Contribution Title

Anonymous

No Institute Given

Abstract. The abstract should briefly summarize the contents of the paper in 150–250 words.

Keywords: First keyword · Second keyword · Another keyword.

1 Introduction

Deep neural networks (DNN) they have achieved exceptional performance in many fields like computer vision, natural language processing, game playing [10], and malware detection. However, it is found that DNNs are often lack of robustness, and are vulnerable to adversarial samples [14]. Even for a well-trained DNN, a very small (and even imperceptible) perturbation on the input may fool the network. This raises the concerns on the safety and reliability of DNNs when we deploy them in safety-critical applications like self-driving cars [15] and medical systems [9].

Formal verification is a way to guarantee the robustness of DNNs. In this work, we focus on local robustness, i.e., given an input and a manipulation region around this input, we determine whether a given DNN works correctly on any input in the region. Constraint solving is an elementary and classical way for DNN verification. In 2010, Luca Pulita et al. proposed the first method based on partition-refinement for DNN verification in [8]. In 2017, Katz et al. [4] and Ehlers [2] independently implemented Reluplex and Planet, two SMT solvers to verify DNNs with the ReLU activation function on properties expressible with SMT constraints. Since 2018, abstract interpretation has been one of the most popular methods for DNN verification in the lead of AI^2 [3], and subsequent works like [12,13,5,1,11,7,6] have improved AI^2 in terms of efficiency, precision and more activation functions (like sigmoid and tanh) so that abstract interpretation based approach can be applied to DNNs of larger size and more complex structures.

2 Preliminary

In this section, we recall some basic definitions on verification of deep neural networks. For a vector $x \in \mathbb{R}^n$, we used x_i to denote its i-th entry for $1 \leq i \leq n$. We write $a \wedge b := \min(a, b)$ and $a \vee b := \max(a, b)$ for $a, b \in \mathbb{R}$.

A deep neural network (DNN) is composed of a sequence of layers, starting with the input layer and ending with the output layer, with several hidden layers

in between. In a layer, there are several neurons, which each represent a real-valued variable. Except for neurons in the input layer, the value of a neuron is obtained by a function of the neurons in the previous layer. We model a DNN as a function $f : \mathbb{R}^m \to \mathbb{R}^n$, which is the composition of the functions between layers. Such functions include affine functions and non-linear activation functions. An affine function is of the form $y = Wx + b$, where W and b are a constant real-valued matrix and vector, respectively. In this work, we only consider the ReLU activation function, defined as $\mathrm{ReLU}(x) = 0 \vee x$ for $x \in \mathbb{R}$. For a ReLU relation $x_j = \mathrm{ReLU}(x_i)$, we say that the neuron x_i is definitely activated/deactivated, if all the possible values of x_i are non-negative/non-positive. If x_i is neither definitely activated nor definitely deactivated, we say that it is uncertain.

A safety property of a DNN is a tuple (f, X, P), where $f : \mathbb{R}^m \to \mathbb{R}^n$ is the neural network, and $X \subseteq \mathbb{R}^m$ and $P \subseteq \mathbb{R}^n$ are a set of input and output, respectively. The property (f, X, P) holds iff for all $x \in X$, $f(x) \in P$. Typically a DNN verification problem is to determine whether a given property (f, X, P) holds. The well-known local robustness is also a safety property of DNNs. For a classification DNN $f : \mathbb{R}^m \to \mathbb{R}^n$, its output usually represents the score of each classification label ranging in $\{1, 2, \ldots, n\}$, and the DNN selects the label with the largest score as its classification result. Formally we define $C_f(x) = \arg\max_{1 \le i \le n} f(x)_i$ as the classification output of f. A classification DNN $f : \mathbb{R}^m \to \mathbb{R}^n$ is locally robust in a given neighborhood $B(x_0)$ of an input $x_0 \in \mathbb{R}^m$, if for all $x \in B(x_0)$, $C_f(x) = C_f(x_0)$. Such a local robustness property is a safety property (f, X, P), where $X = B(x_0)$ and $P = \{y \in \mathbb{R}^n \mid \arg\max_i y_i = C_f(x_0)\}$.

Constraint solving is an elementary way for DNN verification. Given a safety property (f, X, P), we encode the input set X, the functions in the DNN f, and the negation of the property P as equations or inequalities, and determine whether the conjunction of these constraints is satisfiable. If it is satisfiable, there must exist a counterexample that violates the property, or else the property holds. Hereafter, we assume that the input region X as well as the the property P is a polytope, i.e., a conjunction of finitely many linear inequalities. In this case, the negation $\neg P$ is a disjunction $\bigvee_i Q_i$ of polytopes, and we usually divide the problem according to the disjunction into small ones in which every Q_i is a polytope.

3 Incremental Constraint Solving for DNN Verification

In this section, we introduce the incremental SMT solving problem for DNN verification, and present our main algorithm.

3.1 Problem formulation

We start with Reluplex. A Reluplex configuration $\mathcal{C}$ on a set $\mathcal{X}$ of variables is SAT, UNSAT, or a tuple $(\mathcal{B}, T, R, l, u, \alpha)$, where $\mathcal{B} \subseteq \mathcal{X}$ is the set of basic variables, T is the tableau which stores constraints of the form $x_i = \sum_{x_j \notin \mathcal{B}} a_{ij} x_j$ with $x_i \in \mathcal{B}$, $R \subseteq \mathcal{X} \times \mathcal{X}$ is the set of ReLU pairs, $l, u : \mathcal{X} \to \mathbb{R}$ is the lower and

the upper bound of each variable, respectively, and $\alpha : \mathcal{X} \to \mathbb{R}$ is the assignment of each variable. Given a property (f, X, P), we initialize the configuration by introducing slack variables to establish the tableau and invoking DeepPoly to compute the numerical bound. When we reach a figuration $(\mathcal{B}, T, R, l, u, \alpha)$, a local search is conducted by pivoting the variables and adjusting the assigment α of variables to match the constraints given by the tableau T and the ReLU relation R. If an assignment is found with all these constraints satisfied, then a counterexample is found and Reluplex outputs SAT. If there is a linear equation $x_i = \sum_{x_j \notin \mathcal{B}} a_{ij} x_j$ in the tableau T satisfying

$$l(x_i) > \sum_{x_j \notin \mathcal{B}} (a_{ij} \vee 0) \cdot u(x_j) + \sum_{x_j \notin \mathcal{B}} (a_{ij} \wedge 0) \cdot l(x_j) \tag{1}$$

or

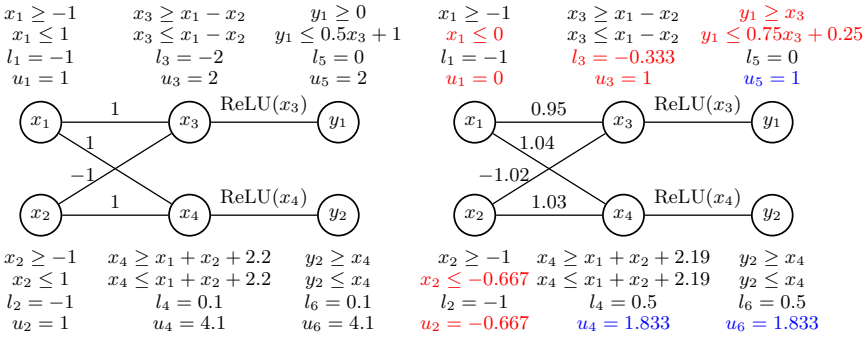
$$u(x_i) < \sum_{x_j \notin \mathcal{B}} (a_{ij} \vee 0) \cdot l(x_j) + \sum_{x_j \notin \mathcal{B}} (a_{ij} \wedge 0) \cdot u(x_j), \tag{2}$$

then no assignment can satisfy the constraints in this configuration, and we mark UNSAT for this configuration. A local search may not determine SAT or UNSAT for a configuration. After a certain number of steps, we stop the local search and choose an uncertain neuron x_j to split its numerical bound into $[0, u(x_j)]$ and $[l(x_j), 0]$. We assert the constraint $x_j \geq 0$ or $x_j \leq 0$ to the current configuration and conduct a new local search. If all the branches are marked with UNSAT, then Reluplex outputs UNSAT.

We formalize a Reluplex solving procedure as a labelled binary tree $\mathcal{T} = (V, E, r, L)$, where V is the set of nodes, $E \subseteq V \times V$ is the set of edges, $r \in V$ is the root, and L is a map assigning each node $v \in V$ to a sequence of configurations and each edge $e \in E$ to an assertion of the form $x_j \geq 0$ or $x_j \leq 0$, where x_j is an uncertain neuron. If the output is UNSAT, all the leaves in this tree have label of the form $\mathcal{C}_1, \ldots, \mathcal{C}_n, \mathsf{UNSAT}$, and we call them UNSAT leaves. If the output is SAT, there exists a leaf whose label is of the form $\mathcal{C}_1, \ldots, \mathcal{C}_n, \mathsf{SAT}$, which we call a SAT leave, and the other leaves are labelled with either $\mathcal{C}_1, \ldots, \mathcal{C}_n, \mathsf{UNSAT}$ or the empty sequence ε. For a sequence ℓ of configurations, we use $\ell\downarrow$ to represent the last configuration that is not SAT or UNSAT in ℓ. For a node $v \in V$, we use $\mathbf{Assert}(v)$ to denote the conjunction of assertions labelling the edges from the root to v.

Incremental constraint solving for DNN verification is aimed to efficiently verify a property on the modified DNN f' by making use of the verification procedure of the original DNN f. In this work, we consider incremental SMT solving based on the Reluplex framework, and assume that the modified DNN f' has exactly the same structure as the original DNN f, and only slightly differs in the weights and bias in the affine functions. We formally state our incremental SMT solving problem for DNN verification as follows:

> Given the Reluplex solving procedure $\mathcal{T} = (V, E, r, L)$ of a safety property (f, X, P), we determine whether the property (f', X, P) holds, where



(a) The original DNN f (b) The modified DNN f'

Fig. 1.

the DNN f' only differs from f in the weights and bias in the affine functions.

3.2 A running example

We show the main idea of our incremental SMT solving with a small example. Consider the DNNs in Fig. 1, and we verify whether $y_2 > y_1$ for all $(x_1, x_2)^{\mathrm{T}} \in [-1, 1] \times [-1, 1]$. For the original DNN f, we invoke a simple Reluplex solving.

3.3 Main algorithm

For this incremental SMT solving problem, the most significant intuition is that, the modified DNN f' is slightly different from the original DNN f, so it is highly possible that the verification procedure of f' inherits that of f in most branches. This motivates us to directly locate the configurations in the leaves, especially the one before the last which immediately infers UNSAT or SAT in the original solving. We check whether such inference still works for the modified DNN f'.

We show the main algorithm in Alg. 1. Given the Reluplex solving procedure $\mathcal{T} = (V, E, r, L)$ of a safety property (f, X, P) and the modified DNN f', we first initialize the configuration for f' and cut the unsatisfiable branches in $\mathcal{T}$. An edge $e \in E$ is eliminated, denoted by $T \setminus \{e\}$, if its label $L(e)$ contradicts with the result of the DeepPoly calculation, and the sub-tree beneath the edge e is deleted from $\mathcal{T}$ at the same time (Line 3-4). If there is a SAT leaf node v in the current tree $\mathcal{T}$, we first heuristically use an attack method (like PGD) to try to find a counterexample. Here we do not invoke the attack method straight to the DNN f', but f' under the constraints of the assertions from the root to this SAT leaf v. When we calculate the gradient of the DNN f', we always use the assertions in **Assert**(v) to determine the derivative of the involved ReLU functions. In this way, we are searching a counterexample which has the same

Algorithm 1 Incremental SMT solving for DNN verification

Input:
The Reluplex solving procedure $\mathcal{T} = (V, E, r, L)$ of a safety property (f, X, P) and the modified DNN f'

Output:
Return SAT if (f', X, P) does not hold, or UNSAT otherwise.

1: function Verify($f', X, P, \mathcal{T}$)
2: **Initialize**(f', X, P) ▷ Standard Reluplex encoding with a DeepPoly calculation
3: for $e \in E$ do
4: if $L(e) \cap$ **DeepPoly**$(f', X) = \varnothing$ then $\mathcal{T} \leftarrow \mathcal{T} \setminus \{e\}$
5: if there is a SAT leaf $v \in V$ then
6: **Attack**($f' \wedge$ **Assert**(v))
7: if a counterexample is found then return SAT
8: for leaf nodes $v \in V$ do ▷ We first deal with the SAT node and the nodes labelled by ε
9: if **Solve**(v) = SAT then return SAT
10: return UNSAT

activation pattern as the node v (Line 5-7). If no counterexample is found via the attack method, we go through all the leaf nodes in $\mathcal{T}$ and and solve the configuration in these nodes.

For the SAT node or the nodes labelled with ε, we invoke a simple Reluplex solving. For an UNSAT node v, we incrementally solve it by checking whether the old proof of UNSAT of v still works for the modified DNN f'. Here we locate ourselves in $L(v)\downarrow$, the configuration that infers UNSAT. We obtain the basic variables **Basic**$(L(v)\downarrow)$ in this configuration and turn the initial tableau T_0 for f' to the form with these basic variables (Line 4). Recall that an UNSAT Reluplex inference derives from Inequality (1) or (2), so we further focus on the linear equation $(L(v)\downarrow)^*$ which infers UNSAT. We use a standard linear approximation encoding and obtain tighter upper and lower bounds of the variables in $(L(v)\downarrow)^*$ by solving the linear programming (Line 2-3). The reason why we need linear programming to obtain tighter bounds is that tighter bounds are beneficial for inferring UNSAT of v for f'. Now we look at the equation $(L(v)\downarrow)^*$ in T with the tighter bounds, and check whether this configuration infers UNSAT (Line 5-6). If not, we continue to conduct a Reluplex solving for v from this configuration.

Alg. 1 is sound and complete. When a counterexample is found, it is either a true counterexample from the attack method, or a true counterexample from a Reluplex solving. When it returns UNSAT, all the leaf nodes in the search tree $\mathcal{T}$ are UNSAT nodes.

Theorem 1. Alg. 1 is sound and complete.

4 Optimizations

In this section, we introduce

Algorithm 2 The function **Solve**(v) for an UNSAT node v

Input:

The Reluplex solving procedure $\mathcal{T} = (V, E, r, L)$, the modified DNN f', and an UNSAT leaf node $v \in V$

Output:

Return SAT if $(f', X \wedge \textbf{Assert}(v), P)$ does not hold, or UNSAT otherwise.

```
function Solve(f', T, v)
    for x_i ∈ Var((L(v)↓)*) do ▷ (L(v)↓)*: the linear equation which infers UNSAT for v
        (l_i, u_i) ← LP(f' ∧ X ∧ ¬P ∧ Assert(v), x_i)   ▷ Output UNSAT if infeasible
    T ← T_0(Basic(L(v)↓))          ▷ Transfer the basic variables the same as L(v)↓
    if (L(v)↓)*(T, l, u) infers UNSAT then
        return UNSAT               ▷ Check whether the old UNSAT proof still works
    else return Reluplex(v)                                   ▷ Reluplex solving from v
```

5 Experimental Evaluation

	property1			property2			property3			property4	
	0.001	0.010	0.030	0.001	0.010	0.030	0.001	0.010	0.030	0.001	0.010
NN1	UNSAT 1857	UNSAT 1858	UNSAT 1859	UNSAT 1860	UNSAT 1861	UNSAT 1862	UNSAT 1863	UNSAT 1864	UNSAT 1865	UNSAT 1866	UNSAT 1867 U
	UNSAT 3641	UNSAT 1859	UNSAT 1860	UNSAT 1861	UNSAT 1862	UNSAT 1863	UNSAT 1864	UNSAT 1865	UNSAT 1866	UNSAT 1867	UNSAT 1868 U
NN2	UNSAT 3642	UNSAT 1860	UNSAT 1861	UNSAT 1862	UNSAT 1863	UNSAT 1864	UNSAT 1865	UNSAT 1866	UNSAT 1867	UNSAT 1868	UNSAT 1869 U
	UNSAT 3643	UNSAT 1861	UNSAT 1862	UNSAT 1863	UNSAT 1864	UNSAT 1865	UNSAT 1866	UNSAT 1867	UNSAT 1868	UNSAT 1869	UNSAT 1870 U
NN3	UNSAT 3644	UNSAT 1862	UNSAT 1863	UNSAT 1864	UNSAT 1865	UNSAT 1866	UNSAT 1867	UNSAT 1868	UNSAT 1869	UNSAT 1870	UNSAT 1871 U
	UNSAT 3645	UNSAT 1863	UNSAT 1864	UNSAT 1865	UNSAT 1866	UNSAT 1867	UNSAT 1868	UNSAT 1869	UNSAT 1870	UNSAT 1871	UNSAT 1872 U
NN4	UNSAT 3646	UNSAT 1864	UNSAT 1865	UNSAT 1866	UNSAT 1867	UNSAT 1868	UNSAT 1869	UNSAT 1870	UNSAT 1871	UNSAT 1872	UNSAT 1873 U
	UNSAT 3647	UNSAT 1865	UNSAT 1866	UNSAT 1867	UNSAT 1868	UNSAT 1869	UNSAT 1870	UNSAT 1871	UNSAT 1872	UNSAT 1873	UNSAT 1874 U
NN5	UNSAT 3648	UNSAT 1866	UNSAT 1867	UNSAT 1868	UNSAT 1869	UNSAT 1870	UNSAT 1871	UNSAT 1872	UNSAT 1873	UNSAT 1874	UNSAT 1875 U

	property1			property2			property3			property4	
	0.001	0.010	0.030	0.001	0.010	0.030	0.001	0.010	0.030	0.001	0.010
NN1	UNSAT 1857	UNSAT 1858	UNSAT 1859	UNSAT 1860	UNSAT 1861	UNSAT 1862	UNSAT 1863	UNSAT 1864	UNSAT 1865	UNSAT 1866	UNSAT 1867 U
	UNSAT 3641	UNSAT 1859	UNSAT 1860	UNSAT 1861	UNSAT 1862	UNSAT 1863	UNSAT 1864	UNSAT 1865	UNSAT 1866	UNSAT 1867	UNSAT 1868 U
NN2	UNSAT 3642	UNSAT 1860	UNSAT 1861	UNSAT 1862	UNSAT 1863	UNSAT 1864	UNSAT 1865	UNSAT 1866	UNSAT 1867	UNSAT 1868	UNSAT 1869 U
	UNSAT 3643	UNSAT 1861	UNSAT 1862	UNSAT 1863	UNSAT 1864	UNSAT 1865	UNSAT 1866	UNSAT 1867	UNSAT 1868	UNSAT 1869	UNSAT 1870 U
NN3	UNSAT 3644	UNSAT 1862	UNSAT 1863	UNSAT 1864	UNSAT 1865	UNSAT 1866	UNSAT 1867	UNSAT 1868	UNSAT 1869	UNSAT 1870	UNSAT 1871 U
	UNSAT 3645	UNSAT 1863	UNSAT 1864	UNSAT 1865	UNSAT 1866	UNSAT 1867	UNSAT 1868	UNSAT 1869	UNSAT 1870	UNSAT 1871	UNSAT 1872 U
NN4	UNSAT 3646	UNSAT 1864	UNSAT 1865	UNSAT 1866	UNSAT 1867	UNSAT 1868	UNSAT 1869	UNSAT 1870	UNSAT 1871	UNSAT 1872	UNSAT 1873 U
	UNSAT 3647	UNSAT 1865	UNSAT 1866	UNSAT 1867	UNSAT 1868	UNSAT 1869	UNSAT 1870	UNSAT 1871	UNSAT 1872	UNSAT 1873	UNSAT 1874 U
NN5	UNSAT 3648	UNSAT 1866	UNSAT 1867	UNSAT 1868	UNSAT 1869	UNSAT 1870	UNSAT 1871	UNSAT 1872	UNSAT 1873	UNSAT 1874	UNSAT 1875 U

6 Related Works

7 Conclusion

References

1. Anderson, G., Pailoor, S., Dillig, I., Chaudhuri, S.: Optimization and abstraction: a synergistic approach for analyzing neural network robustness. In: McKinley, K.S., Fisher, K. (eds.) Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019. pp. 731–744. ACM (2019)

2. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: 15th International Symposium on Automated Technology for Verification and Analysis (ATVA2017). pp. 269–286 (2017)
3. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: AI^2: Safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy (S&P 2018). pp. 948–963 (2018)
4. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: 29th International Conference on Computer Aided Verification (CAV2017). pp. 97–117 (2017)
5. Li, J., Liu, J., Yang, P., Chen, L., Huang, X., Zhang, L.: Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification. In: Chang, B.E. (ed.) Static Analysis - 26th International Symposium, SAS 2019, Porto, Portugal, October 8-11, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11822, pp. 296–319. Springer (2019)
6. Li, R., Li, J., Huang, C., Yang, P., Huang, X., Zhang, L., Xue, B., Hermanns, H.: Prodeep: a platform for robustness verification of deep neural networks. In: Devanbu, P., Cohen, M.B., Zimmermann, T. (eds.) ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020. pp. 1630–1634. ACM (2020)
7. Müller, C., Singh, G., Püschel, M., Vechev, M.T.: Neural network robustness verification on gpus. CoRR abs/2007.10868 (2020), https://arxiv.org/abs/2007.10868
8. Pulina, L., Tacchella, A.: An abstraction-refinement approach to verification of artificial neural networks. In: Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings. pp. 243–257 (2010)
9. Sheikhtaheri, A., Sadoughi, F., Dehaghi, Z.H.: Developing and using expert systems and neural networks in medicine: A review on benefits and challenges. J. Medical Syst. 38(9), 110 (2014)
10. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T.P., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of go with deep neural networks and tree search. Nature 529(7587), 484–489 (2016)
11. Singh, G., Ganvir, R., Püschel, M., Vechev, M.T.: Beyond the single neuron convex barrier for neural network certification. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada. pp. 15072–15083 (2019)
12. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.T.: Fast and effective robustness certification. In: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada. pp. 10825–10836 (2018)
13. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: An abstract domain for certifying neural networks. PACMPL 3(POPL), 41:1–41:30 (2019)
14. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. In: International Conference on Learning Representations (ICLR2014) (2014)

15. Urmson, C., Whittaker, W.: Self-driving cars and the urban challenge. IEEE Intell. Syst. 23(2), 66–68 (2008)