Formal Security Analysis of Neural Networks using Symbolic Intervals

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana
Columbia University

Abstract

Due to the increasing deployment of Deep Neural Networks (DNNs) in real-world security-critical domains including autonomous vehicles and collision avoidance systems, formally checking security properties of DNNs, especially under different attacker capabilities, is becoming crucial. Most existing security testing techniques for DNNs try to find adversarial examples without providing any formal security guarantees about the non-existence of such adversarial examples. Recently, several projects have used different types of Satisfiability Modulo Theory (SMT) solvers to formally check security properties of DNNs. However, all of these approaches are limited by the high overhead caused by the solver.

In this paper, we present a new direction for formally checking security properties of DNNs without using SMT solvers. Instead, we leverage interval arithmetic to compute rigorous bounds on the DNN outputs. Our approach, unlike existing solver-based approaches, is easily parallelizable. We further present symbolic interval analysis along with several other optimizations to minimize overestimations of output bounds.

We design, implement, and evaluate our approach as part of ReluVal, a system for formally checking security properties of Relu-based DNNs. Our extensive empirical results show that ReluVal outperforms Reluplex, a state-of-the-art solver-based system, by 200 times on average. On a single 8-core machine without GPUs, within 4 hours, ReluVal is able to verify a security property that Reluplex deemed inconclusive due to timeout after running for more than 5 days. Our experiments demonstrate that symbolic interval analysis is a promising new direction towards rigorously analyzing different security properties of DNNs.

1 Introduction

In the last five years, Deep Neural Networks (DNNs) have enjoyed tremendous progress, achieving or surpassing human-level performance in many tasks such as speech recognition [19], image classifications [30], and game playing [46]. We are already adopting DNNs in security- and mission-critical domains like collision avoidance and autonomous driving [1,5]. For example, unmanned Aircraft Collision Avoidance System X (ACAS Xu), uses DNNs to predict best actions according to the location and the speed of the attacker/intruder planes in the vicinity. It was successfully tested by NASA and FAA [2,33] and is on schedule to be installed in over 30,000 passengers and cargo aircraft worldwide [40] and US Navy's fleets [3].

Unfortunately, despite our increasing reliance on DNNs, they remain susceptible to incorrect corner-case behaviors: *adversarial examples* [48], with small, human-imperceptible perturbations of test inputs, unexpectedly and arbitrarily changing a DNN's predictions. In a security-critical system like ACAS Xu, an incorrectly handled corner case can easily be exploited by an attacker to cause significant damage costing thousands of lives.

Existing methods to test DNNs against corner cases focus on finding adversarial examples [7,16,31,32,37,39,41,42,51] without providing formal guarantees about the non-existence of adversarial inputs even within very small input ranges. In this paper, we focus on the problem of formally checking that a DNN never violates a security property (e.g., no collision) for any malicious input provided by an attacker within a given input range (e.g., for attacker aircraft's speeds between 0 and 500 mph).

Due to non-linear activation functions like ReLU, the general function computed by a DNN is highly non-linear and non-convex. Therefore it is difficult to estimate the output range accurately. To tackle these challenges, all prior works on the formal security analysis of neural networks [6,12,21,25] rely on different types of Satisfiability Modulo Theories (SMT) solvers and are thus severely limited by the efficiency of the solvers.

We present ReluVal, a new direction for formally checking security properties of DNNs without using SMT solvers. Our approach leverages interval arithmetic [45] to compute rigorous bounds on the outputs of a DNN. Given the ranges of operands (e.g., $a_1 \in [0,1]$ and $a_2 \in [2,3]$), interval arithmetic computes the output range efficiently using only the lower and upper bounds of the operands (e.g., $a_2 - a_1 \in [1,3]$ because $2-1=1$ and $3-0=3$). Compared to SMT solvers, we found interval arithmetic to be significantly more efficient and flexible for formal analysis of a DNN's security properties.

Operationally, given an input range X and security property P, ReluVal propagates it layer by layer to calculate the output range, applying a variety of optimizations to improve accuracy. ReluVal finishes with two possible outcomes: (1) a formal guarantee that no value in X violates P ("secure"); and (2) an adversarial example in X violating P ("insecure"). Optionally, ReluVal can also guarantee that no value in a set of subintervals of X violates P ("secure subintervals") and that all remaining subintervals each contains at least one concrete adversarial example of P ("insecure subintervals").

A key challenge in ReluVal is the inherent overestimation caused by the input dependencies [8, 45] when interval arithmetic is applied to complex functions. Specifically, the operands of each hidden neuron depend on the same input to the DNN, but interval arithmetic assumes that they are independent and may thus compute an output range much larger than the true range. For example, consider a simplified neural network in which input x is fed to two neurons that compute $2x$ and $-x$ respectively, and the intermediate outputs are summed to generate the final output $f(x) = 2x - x$. If the input range of x is $[0,1]$, the true output range of $f(x)$ is $[0,1]$. However, naive interval arithmetic will compute the range of $f(x)$ as $[0,2] - [0,1] = [-1,2]$, introducing a huge overestimation error. Much of our research effort focuses on mitigating this challenge; below we describe two effective optimizations to tighten the bounds.

输入依赖

First, ReluVal uses *symbolic intervals* whenever possible to track the symbolic lower and upper bounds of each neuron. In the preceding example, ReluVal tracks the intermediate outputs symbolically ($[2x, 2x]$ and $[-x, -x]$ respectively) to compute the range of the final output as $[x, x]$. When propagating symbolic bound constraints across a DNN, ReluVal correctly handles non-linear functions such as ReLUs and calculates proper symbolic upper and lower bounds. It concretizes symbolic intervals when needed to preserve a sound approximation of the true ranges. Symbolic intervals enable ReluVal to accurately handle input dependencies, reducing output bound estimation errors by 85.67% compared to naive extension based on our evaluation.

Second, when the output range of the DNN is too large to be conclusive, ReluVal iteratively bisects the input range and repeats the range propagation on the smaller input ranges. We term 称、表明 this optimization *iterative interval refinement* because it is in spirit similar to abstraction refinement [4, 18]. Interval refinement is also amenable to massive parallelization, an additional advantage of ReluVal over hard-to-parallelize SMT solvers.

Mathematically, we prove that interval refinement on DNNs always converges in finite steps as long as the DNN is Lipschitz continuous 收敛 which is true for any DNN with finite number of layers. Moreover, lower values of Lipschitz constant result in faster convergence. Stable DNNs are known to have low Lipschitz constants [48] and therefore the interval refinement algorithm can be expected to converge faster for such DNNs. To make interval refinement even more efficient, ReluVal uses additional optimizations that analyze how each input variable influences the output of a DNN by computing each layer's gradients to input variables. For instance, when bisecting an input range, ReluVal picks the input variable range that influences the output the most. Further, it looks for input variable ranges that influence the output monotonically, and uses only the lower and upper bounds of each such range for sound analysis of the output range, avoiding splitting any of these ranges.

We implemented ReluVal using around 3,000 line of `C` code. We evaluated ReluVal on two different DNNs, ACAS Xu and an MNIST network, using 15 security properties (out of which 10 are the same ones used in [25]). Our results show that ReluVal can provide formal guarantees for all 15 properties, and is on average 200 times faster than Reluplex, a state-of-the-art DNN verifier using a specialized solver [25]. ReluVal is even able to prove a security property within 4 hours that Reluplex [25] deemed inconclusive due to timeout after 5 days. For MNIST, ReluVal verified 39.4% out of 5000 randomly selected test images to be robust against up to $|X|_\infty \leq 5$ attacks.

This paper makes three main contributions.

- To the best of our knowledge, ReluVal is the first system that leverages interval arithmetic to provide formal guarantees of DNN security.
- Naive application of interval arithmetic to DNNs is ineffective. We present two optimizations – symbolic intervals and iterative refinement – that significantly improve the accuracy of interval arithmetic on DNNs.
- We designed, implemented, evaluated our techniques as part of ReluVal and demonstrated that it is on average 200× faster than Reluplex, a state-of-the-art DNN verifier using a specialized solver [25].

2 Background

2.1 Preliminary of Deep Learning

A typical feedforward DNN can be thought of as a function $f : \mathbb{X} \rightarrow \mathbb{Y}$ mapping inputs $x \in \mathbb{X}$ (e.g., images, texts) to outputs $y \in \mathbb{Y}$ (e.g., labels for image classification, texts for machine translation). Specifically, f is composed 组成 of a sequence of parametric functions $f(x;w) = f_l(f_{l-1}(\cdots f_2(f_1(x;w_1);w_2)\cdots w_{l-1}),w_l)$, where l denotes the number of layers in a DNN, f_k denotes the corresponding transformation performed by k-th layer, and w_k denotes the weight parameters of k-th layer. Each $f_{k \in 1,...l}$ performs two operations: (1) a linear transformation of its input (i.e., either x or the output from f_{k-1}) denoted by $w_k \cdot f_{k-1}(x)$, where $f_0(x) = x$ and $f_{k \neq 0}(x)$ is the output of f_k denoting intermediate output of layer k while processing x, and (2) a nonlinear transformation $\sigma(w_k \cdot f_{k-1}(x))$ where σ is the nonlinear activation function. Common activation functions include sigmoid, hyperbolic tangent, or ReLU (Rectified Linear Unit) [38]. In this paper, we focus on DNNs using ReLU ($Relu(x) = max(0,x)$) as the activation function as it is one of the most popular ones used in the modern state-of-the-art DNN architectures [17,20,47].

2.2 Threat Model

威胁模型

目标系统。 在本文中，我们考虑了所有类型的安全关键系统，例如ACAS Xu [33]等无人驾驶飞机的机载避碰系统，该系统在敌手/入侵者在场的情况下使用DNN进行决策。 由于与传统的基于规则的系统相比，DNN具有更高的准确性和更少的性能开销，因此在此类系统中正变得越来越流行[24]。 例如，飞机防撞系统的决策过程可以使用DNN，根据飞机当前速度和航向，敌方的速度以及飞机与附近入侵者之间的距离的传感器数据，预测最佳行动。

Target system. In this paper, we consider all types of security-critical systems, e.g., airborne collision avoidance system for unmanned aircraft like ACAS Xu [33], which use DNNs for decision making in the presence of an adversary/intruder. DNNs are becoming increasingly popular in such systems due to better accuracy and less performance overhead than traditional rule-based systems [24]. For example, an aircraft collision avoidance system's decision-making process can use DNNs to predict the best action based on sensor data of the current speed and course of the aircraft, those of the adversary, and distances between the aircraft and nearby intruders.

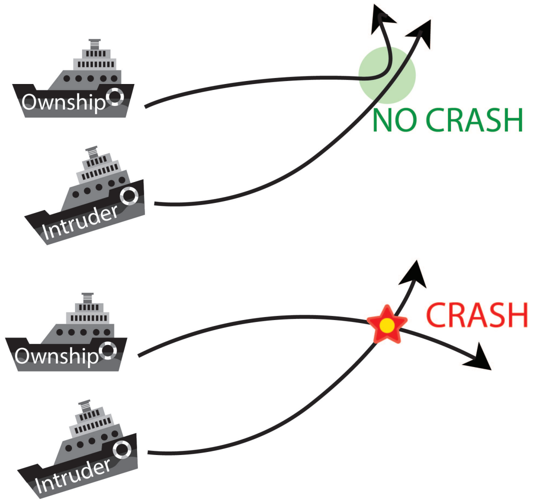


Figure 1: The DNN in the victim aircraft (ownship) should predict a left turn (upper figure) but unexpectedly advises to turn right and collides with the intruder (lower figure) due to the presence of adversarial inputs (e.g., if the attacker approaches at certain angles).

Security properties. In this paper, we focus on input-output-based security properties of DNN-based systems that ensure the correct actions in the presence of adversarial inputs within a given range. Input-output properties are well suited for the DNN-based systems as their decision logic is often opaque even to their designers. Therefore, unlike traditional programs, writing complete specifications involving internal states is often hard.

For example, consider a security property that tries to ensure that a DNN-based car crash avoidance system predicts the correct steering angle in the presence of an approaching attacker vehicle: it should steer left if the attacker approaches it from right. In this setting, even though the final decision is easy to predict for humans, the correct outputs for the internal neurons are hard to predict even for the designer of the DNN.

Attacker model. We assume that the inputs an adversary can provide are bounded within an interval specified by a security property. For example, an attacker aircraft has a maximum speed (e.g., it can only move between 0 and 500 mph). Therefore, the attacker is free to choose any value within that range. This attacker model is, in essence, similar to the ones used for adversarial attacks on vision-based DNNs where the attacker aims to search for visually imperceptible perturbations (within certain bound) that, when applied on the original image, makes the DNN predict incorrectly. Note that, in this setting, the imperceptibility is measured using a L_p norm. Formally, given a computer vision DNN f, the attacker solves following optimization problem: $min(L_p(x' - x))$ such that $f(x) \neq f(x')$, where $L_p(\cdot)$ denotes the p-norm and $x' - x$ is the perturbation applied to original input x. In other words, the security property of a vision DNN being robust against adversarial perturbations can be defined as: for any x' within a L-distance ball of x in the input space, $f(x) = f(x')$.

扰动

Unlike the adversarial images, we extend the attacker model to allow different amounts of perturbations to different features. Specifically, instead of requiring overall perturbations on input features to be bounded by L-norm, our security properties allow different input features to be transformed within different intervals. Moreover, for DNNs where the outputs are not explicit labels, unlike adversarial images, we do not require the predicted label to remain the same. We support properties specifying arbitrary output intervals.

An example. As shown in Figure 1, normally, when the distance (one feature of the DNN) between the victim ship (ownship) and the intruder is large, the victim ship advisory system will advise left to avoid the collision and then advise right to get back to the original track. However, if the DNN is not verified, there may exist one specific situation where the advisory system, for certain approaching angles of the attacker ship, advises the ship incorrectly to take a right turn instead of left, leading to a

一个例子。 如图1所示，通常，当受害船只（自身）与入侵船只之间的距离（DNN的一个特征）较大时，受害船只警报系统将建议左转避免碰撞，然后建议入侵船只返回到原始轨道。 但是，如果未验证DNN，则可能存在一种特殊情况，即警报系统针对入侵船只的某些接近角度，建议自身舰艇错误地右转而不是左转，从而导致致命的碰撞。 如果入侵者知道存在此类对抗样本，则可以专门以该对抗角度接近船只以引起碰撞。

fatal collision. If an attacker knows about the presence of such an adversarial case, he can specifically approach the ship at the adversarial angle to cause a collision.

2.3 Interval Analysis

区间算术研究区间上的算术运算而不是具体的值。 如上所述，由于(1)DNN安全属性检查需要在特定范围内设置输入特征并检查输出范围是否违规，并且(2)DNN计算仅包括加法和乘法(线性变换)和简单的非线性运算(例如ReLUs)，区间分析很自然地适合我们的问题。 我们在下面提供了函数及其属性的区间扩展的一些形式上的定义。 我们在第4节中使用这些定义来证明我们算法的正确性。

Interval arithmetic studies the arithmetic operations on intervals rather than concrete values. As discussed above, since (1) the DNN safety property checking requires setting input features within certain ranges and checking the output ranges for violations, and (2) the DNN computations only include additions and multiplications (linear transformations) and simple nonlinear operations (e.g., ReLUs), interval analysis is a natural fit to our problem. We provide some formal definitions of interval extensions of functions and their properties below. We use these definitions in Section 4 for demonstrating the correctness of our algorithm.

Formally, let x denote a concrete real value and $X := [\underline{X}, \overline{X}]$ denote an interval, where $\underline{X}$ is the lower bound, and $\overline{X}$ is the upper bound. An *interval extension* of a function $f(x)$ is a function of intervals F such that, for any $x \in X$, $F([x,x]) = f(x)$. The ideal interval extension $F(X)$ approaches the image of f, $f(X) := \{f(x) : x \in X\}$.

当Yi包含于Xi时，下面的等式成立，则称F是inclusion isotonic的

Let $f(X_1, X_2, ..., X_d) := \{f(x_1, x_2, ..., x_d) : x_1 \in X_1, x_2 \in X_2, ..., x_d \in X_d\}$ where d is the number of input dimensions. An interval valued function $F(X_1, X_2, ..., X_d)$ is *inclusion isotonic* if, when $Y_i \subseteq X_i$ for $i = 1, ..., d$, we have

$$F(Y_1, Y_2, ..., Y_d) \subseteq F(X_1, X_2, ..., X_d)$$

如果存在一些数字L使得下面的表达式成立，则在区间X0上定义的区间扩展函数F(X)被称为Lipschitz连续的。

An interval extension function $F(X)$ that is defined on an interval X_0 is said to be *Lipschitz continuous* if there is some number L such that:

$$\forall X \subseteq X_0, w(F(X)) \leq L \cdot w(X)$$

where $w(X)$ is the width of interval X, and X here denotes $X = (X_1, X_2, ..., X_d)$, a vector of intervals [45].

3 Overview

Interval analysis is a natural fit to the goal of verifying safety properties in neural networks as we have discussed in Section 2.3. Naively, by setting input features as intervals, we could follow the same arithmetic performed in the DNN to compute the output intervals. Based on the output intervals, we can verify if the input perturbations will finally lead to violations or not (e.g., output intervals go beyond a certain bound). Note that, lack of violations indicates the safety property is verified to be safe due to over-approximations.

However, naively computing output intervals in this way suffers from high errors as it computes extremely loose bounds due to the *dependency problem*. In particular, it can only get a highly conservative estimation of the output range, which is too wide to be useful for checking any safety property. In this section, we first demonstrate the dependency problem with a motivating example using naive interval analysis. Next, based on the same example, we describe how the techniques described in this paper can mitigate this problem.

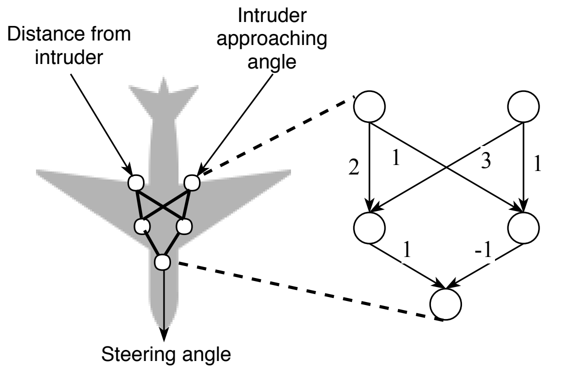


Figure 2: Running example to demonstrate our techniques.

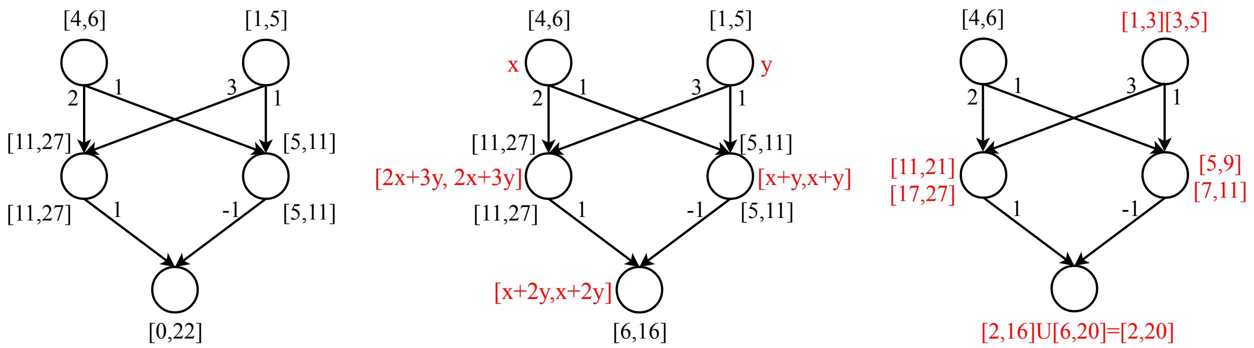
A working example. We use a small motivating example shown in Figure 2 to illustrate the inter-dependency problem and our techniques in dealing with this problem in Figure 3.

Let us assume that the sample NN is deployed in an unmanned aerial vehicle taking two inputs (1) distance from the intruder and (2) intruder approaching angle, while producing the steering angle as output. The NN has five neurons arranged in three layers. The weights attached to each edge is also shown in Figure 3 .

Assume that we aim to verify if the predicted steering angle is safe by checking a property that the steering angle should be less than 20 if the distance from the intruder is in $[4,6]$ and the possible angle of approaching intruder is in $[1,5]$.

Let x denote the distance from an intruder and y denote the approaching angle of the intruder. Essentially, given $x \in [4,6]$ and $y \in [1,5]$, we aim to assert that $f(x,y) \in [-\infty, 20]$. Figure 3a illustrates the naive interval propagation in this NN. By performing the interval multiplications and additions, along with applying the ReLU activation functions, we get the output interval to be $[0,22]$. Note that this is an overestimation because the upper bound 22 cannot be achieved: it can only appear when the left hidden neuron outputs 27 and the right one outputs 5. However, for the left hidden neuron to output 27, the conditions $x = 6$ and $y = 5$ have to be satisfied. Similarly, for the right hidden neuron to output 5, the conditions $x = 4$ and $y = 1$ have to be satisfied. These two conditions are contradictory and therefore cannot be

阐明



(a) Naive interval propagation

(b) Symbolic interval propagation

(c) Iterative bisection and refinement

Figure 3: Examples showing (a) naive interval extension where the output interval is very loose as it ignores the inter-dependency of the input variables, (b) using symbolic interval analysis to keep track of some of the dependencies, and (c) using bisection to reduce the over-approximation error.

satisfied simultaneously and therefore the final output 22 can never appear. This effect is known as the *dependency problem* [45].

As we have defined that a safe steering angle must be less than or equal to 20, we cannot guarantee non-existence of violations, as the steering angle can have a value as high as 22 according to the naive interval propagation described above.

正如我们定义的安全转向角必须小于或等于20一样，我们不能保证不存在违规情况，因为根据上述朴素间隔传播，转向角的值可能高达22。

展示

Symbolic interval propagation. Figure 3b demonstrates how we maintain the *symbolic intervals* to preserve as much dependency information as we can while propagating the bounds through the NN layers. In this paper, we only keep track of linear symbolic bounds and concretize the bounds when it is not possible to maintain accurate linear bounds. We compute the final output intervals using the corresponding symbolic equations. Our approach helps in significantly cutting down the over-approximation errors.

For example, in the current example, the intermediate neurons update their symbolic lower and upper bounds to be $2x+3y$ and $x+y$, denoting the operations performed by the previous linear transformations (taking the dot product of the input and weight parameters). As we also know $2x+3y>0$ and $x+y>0$ for the given input range $x\in[4,6]$ and $y\in[1,5]$, we can safely propagate the symbolic intervals through the ReLU activation functions.

In the final layer, the propagated bound will be $[x+2y, x+2y]$, where we can finally compute the concrete interval $[6,16]$. This is tighter than the naive baseline interval $[0,22]$ and can be used to verify the property that the steering angle will be less than 20.

In summary, *symbolic interval propagation* explicitly represents the intermediate computations of each neuron in terms of the symbolic intervals that encode the inter-dependency of the inputs to minimize overestimation.

However, in more complex cases, there might be intermediate neurons with symbolic bounds whose possible values can potentially be negative. For such cases, we can no longer keep the symbolic interval using a linear equation while passing it through a ReLU. Therefore, we concretize their upper and lower bounds and ignore their dependencies. To minimize the errors caused by such cases, we introduce another optimization, *iterative refinement*, as described below. As shown in Section 7, we can achieve very tight bounds by combining these two techniques.

但是，在更复杂的情况下，可能存在带有符号范围的中间神经元，其可能的值可能为负。 对于这种情况，在通过ReLU时，我们不再可以使用线性方程式保持符号间隔。 因此，我们具体化了它们的上限和下限，而忽略了它们的依赖性。 为了最大程度地减少此类情况引起的错误，我们将介绍另一种优化方法，即迭代优化，如下所述。 如第7节所示，通过结合这两种技术，我们可以实现非常严格的界限。

Iterative refinement. Figure 3c illustrates another optimization that we introduce for mitigating the dependency problem. Here, we leverage the fact that the dependency error for Lipschitz continuous functions decreases as the width of intervals decreases (any DNN with a finite number of layers is Lipschitz continuous as shown in Section 4.2). Therefore, we can bisect the input interval by evenly dividing the interval into the union of two consecutive sub-intervals and reduce the overestimation. The output bound can thus be tightened as shown in the example. The interval becomes $[2,20]$, which proves the non-existence of the violation. Note that we can iteratively refine the output interval by repeated splitting of the input intervals. Such operations are highly parallelizable as the split sub-intervals can be checked independently (Section 7). In Section 4, we provide a proof that the iterative refinement can effectively reduces the width of the output range to an arbitrary precision within finite steps for any Lipschitz continuous DNN.

正确性证明

4 Proof of Correctness

Section 3 demonstrates the basic idea of naive interval extension and the optimization of iterative refinement. In this section, we give the detailed proof about the correctness of interval analysis/estimation on DNNs, also known as interval extension estimation, and the convergence of iterative refinement. The proofs are based on two aforementioned properties of neural networks: *inclusion isotonicity* and *Lipschitz continuity*. In general, the correctness guarantee of interval extension holds for most finite DNNs while the convergence guarantee requires Lipschitz continuity. In the following, we give the proof of correctness for two most important techniques we use throughout the paper, but the proof is generic and works for our other optimizations such as symbolic interval analysis, influence analysis and monotonicity as described in Section 5.

Let f denote an NN and F denote its naive interval extension. We define the naive interval extension as a function $F(X)$ that (1) satisfies for all $x \in X, F([x,x]) = f(x)$ and (2) that only involves naive interval operations during interval variable representations. For all the other types of interval extensions, they can be easily analyzed based on the following proof.

令f表示一个NN，F表示其朴素的区间扩展。我们将朴素区间扩展定义为函数F(X)，其中（1）满足所有x∈X，F([x,x])= f(x)，并且（2）仅涉及区间变量期间的朴素区间运算表示形式。 对于所有其他类型的区间扩展，可以根据以下证据轻松地对其进行分析。

过度估计的正确性

4.1 Correctness of Overestimation

We are going to demonstrate that, for the naive interval extension of f, F always overestimates the theoretically tightest output range f. According to our definition of inclusion isotonicity described in Section 2, it suffices to prove that the naive interval extension of an NN is inclusion isotonic. Note that we only consider neural networks with ReLUs as activation functions for the following proof, but the proof can be easily extended to other popular activation functions like tanh or sigmoid.

我们将证明，对于f的朴素区间扩展，F总是过度估计了理论上最严格的输出范围f。 根据我们在第2节中描述的包含等渗性的定义，足以证明NN的朴素区间扩展是包含等渗性。 请注意，对于以下证明，我们仅将具有ReLU的神经网络视为激活函数，但是可以轻松将证明扩展到其他流行的激活函数，例如tanh或Sigmoid。

First, we need to demonstrate 证明 that F is inclusion isotonic. Because ReLU is monotonic 单调的, so we can simply consider its interval extension to be $Relu_I(X) := [max(0,\underline{X}), max(0,\overline{X})]$. Therefore, $\forall Y \subset X$, we have $max(0,\underline{X}) \leq max(0,\underline{Y})$ and $max(0,\overline{X}) \geq max(0,\overline{Y})$ so that its interval extension $Relu(Y) \subseteq Relu(X)$. Most common activation functions are inclusion isotonic. We refer interested readers to [45] for a list of common functions that are inclusion isotonic.

We note that $f(X)$ is a composition of activation functions and linear functions. And we also see that linear functions, as well as common activation functions, are inclusion isotonic [45]. Because any combinations of inclusion isotonic functions are still inclusion isotonic, thus, we have that the interval representation $F(X)$ of $f(X)$ is inclusion isotonic.

我们注意到f(X)是激活函数和线性函数的组合。 我们还看到线性函数以及常见的激活函数都是等渗的[45]。 由于包含等渗函数的任何组合仍然是包含等渗的，因此，我们认为f(X)的区间表示F(X)是包含等渗的。

Next, we show for arbitrary 任意的 $X = (X_1, \ldots, X_d)$, that:

$$f(X) \subseteq F(X)$$

Applying the previously shown inclusion isotonicity properties of $F(X)$, we get:

$$f(X_1, \ldots, X_d) = \bigcup_{(x_1,\ldots,x_d) \in X} \{f(x_1, \ldots, x_d)\}$$

$$= \bigcup_{(x_1,\ldots,x_d) \in X} F([x_1,x_1], \ldots, [x_d,x_d])$$

Now, for any such $(x_1, \ldots, x_d) \in X$, we have $F([x_1,x_1], \ldots, [x_d,x_d]) \subseteq F(X_1, \ldots, X_d)$, since $([x_1,x_1], \ldots, [x_d,x_d]) \subseteq (X_1, \ldots, X_d)$, and $F(X)$ is inclusion isotonic. We thus get:

$$\bigcup_{(x_1,\ldots,x_d) \in X} F([x_1,x_1], \ldots, [x_d,x_d]) \subseteq F(X_1, \ldots, X_d) \quad (1)$$

which is exactly the desired result.

Now, we get the result shown in Equation 1 that for all input X, the interval extension of f, $F(X)$, always contains the true codomain 陪域 (theoretically tightest bound) for $f(X)$.

有限分割数的收敛

4.2 Convergence in Finite Number of Splits

Now we see that the naive interval extension of f is an overestimation of true output. Next, we show that iteratively splitting input is an effective way to refine and reduce such overestimated error. Empirically, we can see finite number of splits allow us to approximate f with F with arbitrary accuracy, this is guaranteed by Lipschitz continuity property of NNs.

现在我们看到f的朴素区间扩展是对真实输出的过度估计。 接下来，我们证明迭代分割输入是细化和减少这种过度误差的有效方法。 从经验上看，我们可以看到有限数量的分割使我们能够以任意精度用F逼近f，这由NN的Lipschitz连续性属性来保证。

First, we need to prove F is Lipschitz continuous. It is straightforward to show that many common activation functions are Lipschitz continuous [45]. Here, we show the natural interval extension Relu_I is Lipschitz continuous, with a Lipschitz constant $L := 1$. We see, for any input interval X:

首先，我们需要证明F是Lipschitz连续的。 直接表明许多常见的激活函数是Lipschitz连续的[45]。 在这里，我们展示自然区间扩展ReluI是Lipschitz连续的，具有Lipschitz常数L：=1。我们看到，对于任何输入区间X：

$$w(\text{Relu}_I(X)) = max(\overline{X},0) - max(\underline{X},0)$$

$$\leq max(\overline{X},0) - \underline{X} \leq \overline{X} - \underline{X} = w(X)$$

Thus, the interval extension Relu_I of ReLU is Lipschitz continuous. As the NN is a finite composition of Lipschitz continuous functions, its interval extension F is still Lipschitz continuous as well [45].

因此，ReLU的区间扩展ReluI是Lipschitz连续的。 由于NN是Lipschitz连续函数的有限组成，因此其区间扩展F仍然是Lipschitz连续的[45]。

Now we demonstrate that by splitting input X into N smaller pieces and taking the union of their corresponding outputs, we can achieve a refined output estimation with at least N times smaller overestimation error. We define an N-split uniform subdivision of input $X = (X_1, \ldots, X_d)$ as a collection of sets $X_{i,j}$:

现在我们证明，通过将输入X分成N个较小的部分并取其对应输出的并集，我们可以实现精细化输出的估计，而其估计误差至少小N倍。 我们将输入X =(X1,...,Xd)的N均匀细分定义为集合Xi,j的collection：

$$X_{i,j} := [X_i + (j-1)\frac{w(X_i)}{N}, X_i + j\frac{w(X_i)}{N}]$$

where $i \in 1, \ldots, d$ and $j \in 1, \ldots N$. We note that this is exactly a partition of each X_i into N pieces of equivalent width such that $\forall i,j$, $w(X_{i,j}) = w(X_i)/N$ and $X_i = \bigcup_{j=1}^{N} X_{i,j}$. We then define a refinement of F over X with N splits as:

其中，i从1到d，j从1到N。我们注意到，这恰好是将每个Xi划分为N个相等宽度的块，使得

然后，我们定义F在X上的细化，其中N个分割为：

$$F^{(N)}(X) := \bigcup_{i=1}^{N} F(X_{1,i}, \ldots, X_{d,i})$$

最后，我们把将X进行N等份的分割细化后，由朴素区间扩展在NN上产生的过度误差的范围定义为w(E(N)(X))：

Finally, we define the range of overestimated error created by naive interval extension on an NN after N-split refinement as $w(E^{(N)}(X))$:

$$w(E^{(N)}(X)) := w(F^{(N)}(X)) - w(f(X))$$

Because F is Lipschitz continuous, Theorem 6.1 in [45] gives us the following result:

$$w(E^{(N)}(X)) \leq 2L \cdot w(X)/N \qquad (2)$$

Equation 2 shows the error width of the N-split refinement $w(E^{(N)}(X))$ converges to 0 linearly as we increase N. That is, we can achieve arbitrary accuracy when using N-split refinement to approximate $f(X)$ with sufficiently large N.

公式2显示了随着N的增加，N等份分解的误差宽度w(E(N)(X))线性收敛到0。也就是说，当使用N-split的精细度近似于f(X)时，我们可以实现任意精度，只要具有足够大的N。

5 Methodology

Figure 4 shows the main workflow along with the different components of ReluVal. Specifically, ReluVal uses symbolic interval analysis to get a tight estimation of the output ranges based on the input ranges. It declares a security property as verified if the estimated output interval is tight enough to satisfy the property. If the output interval shows potential existence of violations, ReluVal randomly samples a few points from the interval and check for violations. If any adversarial case is detected, i.e., a concrete input violating the security property, it outputs this as a counterexample. Otherwise, ReluVal uses iterative interval refinement to further tighten the output interval to approach the theoretically tightest bound and repeats the same process described above. Once the number of iterations reaches a preset threshold, ReluVal outputs timeout denoting it cannot verify the security property.

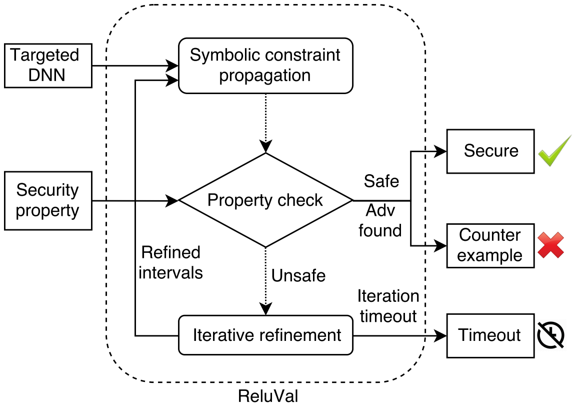


Figure 4: Workflow of ReluVal in checking security property of DNN.

As discussed in Section 3, simple interval extension only obtains loose/conservative intervals due to input dependency problem. Below, we describe the details of the optimizations we propose to further tighten the bounds.

如第3节所述，由于输入依赖性问题，简单的间隔扩展只能获得宽松/保守的间隔。下面，我们描述了为进一步缩小界限而提出的优化的细节。

符号区间传播

5.1 Symbolic Interval Propagation

Symbolic Interval propagation is one of our core contributions to mitigate the input dependency problem and tighten the output interval estimation. If a DNN would only consist of linear transformations, keeping symbolic equation throughout the intermediate computations of a DNN can perfectly eliminate the input dependency errors.

符号区间传播是我们减轻输入依赖性问题并加强输出区间估计的核心贡献之一。如果DNN仅由线性变换组成，则在DNN的所有中间计算过程中保持符号方程式可以完美消除输入相关性错误。

However, as shown in Section 3, passing an equation through a ReLU node essentially involves dropping the equation and replacing it with 0 if the equation can evaluate to a negative value for the given input range. Therefore, we keep the lower and upper bound equations (Eq_{up}, Eq_{low}) for as many neurons as we can and only concretize as needed.

但是，如第3节所示，通过ReLU节点传递方程式实质上涉及删除方程式，如果方程式可以在给定输入范围内求值为负值，则将其替换为0。因此，我们为尽可能多的神经元保留上下界对应的方程（Eq_up, Eq_low），并且仅根据需要具体化。

Algorithm 1 Symbolic interval analysis

```
Inputs:  network ← tested neural network
         input ← input interval
Initialize $eq = (eq_{up}, eq_{low})$;
// cache mask matrix needed in backward propagation
$R$[numLayer][layerSize];
// loops for each layer
for layer = 1 to numlayer do
    // matmal equations with weights as interval;
    $eq$= weight $\otimes$ $eq$;
    // update the output ranges for each node
    if layer != lastLayer then
        for i = 1 to layerSize[layer] do
            if $\overline{eq_{up}[i]} \leq 0$ then
                // Update to 0
                R[layer][i]=[0,0];          ▷ $\frac{d(relu(x))}{dx} = [0,0]$
                $eq_{up}[i]$ = $eq_{low}[i] = 0$;
            else if $\underline{eq_{low}[i]}$ $\geq$0 then
                // Keep dependency
                R[layer][i]=[1,1];          ▷ $\frac{d(relu(x))}{dx} = [1,1]$
            else
                // Concretization
                R[layer][i]=[0,1];          ▷ $\frac{d(relu(x))}{dx} = [0,1]$
                $eq_{low}[i] = 0$
                if $\underline{eq_{up}[i]} \leq 0$ then
                    $eq_{up}[i] = \overline{eq_{up}[i]}$;
    else
        output = {lower, upper};
return R, output;
```

Algorithm 1 elaborates the procedure of propagating symbolic intervals/equations during the interval computation of a DNN. We describe the core components and the details of this technique below.

算法1阐述了在DNN的区间计算过程中传播符号区间/等式的过程。我们在下面描述该技术的核心组件和细节。

Constructing symbolic intervals. Given a particular neuron A, (1) If A is in the first layer, we can compute the

构造符号区间

symbolic bounds as:

$$Eq_{up}^{A}(X) = Eq_{low}^{A}(X) = w_1x_1 + ... + w_dx_d$$

where $x_1,...,x_d$ are the inputs and $w_1,...,w_d$ are the weights of the corresponding edges. (2) If A belongs to the intermediate layer, we initialize the symbolic intervals of A's output as:

$$Eq_{up}^{A}(X) = \overset{a}{W_+Eq_{up}^{A_{prev}}(X)} + \overset{b}{W_-Eq_{low}^{A_{prev}}(X)}$$

$$Eq_{low}^{A}(X) = W_+Eq_{low}^{A_{prev}}(X) + W_-Eq_{up}^{A_{prev}}(X)$$

where $Eq_{up}^{A_{prev}}$ and $Eq_{low}^{A_{prev}}$ are the equations from last layer. W_+ and W_- denote the positive and negative weights of current layer respectively. The output will be $[w_+a, w_+b]$ for multiplying positive weight parameters w_+ with an interval $[a,b]$. For the negative weight parameters, the output will be flipped in terms of a and b, i.e., $[w_-b, w_-a]$.

具体化

总结：在传递区间等式的时候要注意等式的具体值，要符合ReLU的要求。

Concretization. While passing a symbolic equation through the ReLU nodes, we evaluate the concrete value of the equation's upper and lower bounds $\overline{Eq_{up}(X)}$ and $\underline{Eq_{low}(X)}$. If $\underline{Eq_{low}(X)} > 0$, then we pass the lower equation on to the next layer. Otherwise, we concretize it to be 0. Similarly, if $\overline{Eq_{up}(X)} > 0$, we pass the upper equation on to the next layer. Otherwise, we concretize it as $\overline{Eq_{up}(X)}$.

正确性

Correctness. We first clarify (阐明) three different output intervals: (1) theoretically (理论上) tightest bound $f(X)$, (2) naive interval extension bound $F(X)$, and (3) symbolic bound $[\underline{Eq_{low}(X)}, \overline{Eq_{up}(X)}]$. We prove that the symbolic bound is a superset of theoretically tightest bound and a subset of naive interval extension bound:

$$f(X) \overset{(1)}{\subseteq} \overset{(3)}{[\underline{Eq_{low}(X)}, \overline{Eq_{up}(X)}]} \overset{(2)}{\subseteq} F(X) \tag{3}$$

我寻思这不是口头证明？这也有说服力吗？

For a given input range propagated to the output layer, it will involve both computing linear transformations and applying ReLUs. Symbolic interval analysis keeps the accurate bounds for linear transformations and uses concretization to handle non-linearity. Compared to theoretically tightest bound, the only approximation introduced during the symbolic propagation process is due to concretization while handling ReLU nodes, which is an over-approximation as shown before. Naive interval extension, on the other hand, is a degenerate version of symbolic interval analysis where it does not keep any symbolic constraints. Therefore, symbolic interval analysis over-approximates the theoretically tightest bound and, in turn, is over-approximated by naive interval extension as shown in Equation 3.

迭代区间细化

5.2 Iterative Interval Refinement

While symbolic interval analysis helps in computing relatively tight bounds, the estimated output intervals for complex networks may still not be tight enough for verifying properties, especially when the input intervals are comparably large and thus resulting in many concretizations. As discussed above in Section 5, for such cases, we resort to another technique, iterative interval refinement. In addition, we also propose two other optimizations, influence analysis and monotonicity, which further refine the estimated output ranges based on iterative interval refinement.

尽管符号区间有助于计算相对严格的边界，但对于复杂网络依然不够。在这种情况下我们采用另一种技术【区间迭代细化】。此外我们还提出了另外两种优化【影响分析】【单调性】，他们可以再区间迭代细化的基础上进一步细化估计的输出范围。

基线迭代细化

Baseline iterative refinement. In Section 4, we have proved that theoretically tightest bound could be approached by repeatedly splitting the input intervals. Therefore, we perform iterative bisections on each input interval $X_1,...,X_n$ until the output interval is tight enough to meet the security property, or time out, as shown in Figure 4.

在第4节中，我们证明了通过重复分割输入间隔可以达到理论上最严格的界限。因此，我们对每个输入间隔X1...Xn执行迭代二等分，直到输出间隔足够紧以满足安全属性或超时为止，如图4所示。

The iterative bisection process can be represented as a bisection tree as shown in Figure 5. Each bisection on one input yields two children denoting two consecutive sub-intervals, the union of which computes the output bound for their parent. Here, $X^{(i)_j}$ means the jth input interval with split depth i. After one bisection on $X^{(i)_j}$, it creates two children: $X^{(i+1)_{2j-1}} = \{X_1,...,[\underline{X_i}, \frac{\underline{X_i}+\overline{X_i}}{2}],...,X_d\}$ and $X^{(i+1)_{2j}} = \{X_1,...,[\frac{\underline{X_i}+\overline{X_i}}{2}, \overline{X_i}],...,X_d\}$.

迭代二等分过程可以表示为二等分树，如图5所示。一个输入上的每个二等分会产生两个子代，表示两个连续的子区间，其并集计算出其父代的输出范围。 Xij表示深度为i的第j个输入区间。

To identify the existence of any adversarial example in the bisected input ranges, we sample a few input points (the current default is the middle point of each range) and verify if the concrete output leads to any property violation. If so, we output the adversarial example, mark this sub-interval as definitely containing adversarial examples, and conclude the analysis for this specific sub-interval. Otherwise, we repeat the symbolic interval analysis process for the sub-intervals. This default configuration is tailored towards deriving a conclusive answer of "secure" or "insecure" for the entire input intervals. Users of ReluVal can configure it to further split an insecure interval to potentially discover secure sub-intervals within the insecure interval.

为了确定平分输入范围内是否存在任何对抗示例,我们对几个输入点进行采样(当前默认值是每个范围的中间点),并验证具体输出是否导致任何属性违规。如果是这样,我们将输出对抗性示例,将这个子间隔标记为明确包含对抗性示例,然后对该特定子间隔进行分析。否则,我们将对子间隔重复符号间隔分析过程。定制此默认配置是为了在整个输入间隔内得出"安全"或"不安全"的结论性答案。ReluVal的用户可以对其进行配置,以进一步分割不安全间隔,以潜在地发现不安全间隔内的安全子间隔。

优化迭代细化

Optimizing iterative refinement. We develop two other optimizations, namely influence analysis and monotonicity, to further cut the average bisection depths.

我们还开发了另外两个优化，即影响分析和单调性，以进一步减少平均等分深度。

(1) Influence analysis. When deciding which input intervals to bisect first, instead of following a random strategy, we compute the gradient or Jacobian of the output with respect to each input feature and pick the largest one as the first to bisect. The high-level intuition is that the gradient approximates the influence of the input on the output, which essentially measures the sensitivity of the output to each input feature.

Algorithm 2 shows the steps for backward computation of the input feature influence. Note that instead of working on concrete values, this version works with intervals. The basic idea is to approximate the influence caused by ReLUs. If there is no ReLU in the target DNN, the

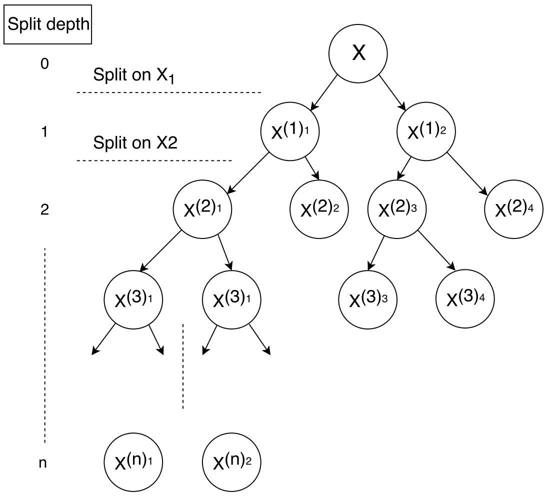


Figure 5: A bisection tree with split depth of n. Each node represents a bisected sub-interval.

Algorithm 2 Backward propagation for gradient interval

```
Inputs:  network ← tested neural network
         R ← gradient mask
// initialize upper and lower gradient bounds
$g_{up} = g_{low}$ = weights[lastLayer];
for layer = numlayer-1 to 1 do
    for 1 to layerSize[layer] do
        // $g$ is an interval containing $g_{up}$ and $g_{low}$
        // interval hadamard product
        $g$=R[layer] $\bigotimes$ $g$;
        // interval matrix multiplication
        $g$=weights[layer] $\bigodot$ $g$;
return $g$;
```

Jacobian matrix is completely determined by the weight parameters, which is independent of the input. A ReLU node's gradient can either be 0 for negative input or 1 for positive input. We use intervals to track and propagate the bounds on the gradients of the ReLU nodes during backward propagation as shown in Algorithm 2.

我们进一步使用估计的梯度区间来计算输入特征的梯度函数[26，27]：

We further use the estimated gradient interval to compute the smear function for an input feature [26, 27]: $S_i(X) = max_{1\le j\le d}\overline{|J_{ij}|}w(X_j)$, where J_{ij} denotes the gradient of input X_j for output Y_i. For each refinement step, we bisect the X_j with the highest smear value to reduce the over-approximation error as shown in Algorithm 3.

单调性。计算雅可比矩阵还有助于我们推断给定输入区间下输出的单调性。

(2) Monotonicity. Computing the Jacobian matrix also helps us to reason about the monotonicity property of the output for a given input interval. In particular, for the cases where the partial derivative of $\frac{\partial F_i}{\partial X_j}$ is always positive or negative for the given input interval X, we can simply replace the interval X_j with two concrete values $\underline{X_j}$ and $\overline{X_j}$. Because, as the DNN output is monotonic in that input interval, it is impossible for any intermediate value to cause a violation without either $\underline{X_j}$ or $\overline{X_j}$ causing one.

部分的 导数

因为，因为DNN输出在该输入间隔内是单调的，所以任何中间值都不可能引起冲突，而Xj或Xj不会引起冲突。 我们在第7节中的经验结果还表明，这种单调性检查可以帮助减少检查不同安全性属性所需的拆分数量。

Algorithm 3 Using influence analysis to choose the most influential feature to split

```
Inputs:  network ← tested neural network
         input ← input interval
         g ← gradient interval calculated by backward propagation
for i = 1 to input.length do
    // r is the range of each input interval
    $r = w(input[i])$;
    // e is the influence from each input to output
    e = $g_{up}[i] * r$;
    if e > largest then                         ▷ most effective feature
        largest = e;
        splitFeature = i;
return splitFeature;
```

Our empirical results in Section 7 also indicate that such monotonicity checking can help decrease the number of splits required for checking different security properties.

我们在第7节中的经验结果还表明，这种单调性检查可以帮助减少检查不同安全性属性所需的拆分数量。

6 Implementation

Setup. We implement ReluVal in `C` and leverage `OpenBLAS`[1] to enable efficient matrix multiplications. We evaluate ReluVal on a Linux server running Ubuntu 16.04 with 16 CPU cores and 256GB memory.

Parallelization. One unique advantage of ReluVal over other security property checking systems like Reluplex is that the interval arithmetic in the setting of verifying DNNs is highly parallelizable by nature. During the process of iterative interval refinement, newly created input ranges can be checked independently. This feature allows us to create as many threads as possible, each taking care of a specific input range, to gain significant speedup by distributing different input ranges to different workers.

However, there are two key challenges that required solving to fully leverage the benefits of parallelization. First, as shown in Section 5.2, the bisection tree is often not balanced leading to substantially different running times for different threads. We found that often several laggard threads slow down the computation, i.e., most of the available workers stay idle while only a few workers keep on refining the intervals. Second, as it is hard to predict the depth of the bisection tree for any sub-interval in advance, starting a new thread for each sub-interval may result in high scheduling overhead. To solve these two problems, we develop a dynamic thread rebalancing algorithm that can identify the potentially deeper parts of the bisection tree and efficiently redistribute those parts among other workers.

Outward rounding. The large number of floating matrix multiplications in a DNN can potentially lead to severe precision drops after rounding [15]. For example, assume that the output of one neuron is [0.00000001,

[1] http://www.openblas.net/

00000002]. If the floating-point precision is $e-7$, then is automatically rounded up to [0.0,0.0]. After one layer propagation with a weight parameter of 1000, the correct output should be [0.00001, 0.00002]. However, after rounding, the output will incorrectly become [0.0, 0.0]. As the interval propagates through the neural network, more errors will accumulate and significantly affect the output precision. In fact, our tests show that some adversarial examples reported by Reluplex [25] are false positives due to such rounding problem.

To avoid such issues, we adopt outward rounding in ReluVal. In particular, for every newly calculated interval or symbolic interval, we always round the bounds outward to ensure the computed output range is always a sound overestimation of the true output range. We implement outward rounding with 32-bit floats. We find that this precision is enough for verifying properties of ACAS Xu models, though it can easily be extended to 64-bit double.

7 Evaluation

7.1 Evaluation Setup

In the evaluation, we consider two general categories of DNNs, deployed for handling two different tasks.

The first category is airborne collision avoidance system (ACAS) crucial for alerting and preventing the collisions between aircraft. We focus our evaluation on ACAS Xu models for collision avoidance in unmanned aircraft [28].

The second category includes the models deployed to recognize hand-written digit from the MNIST dataset. Our preliminary results demonstrate that ReluVal can also scale to larger networks that the solver-based verification tools often struggle to check.

ACAS Xu. The ACAS Xu system consists of forty-five different NN models. Each network is composed of an input layer taking five inputs, an output layer generating five outputs, and six hidden layers with each containing fifty neurons. As shown in Figure 6, five inputs include $\{\rho, \theta, \psi, v_{own}, v_{int}\}$. In particular, ρ denotes the distance between ownship and intruder, θ denotes the heading direction angle of ownship relative to the intruder, ψ denotes the heading direction angle of the intruder relative to ownship, v_{own} is the speed of ownship, and v_{int} is the speed of intruder. Output of the NN includes {*COC*, *weak left*, *weak right*, *strong left*, *strong right*}. *COC* denotes clear of conflict, *weak left* means heading left with angle 1.5^o/s, *weak right* means heading right with angle 1.5^o/s, *strong left* is heading left with angle 3.0^o/s, and *strong right* denotes heading right with angle 3.0^o/s. Each output in NN corresponds to the score for this action (minimal for the best).

MNIST. For classifying hand-written digits, we test a neural network with 784 inputs, 10 outputs, and two hidden layers. Each intermediate layer has 512 neurons. On the MNIST test dataset, it can achieve 98.28% accuracy for classification.

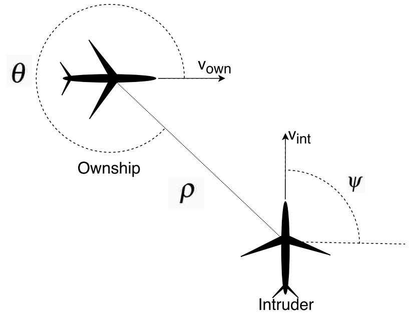


Figure 6: Horizontal view of ACAS Xu operating scenarios.

7.2 Performance on ACAS Xu Models

In this section, we first present a detailed comparison of ReluVal and Reluplex in terms of the verification performance. Then, we compare ReluVal with a state-of-the-art adversarial attack on DNNs, Carlini-Wagner [7], showing that on average ReluVal can consistently find 50% more adversarial examples. Finally, we show that ReluVal can accurately narrow down all possible adversarial ranges and therefore provide more insights on the distribution of adversarial corner-cases.

Comparison to Reluplex. Table 1 compares the time taken by ReluVal with that of Reluplex for verifying ten original properties described in their paper [25]. In addition, we include the experimental results for five new security properties. The detailed description of each property is in the Appendix. Table 1 shows that ReluVal always outperforms Reluplex at checking all fifteen security properties. For the properties on which Reluplex times out, ReluVal is able to terminate in significantly shorter time. On average, ReluVal achieves up to 200× speedup over Reluplex.

Finding adversarial inputs. In terms of the number of adversarial examples detected, ReluVal also outperforms the popular attacks using gradients to find adversarial examples. Here, we compare ReluVal to the Carlini and Wagner (CW) attack [7], a state-of-the-art gradient-based attack that minimizes specialized CW loss function.

As gradient-based attacks start from a seed input and iteratively looking for adversarial examples, the choice of seeds may highly influence the success of the attack at finding adversarial inputs. Therefore, we try different randomly picked seed inputs to facilitate the input generation process. Note that our technique in ReluVal does not need any seed input. Thus it is not restricted

Source	Properties	Networks	Reluplex Time (sec)	ReluVal Time (sec)	Speedup
Security Properties from [25]	ϕ_1	45	>443,560.73*	14,603.27	>30×
	ϕ_2	34*[2]	123,420.40	117,243.26	1×
	ϕ_3	42	35,040.28	19,018.90	2×
	ϕ_4	42	13,919.51	441.97	32×
	ϕ_5	1	23,212.52	216.88	107×
	ϕ_6	1	220,330.82	46.59	4729×
	ϕ_7	1	>86400.0*	9,240.29	>9×
	ϕ_8	1	43,200.01	40.41	1069×
	ϕ_9	1	116,441.97	15,639.52	7×
	ϕ_{10}	1	23,683.07	10.94	2165×
Additional Security Properties	ϕ_{11}	1	4,394.91	27.89	158×
	ϕ_{12}	1	2,556.28	0.104	24580×
	ϕ_{13}	1	>172,800.0*	148.21	>1166×
	ϕ_{14}	2	>172,810.86*	288.98	>598×
	ϕ_{15}	2	31,328.26	876.80	36×

* Reluplex uses different timeout thresholds for different properties.

Table 1: ReluVal's performance at verifying properties of ACAS Xu compared with Reluplex. ϕ_1 to ϕ_{10} are the properties proposed in Reluplex [25]. ϕ_{11} to ϕ_{15} are our additional properties.

# Seeds	CW	CW Miss	ReluVal	ReluVal Miss
50	24/40	40.0%	40/40	0%
40	21/40	47.5%	40/40	0%
30	17/40	58.5%	40/40	0%
20	10/40	75.0%	40/40	0%
10	6/40	85.0%	40/40	0%

Table 2: The number of adversarial inputs CW can find compared to ReluVal on 40 adversarial ACAS Xu properties. The third column shows the percentage of adversarial properties CW failed to find.

by the potentially undesired starting seed and can fully explore the input space. As shown in Table 2, on average, CW misses 61.2% number of models, which do have adversarial inputs exist that CW fails to find.

Narrowing down adversarial ranges. A unique feature of ReluVal is that it can isolate adversarial ranges of inputs from the non-adversarial ones. This is useful because it allows a DNN designer to potentially isolate and avoid adversarial ranges with a given precision (e.g., $e-6$ or smaller). Here we set the precision to be $e-6$, i.e., we allow splitting of the intervals into smaller sub-intervals unless their length becomes less than $e-6$. Table 3 shows the results of the three different properties that we checked. For example, property S_1 specifies `model_4_1` should output strong right with input range $\rho = [400, 10000]$, $\theta = 0.2$, $\psi = -3.09$, $v_{own} = 10$, and $v_{int} = 10$. For this property, ReluVal splits the input ranges into 262,144 smaller sub-intervals and is able to prove that 163,915 sub-intervals are safe. ReluVal also finds that $\rho = [400, 6402.36]$ does not contain any adversarial inputs while $\rho = [6402.36, 10000]$ is adversarial.

P	Adv Range	Adv	Timeout	Non-adv
S_1	$[6402.36, 10000]$	98229	1	163915
S_2	$[-0.2, -0.186]$ and $[-0.103, 0]$	18121	2	14645
S_3	$[-0.1, 0.0085]$	17738	1	15029

Table 3: The second column shows the input ranges containing at least one adversarial input, while the rest of ranges are found by ReluVal to be non-adversarial. The last three columns show the number of total sub-intervals checked by ReluVal with a precision of $e-6$.

MNIST模型的初步测试

7.3 Preliminary Tests on MNIST Model

Besides ACAS Xu, we also test ReluVal on an MNIST model that achieves decent accuracy (98.28%). Given a particular seed image, we allow arbitrary perturbations to every pixel value while bounding the total perturbation by the L_∞ norm. In particular, ReluVal can prove 956 seed images to be safe for $|X|_\infty \leq 1$ and 721 images safe for $|X|_\infty \leq 2$ respectively out of 1000 randomly selected test images. Figure 7 shows the detailed results. As the norm is increased, the percentage of images that have no adversarial perturbations drops quickly to 0. Note that we get more timeouts as the L_∞ norm increase. We believe that we can further optimize our system to work on GPUs to minimize such timeouts and verify properties with larger norm bounds.

[2] We remove model_4_2 and model_5_3 because Reluplex found incorrect adversarial examples due to rounding problems (these models do not have any adversarial case).

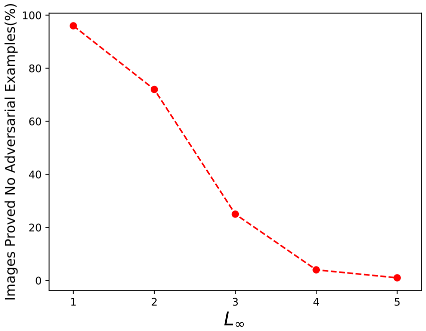


Figure 7: Percentage of images proved to be non-adversarial with $L_\infty = 1,2,3,4,5$ by ReluVal on MNIST test model out of 1000 random test MNIST images.

7.4 Optimizations

In this subsection, we evaluate the effectiveness of the optimizations proposed in Section 5 compared to the naive interval extension with iterative interval refinement. The results are shown in Table 4.

Methods	Deepest Dep (%)	Avg Dep (%)	Time (%)
S.C.P	42.06	49.28	99.99
I.A.	10.65	10.85	96.04
Mono	0.325	0.497	16.91

Table 4: The percentages of the deepest depth, average depth, and average running time improvement caused by the three main components of ReluVal: symbolic interval analysis, influence analysis, and monotonicity compared to the naive interval analysis.

Symbolic interval propagation. Table 4 shows that symbolic interval analysis saves the deepest and average depth of bisection tree (Figure 5) by up to 42.06% and 49.28%, respectively, over naive interval extension.

Influence analysis. As one of the optimizations used in iterative refinement, influence analysis helps prioritize splitting of the most influential input to the output. Compared to the sequential splitting features, influence-analysis-based splitting reduces the average depth by 10.85% and thus cut down the running time by up to 96.04%.

Monotonocity. The improvements from using monotonicity are relatively smaller in terms of tree depth. However, it can still reduce the average running time by 16.91% on average, especially when the average depth is high.

8 Related Work

Adversarial machine learning. Several recent works have shown that even the state-of-the-art DNNs can be easily fooled by adding small carefully crafted human-imperceptible perturbations to the original inputs [7, 16, 37, 48]. This has resulted in an arms race among researchers competing to build more robust networks and design more efficient attacks [7, 16, 31, 32, 39, 41, 51]. However, most of the defenses are restricted to only one type of adversaries/security properties (e.g., overall perturbations bounded by some norms) even though other researchers have shown that other semantics-preserving changes like lightning changes, small occlusions, rotations, etc. can also easily fool the DNNs [13, 42, 43, 49]. However, none of these attacks can provide any provable guarantees about the non-existence of adversarial examples for a given neural network. Unlike these attacks, ReluVal can provide a provable security analysis of given input ranges, systematically narrowing down and detecting all adversarial ranges.

Verification of machine learning systems. Recently, several projects [12, 21, 25] have used customized SMT solvers for verifying security properties of DNNs. However, such techniques are mostly limited by the scalability of the solver. Therefore, they tend to incur significant overhead [25] or only provide weaker guarantees [21]. By contrast, ReluVal uses interval-based techniques and significantly outperforms the state-of-the-art solver-based systems like Reluplex [25].

Kolter et al. [29] and Raghunathan et al. [44] transform the verification problem into a convex optimization problem using relaxations to over-approximate the outputs of ReLU nodes. Similarly, Gehr et al. [14] leverages zonotopes for approximating each ReLU outputs. Dvijotham et al. [11] transform the verification problem into an unconstrained dual formulation using Lagrange relaxation and use gradient-descent to solve the optimization problem. However, all of these works focus on simply over-approximating the total number of potential adversarial violations without trying to find concrete counterexamples. Therefore, they tend to suffer from high false positive rates unless the underlying DNN's training algorithm is modified to minimize such violations. By contrast, ReluVal can find concrete counterexamples as well as verify security properties of pre-trained DNNs.

Recently, Mixed Integer Linear programming (MILP) solvers combined with gradient descent have also been proposed for verification of DNNs [9, 10]. Integrating our interval analysis together with such approaches is an interesting future research problem.

Verivis [43] by Pei et al. is a black-box DNN verification system that leverages the discreteness of image pixels. However, unlike ReluVal, it cannot verify non-existence of norm-based adversarial examples.

Interval optimization. Interval analysis has shown great success in many application domains including nonlinear equation solving and global optimization problems [23, 34, 35]. Due to its ability to provide rigorous bounds on the solutions of an equation, many numerical optimization problems [22,50] leveraged interval analysis to achieve a near-precise approximation of the solutions. We note that the computation inside NN is mostly a sequence of simple linear transformations with nonlinear activation functions. These computations thus highly resemble those in traditional domains where interval analysis has been shown to be successful. Therefore, based on the foundation of interval analysis laid by Moore et al. [36, 45], we leverage interval analysis for analyzing the security properties of DNNs.

9 Future Work and Discussion

Supporting other activation functions. Interval extension can, in theory, be applied to any activation function that maintains inclusion isotonicity and Lipschitz continuity. As mentioned in Section 4, most popular activation functions (e.g., tanh, sigmoid) satisfy these properties. To support these activation functions, we need to adapt the symbolic interval propagation process. We plan to explore this as part of future work. Our current prototype implementation of symbolic interval propagation supports several common piece-wise linear activation functions (e.g., regular ReLU, Leaky ReLU, and PReLU).

Supporting other norms besides L_∞. While interval arithmetic is most immediately applicable to L_∞, other norms (e.g., L_2 and L_1) can also be approximated using intervals. Essentially, L_∞ allows the most flexible perturbations and the perturbations bounded by other norms like L_2 are all subsets of those allowed by the corresponding L_∞ bound. Therefore, if ReluVal can verify the absence of adversarial examples for a DNN within an infinite norm bound, the DNN is also guaranteed to be safe for the corresponding p-norm (p=1/2/3..) bound. If ReluVal identifies adversarial subintervals for an infinite norm bound, we can iteratively check whether any such subinterval lies within the corresponding p-norm bound. If not, we can declare the model to contain no adversarial examples for the given p-norm bound. We plan to explore this direction in future.

Improving DNN Robustness. The counterexamples found by ReluVal can be used to increase the robustness of a DNN through adversarial training. Specific, we can add the adversarial examples detected by ReluVal to the training dataset and retrain the model. Also, a DNN's training process can further be changed to incorporate ReluVal's interval analysis for improved robustness. Instead of training on individual samples, we can convert the training samples into intervals and change the training process to minimize losses for these intervals instead of individual samples. We plan to pursue this direction as future work.

10 Conclusion

Although this paper focuses on verifying security properties of DNNs, ReluVal itself is a generic framework that can efficiently leverage interval analysis to understand and analyze the DNN computation. In the future, we hope to develop a full-fledged DNN security analysis tool based on ReluVal, just like traditional program analysis tools, that can not only efficiently check arbitrary security properties of DNNs but can also provide insights into the behaviors of hidden neurons with rigorous guarantees.

In this paper, we designed, developed, and evaluated ReluVal, a formal security analysis system for neural networks. We introduced several novel techniques including symbolic interval arithmetic to perform formal analysis without resorting to SMT solvers. ReluVal performed 200 times faster on average than the current state-of-art solver-based approaches.

11 Acknowledgements

We thank Chandrika Bhardwaj, Andrew Aday, and the anonymous reviewers for their constructive and valuable feedback. This work is sponsored in part by NSF grants CNS-16-17670, CNS-15-63843, and CNS-15-64055; ONR grants N00014-17-1-2010, N00014-16-1-2263, and N00014-17-1-2788; and a Google Faculty Fellowship. Any opinions, findings, conclusions, or recommendations expressed herein are those of the authors, and do not necessarily reflect those of the US Government, ONR, or NSF.

References

[1] Baidu Apollo Autonomous Driving Platform. https://github.com/ApolloAuto/apollo.

[2] NASA, FAA, Industry Conduct Initial Sense-and-Avoid Test. https://www.nasa.gov/centers/armstrong/Features/acas_xu_paves_the_way.html.

[3] NAVAIR Plans to Install ACAS Xu on MQ-4C Fleet. https://www.flightglobal.com/news/articles/navair-plans-to-install-acas-xu-on-mq-4c-fleet-444989/.

[4] T. Ball and S. K. Rajamani. The SLAM project: debugging system software via static analysis. In *ACM SIGPLAN Notices*, volume 37, pages 1–3. ACM, 2002.

[5] C. Bloom, J. Tan, J. Ramjohn, and L. Bauer. Self-driving cars and data collection: Privacy perceptions of networked

autonomous vehicles. In *Symposium on Usable Privacy and Security (SOUPS)*, 2017.

[6] N. Carlini, G. Katz, C. Barrett, and D. L. Dill. Provably minimally-distorted adversarial examples. *arXiv preprint arXiv:1709.10207*, 2017.

[7] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy*, pages 39–57. IEEE, 2017.

[8] L. H. De Figueiredo and J. Stolfi. Affine arithmetic: concepts and applications. *Numerical Algorithms*, 37(1):147–158, 2004.

[9] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari. Learning and verification of feedback control systems using feedforward neural networks. In *IFAC Conference on Analysis and Design of Hybrid Systems (ADHS)*, 2018.

[10] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*, pages 121–138. Springer, 2018.

[11] K. Dvijotham, R. Stanforth, S. Gowal, T. Mann, and P. Kohli. A dual approach to scalable verification of deep networks. *arXiv preprint arXiv:1803.06567*, 2018.

[12] R. Ehlers. Formal verification of piece-wise linear feedforward neural networks. In *International Symposium on Automated Technology for Verification and Analysis (ATVA)*, pages 269–286. Springer, 2017.

[13] L. Engstrom, D. Tsipras, L. Schmidt, and A. Madry. A rotation and a translation suffice: Fooling cnns with simple transformations. *arXiv preprint arXiv:1712.02779*, 2017.

[14] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. AI2: Safety and robustness certification of neural networks with abstract interpretation. In *IEEE Symposium on Security and Privacy*, 2018.

[15] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys (CSUR)*, 23(1):5–48, 1991.

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

[17] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[18] T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy abstraction. *ACM SIGPLAN Notices*, 37(1):58–70, 2002.

[19] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[20] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, page 3, 2017.

[21] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification (CAV)*, pages 3–29. Springer, 2017.

[22] D. Ishii, K. Yoshizoe, and T. Suzumura. Scalable parallel numerical constraint solver using global load balancing. In *Proceedings of the ACM SIGPLAN Workshop on X10*, pages 33–38. ACM, 2015.

[23] L. Jaulin and E. Walter. Guaranteed nonlinear parameter estimation from bounded-error data via interval analysis. *Mathematics and Computers in Simulation*, 35(2):123–137, 1993.

[24] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer. Policy compression for aircraft collision avoidance systems. In *Digital Avionics Systems Conference (DASC), 2016 IEEE/AIAA 35th*, pages 1–10. IEEE, 2016.

[25] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification (CAV)*, pages 97–117. Springer, 2017.

[26] R. B. Kearfott. *Rigorous global search: continuous problems*, volume 13. Springer Science & Business Media, 2013.

[27] R. B. Kearfott and M. Novoa III. Algorithm 681: Intbis, a portable interval newton/bisection package. *ACM Transactions on Mathematical Software (TOMS)*, 16(2):152–157, 1990.

[28] M. J. Kochenderfer, J. E. Holland, and J. P. Chryssanthacopoulos. Next-generation airborne collision avoidance system. Technical report, Massachusetts Institute of Technology-Lincoln Laboratory Lexington United States, 2012.

[29] J. Z. Kolter and E. Wong. Provable defenses against adversarial examples via the convex outer adversarial polytope. *arXiv preprint arXiv:1711.00851*, 2017.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012.

[31] A. Kurakin, I. J. Goodfellow, and S. Bengio. Adversarial machine learning at scale. In *International Conference on Learning Representations (ICLR)*, 2017.

[32] Y. Liu, X. Chen, C. Liu, and D. Song. Delving into transferable adversarial examples and black-box attacks. In *International Conference on Learning Representations (ICLR)*, 2016.

[33] M. Marston and G. Baca. ACAS-Xu initial self-separation flight tests. *NASA Technical Reports Server*, 2015.

[34] R. Moore and W. Lodwick. Interval analysis and fuzzy set theory. *Fuzzy Sets and Systems*, 135(1):5–9, 2003.

[35] R. E. Moore. Interval arithmetic and automatic error analysis in digital computing. Technical report, Applied Mathematics and Statistics Laboratories Technical Report No. 25, Stanford University, 1962.

[36] R. E. Moore. *Methods And Applications Of Interval Analysis*, volume 2. Siam, 1979.

[37] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2574–2582, 2016.

[38] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 807–814, 2010.

[39] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436, 2015.

[40] M. T. Notes. Airborne Collision Avoidance System X. *MIT Lincoln Laboratory*, 2015.

[41] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 506–519. ACM, 2017.

[42] K. Pei, Y. Cao, J. Yang, and S. Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP)*, pages 1–18. ACM, 2017.

[43] K. Pei, Y. Cao, J. Yang, and S. Jana. Towards practical verification of machine learning: The case of computer vision systems. *arXiv preprint arXiv:1712.01785*, 2017.

[44] A. Raghunathan, J. Steinhardt, and P. Liang. Certified defenses against adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2018.

[45] M. J. C. Ramon E. Moore, R. Baker Kearfott. *Introduction to Interval Analysis*. SIAM, 2009.

[46] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[47] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

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

[49] Y. Tian, K. Pei, S. Jana, and B. Ray. DeepTest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering (ICSE)*, 2018.

[50] R. Vaidyanathan and M. El-Halwagi. Global optimization of nonconvex nonlinear programs via interval analysis. *Computers & Chemical Engineering*, 18(10):889–897, 1994.

[51] W. Xu, Y. Qi, and D. Evans. Automatically evading classifiers. In *Proceedings of Network and Distributed Systems Symposium (NDSS)*, 2016.

A Appendix: Formal Definitions for ACAS Xu Properties ϕ_1 to ϕ_{15}

Inputs. Inputs for each ACAS Xu DNN model are:
ρ: the distance between ownship and intruder;
θ: the heading direction angle of ownship relative to intruder;
ψ: heading direction angle of intruder relative to ownship;
v_{own}: speed of ownshipe;
v_{int}: speed of intruder;

Outputs. Outputs for each ACAS Xu DNN model are:
COC: Clear of Conflict;
weak left: heading left with angle 1.5^o/s;
weak right: heading right with angle 1.5^o/s;
strong left: heading left with angle 3.0^o/s;
strong right: heading right with angle 3.0^o/s.

45 Models. There are 45 different models indexed by two extra inputs a_{prev} and τ, model_x_y means the model used when $a_{prev} = x$ and $\tau = y$:

a_{prev}: previous action indexed as {*COC*, *weak left*, *weak right*, *strong left*, *strong right*}.

τ: time until loss of vertical separation indexed as {0, 1, 5, 10, 20, 40, 60, 80, 100}

Property ϕ_1: If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will always be below a certain fixed threshold.

Tested on: all 45 networks.

Input ranges: $\rho \geq 55947.691$, $v_{own} \geq 1145$, $v_{int} \leq 60$.

Desired output: the output of COC is at most 1500.

Property ϕ_2: If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will never be maximal.

Tested on: model_x_y, $x \geq 2$, except model_5_3 and model_4_2

Input ranges: $\rho \geq 55947.691$, $v_{own} \geq 1145$, $v_{int} \leq 60$.

Desired output: the score for COC is not the maximal score.

Property ϕ_3: If the intruder is directly ahead and is moving towards the ownship, the score for COC will not be minimal.

Tested on: all models except model_1_7, model_1_8 and model_1_9

Input ranges: $1500 \leq \rho \leq 1800$, $-0.06 \leq \theta \leq 0.06$, $\psi \geq 3.10$, $v_{own} \geq 980$, $v_{int} \geq 960$.

Desired output: the score for COC is not the minimal score.

Property ϕ_4: If the intruder is directly ahead and is moving away from the ownship but at a lower speed than that of the ownship, the score for COC will not be minimal.

Tested on: all models except model_1_7, model_1_8 and model_1_9

Input ranges: $1500 \leq \rho \leq 1800$, $-0.06 \leq \theta \leq 0.06$, $\psi = 0$, $v_{own} \geq 1000$, $700 \leq v_{int} \leq 800$.

Desired output: the score for COC is not the minimal score.

***Property* ϕ_5*:** If the intruder is near and approaching from the left, the network advises âĂIJstrong rightâĂİ.

Tested on: model_1_1

Input ranges: $250 \leq \rho \leq 400$, $0.2 \leq \theta \leq 0.4$, $-3.141592 \leq \psi \leq -3.141592 + 0.005$, $100 \leq v_{own} \leq 400$, $0 \leq v_{int} \leq 400$.

Desired output: the score for "strong right" is the minimal score.

***Property* ϕ_6*:** If the intruder is sufficiently far away, the network advises COC.

Tested on: model_1_1

Input ranges: $12000 \leq \rho \leq 62000$, $(0.7 \leq \theta \leq 3.141592) \cup (-3.141592 \leq \theta \leq -0.7)$, $-3.141592 \leq \psi \leq -3.141592 + 0.005$, $100 \leq v_{own} \leq 1200$, $0 \leq v_{int} \leq 1200$.

Desired output: the score for COC is the minimal score.

***Property* ϕ_7*:** If vertical separation is large, the network will never advise a strong turn

Tested on: model_1_9

Input ranges: $0 \leq \rho \leq 60760$, $-3.141592 \leq \theta \leq 3.141592$, $-3.141592 \leq \psi \leq 3.141592$, $100 \leq v_{own} \leq 1200$, $0 \leq v_{int} \leq 1200$.

Desired output: the scores for "strong right" and "strong left" are never the minimal scores.

***Property* ϕ_8*:** For a large vertical separation and a previous "weak left" advisory, the network will either output COC or continue advising "weak left."

Tested on: model_2_9

Input ranges: $0 \leq \rho \leq 60760$, $-3.141592 \leq \theta \leq -0.75 \cdot 3.141592$, $-0.1 \leq \psi \leq 0.1$, $600 \leq v_{own} \leq 1200$, $600 \leq v_{int} \leq 1200$.

Desired output: the score for "weak left" is minimal or the score for COC is minimal.

***Property* ϕ_9*:** Even if the previous advisory was "weak right," the presence of a nearby intruder will cause the network to output a "strong left" advisory instead.

Tested on: model_3_3

Input ranges: $2000 \leq \rho \leq 7000$, $0.7 \leq \theta \leq 3.141592$, $-3.141592 \leq \psi \leq -3.141592 + 0.01$, $100 \leq v_{own} \leq 150$, $0 \leq v_{int} \leq 150$.

Desired output: the score for "strong left" is minimal.

***Property* ϕ_{10}*:** For a far away intruder, the network advises COC.

Tested on: model_4_5

Input ranges: $36000 \leq \rho \leq 60760$, $0.7 \leq \theta \leq 3.141592$, $-3.141592 \leq \psi \leq -3.141592 + 0.01$, $900 \leq v_{own} \leq 1200$, $600 \leq v_{int} \leq 1200$.

Desired output: the score for COC is minimal.

***Property* ϕ_{11}*:** If the intruder is near and approaching from the left but the vertical separation is comparably large, the network still tend to advise "strong right" more than COC.

Tested on: model_1_1

Input ranges: $250 \leq \rho \leq 400$, $0.2 \leq \theta \leq 0.4$, $-3.141592 \leq \psi \leq -3.141592 + 0.005$, $100 \leq v_{own} \leq 400$, $0 \leq v_{int} \leq 400$.

Desired output: the score for "strong right" is always smaller than COC.

***Property* ϕ_{12}*:** If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will be the minimal.

Tested on: model_3_3

Input ranges: $\rho \geq 55947.691$, $v_{own} \geq 1145$, $v_{int} \leq 60$.

Desired output: the score for COC is the minimal score.

***Property* ϕ_{13}*:** For a far away intruder but the vertical distance are small, the network always advises COC no matter the directions are.

Tested on: model_1_1

Input ranges: $60000 \leq \rho \leq 60760$, $-3.141592 \leq \theta \leq 3.141592$, $-3.141592 \leq \psi \leq 3.141592$, $0 \leq v_{own} \leq 360$, $0 \leq v_{int} \leq 360$.

Desired output: the score for COC is the minimal.

***Property* ϕ_{14}*:** If the intruder is near and approaching from the left and vertical distance is small, the network always advises strong right no matter previous action is strong right or strong left.

Tested on: model_4_1, model_5_1

Input ranges: $250 \leq \rho \leq 400$, $0.2 \leq \theta \leq 0.4$, $-3.141592 \leq \psi \leq -3.141592 + 0.005$, $100 \leq v_{own} \leq 400$, $0 \leq v_{int} \leq 400$.

Desired output: the score for "strong right" is always the minimal.

***Property* ϕ_{15}*:** If the intruder is near and approaching from the right and vertical distance is small, the network always advises strong left no matter previous action is strong right or strong left.

Tested on: model_4_1, model_5_1

Input ranges: $250 \leq \rho \leq 400$, $-0.4 \leq \theta \leq -0.2$, $-3.141592 \leq \psi \leq -3.141592 + 0.005$, $100 \leq v_{own} \leq 400$, $0 \leq v_{int} \leq 400$.

Desired output: the score for "strong left" is always the minimal.

***Property* S_1*:**

Tested on: model_4_1

Input ranges: $400 \leq \rho \leq 10000$, $\theta = 0.2$, $\psi = -3.141592 + 0.005$, $v_{own} = 10$, $v_{int} = 10$.

Desired output: the score for "strong right" is the minimal.

***Property* S_2*:**

Tested on: model_4_1

Input ranges: $\rho = 400$, $-0.2 \leq \theta \leq 0$, $\psi = -3.141592 + 0.005$, $v_{own} = 1000$, $v_{int} = 1000$.

Desired output: the score for "strong right" is the minimal.

***Property* S_3*:**

Tested on: model_1_2

Input ranges: $\rho = 400$, $-0.1 \leq \theta \leq 0.1$, $\psi = -3.141592$, $v_{own} = 500$, $v_{int} = 600$.

Desired output: the score for "strong right" is the minimal.