Reluplex made more practical: Leaky ReLU

Jin Xu*, Zishan Li*, Bowen Du†, Miaomiao Zhang* §, Jing Liu‡ §
*School of Software Engineering, Tongji University, Shanghai, China
†Department of Computer Science, University of Warwick, Coventry, United Kingdom
‡School of Software Engineering, East China Normal University, Shanghai, China

***Abstract*—In recent years, Deep Neural Networks (DNNs) have been experiencing rapid development and have been widely used in various fields. However, while DNNs have shown strong capabilities, their security problems have gradually been exposed. Therefore, the formal guarantee of neural network output is needed. Prior to the appearance of the Reluplex algorithm, the verification of DNNs was always a difficult problem. Reluplex algorithm is specially used to verify DNNs with ReLU activation function. This is an excellent and effective algorithm, but it cannot verify more activation functions. ReLU activation function will bring about "Dead Neuron" problem, and Leaky ReLU activation function can solve this problem, so it is necessary to verify DNNs based on Leaky ReLU activation function. Therefore, we propose the Leaky-Reluplex algorithm, which is based on the Reluplex algorithm. Leaky-Reluplex algorithm can verify DNNs based on Leaky ReLU activation function.**

***Index Terms*—Reluplex Algorithm, Leaky ReLU, Deep Neural Networks**

I. Introduction

In recent years, Deep Neural Networks (DNNs) [1] [2] have been experiencing rapid development and have been widely used in various fields. However, while DNNs has shown strong capabilities, their security problems have gradually been exposed. Christian Szegedy put forward the concept of adversarial example [3] in 2014. If a slight disturbance is added to a sample, human beings can correctly distinguish the sample, but the neural network will give a wrong classification with high confidence, we will call the sample causing the wrong classification as the adversarial example [3]. The emergence of adversarial example means that the neural network is not as safe and reliable as we imagined. Using adversarial examples can easily attack an artificial intelligence system, thus it can be seen that the security problem has become an important problem to be solved urgently in the field of artificial intelligence. Therefore, the formal guarantee of neural network output is needed.

Verification of neural networks is difficult as it is experimentally beyond the reach of general-purpose tools such as linear program (LP) solvers or existing satisfiability modulo theories (SMT) solvers [4]–[6]. In 2010, Luca Pulina et al. [7] proposed a verification method for the Multi-layer Perceptron (MLP) [8], and proved that this method can verify the security of multi-layer perceptron in a certain input interval, but it can only process up to 20 hidden nodes [4], so it can only be applied to small neural networks and cannot be applied to large deep neural networks. In 2017, Xiaowei Huang et al. proposed an automated verification framework for detecting the security of deep neural networks, called DLV(Deep Learning Verification) [6]. They propose this approach for verifying the local adversarial robustness of DNNs based on a systematic exploration of a region. The verification process is still exponential in the number of features . In the same year, Katz et al propose an algorithm called Reluplex, which is efficient to verify DNNs with ReLU activation functions [9]. This is achieved by leveraging the piecewise linear nature of ReLU and attempting to gradually satisfy the constraints that they impose as the algorithm searches for a feasible solution. We call the algorithm Reluplex, for "ReLU with Simplex". Compared with [6] and [7] , Reluplex can handle larger deep neural networks and guarantee that there are no irregularities hiding between the discrete points.

Reluplex is considered to be an efficient technique for verifying properties of DNNs with ReLU activation functions which are one of the most commonly used activation functions. However, ReLU units can be fragile during training and can "die" [10]. Leaky ReLU functions are proposed as a remedy to the "Dead Neuron" problem [11]. Since the current Reluplex can only handle verification of DNNs with ReLU functions, we therefore extend the verification process in support of Leaky ReLU functions, which is achieved by designing new derivation rules based on the tool.

So based on the current tool—Reluplex, our contribution can be summarized as follows. We propose Leaky-Reluplex mechanism to verify DNNs with Leaky-ReLU activation function.

The rest of the paper is organized as follows. We begin with some background on ReLU, Leaky-ReLU activation function and Reluplex in Section 2. The extension of verification of DNNs with Leaky ReLU function is described in Section 3. In Section 4, we give the experimental results and analysis. We conclude the paper in the last section.

II. Preliminaries

We first recall some definitions of ReLU, Leaky-ReLU activation function and Reluplex algorithm.

§ Corresponding authors: miaomiao@tongji.edu.cn; jliu@sei.ecnu.edu.cn

978-1-7281-8086-1/20/$31.00

A. ReLU and Leaky-ReLU activation function

ReLU [12] is the most commonly used activation function in neural networks. Compared with other activation functions, such as sigmoid and tanh, RelU holds a faster convergence rate and calculation speed, because of its linear operation. However, Dead Neuron, caused by ReLU, hurt the performance of the neural network. When the input value is negative, the output of ReLU is always zero and its first derivative is thus zero, which will make neuron unable to update parameters.

To overcome this limitation of the ReLU function, Maas [13] introduces a Leaky value in the negative half interval of the ReLU function, called the Leaky ReLU function.

$$\text{Leaky-ReLU}(x) = \max(kx, x) = \begin{cases} x, & \text{if } x > 0 \\ kx, & \text{if } x \leq 0 \end{cases}$$

where $k \in [0, 1]$.

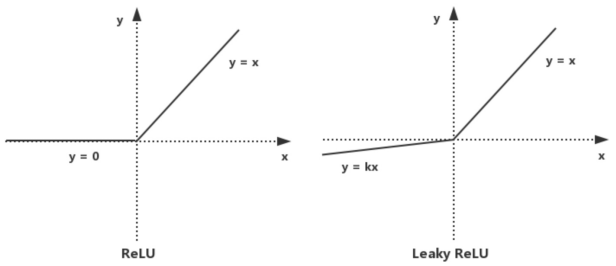

Fig. 1: ReLU and Leaky-ReLU, for Leaky-ReLU k is fixed.

The Leaky ReLU function with leak correction is a variant of the classical ReLU activation function. The output of the function has a small slope to the negative input. Since the derivative is always not zero, this can reduce the occurrence of silent neurons, allow gradient-based learning (although it will be slow), and solve the problem that neurons fail to learn after the ReLU function enters the negative interval.

Kaiming He [14] pointed out that experiments in [13] show that Leaky ReLU has negligible impact on accuracy compared with ReLU. But, Xu suggest that the most popular activation function ReLU is not the end of story: Leaky ReLU consistently outperform the original ReLU. Even though the reasons of their superior performances still lack rigorous justification from theoretic aspect [15], in some cases we do need to use Leaky ReLU.

In other words, the Leaky ReLU activation function performs well, but its effect is not very stable. Compared with ReLU, under what conditions, which one has better performance and effect remains to be discussed. However it is known that in some cases, especially when the learning rate is set so high that "Dead Neuron" problem is unavoidable , Leaky ReLU can be utilized to solve this problem [10].

B. Reluplex Algorithm

Reluplex is an efficient SMT solver for verifying DNNs. The name means "ReLU with Simplex", it is based on the Simplex algorithm [16] [17], and extended to handle the ReLU activation function. DNNs and their properties can be directly encoded as conjunctions of linear formulas and ReLU functions. So when we use Reluplex to verify a property of DNNs, in fact, it is to determine whether there is a set of assignments of variables that satisfies specific linear constraints, ReLU constraints, upper bound and lower bound constraints of variables.

Before Reluplex appeared, there were also some generic SMT solvers [18]–[20] trying to solve the verification problem of DNNs with ReLU activation function. The basic approach is to split a ReLU constraint into two sub-problems according to the piecewise function, and use two different states $Active$ and $Inactive$ to represent the two sub-problems. So a ReLU constraint can be defined as follows:

$$\text{ReLU}(x) = \max(0, x) = \begin{cases} x, & \text{if } x > 0 \quad \textit{(Active state)} \\ 0, & \text{if } x \leq 0 \quad \textit{(Inactive state)} \end{cases}$$

When $x > 0$ and $f(x) = x$, the ReLU constraint is in $Active$ state. When $x < 0$ and $f(x) = 0$, the ReLU constraint is in $Inactive$ state. So a ReLU constraint can be expressed as a disjunction: $Active \vee Inactive$. If one of the states is satisfied, we can say that the ReLU constraint is satisfied.

For a DNN, there may be hundreds of ReLU constraints. These ReLU constraints need to be satisfied at the same time. We can express this condition as $ReLU_1 \wedge ReLU_2 \wedge \cdots \wedge ReLU_n$ in a conjunction. In this way, a DNN can be encoded as a CNF formula [21] [22], and then solved by generic SMT solvers [18]–[20].

This method is feasible in theory, but there are serious performance problems in practice. In the worst case, for every n nodes added, SMT solver needs to traverse 2^n states more, and the complexity increases exponentially, which is easy to reach the performance bottleneck.

The overall solution idea of Reluplex algorithm is consistent with the above method. A DNN will also be encoded into a CNF formula and solved by SMT solver. However, in details, Reluplex proposes its own optimization algorithm to solve the performance bottleneck problem.

The major optimization idea of Reluplex lies in "lazy handling" of ReLU constraints. Reluplex will not actively traverse and split the state of ReLU constraints. Instead, it will first use Simplex algorithm [16] to deal with linear problems. In this process, according to the bound derivation rules in Reluplex, it will tighten the upper bounds and lower bounds of variables in the networks. ReLU variables are the two variables in a ReLU constraint $x^f = max(0, x^b)$, such as x^f and x^b. If the lower bound of a ReLU variable x^f or x^b is found to be greater than 0, the corresponding ReLU constraint is fixed to the $Active$ state. If the upper bound of the ReLU variable x^b is found to be less than 0, the corresponding ReLU constraint is fixed to the $Inactive$ state. In this way, a ReLU constraint can be eliminated without splitting, which greatly reduce the number of ReLU constraints that need to be traversed and splitted, thus improve the efficiency of the algorithm significantly. The details that are crucial to performance and scalability, such

as the use of floating-point arithmetic, bound derivation for ReLU variables, and conflict analysis, are discussed in [9]. The success in verifying properties of the ACAS Xu networks [23] indicates that the technique holds potential for verifying real-world DNNs.

III. From ReLU to Leaky ReLU

Our attempt is to extend Reluplex with Leaky ReLU to tackle rich DNNs. As a consequence, DNNs are verified extended with Leaky-ReLU activation functions. Using two main changes. First, when the Leaky ReLU pair is broken, the fixing derivation rules need to be modified and expanded accordingly. Second, the operations should be changed when the state of Leaky ReLU constraint can be fixed to $Active$ or $Inactive$ according to the upper and lower bounds. Before introducing the concrete modification, we first recall some derivation rules in Reluplex [9].

A. The derivation rules to fix broken ReLU pairs

In Reluplex, variable x_b is used to denote the connection information of a node from the preceding layer, x^f is the connection information of the following layers and $< x_b, x_f >$ is called a ReLU pair. A ReLU pair can unavoidably be broken, which means that their assignments do not satisfy the ReLU constraint $x^f = \max(0, x^b)$. The derivation rules to fix the broken ReLU pairs in Reluplex are given as follows:

$$Update_b : \frac{x_i \notin B, < x_i, x_j > \in R, \alpha(x_j) \neq \max(0, \alpha(x_i)), \alpha(x_j) \geq 0}{\alpha := update(\alpha, x_i, \alpha(x_j) - \alpha(x_i))}$$

$$Update_f : \frac{x_j \notin B, < x_i, x_j > \in R, \alpha(x_j) \neq \max(0, \alpha(x_i))}{\alpha := update(\alpha, x_j, \max(0, \alpha(x_i)) - \alpha(x_j))}$$

where x_i and x_j are non-basic variables [16], $\alpha(x_i)$ is the assignment of variable x_i, $< x_i, x_j >$ is a ReLU pair, R is a set of ReLU pairs, B is a set of basic variables [16].

In general, $Update_b$ is used to update the assignment of x^b (in this rule, x^b corresponds to x_i). When a ReLU pair is broken, calculate the difference value between x_i and x_j, and then add the value to x_i, to make $\alpha(x_i) = \alpha(x_j)$. $Update_f$ is used to update the assignment of x^f (in this rule, x^f corresponds to x_j). Still, in order to fix the broken ReLU pair$< x_i, x_j >$, calculate the difference value between $max(0, \alpha(x_i))$ and $\alpha(x_j)$, and then add the value to x_j, to make x_i and x_j satisfy ReLU constraint.

If all ReLU pairs satisfy the ReLU constraints and all variables satisfy bound constraints, the derivation is said to be successful.

B. New derivation rules to fix broken Leaky ReLU pairs

Since we replace ReLU with Leaky ReLU, obviously Leaky ReLU pairs can be broken. It is needed to redesign the corresponding derivation rules. We extend four rules as follows to handle broken Leaky ReLU pairs, that is, $L - Update_{b1}$, $L - Update_{b2}$, $L - Update_f$ and $L - ReluSuccess$.

L-Update$_{b1}$:

$$\frac{x_i \notin B, < x_i, x_j > \in R, \alpha(x_j) \neq \max(k \cdot \alpha(x_i), \alpha(x_i)), \alpha(x_j) \geq 0}{\alpha := update(\alpha, x_i, \alpha(x_j) - \alpha(x_i))}$$

L-Update$_{b2}$:

$$\frac{x_i \notin B, < x_i, x_j > \in R, \alpha(x_j) \neq \max(k \cdot \alpha(x_i), \alpha(x_i)), \alpha(x_j) < 0}{\alpha := update(\alpha, x_i, (1/k) \cdot \alpha(x_j) - \alpha(x_i))}$$

L-Update$_f$:

$$\frac{x_j \notin B, < x_i, x_j > \in R, \alpha(x_j) \neq \max(k \cdot \alpha(x_i), \alpha(x_i))}{\alpha := update(\alpha, x_j, \max(k \cdot \alpha(x_i), \alpha(x_i)) - \alpha(x_j))}$$

L-ReluSuccess:

$$\frac{\forall x \in X, l(x) \leq \alpha(x) \leq u(x), \forall < x^b, x^f > \in R, \alpha(x^f) = \max(k \cdot \alpha(x^b), \alpha(x^b))}{SAT}$$

$L - Update_{b1}$ is modified based on $Update_b$. Since nonlinear constraint has been changed, when a pair $< x_i, x_j >$ is broken, it is necessary to replace the ReLU function $max(0, \alpha(x_i))$ with the Leaky ReLU function $max(k \cdot \alpha(x_i), \alpha(x_i))$.

$L - Update_{b2}$ is a new design to deal with the case when $\alpha(x_j) \leq 0$. It can be regarded as a supplement rule to $L - Update_{b1}$. Since $L - Update_{b1}$ can only be called when x_j is greater than or equal to 0, it is possbile that x_j has a value less than 0. That is, if $\alpha(x_j) \neq \max(k \cdot \alpha(x_i), \alpha(x_i)) \wedge \alpha(x_j) < 0$, update the assignment of x_i with $\alpha(x_i) = \alpha(x_i) + [(1/k) \cdot \alpha(x_j) - \alpha(x_i)] = (1/k) \cdot \alpha(x_j)$.

$L - Update_f$ and $L - ReluSuccess$ are modified accordingly. The ReLU function $max(0, \alpha(x_i))$ in the original rules has been modified to Leaky ReLU function $max(k \cdot \alpha(x_i), \alpha(x_i))$.

Therefore, using the extended derivation rules specified above, the broken Leaky ReLU pairs can be fixed.

C. New method to eliminate Leaky ReLU pairs

The modification and extension of derivation rules are only part of the work in Leaky-Reluplex. In addition, an important detail is not reflected in the derivation rules. Reluplex will tighten the upper bounds and lower bounds of ReLU variables to directly eliminate the ReLU pairs, and let the ReLU constraints fix at $Active$ or $Inactive$ state. For a ReLU pair $< x^b, x^f >$, it satisfy $x^f = ReLU(x^b)$. If the lower bound of x^b or x^f is strictly positive, it means that in any feasible solution this ReLU constraint will be fixed at $Active$ state. Similarly, if the upper bound of x^b is negative, it implies $Inactive$ state. When the state of a ReLU constraint is fixed, it can be eliminated, and there is no need to traverse and split it again. This is an efficient way to significantly reduce the number of constraints that need to be splitted. In order to inherit this efficient algorithm idea, we also propose some derivation methods in Leaky-Reluplex to eliminate the Leaky ReLU pairs as far as possible.

1) Fixed at the Active state: From the definition of Leaky ReLU function, we can know that if the lower bound of x^b or x^f is positive, then the state of this Leaky ReLU pair is $Active$. In this state, the assignments of x^b and x^f are always equal and the lower bounds should always be equal, too. Therefore, what we need to do is to update the lower

bounds of x^b and x^f, and make sure they are equal. Then update the assignments of x^b and x^f to be identical. At this point, the Leaky ReLU pair can be eliminated. In fact, it is not difficult to find that this situation in Leaky ReLU is the same with ReLU.

2) Fixed at the $Inactive$ state: According to the characteristics of Leaky ReLU function, both x^b and x^f can be negative. When one of them is negative, it implies $Inactive$ state, and the Leaky ReLU constraints can be eliminated. This is different from the ReLU constraints, which only x^b can be negative. So in Leaky-Reluplex, when one of the Leaky ReLU variables is negative, it needs to be classified and discussed according to the type of variables.

- If the current variable is x^f, the new upper bound is a negative number β, we can deduce that this Leaky ReLU constraint is $Inactive$ state. Then the assignments and upper bounds of x^f and x^b should satisfy the linear constraint $x^f = k \cdot x^b$. We need three steps to eliminate this Leaky ReLU pair.

  First, assuming that $u(x)$ represents the upper bound of x, we respectively update the upper bounds of x^f and x^b to be: $u\left(x^f\right) = \beta$ and $u\left(x^b\right) = (1/k) \cdot \beta$.

  Second, in order to ensure that this linear constraint will not be changed in the subsequent derivation, we need to replace all the variables x^f appeared in the $tableau$ T with $k \cdot x^b$, and then add a new equation $x^f = k \cdot x^b$ into the $tableau$ T. ($tableau$ is a table which records all linear constraints in Leaky-Reluplex. For a detailed introduction to $tableau$, see [9] or [16]).

  Third, assuming that $\alpha(x)$ represents the assignment of x, we keep $\alpha(x^f)$ unchanged, update the assignment of x^b to be $\alpha(x^b) = (1/k) \cdot \alpha(x^f)$. So that x^f and x^b satisfy the required linear constraint. And then according to the new assignment of x^b, update the assignments of other variables which have equation relation with x^b in the $tableau$ T. Only in this way can we ensure the correctness of the assignments and corresponding constraints in the $tableau$ T.

  The above measures ensure that if a Leaky ReLU constraint is fixed at $Inactive$ state, then in the subsequent derivation, no matter how the assignments of x^f and x^b change, they will keep the linear constraint $x^f = k \cdot x^b$ all the time.
- If the current variable is x^b, based on the above analysis, we update the assignment of x^f and x^b to satisfy the equation $\alpha\left(x^f\right) = k \cdot \alpha\left(x^b\right)$. And then respectively update the upper bounds of x^f and x^b to be $u\left(x^f\right) = k \cdot \beta$ and $u\left(x^b\right) = \beta$.

Through the above theoretical analysis, the Leaky-Reluplex algorithm proposed by us can already be used to verify DNNs with Leaky ReLU activation function. The main algorithm framework of Leaky-Reluplex is based on Reluplex. So the main idea of the two algorithms is the same. Only when dealing with non-linear constraints, different derivation rules and operations are adopted for different activation functions. In addition, since the original algorithm Reluplex has been proved to be sound and complete, the modifications made by Leaky-Reluplex have not changed this feature. The proof of the soundness and completeness of Leaky-Reluplex is given in APPENDIX A, based on the proof process in [9]. The verification experiment of Leaky-Reluplex will be described in next chapter detailedly.

IV. Experimental Results

Our experiments are conducted on a computer with Inter(R) i5-8250U CPU@1.60GHz with 8GB RAM. The purpose of the experiment is to verify the correctness of the Leaky-Reluplex algorithm and to prove that the improvements and effectiveness of our extensions. There are two steps in this expriment. The first step is to train a DNN with Leaky ReLU activation function and the next one is to use the Leaky-Reluplex to verify the adversarial robustness of the DNN.

A. Training DNNs with Leaky ReLU activation function

The published Cardiotocography data set [24] is employed as our experimental data, which is a data set that records the health status of the fetus, with a total of 2126 samples, and each sample containing 23 attributes. We selected 18 of them as characteristic variables for training, and took the health status of the fetus as the classification target. According to different health status, the fetus can be divided into three categories: normal, suspect and pathologic.

Before training, we normalised the attribute data of different sizes into standard data within $[-1, 1]$. In the training stage, we set up a fully connected neural network with the structure of "18-36-36-3". Leaky ReLU is used as activation function, and the k value of Leaky ReLU is set to 0.1. 80% of the samples are randomly selected as training set, and the remaining 20% are used for testing. The batch size is 24 and the epoch is 50. After the training, the test loss is 0.1827 and the test accuracy is 92.02%.

B. Leaky-Reluplex verifies the adversarial robustness of DNNs

After the training stage, the Leaky-Reluplex is used to verify the adversarial robustness of DNNs. We respectively extract three correctly classified samples from normal, suspect and pathologic samples, and use these samples as test data to verify the robustness of the DNN model. The smaller perturbation neighborhoods are set up for each classification initially, and the robustness of the DNN model can be observed with the different settings of the neighborhoods. The detailed steps of verification are as follows:

- Step 1. The parameters of each layer are extracted from the trained DNNs, and then encoded into Leaky-Reluplex.
- Step 2. The selected samples are normalized, and then input into Leaky-Reluplex respectively.
- Step 3. The perturbation neighborhood is set as: $\delta = 0.01$ to indicate that the value of each variable can perturbed up and down by at most 1%.
- Step 4. The Leaky-Reluplex is applied to verify. Nine selected samples need to be verified nine times to finish

a round of verification. The value of δ is changed and the next round of verification will start. The value of δ is set to increase by 0.005 per round until $\delta = 0.035$, a total of six rounds of verification are carried out.

The verification results are shown in Table I – Table III. The symbol "$\surd$" indicates that there is no adversarial example in the corresponding perturbation neighborhood and the adversarial robustness is satisfied. "Fail" indicates that there is at least one adversarial example in the corresponding perturbation neighborhood, and the adversarial robustness is not satisfied.

According to Table I, when the perturbation neighborhood of normal sample A1 is less than 0.025, the classification of all data in this range are identical to the central point A1 can be ensured, but when the perturbation neighborhood increases to 0.03, there are adversarial examples which have different classification with A1. For the sample A1, $\delta = 0.025$ is a safe area in which the adversarial robustness is satisfied. Once exceeding this area, the adversarial robustness can not be guaranteed.

TABLE I: Verification results of three normal samples

perturbation neighborhood δ	0.01	0.015	0.02	0.025	0.03	0.035
normal sample A1	$\surd$	$\surd$	$\surd$	$\surd$	Fail	Fail
normal sample A2	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	Fail
normal sample A3	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	Fail

TABLE II: Verification results of three suspect samples

perturbation neighborhood δ	0.01	0.015	0.02	0.025	0.03	0.035
suspect sample B1	$\surd$	$\surd$	Fail	Fail	Fail	Fail
suspect sample B2	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	Fail
suspect sample B3	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	Fail

TABLE III: Verification results of three pathological samples

perturbation neighborhood δ	0.01	0.015	0.02	0.025	0.03	0.035
pathological sample C1	$\surd$	$\surd$	$\surd$	$\surd$	Fail	Fail
pathological sample C2	$\surd$	Fail	Fail	Fail	Fail	Fail
pathological sample C3	$\surd$	Fail	Fail	Fail	Fail	Fail

Table I – Table III illustrate that even for the samples with the same classification, different sample individuals have different safe areas. Such as the two samples with the normal class, the safe area of A1 is 0.025, but the safe area of A2 is 0.03. And for the samples with suspect class, the safe area of B1 is 0.015, the safe area of B2 is 0.03.

Based on the expriment, the Leaky-Reluplex algorithm is used to verify the adversarial robustness of DNN with Leaky ReLU activation function. It shows that our extension of Reluplex works well. When the adversarial robustness is not satisfied, Leaky-Reluplex will return an adversarial example.

V. Conclusion

In this paper, we extend the Reluplex algorithm. The derivation rules are modified to fit the operation of Leaky ReLU, so that the Leaky-Reluplex algorithm is able to verify the DNNs with both ReLU and Leaky ReLU activation function. A lot of experiments are conducted to evaluate the validity of this extension. The results of a serial of experiments show that the Leaky-Reluplex algorithm is able to find the counter-examples for the DNNs with the Leaky ReLU.

There are still some further work needed to be done. First of all, this paper proposes the Leaky-Reluplex algorithm, which can verify DNNs with Leaky ReLU as the activation function, but many other activation functions are not supported, which limits the application scope of the verification algorithm. Secondly, because there are some loops in the Recurrent Neural Networks (RNNs), neither the original Reluplex nor Leaky-Reluplex algorithm can verify RNNs, so the verification of RNNs can be completed in future work.

Acknowledgment

We acknowledge the support of NSFC 61972284 and NSFC 61972150.

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[2] M. Riesenhuber and T. Poggio, "Hierarchical models of object recognition in cortex," *Nature neuroscience*, vol. 2, no. 11, p. 1019, 1999.

[3] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[4] L. Pulina and A. Tacchella, "Challenging smt solvers to verify neural networks," *Ai Communications*, vol. 25, no. 2, pp. 117–135, 2012.

[5] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, and A. Criminisi, "Measuring neural net robustness with constraints," in *Advances in neural information processing systems*, 2016, pp. 2613–2621.

[6] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 3–29.

[7] L. Pulina and A. Tacchella, "An abstraction-refinement approach to verification of artificial neural networks," in *International Conference on Computer Aided Verification*. Springer, 2010, pp. 243–257.

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[9] 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*. Springer, 2017, pp. 97–117.

[10] C. Cui and T. Fearn, "Modern practical convolutional neural networks for multivariate regression: Applications to nir calibration," *Chemometrics and Intelligent Laboratory Systems*, vol. 182, pp. 9–20, 2018.

[11] http://cs231n.github.io/neural-networks-1/#actfun.

[12] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[13] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, no. 1, 2013, p. 3.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[15] X. Bing, N. Wang, T. Chen, and L. Mu, "Empirical evaluation of rectified activations in convolutional network," *Computer Science*, 2015.

[16] R. J. Vanderbei, "Linear programming: Foundations and extensions," *Journal of the Operational Research Society*, vol. 49, no. 1, pp. 94–94, 1998.

[17] G. Dantzig, *Linear programming and extensions*. Princeton university press, 2016.

[18] M. Deters, A. Reynolds, T. King, C. Barrett, and C. Tinelli, "A tour of cvc4: how it works, and how to use it," in *2014 Formal Methods in Computer-Aided Design (FMCAD)*. IEEE, 2014, pp. 7–7.

[19] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. Van Rossum, S. Schulz, and R. Sebastiani, "An incremental and layered procedure for the satisfiability of linear arithmetic logic," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2005, pp. 317–333.
[20] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.
[21] B. Pfahringer, "Conjunctive normal form," *Encyclopedia of Machine Learning and Data Mining*, pp. 260–261, 2017.
[22] C. Ansótegui and F. Manyà, "Mapping many-valued cnf formulas to boolean cnf formulas," in *35th International Symposium on Multiple-Valued Logic (ISMVL'05)*. IEEE, 2005, pp. 290–295.
[23] K. D. Julian, M. J. Kochenderfer, and M. P. Owen, "Deep neural network compression for aircraft collision avoidance systems," *Journal of Guidance, Control, and Dynamics*, vol. 42, no. 3, pp. 598–608, 2018.
[24] http://archive.ics.uci.edu/ml/datasets/Cardiotocography.

APPENDIX A
PROOF OF THE SOUNDNESS AND COMPLETENESS OF THE LEAKY-RELUPLEX ALGORITHM

The Leaky-Reluplex algorithm is based on the Reluplex algorithm. The original $Update_b$, $Update_f$, and $ReluSuccess$ rules are changed to $L-Update_{b1}$, $L-Update_f$, and $L-Relusuccess$ respectively. In addition, the $L-Update_{b2}$ rule is added. The remaining derivation rules remain unchanged. In paper [9], Guy Katz has proved the soundness and completeness of the Reluplex algorithm. This paper will analyze the similarities and differences between Leaky-Reluplex and Reluplex based on the existing proof process, and show soundness and completeness of Leaky- Reluplex.

A. Important concepts

(1) The data tuple $\langle \mathcal{B}, T, l, u, \alpha, R\rangle$: records the current value of each data in the Reluplex algorithm derivation process. Here, $\mathcal{B}$ represents the set of underlying variables, T represents a simplex table, and l represents the lower bound of the variable, u represents the upper bound of the variable, α represents the assignment of the variable, and R represents the set of ReLU constraints.

(2) Derivation tree D_i: There is only one initial node s_0 on the initial derivation tree D_0, and $s_0 = \langle \mathcal{B}_0, T_0, l_0, u_0, \alpha_0, R_0\rangle$. As the derivation proceeds, the derivation of the tree D_i is converted to D_{i+1}, and a leaf node s will be added to D_{i+1}.

(3) Derivation process P: indicates the derivation process of the algorithm, consisting of a series of derivation trees D_0, D_1, ..., D_n.

(4) Refutation: If the derivation process P is the last derivation tree D_n, all leaf nodes are UNSAT, then the derivation process is called a refutation.

(5) Witness: If at least one leaf node is SAT on the last derivation tree D_n of the derivation process P, the derivation process P is called a witness.

(6) Property to be verified ϕ: indicates the neural network property to be verified. If the initial node s_0 in D_0 is initialized according to property ϕ, a series of derivation trees D_0, D_1, ...,D_n derived from D_0 as the starting point are called the derivation process P of property ϕ.

(7) Soundness: A calculus is sound if, whenever a derivation D from ϕ is either a refutation or a witness, ϕ is correspondingly unsatisable or satisable, respectively [9].

(8) Completeness: A calculus is complete if there always exists either a refutation or a witness starting from any ϕ [9].

Lemma 1. *Let $\mathcal{D}$ denote a derivation starting from a derivation tree D_0 with a single node $s_0 = \langle \mathcal{B}_0, T_0, l_0, u_0, \alpha_0, R_0\rangle$. Then, for every derivation tree D_i appearing in $\mathcal{D}$, and for each node $s = \langle \mathcal{B}, T, l, u, \alpha, R\rangle$ appearing in D_i (except for the distinguished nodes SAT and UNSAT), the following properties hold:*

(i) an assignment satisfies T_0 if and only if it satisfies T;and

(ii) the assignment α satisfies T (i.e., α satisfies all equations in T). [9]

Lemma 2. *Let $\mathcal{D}$ denote a derivation starting from a derivation tree D_0 with a single node $s_0 = \langle \mathcal{B}_0, T_0, l_0, u_0, \alpha_0, R_0\rangle$. If there exists an assignment α^* (not necessarily α_0) such that α^* satisfies T_0 and $l_0(x_i) \le \alpha^*(x_i) \le u_0(x_i)$ for all i, then for each derivation tree D_i appearing in $\mathcal{D}$ at least one of these two properties holds:*

(i) D_i has a SAT leaf.

(ii) D_i has a leaf $s = \langle \mathcal{B}, T, l, u, \alpha, R\rangle$ (that is not a distinguished node SAT or UNSAT) such that $l(x_i) \le \alpha^*(x_i) \le u(x_i)$ for all i. [9]

Lemma 3. *Let $\mathcal{D}$ denote a derivation starting from a derivation tree D_0 with a single node $s_0 = \langle \mathcal{B}_0, T_0, l_0, u_0, \alpha_0, R_0\rangle$. Then, for every derivation tree D_i appearing in $\mathcal{D}$, and for each node $s = \langle \mathcal{B}, T, l, u, \alpha, R\rangle$ appearing in D_i (except for the distinguished nodes SAT and UNSAT), the following properties hold:*

(i) $R = R_0$; and

(ii) $l(x_i) \ge l_0(x_i)$ and $u(x_i) \le u_0(x_i)$ for all i. [9]

B. Proof of soundness of Leaky-Reluplex

The soundness of Leaky-Reluplex algorithm: if the algorithm is terminated in SAT state, the verification property must be satisfied; If the algorithm terminates in the UNSAT state, the verification property must be unsatisfiable.

Proof: if the algorithm is terminated in SAT state, indicating that the derivation process P of property ϕ is a witness, there is a SAT node on the last derivation tree D in P. Suppose the initial node on the initial derivation tree D_0 is $s_0 = \langle \mathcal{B}_0, T_0, l_0, u_0, \alpha_0, R_0\rangle$, and the previous node of the SAT node is $s = \langle \mathcal{B}, T, l, u, \alpha, R\rangle$ (using the $L-Relusuccess$ rule on s will generate a sat node). According to Lemma 1 [9], it can be seen that α satisfies T_0, and the application condition of $L-Relusuccess$ shows that the assignment of s and the upper and lower bounds satisfy the relationship: $l(x_j) \le \alpha(x_j) \le u(x_j)$, according to property 2 of Lemma 3 [9], can be derived: $l_0(x_j) \le \alpha(x_j) \le u_0(x_j)$, so α satisfies all linear inequalities in ϕ. It can be seen from the operating conditions of $L-Relusuccess$ that α satisfies all ReLU constraints R in s. According to property 1 of Lemma 3 [9], α can also be obtained: α also satisfies all ReLU constraints R_0

in s_0, and R_0 comes from ϕ, so It is concluded that α satisfies all the constraints in ϕ, proving that ϕ is indeed satisfiable in the SAT state.

If the algorithm terminated in the UNSAT state, then ϕ must be unsatisfiable. Because according to Lemma 2 [9], if ϕ is satisfiable, then there must be an assignment α^* that satisfies both T_0 and the initial upper and lower bounds. At this point, any node s that appears in its derivation process is not a sat node. It is the intermediate node that meets certain requirements in the upper and lower bounds, not the unsat node.Therefore, the reverse push is available. When an UNSAT node occurs, ϕ must be unsatisfiable.

C. Proof of completeness of Leaky-Reluplex

Similar to Reluplex [9], by using a strategy for fixing out-of-bounds violations, and by splitting on a Leaky ReLU pair whenever the $L{-}Update_{b1}$, $L{-}Update_f$, or $L{-}Relusuccess$ rules are applied to it more some fixed number of times, termination is guaranteed.

The completeness of Leaky-Reluplex algorithm: After the normal termination of the algorithm, the termination status must be SAT or UNSAT, and there is no third case.

Proof: In Leaky-Reluplex, if you continue to use the $ReluSplit$ rule to decompose Leaky ReLU constraint, when all Leaky ReLU constraints are decomposed, then each leaf node s on the derivation tree using the $Pivot_1$, $Pivot_2$, $Update$, $Failure$ rules [9] of the Simplex module and the new $L - Relusuccess$ rule in the Leaky-Reluplex algorithm to derive the solution, it is guaranteed to reach the SAT or UNSAT state in a limited number of steps. Because when the Leaky ReLU constraint in the problem is decomposed, all the remaining problems are pure linear constraints, which can be solved directly with Simplex. The completeness of the Simplex algorithm has been proved [9]. Like the Reluplex, the Leaky-Reluplex is based on the Simplex algorithm. Even if the Success rule in the Simplex is replaced by the $L - Relusuccess$ rule, the completeness of the algorithm is not affected. Therefore, based on the Simplex algorithm, the Leaky-Reluplex algorithm is also complete.