Getting a-Round Guarantees: Floating-Point Attacks on Certified Robustness

Jiankai Jin
The University of Melbourne
Melbourne, Australia
jiankaij@student.unimelb.edu.au

Olga Ohrimenko*
The University of Melbourne
Melbourne, Australia
oohrimenko@unimelb.edu.au

Benjamin I. P. Rubinstein*
The University of Melbourne
Melbourne, Australia
brubinstein@unimelb.edu.au

ABSTRACT
Adversarial examples pose a security risk as they can alter decisions of a machine learning classifier through slight input perturbations. Certified robustness has been proposed as a mitigation where given an input \( x \), a classifier returns a prediction and a certified radius \( R \) with a provable guarantee that any perturbation to \( x \) with \( R \)-bounded norm will not alter the classifier’s prediction. In this work, we show that these guarantees can be invalidated due to limitations of floating-point representation that cause rounding errors. We design a rounding search method that can efficiently exploit this vulnerability to find adversarial examples against state-of-the-art certifications in two threat models, that differ in how the norm of the perturbation is computed. We show that the attack can be carried out against linear classifiers that have exact certifiable guarantees and against neural networks that have conservative certifications. In the weak threat model, our experiments demonstrate an attack success rate of 50% on random linear classifiers, up to 23% on the MNIST dataset for linear SVM, and up to 15% for a neural network. In the strong threat model, the success rates are lower but positive. The floating-point errors exploited by our attacks can range from small to large (e.g., \( 10^{-13} \) to \( 10^{3} \)) — showing that even negligible errors can be systematically exploited to invalidate guarantees provided by certified robustness. Finally, we propose a formal mitigation approach based on rounded interval arithmetic, encouraging future implementations of robustness certificates to account for limitations of modern computing architecture to provide sound certifiable guarantees.

CCS CONCEPTS
• Security and privacy → Formal methods and theory of security; • Computing methodologies → Machine learning.

KEYWORDS
Certified Robustness, Floating-Point Errors, Adversarial Examples

1 INTRODUCTION
Robustness of modern image classifiers has come under scrutiny due to a plethora of results demonstrating adversarial examples—small perturbations to benign inputs that cause models to mispredict, even when such perturbations are not evident to the human eye [4, 9, 21, 24]. If a learned model is used in critical applications such as self-driving cars, clinical settings or malware detection, such easily added perturbations can have severe consequences. As a result, research focus has shifted to training models robust to adversarial perturbations, that come endowed with certified robustness.

Mechanisms for providing robustness certification aim to bound a model \( f \)'s sensitivity to a certain level of perturbation. At a high level, such mechanisms return a radius \( R \) around a test input \( x \) with a guarantee that for any \( x' \) within \( R \) distance from \( x \), \( f(x) = f(x') \). How \( R \) is computed, whether it is sound and/or complete depends on the mechanism. For example, bound propagation [28, 29] transfers the upper and lower bounds from the output layer to the input layer of a neural network, and gives a lower bound on the perturbation needed to flip the classification.

Given the extensive research on certified robustness, can such mechanisms protect against adversarial examples in practice? In this paper, we show that the limits posed by floating-point arithmetic invalidate guarantees of several prominent mechanisms and their implementations. Despite proofs of robustness guarantees, they mostly assume real numbers can be represented exactly. Unfortunately, this critical (implicit) assumption cannot hold on computers with finite number representations. Since floating-point (FP) numbers can represent only a subset of real values, rounding is likely to occur when computing robust guarantees and can cause overestimation of the certified radius \( R \). Thus, adversarial examples may exist within the computed radius despite claims of certification.

We devise a rounding search method that can efficiently discover such adversarial examples in two threat models, that differ in how the norm of the perturbation is computed. Our method is inspired by the traditional adversarial example search methods such as PGD [21] and C&W [4]. However, we find that such existing methods do not effectively exploit the rounding of a certified radius as the search space they explore is large (i.e., the number of examples to check becomes intractable due to the large number of floating-point values) and instances of inappropriate rounding do not necessarily follow model gradients. To this end, our method is different from these search methods in two aspects: (1) instead of relying on back propagation, it leverages the piecewise linear property of ReLU networks to find coarse-level perturbation directions;
it then searches in a much finer scale by sampling floating-point neighbors of a potential adversarial example. The first aspect allows us to narrow down the search space closer to the certified radius and efficiently find adversarial examples. The second aspect enables our search method to find adversarial examples with perturbation norms that are just smaller than the certified radius (e.g., in the 13th decimal place), which PGD and C&W cannot find. Compared to other works that find robustness violations [17, 30], our attack method is arguably stronger as it works on unmodified target models with unaltered instances as opposed to specially crafted models or instances. We discuss the potential impact of our attacks on robustness guarantees in Section 6.2.

One’s first intuition to mitigate the overestimation of certified radii exploited by the above attacks might be to adopt slightly more conservative radii (e.g., using $R - \gamma$ for some positive constant $\gamma$, e.g., $\gamma = 1.0$). Unfortunately, such radii are not in general sound and choosing $\gamma$ is inherently error prone. That is, we show that the amount of overestimation can depend on the data (e.g., number of features) and model (e.g., number of operations) and that attacking $R - 1.0$ is still possible. To this end, we propose a defense based on rounded interval arithmetic that has soundness guarantees and can be easily integrated into mechanisms for computing certified radii. In summary our contributions are:

- We explore a class of attacks that invalidate the implementations of certified robustness (i.e., find adversarial examples within the certified radius). Our attacks exploit rounding errors due to limited floating-point representation of real numbers.
- We devise a rounding search method that systematically discovers such adversarial examples under two threat models. The weak model assumes that attacks need only have floating-point norms that violate certifications (e.g., in the case where the norm is computed using common software libraries). The strong model makes no such assumption: the true (real-valued) norm of attacks must violate certifications (e.g., in the case where the library that computes the square root for the norm is computed using common software libraries). The norm is inherently error prone. That is, we show that the amount of overestimation can depend on the data (e.g., number of features) and model (e.g., number of operations) and that attacking $R - 1.0$ is still possible. To this end, we propose a defense based on rounded interval arithmetic that has soundness guarantees and can be easily integrated into mechanisms for computing certified radii.
- We show that our attacks work against exact certifications of linear models [5], and against a conservative certified radius returned by a prominent neural network verifier [28] on a network. Our attack success rate differs between linear and neural networks. In the weak threat model, our success rates are over 50% for random linear classifiers and 15% on an MNIST neural network model. In the strong threat model, the attack success rates are lower but are still non-zero. For all cases, in theory, the certification should guarantee a 0% success rate for such attacks within certified radii.
- We propose a defense based on rounded interval arithmetic, with strong theoretical and empirical support for mitigating rounding search attacks.

2 BACKGROUND AND PRELIMINARIES

Let input instance $x = (x_1, x_2, \ldots, x_D)$ be a vector in $\mathbb{R}^D$ with $x_i$ denoting the $i$th component of $x$. We adopt the $\ell_2$ norm $\|x\|_2 = \left(\sum_{i=1}^{D} x_i^2\right)^{1/2}$, written $\|x\|$ where the norm is understood from context, when measuring the length of instance vectors, and its induced metric $\|x - z\|$ for distances. We also use the $\ell_1$ norm $\|w\|_1 = \sum_{i=1}^{D} |w_i|$. We consider the task of learning a classifier $f$ mapping an instance in $\mathbb{R}^D$ to a label in $\{-1, 1\}$ or to a $K$-class label in $|K| = \{1, \ldots, K\}$. Given a training set of $N$ examples $(x^{(j)}, y^{(j)}) \in \mathbb{R}^D \times \{-1, 1\}$ drawn i.i.d. from some unknown distribution $P$, the goal of a learner $A$ is to output some classifier $f$ with low risk $\mathbb{E}_P[\ell(f(X), Y)]$ for some loss $\ell(\cdot)$ of interest.

Linear models. We consider learners $A$ over two-class linear classifiers of the form $f(x) = \text{sign}(w^T x + b)$, for parameters $w \in \mathbb{R}^D$ the model weights and $b \in \mathbb{R}$ model bias. Specifically we consider as learners: (1) uniformly random sampling of $(w, b)$ within a closed, bounded set, without consideration of training data; (2) the linear support vector machine that minimizes the $\ell_1$-regularized squared hinge loss $\sum_{i=1}^{n} (1 - y^{(i)}(w^T x^{(i)} + b))^2 + \lambda\|w\|_1$. Classifiers resulting from learners (1) and (2) directly make predictions as $f(x) = \text{sign}(w^T x + b)$.

Neural networks. We consider fully-connected feed-forward neural networks, which are known to be universal approximators [13]. Such a network forms output scores by composing a sequence of layers $F_1(\cdot) : \mathbb{R}^x \rightarrow \mathbb{R}^y$ as: $F(x) = (F_n \circ F_{n-1} \circ \cdots \circ F_1)(x)$. The neural network forms its classification within classes $|K| = \{1, \ldots, K\}$ by simple majority vote $f(x) = \arg\max_{k \in |K|} F(x)_k$. Layers might take the form $F_i(x) = (\sigma(x^i + b_i))$ for some weights $\theta_i$ and vector of model biases $b_i$. For activation function $\sigma$ we consider either linear $\sigma(x) = x$ or rectified linear units (ReLUs) $\sigma(x) = \text{ReLU}(x) = \max(0, x)$. Gradients of neural network outputs can be computed with respect to weights (for training) or inputs (for finding adversarial examples—see next) using backpropagation as typically implemented by reverse accumulation in autodiff libraries.

Adversarial examples. Given an input instance $x$, a classifier $f$, and a target label $t \neq f(x)$, $x'$ is a targeted adversarial example [24] if $f(x') = t$ where $x'$ is reachable from $x$ according to some chosen threat model. In the vision domain, it is common to assume that small $\ell_p$ perturbations to $x$ will go unnoticed by human observers. In this paper we consider $\ell_2$ distance, i.e., $\|x - x'\|_2 \leq \Delta$ for some small perturbation limit $\Delta$. An adversarial example in the multi-class setting is untargeted if $t$ is not specified, however, it is possible to find an $x'$ such that $f(x') \neq f(x)$ while $x'$ and $x$ are still close. Appropriate methods for finding adversarial examples depend on the model under attack. For a two-class linear classifier $f$, its weight vector $w$ is normal to the decision boundary $w^T x + b = 0$. Therefore, there always exists a closest adversarial example $x'$ just beyond the decision boundary in the direction $w$. See Figure 1 for an illustration. For neural networks, and non-linear classifiers in general, the process is more involved. Two popular white-box approaches are due to Carlini and Wagner (C&W) [4] and Madry et al. called Projected Gradient Descent (PGD) [21]. These methods assume white-box access to the models (that is, access to architectures and model weights) and hence an attacker can run as many queries as they want and observe intermediate gradients on inputs of their choice. At a high level, PGD and C&W view a search for an adversarial example as an optimization problem where one tries to find $x'$ with a flipped label while minimizing the distance to $x$, and differ in how the perturbation norm $\Delta$ is updated during the search.
Floating-point representation. Floating-point values represent reals using three binary numbers: a sign bit b, an exponent e, and a significand $d_1d_2 \ldots d_8$. For example, 64-bit (double precision) floating-point numbers allocate 1 bit for b, 11 bits for e, and 52 bits for the significand. Such a floating-point number is defined to be $(-1)^b \times (1.d_1d_2 \ldots d_8) \times 2^{e-1-1023}$. Floating points can represent only a finite number of real values. Hence, computations involving floating-point numbers often need to be rounded up or down to their nearest floating-point representation, as specified by the IEEE 754 Standard for Floating-Point Arithmetic [14].

2.1 Certified robustness

A robustness certification for a classifier at input $x$ is a neighborhood (typically an $\ell_2$ ball) of $x$ on which classifier predictions are constant. Certifications aim to guarantee that no perturbed adversarial examples exist in this neighborhood, including "slightly" perturbed instances.

**Definition 1.** A pointwise robustness certification for a $K$-class classifier $f$ at input $x \in \mathbb{R}^D$ is a real radius $R > 0$ that is sound and (optionally) complete:

(i) [sound] $\forall x' \in \mathbb{R}^D, ||x' - x|| \leq R \Rightarrow f(x') = f(x)$.

(ii) [complete] $\exists x' \in \mathbb{R}^D, ||x' - x|| \leq R \wedge f(x') \neq f(x)$.

For a given certification mechanism, we will distinguish the idealized certification radius $R$ (i.e., the mapping of Definition 1 under the soundness condition) from a candidate radius $\tilde{R}$ that an implementation of this mechanism computes. As we will see, the latter may not be necessarily sound (or complete).

We categorize certification mechanisms in the literature depending on their specific claims, summarized as follows:

**Exact mechanisms:** claim to output sound and complete radii.

**Conservative mechanisms:** claim to output sound radii that are not necessarily complete.

**Approximate mechanisms:** claim to output random radii that are sound (or abstain), with high probability $1 - \alpha$, and that are not necessarily complete.

We next go into more detail on each category of certification mechanism.

**Exact certification mechanisms.** These mechanisms output sound and complete radii under ideal realization of $\mathbb{R}$ arithmetic. Binary linear classifiers $f(x) = sign(w^T x + b)$ admit an exact certified radius $R = ||w^T x + b|| / ||w||$. Cohen et al. [5] derive this radius and prove its soundness [5, Proposition 4] and completeness [5, Proposition 5] for real arithmetic.

**Conservative certification mechanisms.** These mechanisms are sound and not necessarily complete under real-valued arithmetic. These mechanisms usually view neural network certification as an optimization problem: for input $x$ with $f(x) > 0$, and candidate radius $R$, let

$$f^* = \min_{x' \in \mathbb{R}^D} f(x')$$

s.t. $||x' - x|| \leq R$.

If $f^* > 0$, then $R$ can be certified for $f$ on $x$. In general, this is a non-convex problem due to the non-linear activations used in typical choices of $f$. Hence, conservative certification mechanisms usually relax the non-convexity of neural networks to estimate a tractable lower bound $f \leq f^*$, e.g., approximating non-linear activations with linear functions [3, 7, 26, 28, 29]. If $f > 0$, then $f^* > 0$ and $f$ is certified. Bound propagation [28, 29] aims to estimate $f$ to a certain input, by maintaining at each layer an upper approximation of the set of activation functions and propagating bounds from the output layer to the input layer. Linear programming [3, 7, 26] is another approach that also relaxes non-convex constraints into linear constraints, and solves the certification problem as a linear programming problem. We used $\beta$-Crown and Gurobi MIP solver as example applications of bound propagation and linear programming respectively in Section 4.3.

**Approximate mechanisms.** Approximate certifications output random radii that under $\mathbb{R}$ are sound (or abstain), with high probability $1 - \alpha$. Such mechanisms are not necessarily complete. Researchers have studied approximate certification of non-linear models such as neural networks, under the $\ell_2$ norm [5, 19, 20]. In this work, we consider the randomized smoothing approach of Cohen et al. [5]. Sufficiently stable model predictions lead to certifiable radii; therefore we seek to stabilize the outputs of the base classifier $f$ by forming a smoothed classifier $g$ as follows. To input $x \in \mathbb{R}^D$ add isotropic Gaussian noise, then apply $f$. The input distribution induces a distribution over predictions. Smoothed classifier $g$ then outputs the most likely class under this induced distribution. That is, for

$$\epsilon \sim N(0, \sigma^2 \epsilon^2),$$

$$g(x) = \arg \max_{k \in [K]} \Pr(f(x + \epsilon) = k).$$

Consider lower (upper) bounds on the winning $k^*$ classification’s probability score,

$$\Pr(f(x + \epsilon) = k^*) \geq p_A \geq \Pr(f(x + \epsilon) = k).$$

Then Cohen et al. [5] prove that radius $R = 0.5 \sigma \Pr(\Phi^{-1}(p_A) - \Phi^{-1}(p_B))$ is a sound certification for smoothed classification $g(x)$, under real-valued arithmetic. However the floating-point calculation of a corresponding $\tilde{R}$ is likely to experience some degree of rounding, potentially sufficient to erroneously certify some $\tilde{R} > R$.

3 ROUNDING SEARCH ATTACK

We now present a rounding search method that exploits floating-point rounding errors to find adversarial examples within a computed certified radius $\tilde{R}$.

**Threat model.** Like prior works on adversarial examples [4, 17, 21], we assume that the adversary has white-box access to a classifier $f$, and has white-box access to the certification mechanism that it can query with inputs $f$ and instance $x \in \mathbb{R}^D$, and obtain a certified radius $\tilde{R}$ as an output.

Since there are floating-point rounding errors in the operations for computing a certification, the computed radius $\tilde{R}$ at an instance $x$ could overestimate an intended sound (and possibly complete)
radius $\hat{R} < R$. This creates leeway for an adversary to find adversarial perturbations whose norms are less than or equal to the computed certified radius, but which can change the classifications of the model, invalidating soundness of the computed certification. Our work aims to find a systematic and efficient way to exploit these rounding errors.

A perturbation’s norm $\|\delta\| = \|x' - x\|$ must be estimated when evaluating the perturbation’s success. This norm computation can also suffer floating-point rounding errors, and could be underestimated. To handle this possibility, we conduct attacks in two threat models, one weak one strong.

The weak model makes minimal assumptions on how certification is violated: an attack is ruled successful if the floating-point computation of $\|\delta\|$ is smaller than or equal to $\hat{R}$ (i.e., $\|\delta\| \leq \hat{R}$). We note that this model represents settings where the norm is computed using common software libraries for computing operations on floating numbers (e.g., Numpy’s 32-bit or 64-bit floating-point arithmetic).

The strong model does not make these assumptions, the true (real-valued) norm of attacks must violate certifications. This model considers a setting where the norm is computed using software packages that instead of returning a result that is potentially rounded can return a representation of a real-valued norm or its range. Since we cannot do real arithmetic on machines, we use the upper bound of the norm $\|\delta\|$ instead, which is computed with rounded interval arithmetic and is guaranteed to be greater than or equal to the true norm. That is, a successful attack satisfies $\|\delta\| \leq \hat{R}$, which guarantees that $\hat{R}$ must necessarily exceed the real-valued norm of the attack.

**Figure 1:** The search direction $v$ (blue line) and search area (green area) for finding adversarial examples against a model, whose decision boundary is the orange line. $x$ is the original instance, $\hat{R}$ and $R$ are the computed and real-valued certified radii of the model on $x$, $\delta = \hat{R} v / \|v\|$ is the adversarial perturbation in the search direction $v$, instance $x' = x + \delta$ is the seed for the green search area. Our rounding search method will sample $N$ floating-point neighbors $\delta'$ of $\delta$, and evaluate each $x + \delta'$ to check if any one of them can flip the classification of the model with $\|\delta'\| \leq \hat{R}$ or $\|\delta'\| \leq \hat{R}$ (the red points in the green search area).

**Attack overview.** Consider a classifier $f$, input $x$ and the corresponding computed radius $R$. A naive way to search for an adversarial example would be to try all $x'$ such that $\|x - x'\| \leq \hat{R}$, checking whether $f(x) \neq f(x')$. Unfortunately this exhaustive search is computationally intractable (e.g., there are $\approx 2^{17}$ floating points in a small interval such as $[10, 10 + 2^{32})$). We can avoid some futile search. For example, observe that instances in the gray area, as depicted in Figure 1, are unlikely to flip predictions, as they are in the opposite direction of the decision boundary. A key idea is to find a perturbation direction $v$ that reaches the decision boundary in the shortest distance, and add a perturbation $\delta$ in that direction to $x$, to maximize our chance to flip the classifier’s prediction with perturbation norm $\|\delta\|$ (or $\|\delta'\|$) less than or equal to $\hat{R}$. This baseline method has several challenges. First, computation of perturbation direction $v$ is not easy for NNs which do not typically have linear decision boundaries. To this end, for ReLU networks, we find a local linear approximation prior to computing the gradient for $v$.

In general one may use a standard approach to finding adversarial example directions such as PGD. Second, while $v$ guides a search towards the decision boundary, the search may still be unable to exploit the leeway between the real certified radius $R$ and the computed certified radius $\hat{R}$ to find certification violations. We address this challenge with a tightly-confined randomized floating-point neighborhood search.

In summary, our attack proceeds as follows (depicted in Figure 1).

1. Find an adversarial perturbation direction $v$ that reaches the decision boundary of classifier $f$ in the shortest distance, as a form of PGD attack [21] (Section 3.1).
2. Compute perturbation $\delta$ in the direction $v$ within the computed certified radius $\hat{R}$:
   \[
   \delta = \hat{R} v / \|v\|.
   \]
3. Search for multiple floating-point neighbors $\delta'$ of $\delta$ with $\|\delta'\| \leq \hat{R}$ (or $\|\delta'\| \leq \hat{R}$), and evaluate if any $x + \delta'$ can flip the classifier’s prediction (Section 3.2).

### 3.1 Adversarial perturbation direction

For linear models, direction $v$ is a normal to the decision boundary’s hyperplane $w^T x + b = 0$ and equals $w$. The perturbation direction for neural networks is not as obvious as it is for linear models, as the decision boundary can be highly non-linear. In general one can adopt a PGD-like approach for finding the attack direction. In the rest of this section we describe this approach for finding $v$ for neural networks with ReLU activations that we show to be effective in our experiments. This is an exact instantiation of PGD. The novelty of our attack comes later when we set the attack step size and incorporate a local floating-point search. A neural network with ReLUs can be represented as $F(x) = (F_K \circ F_{K-1} \circ \cdots \circ F_1)(x)$, where $F_i(x) = \text{ReLU}(\theta_i^T x + \hat{\theta}_i)$. Here $x$ and $\hat{\theta}_i$ are vectors, $\theta_i$ is a matrix, and the rectified linear (ReLU) activation function acts pointwise on a vector, returning a vector.

We use the fact that such networks are piecewise linear: therefore a (local) linear approximation at instance $x$ is in fact exact. Then, one can find an adversarial example for $x$ against this linear model as described above and use it to attack the original ReLU network.

**Warmup.** As a warmup, let us consider a network where ReLUs are all activated. For each node $\text{ReLU}(z) = \max(0, z) = z$, and so the network is a combination of $K$ linear models where $K$ is the
number of classes. That is,

\[ F(x) = \theta^T x + \hat{\theta}, \]

where \( \theta^T = \theta^T_0 \theta^T_n \cdots \theta^T_1 \), and \( \hat{\theta} = \sum_{i=1}^{n} \left( \prod_{j=1}^{n-i} \theta^T_j \right) \hat{\theta}_i \). Note that \( \theta^T \) is a \( K \times D \) matrix and \( \hat{\theta} \) is a column vector of length \( K \). Each class \( k \) corresponds to the linear model

\[ F_k(x) = w_k^T x + b_k, \]

where \( w_k^T = (\theta^T)_k \), is the \( k \)th row of \( \theta^T \) and \( b_k = \hat{\theta}_k \).

In order to change this model’s classification from the original class \( l \) to the target class \( t \neq l \), we observe that one can attack the following model:

\[ L(x) = F'(x) - F_l(x) = (w_l^T - w_t^T) x + b_l - b_t. \]

This is a linear model, and \( L(x) < 0 \) when \( F(x) \) classifies \( x \) as \( l \), \( L(x) > 0 \) when \( F(x) \) classifies \( x \) as \( t \), so \( L(x) \) has the decision boundary hyperplane \( L(x) = 0 \). Hence, the most effective perturbation direction to change classification of \( F(x) \) from \( l \) to \( t \), as before for linear models, is \( \nu = w_l^T - w_t^T \), which is the gradient of \( L(x) \) with respect to \( x \).

**Linear approximation of ReLU networks.** ReLUs will all be activated when the weights and biases of each hidden layer of the network are positive, and all values of the input are also positive (e.g., an image, whose pixel value is usually in the range \([0, 1]\)). However, in practice this usually is not the case and some ReLUs will not be activated. For inactive ReLUs, we modify outgoing weights to zero in the calculation of the perturbation direction \( \nu \).

The overall process, LinApproxPerturbDir, is described in Algorithm 1. It proceeds by first finding an exact (local) linear approximation \( F'(x) = \tau^T x + \hat{\tau} \) where \( \tau = \sum_{i=1}^{n} \left( \prod_{j=1}^{n-i} \tau^T_j \right) \hat{\theta}_i \) using the notation in the pseudo-code. The weights of \( F' \) are equal to weights of \( F \) for internal nodes where \( F(x) \) activated the corresponding ReLUs, otherwise they are set to 0. Specifically, we zero out columns of matrix \( \theta^T \) when the corresponding elements of mask \( \hat{\theta}_i \) are zero. Given these weights, LinApproxPerturbDir computes \( \nu \) as explained in the warmup. This direction corresponds to a gradient of the network’s target minus current class scores, with respect to the instance \( x \).

**Projected gradient descent for ReLU networks.** Given \( \nu \) as output by Algorithm 1 and a computed certified radius, one could compute adversarial perturbation \( \delta \) in direction \( \nu \) close to the certified radius as in Equation 1. However, the resulting \( x' = x + \delta \) may activate different ReLUs of \( F \) than \( x \). Hence, the linear approximation \( F' \) on \( x \) may be different to \( F' \) on \( x \); these approximations are only exact in local neighborhoods. To this end we perform a search by iteratively updating \( x' \) and invoking LinApproxPerturbDir until an adversarial example within the input domain \([V_{min}, V_{max}]\) is found or the procedure times out. Algorithm 2 describes this procedure, which we refer to as ReluPGD. The algorithm iteratively performs the following: computes the gradient of the network’s linearization at the current iteration, rescales to the step size \( s \), clips the perturbation to the domain constraint, applies the perturbation.

**Remark 1.** Note that \( \bar{R} \) may not be given, as is the case for some network verifiers that instead of returning \( \bar{R} \), take \( F, x \) and some \( R \) as input and either certify \( R \) or not. In this case, we need to search for the smallest perturbation in the direction of \( \nu \) to find such an \( R \) to attack. Hence, in Algorithm 2 we use \( s \) as an input, which is set to a small initial value (e.g., \( 10^{-5} \) in our experiments) so that \( \nu \) can be updated frequently. If a \( R \) is given, we can set it as a threshold value to stop the algorithm, that is, the algorithm should stop when the total perturbation norm reaches \( \bar{R} \).

### 3.2 Rounding search

Given the direction \( \nu \) and the computed certified radius \( \bar{R} \), an adversarial perturbation \( \delta \) can be computed using Equation 1, and \( x' = x + \delta \) should give an adversarial example so that \( F(x) \neq F(x') \).

If the accumulated rounding errors are large, \( \delta \) can be sufficient to conduct a successful attack (e.g., for neural networks with many neurons). For some attacks, the rounding errors we exploit are much
for linear classifier is \( R = |w^T x + b|/\|w\| \), we compute it ourselves using either 32-bit or 64-bit floating-point arithmetic in Numpy.

For linear classifiers, we will conduct attacks in both the weak and strong threat models. For neural networks, we will conduct attacks in the weak threat model. We show that our rounding search finds adversarial examples within certified radii for all of them. Our linear models are run on an Intel Xeon Platinum 8180M CPU, while our neural network models are run on a Tesla V100 16GB GPU.

Baseline attack rates. The baseline success rate for finding an adversarial example against a linear model within the radius defined in Section 2.1 should be 0% in both threat models, since the mechanism is exact: it claims to be both sound and complete. The baseline success rate for radii returned by conservative mechanisms should also be 0% since they too are claimed to be sound. Though randomized smoothing comes with a failure probability \( \alpha \ll 1 \) to account for sampling error in approximating a smoothed classifier, it does not explicitly take into account errors due to rounding.

Model training. We train (primal) linear SVM with sequential minimal optimization, by using corresponding modules of scikit-learn. Our linear classifiers are trained with \( \ell_1 \) regularization so that model weights are sparse, and perturbations are less likely to move images outside their legal domain (recall that the perturbation direction for a linear classifier is its weights \( v = w \)). All ReLU networks in this section are trained with the SGD optimizer using PyTorch, with momentum 0.9, learning rate 0.01, batch size 64, for 15 epochs. For some controlled experiments we require the weights and biases of the hidden layers to be positive (to activate all ReLUs). In this case weights and biases are clamped with the lower bound 0 after each step of training.

### 4.1 Random linear classifiers
To evaluate the performance of our attack in an ideal scenario, we first conduct our attack on randomly initialized (binary) linear classifiers with randomly generated target instances: \( f(x) = \text{sign}(w^T x + b) \), where weights \( w \) and bias \( b \) are random values drawn from the range \([-1, 1]\). Each value is represented with either 32-bit or 64-bit floating-point precision. For each dimension, we test 10 000 randomly initialized models. For each model we choose one instance \( x \) to attack, where each component \( x_i \) is drawn randomly from \([-1, 1]\). Hence, attack success rate measures the number of models out of 10 000 for which a random instance can result in a successful attack. For each combination of \((w, b, x)\), we sample and evaluate \( N = D^2 \) neighboring perturbations of \( \delta = Rw/\|w\| \) using the Neighbor function (Algorithm 3).

Results are shown in Figure 2a. With higher dimension, our attack success rate first increases and then flattens around 50% in the weak threat model, and around 5% in the strong threat model. We investigate the flattening phenomenon in Section 6.1.

With higher dimension more arithmetic operations are done in computing \( \|\delta^r\| \) and \( R \), which results in accumulation of rounding errors. Figure 2b further shows this influence of \( D \) on the rounding error, which can be accumulated to the magnitude of \( 10^5 \) with increasing \( D \). In summary, with the increasing rounding error, a greater leeway is left between the real certified radius \( R \) and the

---

**Algorithm 3: Neighbor: FP Neighbors Search**

**Input:** perturbation seed vector \( \delta = [\delta_1, \delta_2, \ldots, \delta_D] \);

\( p \): number of signed neighbor values to sample from for each dimension;

\( N \): number of sampled neighbors of \( \delta \) to return;

**Output:** \( \delta \)-neigh, \( N \) neighbors of \( \delta \)

**Function** Neighbor(\( \delta, N, n \)):

1. candidates \( \equiv \{(2p + 1) \times D\}
2. for \( i \leftarrow 1 \) to \( D \) do
   1. candidates\([i] [i] \leftarrow \delta_i
   2. \( r_i \leftarrow \delta_i
   3. \( l_i \leftarrow \delta_i
   4. \( j \leftarrow 2
   5. \) while \( j < 2p + 1 \) do
      1. \( r_i \leftarrow \text{afterFP}(r_i)
      2. \( l_i \leftarrow \text{beforeFP}(l_i)
      3. \) candidates\([j] [i] \leftarrow r_i
      4. \) candidates\([j + 1] [i] \leftarrow l_i
      5. \( j \leftarrow j + 2
   6. \) \end{while}
7. \) \end{for}
8. \( \delta \)-neigh \( \equiv \text{sample(candidates,} N \) \}
9. Construct \( N \) neighbor points: for each point \( \delta' \), randomly sample component \( \delta_i' \) from candidates\([i] [i] \), such that \( \delta \)-neigh contains no duplicates nor copies of \( \delta \).

**Return** \( \delta \)-neigh

**End Function**

In Algorithm 3, beforeFP(\( v \)) returns the first floating-point value before \( v \), and afterFP(\( v \)) returns the first floating-point value after \( v \). In the experiments, we set \( p = 2 \) for Algorithm 3.
We trained 45 models for each combination of distinct labels \( i, j \) with each we can successfully find an adversarial example within certified radius \( \tilde{R} \). That is, the upper bound of image in the test dataset. Our attack samples \( \approx \) pixel intensity within the domain digits where each image has 784 attributes and each attribute is a the MNIST dataset. MNIST [18] contains images of hand-written

In this section, we evaluate our attack on linear SVM trained with

4.2 Linear SVM

In this section, we evaluate our attack on linear SVM trained with the MNIST dataset. MNIST [18] contains images of handwritten digits where each image has 784 attributes and each attribute is a pixel intensity within the domain [0, 255]. We used \( \approx 12,000 \) images for training, and \( \approx 2,000 \) images for validation and evaluation of our attacks, for each combination of the labels \( i, j \in \{0, \ldots, 9\} \). We trained 45 models for each combination of distinct labels \( i, j \in \{0, \ldots, 9\} \) of the MNIST dataset. Validation accuracies range between 91% and 99% for linear SVM.

We then try to find an adversarial image with respect to each image in the test dataset. Our attack samples \( N = 5,000 \) neighbors of \( \delta = \tilde{R} w / \| w \| \) using Algorithm 3.

Computed certified radius \( \tilde{R} \) for our method to exploit, so the attack success rate increases.

The success rates are lower in the strong threat model than in the weak threat model. This is expected, as the leeway (i.e., \( \tilde{R} - \| \delta \| \)) exploited by our attack in the conservative strong model is likely much smaller than that (i.e., \( \tilde{R} - \| \delta \| \)) in the weak model. Why does the success rate first increase \( (D \leq 15) \) and then steadily drop \( (16 \leq D \leq 100) \) in the strong threat model (the triangle-marked dashed curves of Figure 2a)? The reason is that we use the upper bound of the perturbation norm \( \| \delta \| \) estimated with interval arithmetic in the strong model. With increasing dimension, more operations are involved, and interval arithmetic may become more uncertain about its estimation. When interval arithmetic is less certain, it outputs a larger interval so as to contain the real result. That is, the upper bound \( \| \delta \| \) gets larger. Hence, although \( \tilde{R} \) does get larger with increasing dimension, the leeway \( \tilde{R} - \| \delta \| \) exploited by our attack in the strong model first gets larger and then gets smaller, and our success rate decreases after first increasing.

4.3 Certification for neural nets

We now turn our attention to neural network verification mechanisms. In this section we consider neural networks with ReLU activations and rely on their linear approximations. Given a radius \( \tilde{R} \), a neural network \( F \) and an input \( x \), these mechanisms either certify \( \tilde{R} \) or not. Hence, in order to find a tight certified radius for a given model, one can perform a binary search to check multiple radii and call those verifiers multiple times. We avoid the binary search to find a certified radius \( \tilde{R} \) by first finding an adversarial example \( x' \) via ReluPGD (Section 3.1 and Algorithm 2) and then trying to verify the perturbation norms (i.e., \( \| x' - x \| \)) of those adversarial examples using the verifiers. We set ReluPGD to time out after 15 minutes.

Certification with \( \beta \)-CROWN. \( \beta \)-CROWN [28] guarantees sound but not complete robustness certification. That is, it provides a lower bound on the radius and it is possible that a tighter (larger) radius may exist. We use the \( \beta \)-CROWN verifier [28] in the \( \ell_2 \) metric, to verify a 3-layer neural network binary classifier with 1 node in the hidden layer. All model weights and biases in the hidden layers of this classifier are trained to be positive, so the perturbation direction is always \( v = w_f^T - w_i^T \). The classifier has validation accuracy 99.67%. We use ReluPGD, with step size \( s = 1 \times 10^{-5} \), to incrementally add

\[
\text{Figure 2: (a) Rounding search attack success rates against a random binary linear classifier in both weak (W) and strong (S) threat models (Section 4.1). For each dimension } D, \text{ we report the percentage of 10,000 randomly initialized models for which we can successfully find an adversarial example within certified radius } \tilde{R} \text{ for a random instance } x \text{ drawn from } [-1, 1]^D. \text{ Since the attacks are against an exact certified radius, the baseline attack rate should be } 0\% \text{ in both weak and strong threat models. Model weights } w \text{ and biases } b \text{ are randomly initialized with } w \in [-1, 1]^D, b \in [-1, 1]. \text{ All values and computation is done using either 32-bit or 64-bit floating points. (b) Maximum rounding error in the calculation of the certified radius } \tilde{R} \text{ on a sample } x \text{ with each } x_i = 3.3 \times 10^9, \text{ for the linear model with } w_i = 3.3 \times 10^{-9}, b = 3.3 \times 10^9, \text{ where } i \in [1, D] \text{ and } D \in [20, 1000].
\]
perturbation in direction $\nu$ to image $x$, until its prediction is flipped, and we get $x'$. Then we use $\beta$-CROWN to verify the image with respect to $\|\delta\| = \|x' - x\|$. If the certification succeeds, we have a successful attack. ReluPGD times out on 30 out of 2 108 images. When the attack does not time out, it takes $\approx 30$ seconds. A call to a verifier takes $\approx 1$ second. We conduct our attack on all MNIST test images labeled 0 or 1. We find adversarial images for 2 078 images, and $\beta$-CROWN erroneously verifies 53 of them. Our attack success rate is 2.6%.

5 MITIGATION: CERTIFICATION WITH ROUNDED INTERVAL ARITHMETIC

Our attack results demonstrate that floating-point rounding invalidates the soundness claims of a wide range of certification implementations for a variety of common models. How might such rounding errors in certification calculations be mitigated?

Rounding errors violating certifications are sometimes small. For example, the rounding error for the certified radius of the first MNIST image of Figure 3 of Appendix A is in the 13th decimal place. One’s first intuition may be to adopt slightly more conservative radii (e.g., using $\bar{R} - \gamma$ for some positive constant $\gamma$). Unfortunately, such radii are not in general sound, and attacks against $\bar{R} - \gamma$ are still possible. For example, for the same setting as in Section 4.1, it is easy to construct a linear classifier and find adversarial examples against it within $\bar{R} - 1.0$. As we show in Figure 2b, it is possible to construct instance-model pairs that undergo significant rounding errors. Just as a certification $\bar{R}$ should be sound when assuming real arithmetic, an implemented certification $\bar{R}$ must be sound under floating-point computation. Constant corrections $\gamma$ and similar mitigations that are not sound are therefore inappropriate for certifying robustness.

We outline a mitigation applying rounded interval arithmetic [12] to certified robustness. Interval arithmetic is an approach to bounding approximations in numerical analysis. It involves replacing each (possibly approximate floating-point) scalar with an interval with floating-point end-points that are guaranteed to bound the scalar. Our goal is to enable any computation on floating-points to be possible on interval representations, such that this ‘guaranteed bounding’ property on input data, parameters, or intermediate computations, is invariant to further computation.

It is first necessary to extend the standard arithmetic operators to interval arithmetic. The following definition demonstrates this process assuming arithmetic on $\mathbb{R}$: provided two target values $a, b$ are contained in intervals $[a_1, a_2], [b_1, b_2]$ to begin with, then the presented operators are guaranteed to maintain this property on basic arithmetic operations. For example, $a+b \in ([a_1, a_2] + [b_1, b_2])$, where the operator ‘+’ on reals is overloaded to real intervals.

**Definition 2.** For real intervals $[a_1, a_2], [b_1, b_2]$, define the following interval operators for elementary arithmetic:

- **Addition** $[a_1, a_2] + [b_1, b_2]$ is defined as $[a_1 + b_1, a_2 + b_2]$.
- **Subtraction** $[a_1, a_2] - [b_1, b_2]$ is defined as $[a_1 - b_2, a_2 - b_1]$.
- **Multiplication** $[a_1, a_2] \times [b_1, b_2]$ is defined as $[\min\{a_1 b_1, a_2 b_1, a_1 b_2, a_2 b_2\}, \max\{a_1 b_1, a_2 b_1, a_1 b_2, a_2 b_2\}]$.  

Certification with MIP solver. We now consider another method that provides conservative verification via mixed-integer programming (MIP). We use the implementation from [28], which uses the Gurobi MIP solver [10] for neural network verification, with Gurobi feasibility tolerance $2 \times 10^{-5}$. We verify two 3-layer neural network multiclass classifiers with 100 nodes in their hidden layer. For the first classifier, the weights and biases in the hidden layers are all trained to be positive, so the perturbation direction is $\nu = w_f^T - w_1^T$. The second classifier is trained without constraints on its weights and represents a regular network without artefacts. The validation accuracies for the first and second classifiers are 84.14%, and 96.73%, respectively.

We use ReluPGD to attack the two classifiers on all images of the MNIST test dataset, with step size $s = 1 \times 10^{-5}$. We found adversarial images against 8 406 images for the first classifier, and adversarial images against 9 671 images for the second classifier. ReluPGD times out on only 8 and 2 images for the first and second classifier, respectively. We then use MIP to verify each image with respect to their adversarial image’s perturbation norm $\|\delta\|$. Each attack takes $\approx 30$ seconds and each verification takes $\approx 10$ seconds. MIP successfully verified 5 108 out of 8 406 successfully attacked images for the first classifier, and verified 1 531 out of 9 671 successfully attacked images. That is, the attack success rate is 60.76% on an artificially trained network (where RelUs are all activated) and 15.83% on the second classifier trained without artefacts.

Certification with randomized smoothing. Our experiments on approximate certification attack a 3-layer ReLU network with 100 nodes in the hidden layer trained on the MNIST [18] dataset with labels 0 and 1, which has validation accuracy 99.95%. Pixel values of all images are scaled to $[0, 1]$. We adopt the same prediction procedures as [20]. Given an instance $x$, the smoothed classifier $g$ runs the base classifier $f$ on $M$ noise-corrupted instances of $x$, and returns the top class $k_A$ that has been predicted by $f$. The estimation of the certified radius $\bar{R}$ for a smoothed classifier $g$ is based on the probability distribution of the winner and runner-up classes (i.e., $p_A$ and $p_B$). $p_A$ and $p_B$ are estimated via interval estimation [2], with confidence $1 - \alpha$. As recommended in these papers, we set $\alpha = 0.1$, Gaussian prediction noise scale $\sigma_p \in \{1.0, 3.0, 5.0, 7.0\}$, and let the number of Monte Carlo samples $M$ range in $[100, 1000, 10000]$.

We conduct our attack as follows. Given $x$ and $\bar{R}$ as returned by $g$, we first use ReluPGD (maximum runtime is set as 15 minutes) to find an adversarial perturbation $\delta$ against the base classifier $f$, then use Neighbor to find $N = 1000$ neighbors $\delta'$ of $\delta$. We evaluate all generated adversarial examples $x + \delta'$ using the robust classifier $g$. For an image $x$, if any one $x + \delta'$ of the $N = 1000$ adversarial images flips the classification of the model with $\|\delta'\| \leq \bar{R}$, we have a successful attack against the approximate certification on this image. We chose $f$ in the process of perturbation generation as we need a concrete ReLU network with clear weights. We conduct attacks on 2 115 images of the MNIST dataset with labels 0 and 1, and have success rates up to 21.11% against approximate certification. Detailed results are listed in Table 2 of Appendix A.
• Division $[a_1, a_2]/[b_1, b_2]$ is defined as $[a_1, a_2] \times \frac{1}{[b_1, b_2]}$ where by cases:

$$\frac{1}{[b_1, b_2]} = \begin{cases} [\frac{1}{b_1}, \frac{1}{b_2}], & \text{if } 0 \notin [b_1, b_2] \\ \left[ -\infty, \frac{1}{b_1} \right] \cup \left[ \frac{1}{b_2}, \infty \right], & \text{otherwise} \end{cases}$$

Rounded interval arithmetic [12] further employs floating-point rounding when implementing the arithmetic operators, to achieve these sound floating-point extensions.

**Lemma 1.** Consider the interval arithmetic operators in Definition 2 with the resulting lower (upper) limits computed using IEEE754 floating-point arithmetic with rounding down (up), then the resulting rounded interval arithmetic operators are sound floating-point extensions.

That is, given a collection of floating-point intervals representing a collection of corresponding reals, rounded extension operators produce floating-point intervals that are guaranteed to contain the result of corresponding (base, unextended) arithmetic on the given collection of reals.

By representing constants as singleton intervals, rounded interval arithmetic computes floating-point bounds on real-valued results. Applying Definition 2 and Lemma 1, we can compute a sound floating-point interval for the rational 1/3:

$$\frac{[1, 1]}{[3, 3]} = [1.1] \times \frac{1}{[3, 3]} = [1.1] \times \frac{1}{3} = \left[ \frac{1}{3}, \frac{1}{3} \right] = [0.33\ldots33, 0.33\ldots33]$$

Beyond rounded interval arithmetic operators, libraries such as PyInterval offer rounded interval implementations of standard algebraic functions (e.g., the square root) and transcendental functions (e.g., the exponential, logarithm, trigonometric, and hyperbolic functions) using Newton-Raphson approximation that itself applies the above basic operators. Such functionality enables application of rounded interval arithmetic to general machine learning models and their certifications.

**Theorem 1.** Consider a classifier $f$, floating-point instance $x$, and a certification mechanism $R(f, x)$ that is sound when employing real arithmetic. If $R(f, x)$ can be computed by a composition of real-valued operators $\psi_1, \ldots, \psi_L$ with sound floating-point extensions $\hat{\psi}_1, \ldots, \hat{\psi}_L$, then the following certification mechanism $\hat{R}(f, x)$ is sound with floating-point arithmetic under the strong threat model: run the compositions of $\hat{\psi}_1, \ldots, \hat{\psi}_L$ on coordinate-wise intervals $[x, x], [f(x), f(x)]$ to obtain $\hat{R}(x, R)$; return $\hat{R}$.

**Proof.** The result follows by strong induction on the levels of composition implementing $R(f, x)$, with repeated application of Lemma 1. The induction hypothesis is that up to $l \leq L$ levels of composition of interval operators produces intervals that contain the result of the corresponding real operators. In other words, the real-valued composition $(\hat{\psi}_l \circ \hat{\psi}_{l-1} \circ \cdots \circ \hat{\psi}_1)(x)$ is contained in $(\phi_l \circ \phi_{l-1} \circ \cdots \circ \phi_1)(x)$. The base case comes from the (coordinate-wise) intervals $[x, x], [f(x), f(x)]$ containing coordinates of $x$ and $f(x)$; the inductive step follows from repeated application of sound floating-point extensions.

**Remark 2.** While the correction $\hat{R} - R$, may be a straw argument mitigation for constant $\gamma$, our mitigation can be seen as a sound input-dependent correction. To make explicit this required dependence on the input-model pair, we let $\gamma$ is a function of $f$. By taking $\gamma(f(x)) = \hat{R}(f(x)) - \hat{R}(f, x)$, we immediately arrive at a sound certificate via the corresponding correction $\hat{R}(f, x) - \gamma(f, x) = \hat{R}(f, x)$ by simply rearranging terms.

We offer example applications of Theorem 1 on linear classifiers in both weak and strong threat models, with 64-bit floating-point arithmetic. We use the PyInterval library [25] that performs rounded interval arithmetic to compute sound $R$ and $\hat{R}$ for linear classifiers [5]. Our attack success rates for randomly initialized linear classifiers (Section 4.1) drop to 0% for all dimensions in both threat models.

In sum, our theoretical and empirical results provide support for mitigating attacks against exact robustness certifications [5]. However, we should also note that our certificate $\hat{R}$ is both practically and theoretically safe in the strong threat model. In the weak threat model, the rounding errors introduced in calculating the norm of perturbation could potentially invalidate the soundness of $R$. Nonetheless, $\hat{R}$ should still mitigate most attacks in practice, as we have shown for linear classifiers.

6 DISCUSSION

6.1 Revisiting attacks

We now revisit our Section 4.1 experiment on randomized binary linear classifiers. Figure 2a observes an intriguing behavior of fast threat models. Figure 3a observes an intriguing behavior of fast threat models. Figure 4a observes an intriguing behavior of fast threat models. Figure 5a observes an intriguing behavior of fast threat models.

As in Figure 4a of Appendix A, our attack against linear models exploits rounding errors in the magnitude of $10^{-15}$. We find that the rate that $\hat{R}$ is overestimated $(\hat{R} > \hat{R})$ flattens around 50% (Figure 4b of Appendix A). Recall that our certificate works when $\hat{R}$ is overestimated, so there is leeway for our attack to exploit. Therefore, our success rate flattens around 50% because that is the maximum rate that $\hat{R}$ is overestimated.

Why does the overestimated rate flatten around 50%? One intuition is that rounding is effectively random: there is a 50% chance for $\hat{R}$ to be overestimated (rounded up), and another 50% chance for $\hat{R}$ to be underestimated (rounded down). Hence, no matter in 32-bit or 64-bit representation, our attack success rates flatten around 50%.
6.2 Limitations
We identify the following limitations that could be addressed as future work. First, our attacks against methods based on randomized smoothing did not study the relationship between the attack success rate and the soundness probability of these methods due to their probabilistic nature. A possible future direction would be to study how the two interact. Second, we showed how to adopt our mitigation based on interval arithmetic generally, and demonstrate this mitigation for linear models and exact certifications. Detailed exploration of embedding these mitigations in other verifiers and models is left as future work. Third, our evaluations demonstrate the effectiveness of our attacks on relatively small models (e.g., linear SVM) and datasets (e.g., MNIST). Further evaluation on state-of-the-art models and datasets is also left as future work.

Our attacks could potentially be used to find violations of robustness guarantees in system implementations, by exploiting floating-point vulnerabilities in them. Note that to date, we are not aware of any certified robustness implementations yet in deployment. However since certifications could be deployed in the near future, this work serves an important role in highlighting this new vulnerability, and in proposing an effective mitigation against it.

7 RELATED WORK
Several works have explored the influence of floating-point representation on guarantees of verified neural networks. For example, verifiers designed for floating-point representation have been shown to not necessarily work for quantized neural networks [8, 11, 16].

The closest to our work is the independent work by [17] who also exploit rounding errors to discover violations of network robustness certifications. Our work differs from [17] on the adversarial examples we find. As we show in Section 4.3, we are able to find an adversarial example $x'$ for unaltered natural image $x$ from test data, within that image $x$'s certified radius. The work by Jia and Rinard, instead, does not find certification-violating adversarial examples of test instances. It finds perturbed inputs $x_0$ of synthetic inputs $x_0$, that violate certifications of $x_0$. In particular, they adjust brightness of a natural test image $x$ to produce a $x_0$. That is, their attack point $x'_0$ is outside the certified radius of $x$. Hence, our attack can be seen as a stronger attack that is possible due to a novel attack methodology based on accurate perturbation directions. In another parallel and independent work, Voracek and Hein [27] also attacked randomized smoothing by exploiting the finite representation of floating points. However, they also attacked synthetic data and model, while we show attacks on unperturbed networks and test-set images and propose a sound and efficient mitigation based on interval arithmetic.

Research in the area of numerical analysis has proposed approaches to address the limitations of floating-point rounding, with a focus on measuring the stability of calculations. Proposed approaches include replacing floating-point arithmetic with interval arithmetic [15] or affine arithmetic [6]. Both account for rounding errors and return an interval that contains the correct result. Singh et al. [22, 23] use interval arithmetic in neural network certification like what we do for our mitigation, and are sound w.r.t. floating-point arithmetic. However, they use the $l_{inf}$ metric in neural network certification, which avoids the influence of potential rounding errors in the calculation of the perturbation norm. We use the $l_2$ metric, and handle the impact of rounding errors in the calculation of the perturbation norm for certification in the strong threat model. Moreover, our mitigation can be used to make implementations of any valid certification mechanisms sound, by adopting interval arithmetic due to Theorem 1. We adopt interval arithmetic with the implementation PyInterval [25] in the calculation of robustness certification.

Finally, exact robustness certifications without rounding errors can be achieved under special settings. The exact robustness certification problem can be transformed into a mixed integer linear programming (MILP) problem [26], which can be solved over rational numbers without floating-point errors [1]. However to our knowledge, currently there is no work that applies exact MILP solvers to neural network robustness certification, given their very limited performance. Jia and Rinard [16] did exact certification on a special type of neural network — binarized neural networks (BNNs), which quantize their weights and activations to be binary. Such approaches do not involve floating-point arithmetic, and hence are not vulnerable to floating-point errors.

8 CONCLUSION
Certified robustness has been proposed as a defense against adversarial examples. In this work we have shown that guarantees of several certification mechanisms do not hold in practice since they rely on real numbers that are approximated on modern computers. Hence, computation on floating-point numbers—used to represent real numbers—can overestimate certification guarantees due to rounding. We propose and evaluate a rounding search method that finds adversarial inputs on linear classifiers and verified neural networks within their certified radii—violating their certification guarantees. We propose rounded interval arithmetic as the mitigation, by accounting for the rounding errors involved in the computation of certification guarantees. We conclude that if certified robustness is to be used for security-critical applications, their guarantees and implementations need to account for limitations of modern computing architecture.

9 ACKNOWLEDGMENT
This work was supported by the joint CATCH MURI-AUSMURI, and The University of Melbourne’s Research Computing Services and the Petascale Campus Initiative. The first author is supported by the University of Melbourne research scholarship (MRS) scheme.

REFERENCES
[1] David L Applegate, William Cook, Sanjeeb Dash, and Daniel G Espinoza. 2007. Exact solutions to linear programming problems. Operations Research Letters 35, 6 (2007), 693–699.
[2] Lawrence D. Brown, T. Tony Cai, and Anirban DasGupta. 2001. Interval estimation for a binomial proportion. Statistical science 16, 2 (2001), 101–133.
[3] Rudy Bunel, P Mudigonda, Ilker Turkaslan, Philip Torr, Jingyu Li, and Pushmeet Kohli. 2020. Branch and bound for piecewise linear neural network verification. Journal of Machine Learning Research 21, 2020 (2020).
[4] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In IEEE Symposium on Security and Privacy (S&P) 39–57.
A ATTACK RESULTS

The results of rounding search attacks on linear SVM from Section 4.2 for each combination of the distinct labels \(i, j \in \{0, \ldots, 9\}\) of the MNIST dataset are listed in Table 1 for the weak model and the strong model. An example of original and adversarial images together with their perturbation and certified radius information based on linear SVM attack is presented in Figure 3.

Table 1: The success rates of our rounding search attack against linear SVM models on the MNIST dataset in the weak and strong threat models (Section 4.2). In the weak model, an experiment is successful if adversarial perturbation \(|\delta|\) is less than or equal to certified radius \(R = |w^T x + b|/\|w\|\) computed with finite-precision floating-point arithmetic (i.e., \(|\delta| \leq R\)). In the strong model, an experiment is successful if the upper bound of adversarial perturbation \(|\delta|\) is less than or equal to certified radius \(R\). 45 linear SVM models have been trained and attacked for each combination of labels respectively. A cell in row \(i\) and column \(j\) reports the attack success rate for classes original \(i\) and target \(j\), plus the attack success rate for classes original \(j\) and target \(i\). For example, in the weak model, for an SVM model, we can find adversarial perturbations within certified radius for 11.06% of all images labelled 1 (with the model classifying them as 0) or images labelled 0 (with the model classifying them as 1). We use “-” to denote models where rounding search did not find an adversarial example.

| labels | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|--------|---|---|---|---|---|---|---|---|---|
| weak   |   |   |   |   |   |   |   |   |   |
| strong |   |   |   |   |   |   |   |   |   |

(a) weak model

| labels | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|--------|---|---|---|---|---|---|---|---|---|
| weak   |   |   |   |   |   |   |   |   |   |
| strong |   |   |   |   |   |   |   |   |   |

(b) strong model

### References

[5] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. 2019. Certified adversarial robustness via randomized smoothing. In International Conference on Machine Learning (ICML). PMLR, 1310–1320.
[6] Luiz Henrique De Figueiredo and Jorge Stolfi. 2004. Affine arithmetic: concepts and applications. Numerical Algorithms 37, 1 (2004), 147–158.
[7] Ruediger Ehlers. 2017. Formal verification of piece-wise linear feed-forward neural networks. In Automated Technology for Verification and Analysis: 15th International Symposium, ATVA 2017, Pune, India, October 3–6, 2017, Proceedings. Springer, 269–286.
[8] Mirco Giacobbe, Thomas A Henzinger, and Mathias Lechler. 2020. How many bits does it take to quantize your neural network? In International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, 79–97.
[9] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In International Conference on Learning Representations (ICLR). http://arxiv.org/abs/1502.03832
[10] Gurobi Optimization, LLC. 2022. Gurobi Optimizer Reference Manual. https://www.gurobi.com
[11] Luc Jaulin, Michel Kieffer, Olivier Didrit, and Eric Walter. 2001. Interval analysis. Springer Science & Business Media.
[12] Nicholas J. Higham. 2002. Accuracy and stability of numerical algorithms. Society for Industrial and Applied Mathematics (SIAM).
[13] Kurt Hornik. 1991. Approximation capabilities of multilayer feedforward networks. Neural Networks 4. 2 (1991), 251–257.
[14] IEEE. [n. d.]. IEEE Standard for Floating-Point Arithmetic. IEEE Std 754-2019 (Revision of IEEE Std 754-2008) [n. d.], 1–84. https://doi.org/10.1109/IEEESTD.2019.8766229
[15] Luc Jaunin, Michel Kieffer, Olivier Didrit, and Eric Walter. 2001. Interval analysis. In Applied Interval Analysis. Springer, 11–43.
[16] Kai Jia and Martin Rinsard. 2020. Efficient exact verification of binarized neural networks. Advances in neural information processing systems 33 (2020), 1782–1795.
[17] Kai Jia and Martin Rinsard. 2021. Exploiting verified neural networks via floating point numerical error. In International Static Analysis Symposium. Springer, 191–205.
[18] Yann LeCun, Corinna Cortes, and Christopher J. Burges. 2010. MNIST handwritten digit database. ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist
[19] Mathias Leguer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. 2019. Certified robustness to adversarial examples with differential privacy. In IEEE Symposium on Security and Privacy (SP’19), 656–672.
[20] Bai Li, Changyou Chen, Wenlin Wang, and Lawrence Carin. 2019. Certified adversarial robustness with additive noise. In Advances in Neural Information Processing Systems (NeurIPS). Vol. 32.
[21] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards deep learning models resistant to adversarial attacks. In International Conference on Learning Representations (ICLR).
[22] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Pischel, and Martin Vechev. 2018. Fast and effective robustness certification. Advances in neural information processing systems 31 (2018).
[23] Gagandeep Singh, Timon Gehr, Markus Pischel, and Martin Vechev. 2019. An abstract domain for certifying neural networks. Proceedings of the ACM on Programming Languages 3, POPL (2019), 1–30.
[24] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In International Conference on Learning Representations (ICLR). http://arxiv.org/abs/1312.6199
[25] Stefano Tadimin. 2008. PyInterval, interval arithmetic in Python. https://pypi.org/project/pyinterval/; version 1.2.0 released 2017-03-05.
[26] Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In International Conference on Learning Representations. https://openreview.net/forum?id=H7yGdOfiqm
[27] Václav Voráček and Matthias Hein. 2022. Sound randomized smoothing in floating-point arithmetics. arXiv preprint arXiv:2207.07209 (2022).
[28] Shiqi Wang, Huan Zhang, Kai Xu, Xin Lin, Suman Jana, Chao-Jui Hsieh, and J Zico Kolter. 2021. Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. In Advances in Neural Information Processing Systems (NeurIPS).
[29] Huan Zhang, Tsou-Wei Weng, Pin-Yo Chen, Cho-Jui Hsieh, and Luca Daniel. 2018. Efficient neural network robustness certification with general activation functions. In Advances in Neural Information Processing Systems (NeurIPS). Vol. 31.
[30] Dániel Zombori, Balázs Bánhelyi, Tibor Csendes, István Megyesi, and Mark Jelali. 2020. Fooling a complete neural network verifier. In International Conference on Learning Representations (ICLR).
Figure 3: (a) Original images from the MNIST dataset. (b) Corresponding adversarial images in the weak threat model, with perturbations within the exact certified radius (i.e., $\|\delta\| \leq \tilde{R}$) but the linear SVM model misclassifies them. For example, the first top-left image in (a) has certified radius of $\tilde{R} = 333.6087764918925$, and is classified as 1, while the corresponding adversarial image in (b) has perturbation $\|\delta\| = 333.6087764918924$, and is classified as 0. Labels at the bottom right of each image are the classifications of the linear SVM model (Section 4.2).

The results of rounding search attacks on neural nets with approximate certification (i.e., randomized smoothing) are listed in Table 2.

Table 2: The attack success rates of our rounding search within the certified radius $\tilde{R}$ ($\alpha = 0.1\%$) [5]. The number of Monte Carlo samples $M$ used in certified radius estimation ranges in $\{100, 1000, 10000\}$. Prediction noise scale $\sigma_P \in \{1.0, 3.0, 5.0, 7.0\}$. We randomly sample and evaluate $N = 1000$ neighbors $\delta'$ of $\delta$ in the search area for each image. $N_A$ and $N_V$ denote the number of examples that have been successfully attacked and verified respectively.

| $\sigma_P$ | $M$  | $N_A$ | $N_V$ | success rate |
|------------|------|-------|-------|--------------|
| 1.0        | 100  | 5     | 2111  | 0.24%        |
|            | 1000 | 1     | 2113  | 0.05%        |
|            | 10000| 1     | 2115  | 0.05%        |
| 3.0        | 100  | 182   | 2066  | 8.81%        |
|            | 1000 | 89    | 2111  | 4.22%        |
|            | 10000| 8     | 2113  | 0.38%        |
| 5.0        | 100  | 273   | 1606  | 17.00%       |
|            | 1000 | 162   | 2097  | 7.73%        |
|            | 10000| 54    | 2109  | 2.56%        |
| 7.0        | 100  | 220   | 1042  | 21.11%       |
|            | 1000 | 248   | 2026  | 12.24%       |
|            | 10000| 99    | 2066  | 4.79%        |

The results of exploration of flattening success rate phenomenon appear in Figure 4.