Improving Neural Network Verification Efficiency Through Perturbation Refinement

Minal Suresh Patil(✉) and Kary Främling

Umeå universitet, UNIVERSITETSTORGET 4, Umeå, Sweden
{minal.patil,kary.framling}@umu.se

Abstract. This paper presents a novel approach to efficient neural network verification through the use of adversarial attacks and symbolic interval propagation. The proposed method leverages low-cost adversarial attacks to quickly obtain a rough estimate of the first set of bounds, and then utilizes symbolic interval propagation to compute tighter bounds. We demonstrate the effectiveness of our proposed method on the popular MNIST dataset, which contains hand-written digit images. The results show that the proposed method achieves state-of-the-art verification accuracy with significantly reduced computational cost, making it a promising approach for practical neural network verification.

Keywords: Perturbation Refinement · Neural Network Verification · Adversarial Robustness

1 Introduction

Deep neural networks (DNNs) are widely used today. Their ability to generalise and thus work well even on previously unknown inputs is a key factor in their widespread use. Although this has many useful advantages, it could occasionally render DNNs unreliable. This dearth of dependability can actually come at a terrible price in applications that are either safety- or business-critical. Evidently, a trained network's instability is primarily caused by its inability to withstand input perturbations, or the fact that even minor changes to some inputs can significantly alter the network's output. In a lot of application domains, this is not ideal. Consider, for instance, a network that has been taught to alert aircraft to change their paths in response to approaching intruder aircraft. It is reasonable to anticipate that such a network will be capable of making sound decisions, meaning that the advice given in two situations that are strikingly similar should not diverge greatly. However, if that is not the case, then showing the network's lack of resilience through adversarial inputs can aid in both network improvement and determining when the network should hand over control to a more dependable entity.

When a network and an input are provided, an adversarial input is one that is very similar to the input but the outputs of the network for the two inputs

L. Iliadis et al. (Eds.): ICANN 2023, LNCS 14254, pp. 504–515, 2023.
https://doi.org/10.1007/978-3-031-44207-0_42

are very distinct. Finding adversarial sources has been the subject of extensive research in the past [3,7,9,17]. Depending on whether they take into account the architecture of the network during the study or not, these approaches can be categorised as black-box or white-box techniques. Both of these groups have produced a wide range of techniques, from the creation of random attacks [14] and gradient-based approaches [1] to symbolic execution [18,22,26], fault localization [24], coverage-guided testing, SMT, and ILP solving [10,23].

Interval analysis is a method that's used to verify the safety and robustness of neural networks by estimating their output ranges for a given input. This is achieved by calculating both the upper and lower bounds of the output using interval arithmetic, as described in the literature [11]. To improve the accuracy of this approach, researchers have proposed a related technique known as symbolic interval analysis [22]. This involves approximating the output range of a neural network by computing the upper and lower bounds using symbolic mathematical expressions, which can result in tighter bounds and improved verification outcomes.

Robustness verification of neural networks is essential to ensure that they behave correctly and reliably in the presence of adversarial attacks or unexpected inputs. However, the verification process can be computationally expensive, especially for large and complex neural networks. Therefore, accelerating robustness verification of neural networks is crucial to make it feasible for practical applications. Numerous techniques have been proposed to utilize abstraction to achieve robustness [5,13]. Since ReLU activation function is commonly used in neural networks, it is more practical to investigate the problem of verifying robustness [15]. As ReLU networks have a piecewise-linear structure, the problem of verifying robustness can be transformed into a standard Mixed-Integer Linear Programming (MILP) problem, which can be tackled using branch-and-bound methods [6]. However, for large-scale ReLU networks, solving MILP problems for verifying robustness is still challenging. The difficulty of systematically searching the high dimensional and continuous input space makes it challenging to ensure that an adversarial example can be found, even if it exists. Therefore, machine learning models that appear robust to existing attacks may still have security weaknesses in practice. Off-the-shelf MILP solvers cannot make use of solutions gathered at a low cost via gradient-based adversarial attacks to quicken up its search. To tackle this issue, we propose *warm-starting* and *bounds tightening* techniques by integrate symbolic interval analysis to obtain tighter bounds for a gradient-based adversarial example which is formulated as MILP formulation. This can reduce the number of iterations required to converge to the optimal solution, and hence the computation time. The contribution is summed up as follow:

– An approach called *warm-starting* has been proposed to incorporate cheap solutions obtained from adversarial attacks, with the goal of reducing the search space that a MILP solver would otherwise have to explore. Additionally, a technique called *bound tightening* has been introduced to tighten the

bounds on the neurons, which can further improve the accuracy and efficiency of the MILP solver.
– A framework verifier for generating adversarial examples has demonstrated superior performance and has been validated on the MNIST dataset using three distinct neural network architectures and three different verification methods.

2 Related Work

Our work is connected to prior studies on attacking and defending deep neural networks, which encompass topics such as verification, testing, and creating adversarial examples. The existing research on neural verification can be categorized into two types based on constraint solving: methods based on Satisfiability Modulo Theories (SMT) problems [8,13] and methods based on Linear Programming (LP) [2,4]. These techniques are generally sound and complete i.e., no false negatives and no false positive respectively. But owing to the computational complexity, they have little capacity to scale. There are two methods, namely *approximation* and *abstraction*, that can be employed to achieve better scalability when verifying robustness. These techniques are known to be effective in achieving this objective [12,25]. Furthermore, there are numerous efforts aimed at either attacking deep neural networks (DNNs) or enhancing their resilience through the creation of adversarial examples. L-BFGS [19] was the earliest method developed for producing adversarial examples, while FGSM [9] utilizes gradient updates to create such examples. FGSM is capable of generating an adversarial example from an input with just one update, making it a relatively efficient technique. In our work, we employ FGSM attack to produce the rough set of bounds for the neurons before formulating into a LP problem.

3 Background

3.1 Robustness Against Adversarial Perturbations

The characteristics of a neural network can be inferred from the meaning and context of its specification. Typically, these characteristics are input-output (IO) properties that specify a particular relationship between the input and output of the network. One of the earliest IO properties that has been investigated is robustness, which requires the model's output to remain consistent even when minor modifications are made to the input value [7,16].

3.2 Gradient-Based Adversarial Attack

A small change made to the input to deceive the classifier's prediction is called an adversarial attack. If a neural network can withstand such attacks, it is probable that it can also handle other types of changes. However, this is not guaranteed, and therefore it is necessary to formally test the network's robustness against

all potential alterations. In our work, the purpose of the finding adversarial examples is to reduce the search space by obtaining the perturbation bounds of the adversarial example before encoding it to a MILP solver. Numerous methods exist for creating adversarial attacks, which can be classified into two categories based on the attacker's objective: targeted attacks and untargeted attacks.

- *Targeted attack*: A targeted attack aims to cause the input sample to be misclassified to a specific target class, rather than just away from its original class.
- *Untargeted attack*: An untargeted attack does not have a specific desired output class, but rather aims to cause the input sample to be misclassified from its original class, regardless of what new output class it ends up being classified as.

Fast Gradient Sign Method. To create boundaries for the perturbations, we use the Fast Gradient Sign Method for the adversarial attack [9].
To produce a modified version of an original sample represented by x, we introduce a slight perturbation ϵ to each of its components through either addition or subtraction.

The technique involves analyzing the sign of the gradient of the loss function, which is denoted as $\nabla_x \mathcal{L}(x, y)$:

- If the gradient $\nabla_x \mathcal{L}(x, y)$ is positive, it indicates that an increase in x results in an increase in the loss function $\mathcal{L}$.
- Conversely, if the gradient $\nabla_x \mathcal{L}(x, y)$ is negative, it implies that an increase in x leads to a decrease in the loss function $\mathcal{L}$.

3.3 Symbolic Interval Analysis for Bound Tightening

Interval analysis is a method utilized in the verification of neural networks to study their behavior and ensure their safety and robustness. The process involves an estimation of the output range of a neural network for a given input through computing the upper and lower bounds of the output using interval arithmetic. In interval analysis, each neuron in the network is treated as a function that takes an input and gives an output. The input and output are both represented as intervals that express a range of possible values. Interval arithmetic operations are then used to propagate the input intervals through the network, resulting in the output intervals. The calculated output intervals are compared to the desired output range to determine whether the network is safe and robust. If the output intervals include the desired output range, then the network is considered safe; otherwise, it may be potentially unsafe, requiring further analysis. Interval analysis is a robust technique for verifying neural networks as it can handle non-linear activation functions and multiple layers. However, it may be computationally demanding and not scalable to larger networks.

The given Fig. 1(a), depicts a Naive Interval analysis of a three-layer Deep Neural Network (DNN) with weights assigned to edges, and bias vectors containing all elements as 0. Assuming the input intervals to the first layer to be $[2, 4]$

and $[3, 6]$, the output interval obtained after performing scalar operations over intervals layer-wise, is $[-5, 7]$. However, here the output bound includes certain specific values that are infeasible in practical scenarios due to overestimation. For example, the value of -5 can only be achieved when neuron n_3 outputs 13 and neuron n_4 outputs 8. But to output 10 for n_3, the neurons n_1 and n_2 must output 4 and 5 simultaneously, and to output 8 for n_4, the neurons n_1 and n_2 should output 1 and 2 at the same time which also referred to as the *dependency problem.*

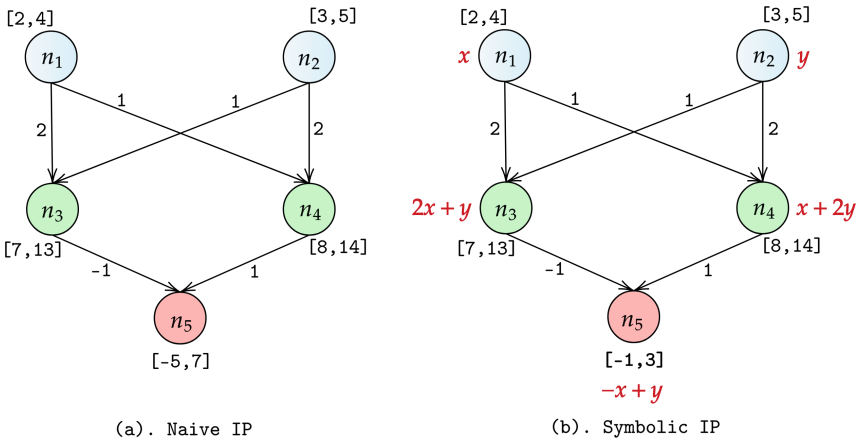

Fig. 1. Naive Interval Propagation vs. Symbolic Interval Propagation.

Symbolic interval analysis [22] or Symbolic Interval Propagation (SIP) is an approach utilized in the verification of neural networks to ensure their safety and robustness. This method involves approximating the output range of a neural network by calculating the upper and lower bounds of the output through the use of symbolic mathematical expressions. Symbolic interval analysis employs interval arithmetic to generate a group of mathematical expressions that represent the output range of the neural network. These expressions can be utilized to calculate the output range of the network for a given input and compare it to the desired output range to determine the safety and robustness of the network. Symbolic interval analysis is a powerful technique for verifying neural networks as it enables the analysis of intricate networks with non-linear activation functions and multiple layers. It is particularly effective for analyzing networks with piece-wise linear activation functions such as ReLU, as these networks can be challenging to evaluate using other verification methods. Figure 1(b), represents a symbolic approach to address the dependency problem. For neurons n_1 and n_2, let x and y represent the input variables. For neurons n_3 and n_4 can be symbolically represented as $2x + y$ and $x + 2y$ correspondingly and greater than zero since $x \in [1, 3]$ and $y \in [2, 4]$. Therefore, the symbolic interval for n_3 and n_4 is $[2x + y, 2x + y]$ and $[x + 2y, x + 2y]$ correspondingly and, similarly, the

symbolic interval for n_5 is $[-x + y]$. Hence, for $x \in [1, 3]$ and $y \in [2, 4]$, the output interval is $[-1, 3]$ which is computed as a tighter bound as compared to the naive approach of $[-5, 7]$.

3.4 Mixed-Integer Linear Programming

Unstable Neurons. The non-linearity of activation functions $\mathcal{A}$ is a significant obstacle in the process of verification. Specifically, the ReLU activation function $\mathcal{A}\left(z_j^{(i)}\right) = \text{ReLU}\left(z_j^{(i)}\right) = \max\left(0, z_j^{(i)}\right)$ introduces complexities that must be addressed during verification. To tackle this issue, we define intermediate layer bounds $\mathbf{l}_j^{(i)} \leq z_j^{(i)} \leq \mathbf{u}_j^{(i)}$ that constrain the input of each ReLU neuron for a given input $x \in \mathcal{C}$. With these bounds, we can categorize the activation space of each ReLU neuron.

- Active and Inactive: When the bounds of an intermediate layer for a ReLU neuron satisfy the condition $\mathbf{l}_j^{(i)} \geq 0$ or $\mathbf{u}_j^{(i)} \leq 0$, it indicates that the ReLU neuron lies in either the linear active region where its output is equal to its input ($\hat{z}_j^{(i)} = z_j^{(i)}$) or the inactive region where its output is zero ($\hat{z}_j^{(i)} = 0$).
- Unstable: If $\mathbf{l}_j^{(i)} \leq 0 \leq \mathbf{u}_j^{(i)}$, we call this ReLU neuron as an unstable neuron, this circumstance frequently presents challenges to the process of certification.

We follow the MILP-based reformulation of ReLU networks [20], to encode the unstable neuron. We formulate a ReLU activation function as:

$$\begin{aligned} \boldsymbol{z}_0 &= \boldsymbol{x} \\ \hat{\boldsymbol{z}}_{k+1} &= \boldsymbol{W}_{k+1}\boldsymbol{z}_k + \boldsymbol{b}_{k+1}, \forall k = 0, 1, \ldots, K-1 \\ \boldsymbol{z}_k &= \max\left(\hat{\boldsymbol{z}}_k, 0\right), \quad \forall k = 1, \ldots, K \\ \hat{\boldsymbol{y}}_{\boldsymbol{x}} &= \boldsymbol{W}_K \boldsymbol{z}_K + \boldsymbol{b}_K, \end{aligned} \tag{1}$$

where the variable K denotes the number of layers. Each layer is determined by a weight matrix $\boldsymbol{W}_k$ and a bias vector $\boldsymbol{b}_k$. The size of the weight matrix is $[N_{k+1} \times N_k]$, while the size of the bias vector is $[N_{k+1} \times 1]$. Here, N_k refers to the number of neurons in the k^{th} layer. The specifications that define the encoding of the neuron are as follows:

$$\boldsymbol{z}_k = \max\left(\hat{\boldsymbol{z}}_k, 0\right) \Rightarrow \begin{cases} \boldsymbol{z}_k \leq \hat{\boldsymbol{z}}_k - \hat{\boldsymbol{z}}_k^{\min}\left(1 - \mathbf{b}_k\right) \\ \boldsymbol{z}_k \geq \hat{\boldsymbol{z}}_k \\ \boldsymbol{z}_k \leq \hat{\boldsymbol{z}}_k^{\max}\mathbf{b}_k \\ \boldsymbol{z}_k \geq \mathbf{0} \\ \mathbf{b}_k \in \{0, 1\}^{N_k}. \end{cases} \tag{2}$$

where $\boldsymbol{b}_k$ is a binary variable.

4 The Perturbation Refinement Verification Framework

The methodology we have adopted involves the combination of gradient attack and symbolic interval analysis with the MILP-based method. The use of adversarial example from an attack aids in the provision of rough perturbation values,

which we then utilize to establish the primary bounds. Following this, we leverage SIP to obtain more precise bounds for the hidden neurons. The application of tighter bounds results in decreased activation search space in the verification problem, thereby enhancing verification efficiency.

Algorithm 1: Perturbation Refinement

Input: DNN $\mathcal{N}$, input x, perturbation threshold ϵ
Output: Robust

1 $\widehat{\epsilon} :=$ FGSM_attack(x); // Initial adversarial perturbation
2 $SymbolicBounds := SymbolicBoundpPropogation(x, \widehat{\epsilon})$
3 $MIPFormulation := MIPModel(\mathcal{N}, x, \widehat{\epsilon}, symbolic\_bounds)$
4 $\widehat{solver} := Constraint\&Objective(MIP\_formulation)$
5 $output := optimise(\widehat{solver})$
6 **if** *UNSAT* **then**
7 | **return** Robust $\mathcal{N}$; // returns a robust network
8 **end**
9 **else**
10 | x' :=get_adversarial($\widehat{solver}$); // returns an adversarial example
11 | **return** x';
12 **end**

Algorithm 1 displays an outline of our approach. Given an neural network $\mathcal{N}$, an input $x \in \mathbb{R}^n$ and a perturbation threshold ϵ. The FGSM attack is responsible for creating the initial boundaries for the adversarial example in line 1. In line 2, the SIP is utilized to establish tighter boundaries for the neurons than those from the original adversarial example. Between lines 3-6, an MILP problem is formulated that takes into account the input, $\mathcal{N}$, $\widehat{\epsilon}$, and the symbolic boundaries from line 2. The problem returns UNSAT if no adversarial example can be found, but it returns an adversarial example between lines 7-12.

FGSM Attack. In our method, we first compute a rough estimate ϵ using adversarial examples generated by the FGSM attack. This value is typically very close to the optimal robust radius, which helps to establish more precise input bounds. By having tighter input bounds, the number of binary variables is reduced, which in turn reduces the activation search space. Further, we constraint the adversarial attack that limits the magnitude of the perturbation that can be added to the input features of the neuron using the L-∞ norm. Mathematically, we can represent it as:

$$\|\delta\|_\infty = \max\left(|\delta_1|, |\delta_2|, \ldots, |\delta_n|\right) \tag{3}$$

where δ_i is the perturbation added to the $i-th$ neuron, and n is the total number of neurons. First, we define the L-∞ norm as the maximum absolute deviation

between the original input and the perturbed input as shown in 3. Next we define the ϵ from the attack as the maximum allowable deviation between the original input and the perturbed input. Next, we set the bounds for each neuron in the network by taking into account both the L-∞ norm and ϵ. For example, if the L-∞ norm is 0.1 and ϵ is 0.05, the bounds for a neuron would be $[-0.05, 0.05]$, since any perturbation greater than 0.05 in either direction would violate both the L-∞ norm and ϵ. This ensures that the perturbed input stays within the allowable range.

By constraining the L-∞ norm of the perturbation added to the input of the neuron, the FGSM attack ensures that the resulting adversarial example remains within a certain "perceptual distance" of the original input, or that the output of the neuron remains close to its original output for small perturbations.

Epsilon-Robustness ϵ. Epsilon perturbation or epsilon-robustness is employed to represent the perturbation limits, ϵ, for a neuron x. Epsilon perturbation involves adding a small perturbation to the input of the neuron such that the output remains approximately the same. The procedure to encode perturbation bounds ϵ for a neuron x using ϵ perturbation:

- Determine the range of values that x can take. For example, if x is a pixel in an image, it might take values between 0 and 255.
- Choose a value for ϵ (this is obtained from the FGSM attack). This is the maximum amount of perturbation that is allowed for x.
- Scale epsilon to the same range as x. For example, if x takes values between 0 and 255 you can allow a maximum perturbation of 10%, you would scale epsilon to 25.5.
- Add or subtract the scaled epsilon value to x to create two new values: $x\_min = x - \epsilon$ and $x\_max = x + \epsilon$
- Use x_min and x_max as the new input values for the neuron x. This ensures that the output of the neuron will remain within a certain range, even if the input is perturbed.

By using ϵ perturbation to encode perturbation bounds for a neuron x, we can ensure that the neuron is robust to small perturbations in its input. This is useful in speeding up the verification process because since it provides low-cost solutions or information gathered via a gradient-based adversarial attack.

MILP Formulation. To formulate the bounded neurons bounded by ϵ into MILP solver[1][2], we follow the following steps:

1. The binary decision variables: We define binary decision variables for each neuron in the network, where the variable takes a value of 1 if the neuron is active and 0 otherwise.

[1] We use the Gurobi solver to tackle the MILP problem.
[2] https://www.gurobi.com/resources/chapter-1-why-mixed-integer-programming-mip/.

2. The objective function: The objective function can be defined to minimize the distance between the original input and the perturbed input subject to the constraints that we define in the following steps.
3. Define the constraints for the epsilon value: We can formulate the epsilon value constraint as a set of linear constraints that ensure that the perturbation of each neuron is within a specified bound. For each neuron, we can define two linear constraints to enforce the upper and lower bounds on the perturbation of the neuron obtained from the FGSM attack.
4. Solve the MILP: Once the MILP is formulated, we can solve it using an optimization solver to find the input that minimizes the distance between the original input and the perturbed input subject to the constraints that we have defined.

By formulating bounded neurons bounded by epsilon into MILP, we can find adversarial examples that are constrained by the epsilon value while preserving the behavior of the neural network.

5 Experimentation, Dataset and Evaluation

We compare our verification procedure's implementation with three existing verifiers, namely Venus [4], Neurify [21], and MIPVerify [20]. Venus Verifier is a software tool for verifying neural network models using a combination of abstract interpretation and SMT-based techniques. It is based on a novel approach that combines interval arithmetic and constraint propagation with SMT-based techniques such as CEGAR and IC3. Neurify is a software tool for verifying neural network models using abstract interpretation. It is based on the ReluVal algorithm, which is an abstract interpretation-based approach for analyzing ReLU neural networks. MIPVerify is a software tool for verifying neural network models using mixed-integer programming (MIP). It is a verification framework that is based on solving a sequence of MIP problems, where each problem checks if the output of the neural network model is within a certain range. In this work, we evaluate the effectiveness of our verification algorithm on the MNIST dataset, which contains handwritten digits ranging from 0 to 9. To ensure consistency, the images are preprocessed to have a size of 28×28 pixels and are normalized and centered. Each pixel of the image has a value between 0 and 255, with 0 representing black, 255 representing white, and intermediate values representing different shades of gray.

Table 1 displays the verification results on the first 100 instances using four different neural architectures and three different verifiers, with an epsilon value of 0.05. The metrics used to evaluate the verifiers are $V_{t(sec)}$ (total verification time), $\#Adv$ (number of adversarial examples computed), $\#SAT$ (number of instances verified as satisfied), and $\#UNK$ (number of instances for which verification was inconclusive). The experiments were conducted on a Linux workstation equipped with a Dual Xeon E5-2673 v3 (24 cores) and 64GB of memory. A time-limit was set to 120 min for each instance and an overall limit of 720 min. To minimize experimental errors resulting from parallel tasks, each verification task

Table 1. Verification on the MNIST dataset.

Method($\mathcal{N}_1$)$\langle 784, 24, 24, 10\rangle \epsilon = 0.05$	$V_{t(sec)}$	$\#Adv$	$\#SAT$	$\#UNK$
Ours	101.43	49	2	0
Venus	24.32	47	2	0
Neurify	398.42	47	2	0
MIPVerify	723.48	49	2	0
Method($\mathcal{N}_2$) $\langle 784, 40, 20, 10\rangle$ $\epsilon = 0.05$	$V_{t(sec)}$	$\#Adv$	$\#SAT$	$\#UNK$
Ours	282.36	43	7	0
Venus	34.55	44	7	0
Neurify	timelimit	-	-	-
MIPVerify	1130.45	43	7	0
Method($\mathcal{N}_3$) $\langle 784, 512, 512, 10\rangle$ $\epsilon = 0.05$	$V_{t(sec)}$	$\#Adv$	$\#SAT$	$\#UNK$
Ours	12335.32	41	7	0
Venus	2515.86	44	7	1
Neurify	memlimit	-	-	-
MIPVerify	34525.35	44	7	1
Method($\mathcal{N}_4$) $\langle 784, 500, 10\rangle$ $\epsilon = 0.05$	$V_{t(sec)}$	$\#Adv$	$\#SAT$	$\#UNK$
Ours	513.43	2	46	0
Venus	18188.76	6	45	5
Neurify	timelimit	-	-	-
MIPVerify	18187.76	4	46	0

was run five times. The average of these results was then used as the experimental outcome. Our verifier performs better than the other verification methods on all four neural architectures. Additionally, we notice that Neurify reaches a time limit for the second and third architectures, denoted as $\mathcal{N}_2$ and $\mathcal{N}_3$, respectively, and a memory limit for the fourth architecture, denoted as $\mathcal{N}_4$. Venus, however, failed on $\mathcal{N}_4$ but outperformed on $\mathcal{N}_1$, $\mathcal{N}_2$, $\mathcal{N}_3$. On $\mathcal{N}_4$, Venus returns five $\#UNK$ case whereas our verifier returns zero $\#UNK$ thus ensuring completeness is achieved by always providing a solution to the MILP problem.

6 Conclusion

In conclusion, our paper has successfully demonstrated the potential of low-cost solutions derived from adversarial attacks to reduce the search space and streamline the verification process, while still maintaining high levels of accuracy. Future work in this area will involve comparing our approach with different adversarial attacks to further optimize the effectiveness of our method. By leveraging the insights gained from this study, we hope to contribute to the ongoing efforts to enhance the security and robustness of machine learning systems. In future

work, we focus on robust optimization i.e., variability in the data or parameters of the problem, can lead to sub-optimal or even infeasible solutions. One way to address this is the objective function and constraints are reformulated to explicitly account for the worst-case scenarios of the uncertain data. Adversarial attacks can be seen as a way of generating such worst-case scenarios, and hence the adversarial solutions can be used as inputs to the robust optimization formulation. This can lead to more robust and reliable MILP solutions that perform well under various scenarios.

References

1. Alparslan, Y., Alparslan, K., Keim-Shenk, J., Khade, S., Greenstadt, R.: Adversarial attacks on convolutional neural networks in facial recognition domain. arXiv preprint arXiv:2001.11137 (2020)
2. Anderson, R., Huchette, J., Ma, W., Tjandraatmadja, C., Vielma, J.P.: Strong mixed-integer programming formulations for trained neural networks. Math. Program. **183**(1–2), 3–39 (2020)
3. Biggio, B., et al.: Evasion attacks against machine learning at test time. In: Blockeel, H., Kersting, K., Nijssen, S., Železný, F. (eds.) ECML PKDD 2013. LNCS (LNAI), vol. 8190, pp. 387–402. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40994-3_25
4. Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., Misener, R.: Efficient verification of ReLU-based neural networks via dependency analysis. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, pp. 3291–3299 (2020)
5. Bunel, R., Mudigonda, P., Turkaslan, I., Torr, P., Lu, J., Kohli, P.: Branch and bound for piecewise linear neural network verification. J. Mach. Learn. Res. **21**(2020) (2020)
6. Bunel, R.R., Turkaslan, I., Torr, P., Kohli, P., Mudigonda, P.K.: A unified view of piecewise linear neural network verification. Adv. Neural Inf. Process. Syst. **31** (2018)
7. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 39–57. IEEE (2017)
8. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: D'Souza, D., Narayan Kumar, K. (eds.) ATVA 2017. LNCS, vol. 10482, pp. 269–286. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68167-2_19
9. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
10. Gopinath, D., Pasareanu, C.S., Wang, K., Zhang, M., Khurshid, S.: Symbolic execution for attribution and attack synthesis in neural networks. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), pp. 282–283. IEEE (2019)
11. Hernandez, C., Espf, J., Nakayama, K., Fernandez, M.: Interval arithmetic backpropagation. In: Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan), vol. 1, pp. 375–378. IEEE (1993)
12. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 3–29. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_1

13. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 97–117. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_5
14. Kurakin, A., Goodfellow, I.J., Bengio, S.: Adversarial examples in the physical world. In: Artificial Intelligence Safety and Security, pp. 99–112. Chapman and Hall/CRC (2018)
15. Lin, W., et al.: Robustness verification of classification deep neural networks via linear programming. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 11418–11427 (2019)
16. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083 (2017)
17. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 372–387. IEEE (2016)
18. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. Proc. ACM Program. Lang. **3**(POPL), 1–30 (2019)
19. Szegedy, C., et al.: Intriguing properties of neural networks. corr abs/1312.6199, arXiv preprint arXiv:1312.6199 (2013)
20. Tjeng, V., Xiao, K., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. arXiv preprint arXiv:1711.07356 (2017)
21. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. In: Advances in neural information processing systems, vol. 31 (2018)
22. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. In: 27th {USENIX} Security Symposium ({USENIX} Security 18), pp. 1599–1614 (2018)
23. Wang, S., Su, Z.: Metamorphic testing for object detection systems. arXiv preprint arXiv:1912.12162 (2019)
24. Wardat, M., Le, W., Rajan, H.: Deeplocalize: fault localization for deep neural networks. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp. 251–262. IEEE (2021)
25. Weng, L., et al.: Towards fast computation of certified robustness for ReLU networks. In: International Conference on Machine Learning, pp. 5276–5285. PMLR (2018)
26. Yang, P., et al.: Enhancing robustness verification for deep neural networks via symbolic propagation. Formal Aspects Comput. **33**(3), 407–435 (2021)