisoning attacks. In: International Conference on Learning Representations (ICLR) (2021)

[20] Mirman, M., Gehr, T., Vechev, M.: Differentiable abstract interpretation for provably robust neural networks. In: International Conference on Machine Learning (ICML) (2018)

[21] OpenAI: Chatgpt: Optimizing language models for dialogue. https://openai.com/blog/chatgpt/ (2022), accessed: 2023-05-17

[22] Papernot, N., McDaniel, P., Sinha, A., Wellman, M.: Sok: Security and privacy in machine learning. In: European Symposium on Security and Privacy (EuroS&P) (2018)

[23] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Advances in Neural Information Processing Systems (NeurIPS) (2019)

[24] Rosenfeld, E., Winston, E., Ravikumar, P., Kolter, Z.: Certified robustness to label-flipping attacks via randomized smoothing. In: International Conference on Machine Learning (ICML) (2020)

[25] Rump, S.M.: Fast and parallel interval arithmetic. BIT Numerical Mathematics (1999)

[26] Schwarzschild, A., Goldblum, M., Gupta, A., Dickerson, J.P., Goldstein, T.: Just how toxic is data poisoning? a unified benchmark for backdoor and data poisoning attacks. In: International Conference on Machine Learning (ICML) (2021)

[27] Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. In: Proceedings of the ACM on Programming Languages (PACMPL) (2019)

[28] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I.J., Fergus, R.: Intriguing properties of neural networks. In: International Conference on Learning Representations (ICLR) (2014)

[29] Tian, Z., Cui, L., Liang, J., Yu, S.: A comprehensive survey on poisoning attacks and countermeasures in machine learning. ACM Computing Surveys (2022)

[30] Wang, B., Cao, X., Gong, N.Z., et al.: On certifying robustness against backdoor attacks via randomized smoothing. arXiv preprint arXiv:2002.11750 (2020)

[31] Wang, W., Feizi, S.: Temporal robustness against data poisoning. In: Advances in Neural Information Processing Systems (NeurIPS) (2023)

[32] Wang, W., Levine, A.J., Feizi, S.: Improved certified defenses against data poisoning with (deterministic) finite aggregation. In: International Conference on Machine Learning (ICML) (2022)

[33] Weber, M., Xu, X., Karlaš, B., Zhang, C., Li, B.: Rab: Provable robustness against backdoor attacks. In: IEEE Symposium on Security and Privacy (S&P) (2023)

[34] Weng, L., Zhang, H., Chen, H., Song, Z., Hsieh, C.J., Daniel, L., Boning, D., Dhillon, I.: Towards fast computation of certified robustness for relu networks. In: International Conference on Machine Learning (ICML) (2018)

[35] Zhang, H., Weng, T.W., Chen, P.Y., Hsieh, C.J., Daniel, L.: Efficient neural network robustness certification with general activation functions. Advances in neural information processing systems (NeurIPS) (2018)

[36] Zhang, Y., Albarghouthi, A., D'Antoni, L.: Bagflip: A certified defense against data poisoning. In: Advances in Neural Information Processing Systems (NeurIPS) (2022)

[37] Zhong, H., Liao, C., Squicciarini, A.C., Zhu, S., Miller, D.: Backdoor embedding in convolutional neural network models via invisible perturbation. In: Conference on Data and Application Security and Privacy (CODASPY) (2020)

Appendix

A Interval Relaxations

This section provides additional information on the constraints introduced in section 3.3, as well as proofs for the soundness and tightness of these bounds.

A.1 Basic Operations

The most basic form of abstract operation on these sets is the standard interval arithmetic, with the same soundness concept as introduced before. Given two intervals $\underline{\overline{A}}, \underline{\overline{B}} \in \mathbb{I}^m$, we can compute their sum and differences as

$$\begin{aligned} \underline{\overline{A}} + \underline{\overline{B}} &= [\underline{A} + \underline{B}, \overline{A} + \overline{B}] \\ \underline{\overline{A}} - \underline{\overline{B}} &= [\underline{A} - \overline{B}, \overline{A} - \underline{B}]. \end{aligned} \tag{17}$$

Similar interval operations can be defined for multiplication and division:

$$\begin{aligned} \underline{\overline{A}} * \underline{\overline{B}} = \big[&\min\{\underline{A} * \underline{B}, \underline{A} * \overline{B}, \overline{A} * \underline{B}, \overline{A} * \overline{B}\}, \\ &\max\{\underline{A} * \underline{B}, \underline{A} * \overline{B}, \overline{A} * \underline{B}, \overline{A} * \overline{B}\}\big] \\ \frac{\underline{\overline{A}}}{\underline{\overline{B}}} = \underline{\overline{A}} * \frac{1}{\underline{\overline{B}}}, &\qquad \frac{1}{\underline{\overline{B}}} = \left[\frac{1}{\overline{B}}, \frac{1}{\underline{B}}\right], 0 \notin \underline{\overline{B}}. \end{aligned} \tag{18}$$

A.2 Monotonic Functions

Monotonically increasing or decreasing functions can be evaluated at the bounds of the input interval. For a monotonically increasing function f:

$$f(\underline{\overline{x}}) = [f(\underline{x}), f(\overline{x})]. \tag{19}$$

This directly follows from the monotonic property, i.e., $\forall a, b : a \leq b \Rightarrow f(a) \leq f(b)$:

$$\begin{aligned} \forall u \in \underline{\overline{x}} : u \leq \overline{x} &\Rightarrow f(u) \leq f(\overline{x}) \\ \underline{x} \leq u &\Rightarrow f(\underline{x}) \leq f(u) \end{aligned}$$

□

These bounds are also tight, as they are realized by the upper and lower bounds of the input.

For monotonically decreasing functions, the opposite holds with the same argument:

$$f(\underline{\overline{x}}) = [f(\overline{x}), f(\underline{x})]. \tag{20}$$

A.3 Cross-Entropy Loss

The typical loss function to train classifiers is the cross-entropy-loss:

$$\begin{aligned} J(x, y) &= -\sum_{i=1}^{m} y_i \log(p_i) \\ p_i := \text{softmax}_i(z) &= \frac{e^{z_i}}{\sum_{j=1}^{m} e^{z_j}} \end{aligned} \tag{21}$$

Computing the last-layer softmax function followed by the cross entropy loss is numerically unstable since $\exp(z) = \inf \; \forall z > 88$ and $\exp(z) = 0.0 \; \forall z < -104$. This is already a challenge in regular model training, but bound-based training is especially sensitive.

Deep learning frameworks, therefore, combine the two operations into a single, numerically stable operation using the logsoftmax trick [11]:

$$\text{softmax}_c(z) = \frac{\exp(z_c)}{\sum_{i=1}^{m} \exp(z_i)} = \frac{\exp(z_c - a)}{\sum_{i=1}^{m} \exp(z_i - a)} \tag{22}$$

$$\begin{aligned} \text{logsoftmax}_c(z) &= \log\left(\frac{\exp(z_c)}{\sum_{i=1}^{m} \exp(z_i)}\right) \\ &= (z_c - a) - \log\left(\sum_{i=1}^{m} \exp(z_i - a)\right) \end{aligned} \tag{23}$$

$$a = \max_{i=1..m} z_i$$

Naïve bounds for softmax would be

$$\begin{aligned} \text{softmax}_c(\overline{\underline{z}}) &= \left[\frac{\exp(\underline{z}_c)}{\sum_{i=1}^{m} \exp(\overline{z}_i)}, \frac{\exp(\overline{z}_c)}{\sum_{i=1}^{m} \exp(\underline{z}_i)}\right] \\ &= \left[\frac{\exp(\underline{z}_c - a)}{\sum_{i=1}^{m} \exp(\overline{z}_i - a)}, \frac{\exp(\overline{z}_c - a)}{\sum_{i=1}^{m} \exp(\underline{z}_i - a)}\right] \end{aligned} \tag{24}$$

However, these bounds are not tight. The upper and lower bounds of z_c are used in the same term but can never be realized simultaneously. Also, using a single offset a does not fully mitigate the stability issue, as the terms in the upper and lower bounds can be quite different for larger intervals.

We, therefore, refine the solution with tight upper and lower bounds:

$$\begin{aligned} \text{softmax}(\overline{\underline{z}}) &= \left[\frac{\exp(\underline{z}_c)}{\sum_{i \neq c} \exp(\overline{z}_i) + \exp(\underline{z}_c)}, \frac{\exp(\overline{z}_c)}{\sum_{i \neq c} \exp(\underline{z}_i) + \exp(\overline{z}_c)}\right] \\ &= \left[\frac{\exp(\underline{z}_c - a_l)}{\sum_{i \neq c} \exp(\overline{z}_i - a_l) + \exp(\underline{z}_c - a_l)}, \frac{\exp(\overline{z}_c - a_u)}{\sum_{i \neq c} \exp(\underline{z}_i - a_u) + \exp(\overline{z}_c - a_u)}\right] \\ a_l &= \max\{\underline{z}_c, \overline{z}_{i \neq c}\}, \;\; a_u = \max\{\overline{z}_c, \underline{z}_{i \neq c}\}. \end{aligned} \tag{25}$$

We prove soundness through monotonicity. Softmax is monotonically increasing in z_c:

$$\operatorname{softmax}(z) = \frac{\exp(z_c)}{\sum_{i=1}^{m} \exp(z_i)} = \frac{\exp(z_c)}{\exp(z_c) + b} = 1 - \frac{b}{b + \exp(z_c)}, b \in \mathbb{R}_{\geq 0}, \tag{26}$$

and monotonically decreasing in $z_{i \neq c}$:

$$\operatorname{softmax}(z) = \frac{\exp(z_c)}{\sum_{i=1}^{m} \exp(z_i)} = \frac{b_1}{\exp(z_i) + b_2}, \{b_1, b_2\} \in \mathbb{R}_{\geq 0}. \tag{27}$$

Therefore, using eq. (19) and eq. (20), our bounds are sound and tight.

The same concept can be extended to the logsoftmax function:

$$\begin{aligned}
&\operatorname{logsoftmax}(\underline{\overline{z}}) \\
&= \left[\underline{z}_c - \log\left(\sum_{i \neq c}^{m} \exp(\overline{z}_i) + \exp(\underline{z}_c)\right), \overline{z}_c - \log\left(\sum_{i \neq c}^{m} \exp(\underline{z}_i) + \exp(\overline{z}_c)\right)\right] \\
&= \left[(\underline{z}_c - a_l) - \log\left(\sum_{i \neq c}^{m} \exp(\overline{z}_i - a_l) + \exp(\underline{z}_c - a_l)\right),\right. \\
&\qquad \left.(\overline{z}_c - a_u) - \log\left(\sum_{i \neq c}^{m} \exp(\underline{z}_i - a_u) + \exp(\overline{z}_c - a_u)\right)\right] \\
&a_l = \max\{\underline{z}_c, \overline{z}_{i \neq c}\},\ a_u = \max\{\overline{z}_c, \underline{z}_{i \neq c}\}
\end{aligned} \tag{28}$$

The log function is monotonically increasing. That implies that log ∘softmax has the same monotonicity as softmax, which proves both the soundness and tightness of our bounds.

A.4 Binary Cross-Entropy

For binary classification problems, we use the binary cross-entropy (BCE) loss:

$$J(p, y) = -(y \log(p) + (1 - y) \log(1 - p),\ p = \sigma(z) = \frac{1}{1 + \exp(-z)}. \tag{29}$$

Combined with the sigmoid activation of the last layer, it has the same numeric stability issue as cross-entropy with softmax. The typical solution looks, therefore, similar:

$$\begin{aligned}
J(z, y) &= -(y \log(\sigma(z)) + (1 - y) \log(1 - \sigma(z)) \\
&= z - zy + \log(1 + \exp(-z)) \\
&= z - zy + a + \log(\exp(-a) + \exp(-z - a)) \\
a &= \max(-z, 0).
\end{aligned} \tag{30}$$

BCE is monotone in z. Therefore, we can directly define tight and sound bounds using eq. (19):

$$J(\underline{\overline{z}}, y) = [\min\{L(\underline{z}, y), L(\overline{z}, y)\}, \max\{L(\underline{z}, y), L(\overline{z}, y)\}] \tag{31}$$

For the backwards pass, we get the derivative of BCE as

$$\frac{\partial J}{\partial z}(z) = \sigma(z) - y \tag{32}$$

This function is, again, monotonically increasing in z. Therefore

$$\frac{\partial J}{\partial z}(\overline{\underline{z}}) = [\sigma(\underline{z}) - y, \sigma(\overline{z}) - y] \tag{33}$$

are tight and sound bounds.

B Additional Experiments

We supplement our main experiments in section 5 with additional experiments to analyze the influence of different aspects of model training on FullCert.

B.1 Complexity and Compute

Training with interval bounds has the same asymptotic complexity as regular model training. The overhead due to computing upper and lower bounds for each intermediate result is a constant factor.

In practice, the overhead is larger, as our BoundFlow library is not as heavily optimized as modern machine learning libraries like PyTorch. For practical comparison, training a fully connected network with two layers for 100 epochs on Two-Moons with FullCert takes 7.6 seconds, while the same model without bounds in PyTorch trains in 0.5 seconds. For MNIST 1/7, a single epoch with FullCert takes 2 seconds, while the same model without bounds in PyTorch trains in 0.38 seconds per epoch.

We run all experiments on a Linux machine with an 8-core 3.6 GHz CPU, 32GB of RAM, and an Nvidia Titan RTX GPU. For exact software versions, please refer to the environment file provided with the implementation.

B.2 Training Details

The experiments in section 5 use the following hyper-parameters as defaults:

- batch size: 100
- learning-rate: 0.01 Two-Moons, 0.05 MNIST 1/7
- max-epochs: 100
- architecture: 3-layer MLP with ReLU activation

For Two-Moons, we use a training set size of 1000 samples, and 200 samples for validation and test sets, respectively. For MNIST 1/7, we use the standard train and test splits of the full MNIST dataset, filtered to only include samples from classes 1 and 7. We use 2000 images from the training set for validation, which leaves us with 9831 samples for training and 1938 samples for the test set.

B.3 Influence of Hyper-Parameters

In this set of experiments, we evaluate the influence of different training parameters on FullCert. To this end, we train fully connected models on the Two-Moonsdataset without pertaining while varying different hyper-parameters.

ϵ	0.0001	0.001	0.01	0.1
certified accuracy	85.1 ± 3.7	83.7 ± 4.0	72.4 ± 11.7	36.8 ± 14.8

Table 1. Certified accuracy of 2-layer MLPs trained on Two-Moons for different ϵ. All values show mean and standard deviation over 10 independent training runs with different, randomly chosen seeds. For smaller perturbations budgets, the algorithm consistently converges toward an optimum. For large perturbations, some initializations cause training to converge due to one of the effects analyzed in section 4, leading to lower mean accuracy and higher variance.

The most influential hyper-parameter is the perturbation radius ϵ. ϵ is defined by the threat model, and it directly controls the adversary's capabilities. The larger ϵ, the stronger the attacker's potential influence on the prediction. Table 1 shows certified accuracies for 2-layer fully-connected models with ReLU activation with different perturbation budgets. The certified accuracy is averaged over 10 runs with different, randomly chosen seeds for initialization. For small ϵ, FullCert consistently converges towards an optimum with high certified accuracy. This changes for large ϵ, where some runs fail to converge depending on initialization. This is likely due to one of the effects discussed in section 4 and results in a lower average certified accuracy and higher standard deviation.

learning rate	0.01	0.1	1.0	5.0	10.0
certified acc.	71%	72%	75%	74%	77%

Table 2. Certified accuracy of 2-layer MLPs trained on Two-Moons with $\epsilon = 0.01$ and different learning rates. The experiments show convergence and good certified accuracy across four orders of magnitude.

Table 2 shows certified accuracy when trained with different learning rates. Training is stable across four magnitudes of learning rates with certified accuracies consistently above 70%. The increase in certified accuracy with higher training rates is likely due to the much faster convergence towards the optimum, which decreases the number of learning steps and therefore the number of accumulated over-approximations.

Table 3 shows certified accuracy when training with different batch sizes. The results are nearly identical, demonstrating that FullCert works independently of batch size.

batch size	50	100	500	1000
certified accuracy	72.0%	72.5%	72.5%	71.5%

Table 3. Certified accuracy of 2-layer MLPs trained on Two-Moons with $\epsilon = 0.01$ and different batch sizes. The results show that certified accuracy is independent of batch size.

architecture	certified accuracy
2 layer MLP, 10 hidden nodes per layer	75.5%
2 layer MLP, 20 hidden nodes per layer	72.0%
2 layer MLP, 30 hidden nodes per layer	69.5%
3 layer MLP, 30 hidden nodes per layer	61.0%

Table 4. Certified accuracy for MLPs with different architectures trained on Two-Moons with $\epsilon = 0.01$. The larger the model, the lower the certified accuracy due to the increasing imprecisions caused by over-approximations.

Lastly, we show that FullCert works on different model architectures. Table 4 shows certified accuracy for MLPs with different layer and node counts. In general, the larger the model, the lower the certified accuracy. This is due to the fact that the effects of over-approximation increase with increasing model size, and the bounds become less precise as a result.