Verification of Neural Nets: Towards Deterministic Guarantees

Hussein Sibai
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Urbana, IL 15213
sibai2@illinois.edu

Abstract

The success of neural networks in different classification and regression tasks encouraged different researchers and companies to incorporate them in safety-critical systems such as autonomous vehicles. Because stochastic guarantees are not enough to ensure safety in these settings, the formal methods started to develop methods and tools to formally verify properties of these neural nets. This report is a summary of the recent paper "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks" by Katz et al. The authors present an extension of the Simplex method to formally verify or falsify properties of Rectified Linear Unit (ReLU) based networks. They used it to verify some interesting properties of a prototype deep neural network implementation of the next-generation Airborne Collision Avoidance System for unmanned aircraft (ACAS Xu).

1 Introduction

Deep neural networks have shown a lot of success in accomplishing challenging tasks such as image classification, speech recognition and game playing. They are trained on a finite set of the input space, called the training set, and are expected to generalize, i.e. result in a correct answer even for new data that was not part of the training dataset. Their popularity comes from the fact that they met this expectation for random input data in many important tasks. However, it has been shown [5] that a carefully chosen small perturbation of the input could cause a network to result in a different answer. For example, one can add adversarial noise with small norm to an image that would not normally change its class from a human perspective, while changing it from the network one. This raises safety concerns especially when these networks are incorporated in safety-critical systems. It also motivates the need for methods and tools to verify specific properties of neural nets. Actually, it is an instance of a model checking problem, the model here being the neural net. And, there is a mature field of research about model checking and theorem proving which provide tools and methods that may be useful in our situation.

Checking even simple properties of ReLU-based networks is NP-complete as proved in Appendix A of the paper by the reduction to the SAT problem. The difficulty of verification is caused by the ReLU nonlinear activation units which makes the problem nonconvex. A neural netowrk network is a stack of layers each containing a set of nodes. The output of a node is the value of a nonlinear activation function applied to a linear combination of the outputs of the nodes of the previous layer. The input is fed to the first layer and the output of the network is the output of the last layer. In the paper, they consider networks where ReLU is the only nonlinear activation unit allowed. ReLU is a function that maps a real number x to zero if it is negative (*inactive* case) and to itself otherwise (*active* case). These type of networks are used widely in the machine learning community. Their piecewise linearity is believed to be a cause for their good generalization abilities. Also, this piecewise linearity makes

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

them only one step away from being instances of a linear programming problem where one wants to check the satisfiability of a conjunction of linear inequalities.

Satisfiability Modulo Theories (SMT) solvers and linear programming solvers are tools used in the model checking and optimization communities respectively to check the satisfiability of properties written in some theory. A natural thing to do is to try to use these solvers to verify properties of neural nets as a black box without utilizing the structure of the neural nets. In [4], the authors tried that approach while linearizing the sigmoid activation function, but the largest network they were able to verify is with only one hidden layer and 20 hidden units. So, a more scalable approach that utilizes the structure inherent in the neural network is needed. Katz et al. in [3] extended the Simplex method with few rules to handle constraints that include ReLU activation units. The worst case time complexity is still exponential in the number of nodes because of the NP-completeness of the problem. However, their heuristic helps in eliminating large branches of the search tree. They showed the effectiveness of their tool by verifying several interesting safety and robustness properties of a prototype DNN implementation of Airborne Collision Avoidance System for Unmanned Aircraft (ACAS Xu).

2 Background

2.1 Neural Networks

A neural network is a stack of layers. The first layer is the input layer and the last one is the output layer. The layers between the input and the output are called hidden layers. Each one consists of a set of nodes. Each of these nodes takes as input a linear combination of the output of the nodes in the previous layer to which it applies a nonlinear activation function. In the case of the paper under consideration, this nonlinear function is always a ReLU. Formally, ReLU maps any real number x to $\max(0, x)$. Thus, the output of a node is the output of the ReLU when applied to the dot product between a weight vector and the output of the previous layer plus some bias term. They denote the number of layers (including the input and the output layers) by n, the input layer by Layer 1, the output layer by Layer n, the size (number of nodes) of the i^{th} layer by s_i, the output of the j^{th} node in the i^{th} layer by $v_{i,j}$, the column vector $[v_{i,1}, \ldots, v_{i,s_i}]^T$ by V_i, the $s_i \times s_{i-1}$ weight matrix of the i^{th} layer (where $2 \leq i \leq n$) by W_i and its bias vector by B_i. Therefore, the output of the i^{th} layer V_i is equal to ReLU$(W_i V_{i-1} + B_i)$, where ReLU is applied element-wise. The weight matrices W_i's and bias vectors B_i are the result of an optimization algorithm applied to a finite data set called the training data set. For any new data point, the input is fed to Layer 1, then propagated through the network to the output layer.

2.2 Satisfiability Modulo Theories (SMT)

A theory T is a pair $\langle \Sigma, I \rangle$, where Σ is a signature and I is a class of Σ-interpretations that is closed under variable reassignment. For the purpose of the paper, they only care about the theory of real numbers. A Σ-formula is a first order logic with equalities formula with respect to an underlying theory. It is T-satisfiable (T-unsatisfiable) if it is satisfied (unsatisfied) by some (any) interpretation. They only consider quantifier free formulas. A theory solver is a tool to determine if a Σ-formula is T-satisfiable or T-unsatisfiable. The DPLL(T) architecture is a famous approach to determine T-satisfiability of a Σ-formula. It is a combination of a theory solver and SAT solver. In that architecture, the SAT solver operates on a boolean abstraction of the formula, then the theory solver checks it if it can correspond to some interpretation in the theory. Splitting-on-demand framework allows the theory solver to guide the search of the SAT solver.

2.3 Linear Real Arithmetic and Simplex

They consider the theory of real arithmetic denoted by $\mathcal{T}_{\mathbb{R}}$ as it is the most relevant in DNN verification. The signature of this theory consists of all rational numbers and the symbols $\{+, -, \cdot, \leq, \geq\}$ while I is the set of all real numbers. They focus on linear formulas where the atoms are of the form $\sum_{x_i \in \mathcal{X}} c_i x_i \bowtie d$, where $\mathcal{X}$ is the set of variables, $\bowtie \in \{=, \leq, \geq\}$ and c_i, d are rational numbers. For example, if the network is linear, i.e. the activation function is the identity mapping, the output is a linear combination of the input. Then, any property of the output of the network can be written as a Σ-formula in this theory. Simplex method is an efficient method to optimize some linear objective

function under linear constraints, i.e. a conjunction of linear inequalities called linear atoms. The first part of the method which finds a feasible set that satisfies the constraints is the relevant one to the work. Moreover, the second part which finds the optimal feasible solution is not used. It consists of passing over some rules that update a data structure called *configuration*. For a given set of variables $\mathcal{X} = \{x_1, \ldots, x_n\}$, in all but the last iteration, a configuration is a tuple $\langle \mathcal{B}, T, l, u, \alpha \rangle$ where $\mathcal{B} \subseteq \mathcal{X}$ is the set of basic variables, T is a matrix representing each basic variable as a linear combination of nonbasic variables and is called *tableau*, l and u are vectors mapping each variable to its lower and upper bound respectively, and α is a vector mapping each variable to a real number. In the last iteration, the configuration is either SAT or UNSAT. The initial configuration is generated as follows: for each linear atom, a basic variable is created and set in T to be equal to the left hand side of the inequality. Then, the right hand side is set to be an upper bound, lower bound, or both for that basic variable, depending on the operator. α is initialized to zero so that all the equations in T are satisfied although their bounds in l and u may not.

A rule consists of a premise and a conclusion. If the premise is satisfied by a configuration, the configuration is updated according to the conclusion. The rules of the Simplex algorithm are shown in Figure 1 where slack^+ and slack^- are defined as follows:

$$\text{slack}^+(x_i) = \{x_j \notin \mathcal{B} \mid (T_{i,j} > 0 \wedge \alpha(x_j) < u(x_j)) \vee (T_{i,j} < 0 \wedge \alpha(x_j) > l(x_j))$$
$$\text{slack}^-(x_i) = \{x_j \notin \mathcal{B} \mid (T_{i,j} < 0 \wedge \alpha(x_j) < u(x_j)) \vee (T_{i,j} > 0 \wedge \alpha(x_j) > l(x_j))$$

$$\text{Pivot}_1 \quad \frac{x_i \in \mathcal{B}, \quad \alpha(x_i) < l(x_i), \quad x_j \in \text{slack}^+(x_i)}{T := pivot(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}}$$

$$\text{Pivot}_2 \quad \frac{x_i \in \mathcal{B}, \quad \alpha(x_i) > u(x_i), \quad x_j \in \text{slack}^-(x_i)}{T := pivot(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}}$$

$$\text{Update} \quad \frac{x_j \notin \mathcal{B}, \quad \alpha(x_j) < l(x_j) \vee \alpha(x_j) > u(x_j), \quad l(x_j) \leq \alpha(x_j) + \delta \leq u(x_j)}{\alpha := update(x_j, \delta)}$$

$$\text{Failure} \quad \frac{x_i \in \mathcal{B}, \quad (\alpha(x_i) < l(x_i) \ \wedge \ \text{slack}^+(x_i) = \emptyset) \vee (\alpha(x_i) > u(x_i) \ \wedge \ \text{slack}^-(x_i) = \emptyset)}{\texttt{UNSAT}}$$

$$\text{Success} \quad \frac{\forall x_i \in \mathcal{X}.\ l(x_i) \leq \alpha(x_i) \leq u(x_i)}{\texttt{SAT}}$$

Figure 1: Simplex Rules

The Pivot rules are used to switch a basic variable with a nonbasic variable. The Update rule is used to update the value of a nonbasic variable. The Failure rule checks if there is no way to do a feasible change anymore and declare the property unsatisfied if that is true. Finally, the Success rule checks if the assignment vector α satisfies the bounds in l and u and if that is true, declares the property as satisfied and the values of the variables in α are a witness of this satisfaction. All the rules preserves the equalities in T while the lower and upper bounds in l and u may be violated temporarily and then fixed by the Update rule. It is well known that there exist strategies where the algorithm will always terminate. Besides, it is sound, i.e. when it results in SAT, then the formula is satisfied and the values in α are a witness and when it outputs UNSAT then there is no assignment of the variables in real numbers that makes the formula true. It is also complete, i.e. for every possible initial configuration resulting from some formula, it will output SAT or UNSAT.

3 Reluplex (ReLU with Simplex)

The key contribution of the paper is covered in this section. It discusses the modifications done for the Simplex method so that it can handle ReLUs. A brute force way to solve the problem of the existence of ReLUs is to encode them as disjunctions of the active and inactive states. Then, feed the resulting formula to an SMT solver. Theoretically, this will lead to an exponential time complexity in the number of ReLUs in the network. This behavior was also seen in practice. So, instead, they defined the binary predicate $\text{ReLU}(x, y)$ which is true iff $y = \max(0, x)$. Then, they allowed the constraints

to include the application of ReLU to linear terms in addition to the linear inequalities. Now, each ReLU unit is encoded as two variables, a backward facing variable x^b and a forward facing variable x^f. Then, ReLU(x^b, x^f) is added to the constraints.

Reluplex do similar steps as Simplex. Its configuration is the same as that of the Simplex with an additional matrix $R \subseteq \mathcal{X} \times \mathcal{X}$ in the tuple representing the ReLU connections. As Simplex, it allows the variables two violate their bounds temporarily. It also allows them to violate their ReLU semantics temporarily. They kept the rules Pivot$_1$, Pivot$_2$ and Update to handle the out-of-bound violations. Moreover, they modified the Success rule to check if the semantics of the ReLUs are satisfied in addition to the bounds and they called it ReluSuccess. They added the rules Update$_b$ and Update$_f$ to allow a backward or forward facing variable to correct the semantics of the ReLU unit. They also added the PivotForRelu rule to switch a basic variable appearing as a backward facing or forward facing for some ReLU unit to a nonbasic variable so that the Update rules can be applied. Finally, they added the ReluSplit rule that splits a ReLU as a disjunction of the active and inactive states. Formally, the new rules are as follows:

$$\textsf{Update}_b \quad \frac{x_i \notin \mathcal{B}, \quad \langle x_i, x_j \rangle \in R, \quad \alpha(x_j) \neq \max(0, \alpha(x_i)), \quad \alpha(x_j) \geq 0}{\alpha := update(x_i, \alpha(x_j) - \alpha(x_i))}$$

$$\textsf{Update}_f \quad \frac{x_j \notin \mathcal{B}, \quad \langle x_i, x_j \rangle \in R, \quad \alpha(x_j) \neq \max(0, \alpha(x_i))}{\alpha := update(x_j, \max(0, \alpha(x_i)) - \alpha(x_j))}$$

$$\textsf{PivotForRelu} \quad \frac{x_i \in \mathcal{B}, \quad \exists x_l.\ \langle x_i, x_l \rangle \in R \vee \langle x_l, x_i \rangle \in R, \quad x_j \notin \mathcal{B}, \quad T_{i,j} \neq 0}{T := pivot(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}}$$

$$\textsf{ReluSplit} \quad \frac{\langle x_i, x_j \rangle \in R, \quad l(x_i) < 0, \quad u(x_i) > 0}{u(x_i) := 0 \qquad l(x_i) := 0}$$

$$\textsf{ReluSuccess} \quad \frac{\forall x \in \mathcal{X}.\ l(x) \leq \alpha(x) \leq u(x), \quad \forall \langle x^b, x^f \rangle \in R.\ \alpha(x^f) = \max(0, \alpha(x^b))}{\texttt{SAT}}$$

Figure 2: Reluplex Rules

Their strategy was to fix all out-of-bounds violations first and then fix the ReLU relations. They proved in Appendix B of the paper that their method is also sound and complete. The proofs consist of three key lemmas. First Lemma proves that the assignment in α is always satisfying the equalities in T. Second Lemma proves by induction that the leafs of the derivation trees are either SAT or configurations that satisfy the bounds. The third Lemma proves that the ReLU property will be preserved by all the rules over the derivation tree, and the upper and lower bounds can only get tighter (because of the optimizations shown later) down the derivation tree. They use these three lemmas to prove that it is a sound procedure. Its completeness follows from the completeness of Simplex and the fact that the split rule can be applied till there is not anymore ReLUs in the property when Simplex can be applied.

4 Efficient Implementation of Reluplex

They used several techniques to improve the performance of the algorithm.

4.1 Floating Point Arithmetic

They suggested that considering the inputs, weights and the outputs are floating point numbers rather than real numbers reduce significantly the number pivot operations needed. Each of these operations is expensive especially when requiring the division with very small numbers or multiplication of very large ones. However, using floating point arithmetic reduces the precision of the results. Specifically, the algorithm will not be anymore sound. For example, if it results in UNSAT, that means there is no floating point number assignment of the variables that makes the formula true, but there may be a real number assignment that makes it true. To tackle partially this issue, even without recovering soundness, they keep track of the roundoff error. If it exceeds some predefined threshold, the steps are undone, and since they know the initial tableau and the final one, a shorter path (fewer number of pivot operations) is taken leading to the same result with a better accuracy.

4.2 Tighter Bound Derivations and Derived Bounds and Conflict Analysis

They added two rules to optimize the bounds on each variable. Although tightening the bounds takes time, it has a huge benefit in eliminating ReLUs from the search process as a lot of them will have bounds in either the active or the inactive state.

In case of contradictions in the bounds of a variable (the upper bound is smaller than the lower bound), instead of undoing only one step, multiple steps are undone till there is no more contradiction. This is a standard techniques used in SAT and SMT solvers.

4.3 Bound Under-Approximation

Instead of just tightening the bounds, sometimes under-approximating them might be helpful in eliminating more ReLUs. This technique will not preserve soundness since a feasible solution may still exist and be missed by the under-approximation of the bounds.

5 Case Study: The ACAS Xu System

They implemented their tool over the open source LP solver GLPK [1]. As a case study, they considered a prototype implementation of ACAS Xu consisting of 45 DNNs. The implementation of this system as DNNs reduced the memory size needed from 2GB to 3MB. The inputs of the system are the values of seven sensors of an unmanned aircraft representing its state and the state of another intruder aircraft. The system uses the values of two of these sensors to choose which of the 45 DNNs to use. Then, the chosen DNN is fed with the values of the other 5 sensors to compute 5 outputs representing the advisories: Clear-of-conflict (COC), weak right, strong right, weak left and strong left. The output with the least value is taken as the recommended advisory.

In their implementation, they used floating point arithmetic and a round-off-error threshold of 10^{-6}. Moreover, they fixed all the out-of-bounds violations before fixing the ReLU semantics. Also, they did bound tightening after each pivot operation on the corresponding variable and the whole tableau every few thousand operations. They split a ReLU if 5 operations were applied to fix it. Interestingly, they observed that splitting upto 10% of the ReLUs led to the elimination of the rest.

To show the practical benefit of their heuristic, they tried to use state of the art SMT solvers to verify simple properties of the form $x < c$ where x is one of the outputs of 2 of the 45 DNNs chosen arbitrarily. Most of them timed-out with a threshold of 4 hours while Reluplex was able to verify them in few seconds as shown in the following table:

	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6	φ_7	φ_8
CVC4	-	-	-	-	-	-	-	-
Z3	-	-	-	-	-	-	-	-
Yices	1	37	-	-	-	-	-	-
MathSat	2040	9780	-	-	-	-	-	-
Gurobi	1	1	1	-	-	-	-	-
Reluplex	11	3	9	10	155	7	10	14

Figure 3: Comparison with SMT and LP solvers (in seconds)

Then, they considered more interesting properties and used Reluplex to verify them. These properties concerns safety and convenience. For example, if the intruder is near, the advisory will be “strong left” or “strong right” depending on from which side the intruder is approaching. Also, if the intruder is far, the advisory will not be “strong”. The results are shown in the following table:

The Networks column represent the number of networks they tried to verify. The Result column represent if the networks violated the property, i.e. if it is SAT it means that the network violated the property and the solver generated a counter example, otherwise it did satisfy it. The time is in

[1] `www.gnu.org/software/glpk/`

	Networks	Result	Time	Stack	Splits
ϕ_1	38	UNSAT	278403	45	1090082
	7	TIMEOUT			
ϕ_2	35	SAT	82419	44	284515
ϕ_3	42	UNSAT	28156	22	52080
ϕ_4	42	UNSAT	12475	21	23940
ϕ_5	1	UNSAT	19355	46	58914
ϕ_6	1	UNSAT	180288	50	548496
ϕ_7	1	TIMEOUT			
ϕ_8	1	SAT	40102	69	116697
ϕ_9	1	UNSAT	99634	48	227002
ϕ_{10}	1	UNSAT	19944	49	88520

Figure 4: Verifying properties of the ACAS Xu networks

seconds. The Stack and Splits columns represent the maximal number of nested case-splits averaged over the number of networks tested and the total number of case-splits respectively.

Finally, they tried to check the robustness of the DNNs. Robustness and adversarial examples had got a lot of attention in the past few years because of the security and safety concerns that they cause. They use the following definition of local robustness: a network is δ-locally-robust at a data point x if for every x' such that $\|x - x'\| \leq \delta$, the output of the network is the same. They formulated this as a property of the output of the network and tested the robustness of one of the networks over 5 arbitrary points and five different values of δ. The results are shown in the following table:

	$\delta = 0.1$		$\delta = 0.075$		$\delta = 0.05$		$\delta = 0.025$		$\delta = 0.01$		Total
	Result	Time	Result	Time	Result	Time	Result	Time	Result	Time	Time
Point 1	SAT	135	SAT	239	SAT	24	UNSAT	609	UNSAT	57	1064
Point 2	UNSAT	5880	UNSAT	1167	UNSAT	285	UNSAT	57	UNSAT	5	7394
Point 3	UNSAT	863	UNSAT	436	UNSAT	99	UNSAT	53	UNSAT	1	1452
Point 4	SAT	2	SAT	977	SAT	1168	UNSAT	656	UNSAT	7	2810
Point 5	UNSAT	14560	UNSAT	4344	UNSAT	1331	UNSAT	221	UNSAT	6	20462

Figure 5: Local adversarial robustness tests. All times are in seconds.

As before, SAT means that Reluplex found an adversarial example that violates the robustness property. At different points, the robustness of the network is different which is expected. They suggested a brute force method to check the global robustness of the network, but it is not feasible in practice even for normal sized networks.

6 Related Work

There has been few attempts to verify neural networks in the past few years. Probably the first one, [4] used piecewise linear approximation of the sigmoid function and invoked SMT solvers, but they were not able to verify properties for networks with more than 20 hidden units and 1 hidden layer.

In [1], the authors restricted the search over a region where each of the ReLUs is either in the active or inactive state to check the robustness of a ReLU-based network at a certain point. However, it is not clear what can be done using their method if there is no such region around the point.

In [2], the authors checked if a network have consistent labeling over a finite set of points around some data point of interest. These points represent certain "common" perturbations of the input such as camera scratches. However, there may be some points in the infinite domain around the point that were not modeled that are mislabeled.

7 Discussion

The work presented is a first step towards the goal of making the formal verification of neural networks practical. The problem is becoming more important as DNNs are becoming a natural part of the control systems of autonomous vehicles and unmanned aircraft. The standard stochastic guarantees in statistical learning theory are not enough when human lives may be in danger and the recent papers about adversarial examples in neural nets are an example of what can go wrong.

However, from the few attempts to formally verify interesting properties of medium sized networks clearly shows that aiming for absolute correctness of the neural nets is a hard goal to achieve. Even in this work, the reported results are when using floating point arithmetic and for only ReLU-based networks. As said by the authors, verifying it for real numbers would take much more time. Also, current networks used in autonomous vehicles are not only ReLU-based but contain other types of activation functions and are much larger in size. Moreover, the input of the network considered is only 5-dimensional while current networks take as input a video feed which have a much larger dimension. The first thing to be done is define what is the type of correctness is needed in practice. For example, it may be acceptable if a neural network used as a controller of a car outputs a wrong class for a few number of the inputs over a certain trajectory of inputs as long as the trajectory of the car is still safe (avoiding an obstacle or stopping at a stop sign). For example, an experiment we conducted in another class project, showed that adding adversarial noise to a stop sign did not cause the network to miss classify all the frames, only few of them. That for example, would not cause the car to not stop at the stop sign. This is usually the case in hybrid systems verification, where people are concerned about the safety of the whole trajectory rather than individual decisions. It is interesting to look at the methods and approaches in that area to verify the safety of the system as a whole instead of the safety of the neural net alone.

References

[1] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, and A. Criminisi. Measuring neural net robustness with constraints. *arXiv preprint arXiv:1605.07262*, 2016.

[2] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety verification of deep neural networks. *CoRR*, abs/1610.06940, 2016.

[3] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. *CoRR*, abs/1702.01135, 2017.

[4] L. Pulina and A. Tacchella. Challenging smt solvers to verify neural networks. *AI Commun.*, 25(2):117–135, Apr. 2012.

[5] 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.