Attack-Guided Efficient Robustness Verification of ReLU Neural Networks

Yiwei Zhu [1], Feng Wang [2], Wenjie Wan [1], Min Zhang* [1,3]
[1] *Shanghai Key Laboratory of Trustworthy Computing, ECNU, Shanghai, China*
[2] *Beijing Institute of System Engineering, Beijing, China*
[3] *Shanghai Institute of Intelligent Science and Technology, Tongji University, Shanghai, China*
zhangmin@sei.ecnu.edu.cn

***Abstract*—Nowadays the robustness of Deep Neural Networks (DNN) is gaining much more attention than ever. That is because DNNs are intensively adopted in safety-critical AI-enabled applications such as autonomous driving and authentication control. Formal methods have been proved to be effective to provide provable guarantee to the robustness of DNNs. However, they are suffering from bad scalability due to intrinsic high computational complexity of the verification problem. In this paper, we propose a novel attack-guided approach for efficiently verifying the robustness of neural networks. The novelty of our approach is that we use existing attack approaches to generate coarse adversarial examples, by which we can significantly simply final verification problem. In particular, we are focused on the neural networks that take ReLU activation functions, which are widely adopted for solving classification problems. The experimental results show that our approach outperforms those verification tools based on constraint solving by up to 69 times speedup, while it can compute minimum adversarial examples. The improvement is particularly significant on those adversarially trained networks.**

I. Introduction

Over the last few years, deep neural networks (DNNs) are increasingly employed as intelligent components in software applications by safety critical domains such as medical diagnose [1], autonomous vehicles [2], and security authentication [3]. Applications in these domains require certifications to both security and reliability. However, many works have shown that DNNs often make dangerous mistakes, especially for rare corner case inputs. An even worse fact is that they are also vulnerable to adversarial examples: perturbed inputs that are very similar to some original input but fool DNNs to produce incorrect outputs [4], [5]. Bugs in DNNs may lead to disasters. For example, a woman was struck and killed by an Uber self-driving SUV while crossing the road recently, and the root reason for the accident is that the driving system misclassified the pedestrian as an unknown object due to bad light condition and the pedestrian's black clothes.

It becomes increasingly important to guarantee DNNs can always make correct perception even if target objects are slightly perturbed. This requirement is called robustness. Increasing and guaranteeing the robustness of neural networks are essential tasks. Many approaches have been investigated and proposed to improve and evaluate neural networks' robustness. [6] However, most of these works do not provide provable guarantee because they are based on either testing or statistics.

There have been a lot of works concerning about defense methods to make DNNs more robust to adversarial examples. Currently, the defense methods are being developed along three main directions [7]: i) using modified dataset; ii) modifying network architecture; iii) adding external network model to assist with classification tasks. Attack-based approaches are useful to detect potential flaws in networks by finding real adversarial examples. However, they are evaluated via heuristic attacks, such as the Fast Gradient Sign Method [4] or Projected Gradient Descent [8]. When failing to find adversarial examples, they cannot guarantee whether the robustness of a neural network is actually improved or not.

Recent works have shown formal methods are effective in providing provable robustness guarantee to DNNs by mathematically proving the absence of adversarial examples when a neural network is robust. A comprehensive survey on formal verification of neural networks [9] summarizes various state-of-the-art techniques in this direction. Although formal methods are promising to robustness verification, there are several technical challenges such as low scalability of algorithms and unexplainability of neural network models. In particularly, the scalability issue is a realistic obstacle in robustness verification due to the drastic increase of neural networks' scales. Many existing approaches adopt abstraction and approximation techniques to improve efficiency at the price of sacrificing completeness, i.e., no robustness guarantee when no real adversarial examples are found [10]–[12]. It is hard to achieve balance between accuracy and efficiency.

In this paper, we propose a novel attack-guided approach to accelerate robustness verification of neural networks. The basic idea of our approach is to compute a coarse adversarial example using existing attack techniques then accelerate the solving process by simplifying the constraints in the problem. In particular, we are focused on feedforward ReLU-based DNNs and MILP (Mixed Integer Linear Programming) based verification approaches. We choose feedforward ReLU-based DNNs as they are prevalently adopted in classification problems, and MILP-based approaches are both sound and complete [13]. The robustness verification problem of ReLU-based DNNs can be equivalently deduced to a MILP problem, whose

978-0-7381-3366-9/21/$31.00 ©2021 IEEE

computational complexity is proved NP-complete [14]. The solving time sensitively depends on the number of variables and constraints. In our approach, we simplify the deduced MILP problems before feeding them into MILP solvers such as Gorubi. To simplify the problems, we leverage efficient attack techniques such as gradient attack to compute a coarse adversarial example. Then, we estimate a rough perturbation bound using the coarse adversarial example. The estimated bound helps to remove redundant variables and constraints in the MILP problems, which significantly reduces the solving time.

Our approach inherits the efficiency of gradient attack and the completeness of MILP-based verification. We implement our approach in a prototype tool Agrify (Attack-guided verification). We compare it with three state-of-the-art complete NN verification tools including MIPVerify [13], Neurify [15], Venus [16]. Experimental results show that our approach outperforms other three tools by up to 69 times, and meanwhile it computes minimum adversarial perturbation without loss of completeness.

In summary, this work stays in line with previous sequel of works neural network robustness verification and makes the following two major contributions:

1) A novel approach of combing attack and verification techniques to improve the efficiency of neural network verification.
2) An efficient prototype tool for robustness verification of ReLU-based FNNs, which can compute minimum adversarial examples in non-robust cases.

The remaining part of this paper is organized as follows. Section II introduces key concepts on neural networks and background knowledge related to robustness verification. Section III presents our verification framework. Section IV reports the tool and experimental results. Section V discusses some related work, and Section VI concludes the paper and mentions some future work inspired by our approach.

II. Preliminaries

In this section, we briefly describe some preliminaries that are necessary to understand our approach, including feed-forward deep neural networks, gradient attack, interval analysis and symbolic interval propagation.

A. Robustness of neural networks

A feed-forward deep neural network (FNN) is mathematically an acyclic graph with multiple layers between the input and output layers, with each layer consisting of a number of neurons. The semantics of an FNN can be defined as a function $f\colon \mathbb{R}^m \to \mathbb{R}^n$. The first layer is called input layer, denoted as layer 0, and the last layer is the output layer, denoted as layer l. Every layer between the input and output layers is called a hidden layer, denoted as layer i, for $1 \le i \le l$. Each neuron in hidden layers is connected to all neurons in the previous layer and associated with a bias. Each edge connecting two neurons is associated with a weight. All weights and biases are learned during training phase. The calculation of each layer can be expressed as a function f, which is defined as follows:

$$\begin{aligned} &x^0 = x, \\ &x^{k+1} = \phi(W^k x^k + b^k) \quad for \quad k = 0, ..., l-1, \\ &f(x) = W^l x^l + b^l \end{aligned} \tag{1}$$

where, $x^0 = x \in \mathbb{R}^m$ is the input, W^k and b^k respectively are the weight matrix and bias vector of the k-th layer, and $\phi(.)$ is an activation function applied to the input vector. Here, we only consider the Rectified Linear Units (ReLU) activation function, which is the most commonly-used activation functions in DNNs. It is defined as $ReLU(x) \triangleq \max(x, 0)$ for $x \in \mathbb{R}$.

Given an FNN $f\colon \mathbb{R}^m \to \mathbb{R}^n$, it is robust if and only if it can return the same classification result for a slightly perturbed input and the original input. Namely, the network's output is not affected by small perturbation of given inputs. The perturbation range of an input is usually evaluated by an L-norm distance. There are three widely used L-norms: L_1, L_2 and L_∞ norms. They are mainly used to measure the similarity between two inputs. In this work, we mainly consider L_∞ norm. That is, for each pair of vectors x, x' with the same size,

$$\|x - x'\|_\infty \equiv \max_{1 \le i \le n}\{|x_i - x'_i|\} \tag{2}$$

where i is a vector component index. And our work also supports the other two norms.

For an input x and a distance threshold ϵ, we define a norm-ball is an input region which contains all the inputs x' such that $\|x - x'\|_\infty \le \epsilon$. An FNN is called local robust w.r.t. the input x and distance threshold ϵ, if and only if for all inputs x' such that $\|x - x'\|_\infty \le \epsilon$, the FNN always returns the same classification result [17]. In the literature, an FNN is called global robust if it is local robust for each input of the given test dataset [18]. In this work, we focus on the local robustness.

Definition 1 (Robustness): Given an FNN $f\colon \mathbb{R}^m \to \mathbb{R}^n$, an input $x \in \mathbb{R}^m$, and an L_∞ distance threshold ϵ, f is robust w.r.t. the input x and distance threshold ϵ if

$$\mathcal{L}(f(x)) = \mathcal{L}(f(x')) \tag{3}$$

for all $x' \in I$ with $\|x - x'\|_\infty \le \epsilon$ and $\mathcal{L}(.)=\text{argmax}_i(f_i(x))$.

If there exists an input $x' \in \mathbb{R}^m$ such that $\|x - x'\|_\infty \le \epsilon$ and $\mathcal{L}(f(x)) \ne \mathcal{L}(f(x'))$, then we call x' an adversarial example of x. Therefore, the robustness verification is equivalent to the problem of checking the satisfiability of the conjunction of the two constraints $\|x - x'\|_\infty \le \epsilon$ and $\mathcal{L}(f(x)) \ne \mathcal{L}(f(x'))$.

B. Project Gradient Descent (PGD) Attack

Gradient attacks mainly refer to artificially creating interference and confusing models to produce wrong results. The best way to make attack effectively is to maximize the loss function. The classification model keeps parameters unchanged in the process of gradient attacking. The method misclassifies model by changing the input value.

There are some typical gradient-based adversarial attack methods. These methods aim to find adversarial example of

DNNs. Project Gradient Descent [8] is an iterative attack method. It's an advanced fast gradient sign method with several iterations. The PGD attack is mainly a loop, it first initializes the search at a random point within the allowed norm ball, then it runs several iterations, each iteration runs basic method to find the adversarial example. Compared with the basic method, this iteration can correct the direction of the gradient in time. The process can be formulated as

$$x_{t+1} = \prod_{x+S} (x^t + \alpha sign(\nabla_x L(\theta, x, y))) \tag{4}$$

Although the gradient attack method can efficiently find adversarial examples, it cannot guarantee the minimum distance between the found adversarial example and the original input, so the minimum adversarial perturbation cannot be determined.

C. Neural Network Verification Techniques

1) Symbolic Interval Analysis: Interval arithmetic [19] is an efficient way of deriving pre-activation bounds by computing and propagating the interval of the input range through the network. It can be used to for verification by estimating output range of neural networks. However, the resulting bounds often suffer from large over-approximated error because naive interval analysis ignores the input dependencies of input nodes during propagation. To minimize over-approximation of output intervals, approaches based on symbolic interval analysis [20] replace concrete interval by symbolic ones during propagation in order to preserve variable dependencies.

Figure 1 shows an example of symbolic interval analysis. By propagating variables x, y layer by layer, we can obtain the expression $x - y$ for node n_5, which gives an output interval $[-3, 1]$. This result is much tighter than the one $[-7, 5]$, which is computed using the naive interval arithmetic approach.

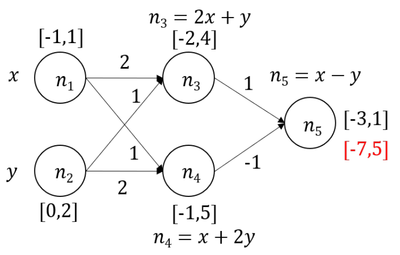


Fig. 1. Example of symbolic interval propagation

2) MILP Formulation: The main idea of MILP-based method is to express piece-wise linear activation function as a Mixed Integer Linear Program (MILP). Specifically, the verification property is verified if and only if the corresponding MILP problem is unsatisfiable. The MILP encoding of a neuron depends on its state, and in our approach the pre-activation bounds of the neurons have already been calculated. We introduce the way of formulating ReLU in MIPVerify [13]. Let $y = \max(x, 0)$, and $l \le x \le u$, if the neuron is strictly active, then $y \equiv x$. Similarly, if the neuron is strictly inactive, then $y \equiv 0$. Otherwise, the neuron is called *unstable*. The encoding of a neuron is given by the following constraints:

$$\begin{aligned} & y \le x - l(1 - a) \\ & y \ge x \\ & y \le u \cdot a \\ & y \ge 0 \\ & a \in \{0, 1\} \end{aligned} \tag{5}$$

where, a is a binary variable such that $a = 0$ iff $y = 0$ and $a = 1$ iff $y = x$.

In the context of FNNs, the efficiency of solving an MILP problem i) the number of binary variables caused by unstable nodes, and ii) the accuracy of the pre-activation bounds, it related to the method used to calculate the bounds.

D. Minimum Adversarial Perturbation

Let $d(.)$ denotes a distance metric that measures the distance between two input images. The minimum adversarial perturbation under d w.r.t the input x, adversarial example x', true label $\mathcal{L}(f(x))$ is the solution to the optimization:

$$\begin{aligned} & min_{x'} d(x, x') \\ subject \quad to \quad & \mathcal{L}(f(x')) \neq \mathcal{L}(f(x)) \\ & x' \in \mathbb{R}_{valid} \end{aligned} \tag{6}$$

We can target attacks to generate adversarial examples corresponding to the label l by replacing $\mathcal{L}(f(x')) \neq \mathcal{L}(f(x))$ with $\mathcal{L}(f(x')) = l$.

III. The Verification Framework

This section presents our approach of integrating attack and verification techniques to the robustness verification of FNNs.

A. Overview

Our approach combines gradient attack and symbolic interval analysis to the MILP-based method. The gradient attack provides us with the initial perturbation values, we use this information to define initial bounds. And then, we obtain tighter bounds for hidden neurons through symbolic propagation interval analysis. Tighter bounds lead to fewer binary variables in the verification problem. Verification efficiency is consequently improved by reducing the search space.

The overview of our approach is shown in Algorithm 1. Given an FNN f, an input $x \in \mathbb{R}^n$ and an epsilon threshold ϵ, Algorithm 1 outputs one of the following results:

- Robust, indicating that the robustness property is satisfied.
- An adversarial example, indicating that the violation of the robustness property.

We first get the rough perturbation $\widehat{\epsilon}$ using the PGD attack approach (line 1). Then, we calculate node bounds according to the input x and $\widehat{\epsilon}$ using symbolic propagation interval analysis (line 2). After obtaining bounds, we build the MILP model, construct necessary constraints, and set objective function (lines 3–5), where *milp* is the initial model that cannot return a preconceived result. The build_MILP_model function takes

Algorithm 1: Robustness verification of DNN

```
input  : A DNN $f$, an input $x$, a distance threshold $\epsilon$
output: {Robust, an adversarial example}
1  $\widehat{\epsilon}$:=PGD_Attack($x$);
2  node_bounds:=symbolic_propagation($x$, $\widehat{\epsilon}$);
3  milp:=build_MILP_model($f$, $x$,$\epsilon$,node_bounds);
4  milp′ :=add_constraints(milp);
5  milp″ :=set_objective_function(milp′);
6  result:=milp_solver(milp″);
7  if result is infeasible then
8      return Robust;
9  else
10     $x'$ :=get_adv_example(milp″);
11     return $x'$ ;
```

the DNN f, the input x, the distance ϵ as inputs, and the output initial model *milp*. It checks whether the FNN f is robust or not by invoking milp_solver (line 6), the solver solves the MILP model *milp* and output infeasible or optimal objective value. If the model is unsatisfiable, it returns robust. Otherwise, we exact an adversarial example from the solver and return it (lines 10–11).

B. *Initial Perturbation Bound Estimation*

Our approach relies on an initial perturbation bound, which is expected to be close to the optimal one. We estimate the bound using attack techniques. In our approach, we choose the PGD gradient attack method. At first, we use the PGD gradient attack to produce an adversarial example for each input. Each adversarial example returned by the PGD attack has minimum perturbation that the PGD attack can find. We do not use the adversarial example directly but leverage the information that comes from the adversarial example to find a rough perturbation range $\widehat{\epsilon}$. This information can be seen as a rough robust radius, and we leverage these $\widehat{\epsilon}$ to calculate the bounds of all the neurons in networks.

We use the example in Figure 2 to explain the process of our method. We assume that the input $x = -2, y = 0, z = 3$, then through the PGD attack, we obtain an adversarial example $x' = -1, y' = 0, z' = 2.5$. According to L-∞ norm, we get the $\widehat{\epsilon} = max\{|x = x'|, |y - y'|, |z - z'|\} = 1$. The $\widehat{\epsilon}$ will be used for subsequent bounds calculations.

In MIPVerify, there are two ways of determining the initial perturbation range (the perturbation range determines the size of the solver's search space). One is to simply set $\widehat{\epsilon} = 1$. It will produce more binary variables, and the solver will find an adversarial example in hole search space. This way consumes much time on solving. The other one is to allow users to provide an initial value $\widehat{\epsilon}$, which is hard for users to choose an appropriate value.

To the best of our knowledge, most existing robustness verification approaches use formal methods without considering the AI domain's attack methods. Actually, in terms of finding adversarial examples, gradient attacks have an extremely high efficiency compared to formal verification methods. Although the gradient attack method does not guarantee that the adversarial examples have minimum perturbation, it does not affect our use of effective information in the adversarial examples.

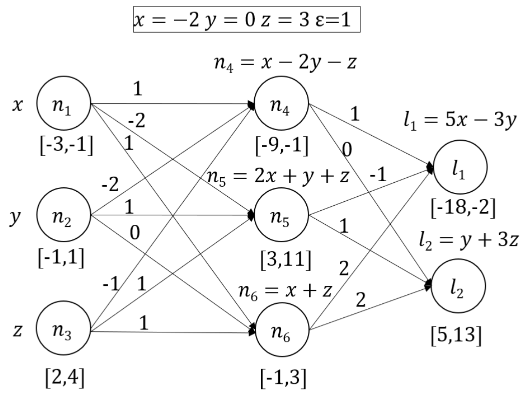


Fig. 2. Example of calculating the bounds of neurons

C. *Internal Neuron Bound Estimation*

In the MILP model solving problem, one of the key factors affecting the efficiency of the solution is the number of integer variables (here is binary variables) [14]. If we can prove that the phase of ReLU is stable, we can avoid introducing binary variables. More generally, the loose bound input to a neuron will propagate downstream, resulting in a looser bound for the neurons in the following layer. Therefore, calculating tight bounds for internal neurons is a key point to improve verification efficiency.

We calculate the bounds using symbolic interval propagation analysis techniques. First, we set the initial interval $[x - \widehat{\epsilon}, x + \widehat{\epsilon}]$. Compared with other MILP-based methods such as MIPVerify, we use approaches previously mentioned instead of using MILP solver or naive interval propagation to calculate the bounds of the nodes in hidden layers of DNNs. Because using naive interval arithmetic leads to large over-approximations, while using solver consumes too much time and causes query timeouts. The combination of gradient attack and symbolic interval propagation analysis can significantly improve efficiency and produce tighter bounds.

Let us consider the example in Figure 2, we assume that the inputs $x = -3, y = 0, z = 3$ and $\widehat{\epsilon} = 1$. Then the initial intervals for neurons n_1, n_2, n_3 are $[-3, -1]$, $[-1, 1]$, and $[2, 4]$, respectively. The input bounds are propagated layer by layer using symbolic propagation interval analysis as we introduced in Section II-C. We obtain the bounds for all the internal neurons in the network.

In our approach, we calculate an initial epsilon $\widehat{\epsilon}$ from adversarial examples produced by PGD gradient attack. The $\widehat{\epsilon}$ is usually close to the optimal robust radius, providing tighter input bounds. A tighter input bound produces fewer binary variables, by which search space is further reduced. We can effectively reduce the build time and solution time and made a trade-off between the two.

D. MILP Problem Building and Solving

After obtaining all the bounds of internal neurons, we build the MILP model based on these bounds. We create a set of variables for the neurons in the input layer, these variables propagate through the network, each neuron thus can be represented by a variable or expression according to the MILP formulation. We construct constraints to ensure that the returned results are adversarial examples, i.e., line 2 of the equation 6. By default, we target attack the label l_s corresponding to the second-highest confidence value, $l_s = argmax_i(f_i(x))$ and $i \neq j$ where j is the true label of the input. The objective function is set to find minimum adversarial perturbation. So far, the MILP model is constructed, which is essentially a series of variables, expressions, constraints, and an objective function.

Considering the example shown in Figure 2, we create three MILP variables $v_1 = -3 + \epsilon_1, v_2 = 0 + \epsilon_2 . v_3 + \epsilon_3$ for the neurons in the input layer. When the variable propagates to the second layer, we create a set of expressions and constraints for neuron n_4, n_5, n_6, $v_1 - 2v_2 - v_3$ for n_4, $-2v_1 + 2v_2 + v_3$ for n_5, $v_1 + v_3$ for n_6. Then, we add constraints to the unstable neuron based on the previously calculated bounds. The neuron n_4 with bounds $[-9, -1]$ is strictly inactive, n_5 with bounds [3,11] is strictly active, and n_6 with bounds [-1,3] is unstable. We only need to introduce a binary variable a for n_6, and consequently construct the following constraints:

$$\begin{aligned} & v_{rect} \leq v_1 + v_3 - (-1)(1 - a) \\ & v_{rect} \geq v_1 + v_3 \\ & v_{rect} \leq u \cdot a \\ & v_{rect} \geq 0 \\ & a \in \{0, 1\} \end{aligned} \tag{7}$$

We use v_{rect} to represent n_6 when propagating it to next layer. Therefore, the label l_1 and l_2 can be expressed as $3v_1 - 3v_2 - 2v_3 + 2v_{rect}$, $-2v_1 + v_2 + v_3 + 2v_{rect}$ respectively. We assume that the true label of the input is l_2, then we construct constraint $3v_1 - 3v_2 - 2v_3 + 2v_{rect} > -2v_1 + v_2 + v_3 + 2v_{rect}$ for verifying robustness. The objective function is set to $min(\epsilon_1 + \epsilon_2 + \epsilon_3)$.

We use Gurobi to solve this model, if solver returns an adversarial example, the network does not satisfy robustness; if solver returns that the model is not feasible, the network is robustness on this input.

E. Completeness of the Approach

We argue that the techniques used in our approaches preserve completeness. We only get the perturbation range closer to the true robustness radius for the initial perturbation range. It reasonably eliminates unnecessary search space. For bounds of nodes in hidden layers, we illustrate this with the example above, as shown in Figure 2. Assuming that initial input bounds are n_1 ∈[-3,1], n_2 ∈[-1,1] and n_2 ∈[2,4]. Using symbolic interval propagation analysis, it can get that the bounds of neuron n_6 is [-1,3]. If we use the solver to computing the bounds of neuron n_6, they are [0,3]. Then our approach produces one more binary variable in MILP formulation than the most accurate one (cause that the lower bound of neuron n_5 is less than 0, and the upper bound is greater than 0). For a stable neuron, adding the MILP constraints does not affect the final result. Therefore, our approach guarantees completeness like other MILP-based methods.

IV. Implementation and Evaluation

This section aims to demonstrate the effectiveness of our approach to verify the robustness of DNNs and obtain an adversarial example with minimum adversarial perturbation when the DNN does not satisfy robustness.

A. Implementation

We implement our verification approach in a toolkit called Agrify. The implementation language is Python. Our tool is applicable to ReLU-based FNNs, which can be adversarially trained, and the last layer can contain softmax activation functions. We choose Gurobi as the back-end MILP solver.

B. Compared Methods

We only compare our approach with complete methods, which always return a definite result that whether an FNN satisfies robustness and guarantees the correctness of the result. Particularly, we choose the three promising tools MIPVerify [13], Neurify [15] and Venus [16] for comparison.

- MIPVerify formulates the robustness verification problem as a MILP program. It improves plain MILP-based approaches via a tighter formulation for non-linearities and a novel presolve algorithm.
- Neurify is a verification tool that can choose between completeness and efficiency. It introduces symbolic interval analysis and linear relaxation to compute tighter bounds of outputs. Neurify is theoretically complete under a fixed number of refinements, but the refinement process might take too much time and thereby a threshold is set to force termination.
- Venus is one of the most efficient approximation-based DNN verification tools. Its main idea is to reduce the search space via dependency analysis and dependency cuts during a branch-and-bound approach.

C. Datasets

We use the most commonly used benchmarks in the context of DNNs verification:

- MNIST [21] is a dataset of handwritten digits 0-9. Each image in MNIST is formatted as a 28x28x1-pixel grayscale image. In our experiment, we choose four neural networks that have been verified by MIPVerify, Neurify, and Venus. They have different architectures denoted by FNN$_i$ with i=1,2,3,4. Among them, FNN$_4$ is adversarially trained with SDP when the dual of a semidefinite relaxation is used, as in [22].
- Fashion-MNIST [23] is a dataset of clothing. Each image in Fashion-MNIST is also formatted as a 28x28x1-pixel grayscale image. We trained two neural networks for the experiment, denoted as FNN$_5$ and FNN$_6$.

TABLE I
EXPERIMENT RESULTS ON FOUR FNNS WITH MNIST DATESET

	FNN_1					FNN_2					FNN_3					FNN_4				
	n_s	t_{all}	n_a	n_n	n_u	n_s	t_{all}	n_a	n_n	n_u	n_s	t_{all}	n_a	n_n	n_u	n_s	t_{all}	n_a	n_n	n_u
Agrify	99	202.769	96	3	0	99	565.763	87	12	0	99	24731.163	83	15	1	98	1027.622	4	94	0
MIPVerify	99	1451.656	96	3	0	99	2261.611	87	12	0	99	69054.317	83	15	1	98	36494.919	4	94	0
Neurify	100	798.837	97	3	0	100	OOT	-	-	-	100	OOM	-	-	-	100	OOT	-	-	-
Venus	100	49.931	97	3	0	100	67.277	88	12	0	100	5031.701	84	15	1	100	36377.141	6	90	4

Remarks: $n_s, t_{all}, n_a, n_n, n_u$ mean the number of valid input images, time (in second) on verification, the numbers of computed adversarial examples, certified cases and unknown cases. OOT means out-of-time, and OOM means out-of-memory.

TABLE II
DETAILS OF THE FNNS USED IN OUR EXPERIMENTS

Model	**ReLUs**	**Network Architecture**	**Source**	**Note**
		MNIST		
FNN_1	48	⟨784,24,24,10⟩	Neurify	
FNN_2	60	⟨784,40,20,10⟩	MIPVerify	
FNN_3	1024	⟨784,512,512,10⟩	Venus	
FNN_4	500	⟨784,500,10⟩	MIPVerify	SDP
		Fashion-MNIST		
FNN_5	48	⟨784,24,24,10⟩	This work	
FNN_6	60	⟨784,40,20,10⟩	This work	

Details of these networks are given in Table II, where column 2 gives the number of ReLU activation functions, and column 3 gives the number of layers and neurons in each layer.

D. Experimental Settings

We verified the network against local robustness for a perturbation radius of 15 on the first 100 images from the test set of MNIST. Due to normalization, some parameters need to be set to 15/255.0. For the experiments, MIPVerify was running with *LInfNormBoundedPerturbationFamily* (15/255.0), this parameter means MIPverify will limit L-∞ norm bound; all pixel will not be allowed to modify larger than the parameter; Neurify was running with MAX_THREAD set to 1, depth set to 500, radius set to 15; Venus was running with the radius set to 15/255.0, and other parameters reported in [16]. Agrify was running with the radius set to 15/255.0.

We also choose the first 100 images from the test set of Fashion-MNIST for experiments, the perturbation radius set to 15. We focus on performance on evaluating minimum adversarial perturbation on this dataset. MIPVerify was running with *LInfNormBoundedPerturbationFamily* set to 15/255.0; Agrify was running with the radius set to 15/255.0.

The timeout threshold is set one hour for each image and twenty-four hours for the whole 100 queries. All the experiments were conducted on a workstation running Ubuntu 18.04 with a 32-core AMD Ryzen Thread-ripper 3970X CPU @ 3.7GHz and 128 GB of RAM.

E. Result Analysis

Table I depicts the performance of four tools on the four networks mentioned above. The tables give the number n_s of verification queries that were solved (note that Agrify and MIPVerify will skip queries misclassified by neural networks), the overall time t_{all} taken for all queries, the number n_{adv}, $n_{non\text{-}adv}$, $n_{unknown}$ of verification queries that return adversarial example, robust, and unknown when verification fails due to out of time or memory.

The results show that Agrify's performance was superior on all four networks. MIPVerify takes the longest of all experiments that have not timed out. Neurify was timed out on FNN_2 and FNN_4, and for FNN_3, it consumed excessive memory. Venus was the most performing of the toolkits on the first three networks. However, its performance on FNN_4 was not well, because the network is adversarially trained, and Venus can hardly cope with such type of structured network.

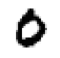
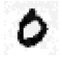
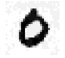
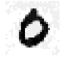
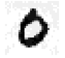
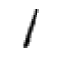
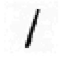
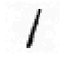
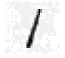
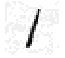
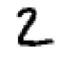
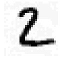
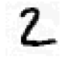
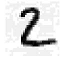
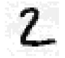
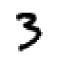
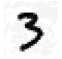
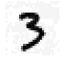
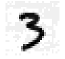
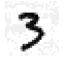
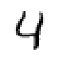
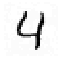
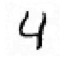
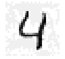
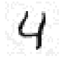
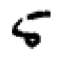
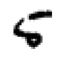
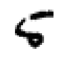
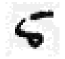
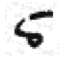
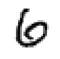
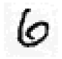
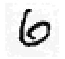
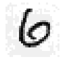
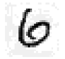
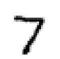
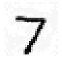
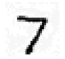
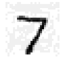
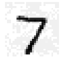
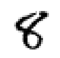
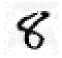
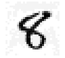
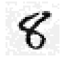
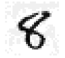
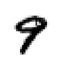
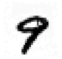
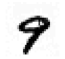
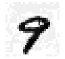
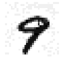
In terms of verification accuracy, the performance results of Agrify and MIPVerify are the same, and Venus shows similar results. That is because both methods in experiments are complete. On FNN_4, Venus returns 4 unknown results where Agrify and MIPVerify return 0 unknown results. It can be considered that our approach can handle some corner cases.

In addition to the accuracy and efficiency of the verification, the perturbation value of the returned adversarial example is also one of the important factors we consider.

Table III shows the images of the MNIST test set, which contains one image for each label and the visual results of adversarial examples returned by the different tools. To see the difference more clearly, we made the image color-inverted. There is a clear difference between them: the adversarial perturbation. Adversarial examples returned by Neurify and Venus are more distorted than ones returned by Agrify and MIPVerify. That is because Agrify and MIPVerify can evaluate the minimum adversarial perturbation, they return adversarial examples with minimum perturbation while Neurify and Venus return adversarial examples with larger perturbation (according to the user's parameter). Note that some adversarial examples returned by Venus are not really adversarial examples (after the second confirmation), the reason for this phenomenon is the accuracy of floating-point numbers in the Venus code. Furthermore, solving the optimal result is costly at the expense of efficiency.

Figure 3 shows the adversarial perturbations returned by the four tools on FNN_1 for 100 queries. Since the result visualization curves are highly overlapped, we display them separately. The results show that Venus and Neurify return similar curves because they have the same strategy: they will return adversarial examples based on hyperparameters. In our experiment, we set perturbation radius 15/255.0, so all adversarial examples they returned are with 15 pixel perturbation. And Agrify and MIPVerify returns similar results, which are close to the optimal ones. Some of the returned results are even trivial as they are close to 0. That means the adversarial examples returned by Agrify and MIPVerify are close to the input images.

TABLE III
Example of adversarial examples on MNIST

Original Image	Agrify	MIPVerify	Neurify	Venus
 0	 6	 6	 6	 6
 1	 8	 8	 8	 8
 2	 5	 5	 5	 6
 3	 5	 5	 5	 3
 4	 9	 9	 6	 9
 5	 6	 6	 6	 5
 6	 2	 2	 0	 0
 7	 3	 3	 0	 7
 8	 6	 6	 2	 6
 9	 4	 4	 8	 4

Tables IV and V show the performance of our tool and MIPVerify on the FNN$_5$ and FNN$_6$, trained on the dataset Fashion-MNIST. We cannot compare with other two tools because they do not support this network architecture. It can be seen that our tool takes much less time to verify 100 images than MIPVerify, while it can compute almost the same adversarial examples. There is basically no difference between the adversarial examples and the original image. This is back to the original intention of verifying the robustness of neural networks. The subtle human-imperceptible perturbations will cause the neural network to make wrong judgments, resulting in unpredictable consequences. We believe that these minimally perturbed adversarial examples can help improve the robustness of neural networks.

TABLE IV
Experiment Results on Fashion-MNIST

	FNN_5					FNN_6				
	n_s	t_{all}	n_a	n_n	n_u	n_s	t_{all}	n_a	n_n	n_u
Agrify	87	350.75	85	2	0	87	1041.99	86	0	0
MIPVerify	87	1548.15	85	2	0	87	3302.66	86	0	0

In summary, the above experiments show that our approach can verify the robustness property of DNNs more efficiently. Moreover, it can evaluate the mean minimum adversarial perturbation.

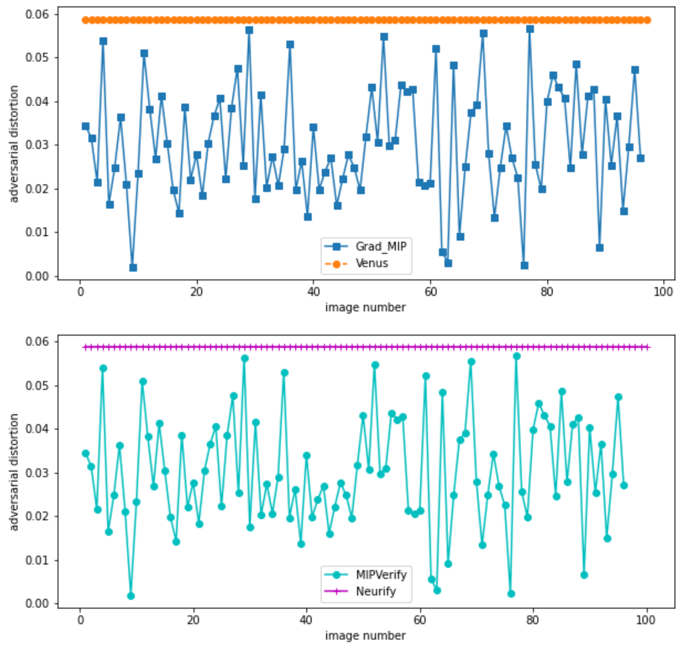


Fig. 3. Adversarial perturbation of adversarial examples returned by four tools on FNN$_1$

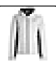
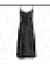
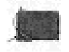
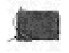
V. Related Work

Our work relates most closely to other work on verification of ReLU-based DNNs. Katz *et al.* [24] proved that the verification problem of FNNs against simple properties such as robustness is NP-complete. Due to high computational complexity, many verification methods have The robustness verification methods of neural networks can be divided into two types: complete and incomplete.

Complete methods can be divided into three main groups: (i) MILP solving [13], [25]–[27] that formulate the verification problem at hand as a mixed-integer linear program; (ii) SMT-based methods that encode the verification problem as the satisfiability modulo theory problem [24], [28]; (iii) methods that use a combination of over-approximating and refinement to get a definite result [15], [20]. Recently, a novel dependency analysis based framework [16] has been proposed, which is also based on MILP-solving. Although these methods suffer from limited scalability, they return a definite answer as to whether the neural networks satisfy robustness property.

In contrast, incomplete ones may erroneously conclude that the neural network is not robust when it actually is. Approximation and abstraction are two effective techniques for robustness verification with better scalability. Incomplete methods include duality [29], abstract interpretation [11] [10], [30], symbolic interval analysis [20], layer-by-layer approximations of the adversarial polytope [31], discretizing the search space [17], linear approximations [12], bounding the local Lipschitz constant [12], or bounding the activation of the ReLU activation function with linear functions [12]. All these methods have a common feature. They tune the verification problem into a linear programming problem by over-approximating. The result is improved scalability at the expense of losing completeness.

TABLE V
Example of adversarial examples on Fashion-MNIST.

Original Image	T-Shirt	Trouser	Pullover	Dress	 Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
Agrify	Dress	T-Shirt	Shirt	 Trouser	Shirt	Dress	T-Shirt	Ankle boot	 Shirt	Sneaker
MIPVerify	Dress	T-Shirt	Shirt	Trouser	Shirt	Dress	T-Shirt	Ankle boot	 Shirt	Sneaker

VI. Conclusion

In this paper, we have proposed an efficient approach to verify the robustness of neural networks by integrating attack and verification techniques. We simplify verification problem using attack result, which significantly reduces the verification time. We implemented our approach into a prototype tool Agrify. Experimental results showed that Agrify can verify neural networks efficiently and evaluate minimum adversarial perturbation, compared with other relevant tools. In particular, Agrify performs better in terms of combining verification and minimum adversarial perturbation evaluation, achieving high efficiency without losing verification accuracy. Due to the intrinsic high computational complexity of neural network verification, we believe that attack techniques are algorithmically helpful to improve verification efficiency by simplifying the formal models to verify. We would investigate integrating other existing attack approaches into verification in our future work.

Acknowledgments

This work is supported by National Key Research and Development Program (2020AAA0107800), Huawei Technologies Co., Ltd., and NSFC projects (No. 61872146, 61872144).

References

[1] D. C. Ciresan, A. Giusti, L. M. Gambardella, and J. Schmidhuber, "Deep neural networks segment neuronal membranes in electron microscopy images," in *26th NIPS*, 2012, pp. 2852–2860.

[2] P. Kebria, A. Khosravi, S. Salaken, and S. Nahavandi, "Deep imitation learning for autonomous vehicles based on convolutional neural networks," *IEEE CAA J. Autom. Sinica*, vol. 7, no. 1, pp. 82–95, 2020.

[3] T. Goel and R. Murugan, "Classifier for face recognition based on deep convolutional - optimized kernel extreme learning machine," *Comput. Electr. Eng.*, vol. 85, p. 106640, 2020.

[4] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *3rd ICLR*, 2015.

[5] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *2nd ICLR*, 2014.

[6] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, and X. Yi, "A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability," *Comput. Sci. Rev.*, vol. 37, p. 100270, 2020.

[7] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," *CoRR*, vol. abs/1801.00553, 2018.

[8] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *6th ICLR*, 2018.

[9] C. Liu, T. Arnon, C. Lazarus, C. Barrett, and M. Kochenderfer, "Algorithms for verifying deep neural networks," *CoRR*, vol. abs/1903.06758, 2019.

[10] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev, "Fast and effective robustness certification," in *31th NeurIPS*, 2018, pp. 10 825–10 836.

[11] 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 *SP'18*, 2018, pp. 3–18.

[12] T. Weng, H. Zhang, H. Chen, Z. Song, C. Hsieh, L. Daniel, D. S. Boning, and I. S. Dhillon, "Towards fast computation of certified robustness for relu networks," in *35th ICML*, ser. Proceedings of Machine Learning Research, vol. 80, 2018, pp. 5273–5282.

[13] V. Tjeng, K. Y. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," in *7th ICLR*, 2019.

[14] J. P. Vielma, "Mixed integer linear programming formulation techniques," *Siam Review*, vol. 57, no. 1, pp. 3–57, 2015.

[15] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Efficient formal safety analysis of neural networks," in *31th NeurIPS*, 2018, pp. 6369–6379.

[16] E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, and R. Misener, "Efficient verification of relu-based neural networks via dependency analysis," in *34th AAAI*, 2020, pp. 3291–3299.

[17] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *29th CAV*, vol. 10426, 2017, pp. 3–29.

[18] W. Ruan, M. Wu, Y. Sun, X. Huang, D. Kroening, and M. Kwiatkowska, "Global robustness evaluation of deep neural networks with provable guarantees for the hamming distance," in *28th IJCAI*, 2019, pp. 5944–5952.

[19] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to Interval Analysis*. SIAM, 2009.

[20] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *27th USENIX Security Symposium, USENIX Security 2018*, 2018, pp. 1599–1614.

[21] Y. Lecun and L. Bottou, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, 1998.

[22] A. Raghunathan, J. Steinhardt, and P. Liang, "Certified defenses against adversarial examples," in *6th ICLR*, 2018.

[23] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *CoRR*, vol. abs/1708.07747, 2017.

[24] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *29th CAV*, ser. LNCS, vol. 10426, 2017, pp. 97–117.

[25] A. Lomuscio and L. Maganti, "An approach to reachability analysis for feed-forward relu neural networks," *CoRR*, vol. abs/1706.07351, 2017.

[26] C. Cheng, G. Nührenberg, and H. Ruess, "Maximum resilience of artificial neural networks," in *15th ATVA*, ser. LNCS, vol. 10482, 2017, pp. 251–268.

[27] M. Fischetti and J. Jo, "Deep neural networks and mixed integer linear optimization," *Constraints An Int. J.*, vol. 23, no. 3, pp. 296–309, 2018.

[28] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," in *15th ATVA*, ser. LNCS, vol. 10482, 2017, pp. 269–286.

[29] K. Dvijotham, R. Stanforth, S. Gowal, T. A. Mann, and P. Kohli, "A dual approach to scalable verification of deep networks," in *34th UAI*, 2018, pp. 550–559.

[30] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "An abstract domain for certifying neural networks," *Proc. ACM Program. Lang.*, vol. 3, no. POPL, pp. 41:1–41:30, 2019.

[31] W. Xiang, H. Tran, and T. T. Johnson, "Output reachable set estimation and verification for multilayer neural networks," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 29, no. 11, pp. 5777–5783, 2018.