Seminar Formal Methods for Learned Systems, Project proposal

Iterative Marabou

tightening verifiable bounds

Jop Schaap

April 18, 2024

1 Introduction

Deep Neural Networks(DNN) show a very high performance on a wide variety of tasks. However, the complexity of DNNs, makes it difficult to know whether the mapping from the inputs to outputs is correct for all possible input output pairs. This in turn prevents us from reasoning whether or not the DNN is correctly functioning over an input space [1]. However, DNNs are often used in safety critical systems, such as for autonomous driving [2]. Thus it is important to find and verify properties of these DNNs.

Katz, Barrett, Dill, *et al.* [3] devised a way to formally verify such properties. They devised the algorithm Reluplex. Reluplex is able to verify properties in the form of linear constraints on the input and output variables of a DNN. The Reluplex algorithm is sound and complete. Meaning, that whenever there exists a violation on the constraints then Reluplex is able to find it, and if there exists no such violation of the constraints then Reluplex proofs this.

The Reluplex algorithm nevertheless, requires the user to define both an input space and a safe output space. However, the input space might not be clear when examining a problem. And instead, we might want to find the safe input space in which the DNN can be used, without violating some constraints on the output space. For instance one might seek to verify at which speeds an autonomous vehicle can safely drive while preventing the vehicle to brake too abruptly in any situation.

The approach presented in this paper uses the Marabou[4] tool, which is an extension of Reluplex[3] to find these input spaces. Here the main idea is to iteratively apply the Marabou solver with larger input bounds until the solver finds a counter example showing the new bound to be too wide, and thus violating the output constraints.

Research Question

Thus the research question I would like to answer in this project is:

- How effective is the proposed framework in combination with Marabou to construct safe input spaces?

Here I want to measure effectiveness in how wide the proposed approach would be able to construct the input space, within a limited time.

To help answer the research question I want to answer the following sub-questions.

- How big is the performance impact from the proposed framework?
- How well does the framework converge?

Approach

Finding this box can be seen as a multiple objective optimization problem, since we want to increase the width of the box in all dimensions as much as possible. But when we increase the width of the box in one direction we might no longer be able to increase the size of the box in a different direction. In this paper I present an approach that guarantees converging to a pareto optimal solution for this problem. Here a pareto optimal solution is a box with certain dimensions such that one cannot increase one of the dimensions without reducing the other dimensions

Paper Structure

This paper is organized as follows. First, in Section 2, I will discuss the preliminaries necessary to understand the rest of the paper. Hereafter, in Section 3, I will introduce the topic and method of the framework. Then, in Section 4, I will explain and discuss experimental results. Finally, in Section 5, I will give some concluding remarks and ideas for future work.

2 Preliminaries

In this section I will introduce terms and definition necessary to understand this work. I will start by first describing the inner workings of *Simplex*, hereafter I will introduce *Reluplex* and *Marabou*. Finally, I will discuss the *quick find* algorithm and how this helps our search.

Reluplex

The classical method of verifying properties come in the form of putting some constraints on the input and verifying that there exist no inputs in the constrained region that fall result into an illegal output region. Here the output region is similairly constrained. These constraints are herafter put into an solver that is able to verify if there exists any Assignment that both adheres to the input and to the output.

Reluplex is such a solver designed to verify these properites, it does this by applying a technique based on the Simplex method. Reluplex is sound and complete, meaning that if there exists a violating input it will eventually find it, and if there does not exists such an input then it will prove this.

The main strengths of Reluplex over that of regular Mixed Integer Programming(MIP), is that Reluplex is better able to deal with the ReLu activation functions.

Reluplex however does have some limitations, mainly it is not able to verify non linear constraints. Furthermore, it only supports the ReLu activation function. Finally, the process of verifying networks is computationally expensive and generally does not scale to large DNNs.[3]

Marabou

Marabou is an enhancement build on top of the Reluplex algorithm. It uses the same underlying verification method based on Simplex. It furthermore comes with additional improvements that allow it to solve more complex DNNS. The most impactfull improvement is a divide and conquer mechanism that divides the input/output space into smaller sections when the overal network proofs to difficult to solve. Thus Marabou is a more sophisticated version of the original Reluplex algorithm, and hence for the experiments I will use Marabou.[4]

Multi Objective Optimization

The task as described in Section 1, can be formulated as a multi objective optimization problem. This is formulated in the following way. The optimization problem we are trying to solve, is that we want to maximize the bounds on each input variable. However, these bounds are related, since if the bounds on one variable increases then the possible bounds on another variable might decrease.

A releated concept to Multi Objective Optimization, is Pareto Optimality. A solution is said to be Pareto optimal, if there exist no way to increase any of the objective values without decreasing another objective value[5]. For our problem this thus entails that we cannot increase the size of the box in any dimensions without decreasing the size of the box in the other dimensions.

3 Topic

In the following Section I will start with first explaining the general structure of the framework. Hereafter, I will explain how to algorithm works. And finally I will explain why the algorithm works. And I will finish with an explanation on why this approach produces a Pareto optimal solution.

Iterative Marabou

The iterative Marabou algorithm works by separating the upper and lower bound of each variable. These bounds are then again split into an optimistic bound and an pessimistic bound. The pessimistic bound is the lowest bound for which the Marabou solver has returned UNSAT, e.g. the largest values such that the output is restricted by the constraints. Meanwhile, the optimistic bounds contain the smallest bounds for which Marabou has returned SAT.

The algorithm, during execution iteratively tries a new higher pessimistic bounds. The algorithm is described schematically in Figure 1 and checks Marabou whether this new bound is still UNSAT, and if so increases the pessimistic bound to this new bound, if on the other hand the problem turns out to be SAT after setting the new bound then the algorithm reduces the pessimistic bound to this new value.

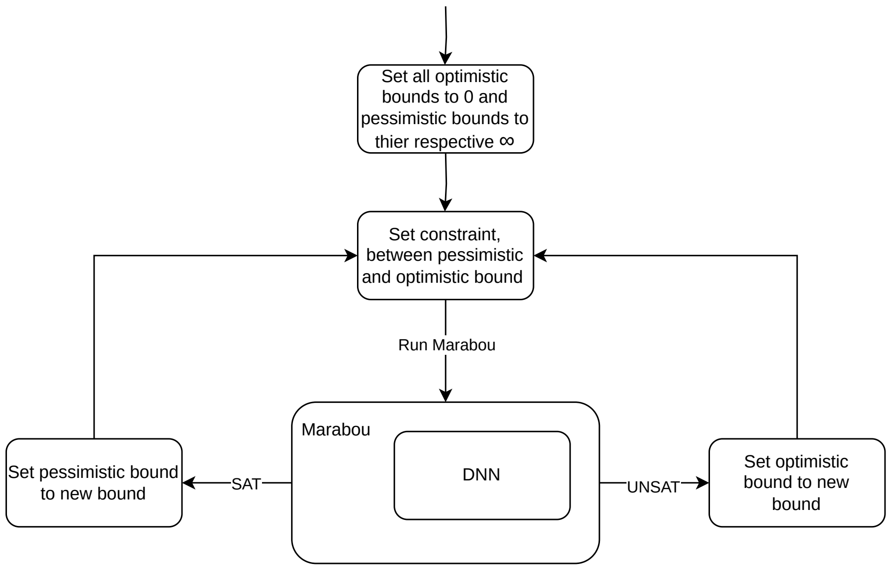
Set all optimistic bounds to 0 and pessimistic bounds to thier respective ∞
Set constraint, between pessimistic and optimistic bound
Run Marabou
Marabou
DNN
SAT
UNSAT
Set pessimistic bound to new bound
Set optimistic bound to new bound

Figure 1: The main loop of the Iterative Marabou, algorithm.

The main intuition behind iterative Marabou is that increasing the optimistic bound of a variable can only every decrease the pessimistic bounds of all other problems, and never increase it. This intuition thus allows us to continuously tighten the space in which a true Pareto optimal solution lies.

To let the technique work we require that we have at least one point for which the properties hold. Finding of this point is outside the scope of this report and I instead assume this starting point to be 0. For the project I want to create an algorithm as described below:

Algorithm 1, shows how the algorithm works. Here the algorithm is defined to take each iteration a random dimension, and then splits the search space in this dimension in two. On this bound the Marabou solver is called to check whether it upholds the constraints. If the constraints are upheld we have thus found a better pessimistic bound and can thus increase this. If the constraints are not upheld we know that there will be no bound greater than our tested bound that upholds the constraints and thus we reduce the optimistic bounds.

Algorithm 1 Iterative Marabou algorithm

```
1: $LB_{pessimistic}[\#parameters] \leftarrow [0]$
2: $LB_{optimistic}[\#parameters] \leftarrow [-\infty]$
3: $UB_{pessimistic}[\#parameters] \leftarrow [0]$
4: $UB_{optimistic}[\#parameters] \leftarrow [\infty]$

5: while $time\_to\_run > 0$ do
6:     $dim \leftarrow$ random_dim()                                   ▷ Optimize the lower bound
7:     $mid \leftarrow (LB_{pessimistic}[dim] + LB_{optimistic}[dim])/2$            ▷ Get the midpoint
8:     addProperty$(input\_parameters[dim] \geq mid)$
9:     $upholds \leftarrow$ check_properties()
10:    if $upholds$ then
11:        $LB_{pessimistic}[dim] \leftarrow mid$        ▷ If the properties still hold we found a better bound
12:    else
13:        remove_last_property()
14:        $LB_{optimistic} \leftarrow mid$      ▷ If the properties donnot hold we find a limitation of the bound
15:    end if

16:    $mid \leftarrow (UB_{pessimistic}[dim] + UB_{optimistic}[dim])/2$            ▷ Optimize the upper bound
17:    addProperty$(input\_parameters[dim] \leq mid)$
18:    $upholds \leftarrow$ check_properties()
19:    if $upholds$ then
20:        $UB_{pessimistic}[dim] \leftarrow mid$
21:    else
22:        remove_last_property()
23:        $UB_{optimistic} \leftarrow mid$
24:    end if
25: end while
26: return $LB_{pessimistic}, UB_{pessimistic}$
```

Pareto OPtimality

This algorithm converges to a Pareto optimal solution. This is the case because of the following two facts. The algorithm never lowers the pessimistic bound such that a feasible value would be removed from the search space, as shown in Proof 3. And the lowering the optimistic bound cannot result into a better pareto optimal solution. These two facts together mean that the search space of all bounds decrease every iteration, and thus eventually converges to a Pareto optimal solution.

Proof.
Take B_i^O for the optimistic bound for input variable i.
Take B_i^P for the pessimistic bound for input variable i.
I denote B^O as the set of all optimistic bounds.
And I use $B^O - B_i^O + B'$ to denote the optimistic bound set wher bound B_i^O is replaced with B'.
Finally I use $UNSAT(B)/SAT(B)$ to denote that the bounds defined by B are proven to be UNSAT and SAT respectively.
We now need to proof the following:

$$\forall B_i' | (B_i^P < B_i' < B_i^O) \nexists B_j^P \text{ S.T.} \tag{1}$$
$$UNSAT(B^P - B_i^P + B_i') \rightarrow UNSAT(B^P - B_i^P + B_i' - B_j^O + B_j^P) \tag{2}$$

In other words, for all new bounds that result into an UNSAT solution it should be impossible to add a bound that previously caused the problem to be SAT to result into an UNSAT solution.
I proof this by noting that:

$$UNSAT(B^P) \tag{3}$$

and

$$\forall B_i^O SAT(B^P - B_i^P + B_i^O) \tag{4}$$

Thus there must be an assignment outside of the bounds of B^P but inside the bounds of $B^P - B_i^P + B_i^O$. From 1 we see that the newly added bounds are always strictly higher than the old bounds, thus we get that:

$$B^P - B_i^P + B_i^O \subset B^P - B_i^P + B_i' - B_j^O + B_j^P \tag{5}$$

Thus since there exist a satisfying assignment in $B^P - B_i^P + B_i^O$ there also must exists a satisfying assignment to $B^P - B_i^P + B_i' - B_j^O + B_j^P$. Thus we never prune a solution that is more Pareto optimal than every solution in our search space. $\square$

4 Experiments and Results

In this section I will first start by explaining the experiments I performed on my algorithm. Hereafter, I will show and discuss the results obtained.

Experimental Setup

I used a proof-of-concept implementation of the algorithm proposed. For the experiments I used the `ACASXU_experimental_v2a_1_1` DNN[6] to generate bounds for. I adapted this DNN by removing two second to last layers, since the original problem showed to take to long to obtain meaningful results. The resulting DNN had 5 layers, with a total of 410 parameters.

For the experiments I chose one of the outputs and set a constant bound (in such a way that the $\mathbf{0}$ vector adheres to this property). Hereafter, I ran the Iterative Marabou algorithm for 50 iterations, and recorded the volume of the pessimistic and optimistic bounds.

During preliminary testing I noticed that the Marabou verifier slowed down, for later iterations. To ensure I could still produce meaningful results I decided that for each iteration, to set a timeout of 240 seconds for Marabou, and let the algorithm assume the solver returned SAT, whenever Marabou timed out. It should be noted that this breaks the property that we converge to a pareto optimal solution.

All experiments where run on a HP ZBook Studio G5, with an *Intel(R) Core(TM) i7-8750H* cpu. The code ran in a docker container with the host os being Ubuntu 22.04.4. The code for the experiments, together with with instructions on how to run, can be accessed on github[1].

[1]`https://github.com/JopSchaap/iterative_marabou`

Results

The resulting convergence graph is shown in Figure 2. This graph shows the lower bound box size, and the upper bound box size. Here we see that the algorithm converges since the optimistic and pessimistic values are nearly equal after about 100 iterations.

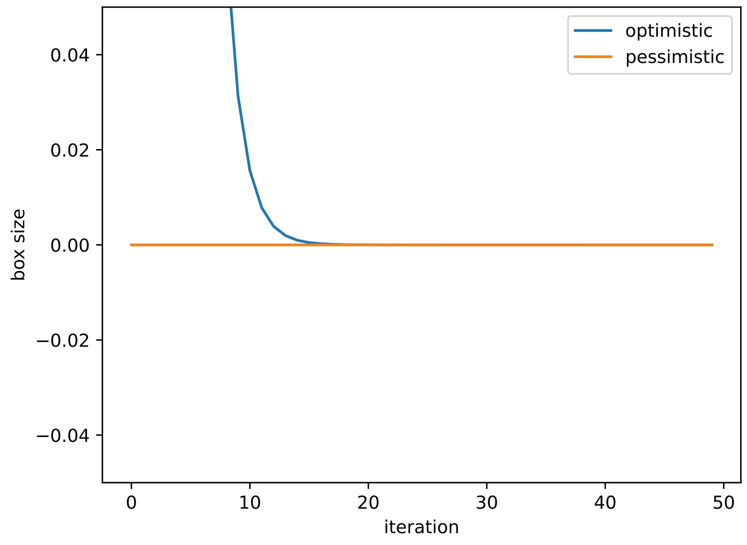

Figure 2: Convergence graph showing the size of the box for `ACASXU_experimental_v2a_1_1`, with the output constraint of $y_2 < 0.8$

Very Simple Net

To show the algorithm working I also did some experiments on the `ACASXU-_run2a-_1_1_tiny`, which is part of the test DNNs for Marabou. This DNN can be found on the Marabou github[2]. This DNN has just 2 layers, with a total of 90 parameters.

Figure 3, shows that the algorithm is as good as converged after only about 100 iterations.

[2]`https://github.com/NeuralNetworkVerification/Marabou`

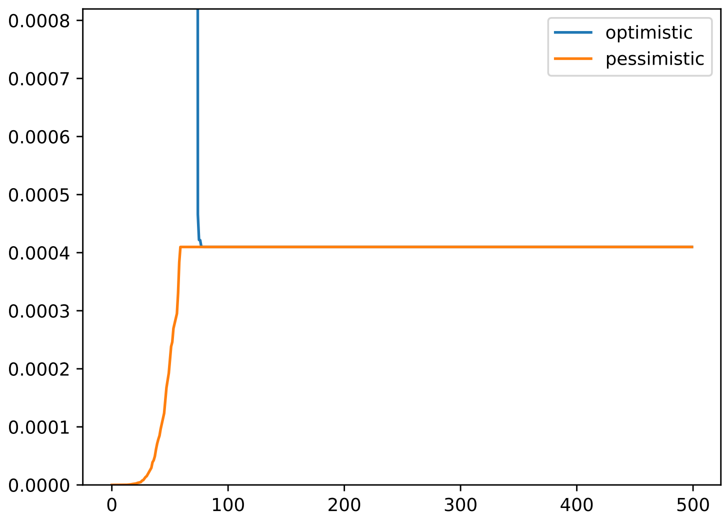

Figure 3: Convergence graph showing the size of the box for `ACASXU-_run2a-_1_1_tiny`, with the output constraint of $y_2 < 0.8$

Figure 4 shows the resulting box for the first two dimensions, together with the output values of the DNN. The background for Figure 4 shows the DNN output values where the input variables not listed in the particular picture are fixed at zero. This figure shows that while the box is of substantial size, there exists differencing values for the box widths, that are pareto optimal. This can be seen since the non white area is substantially larger than the red box, and This likely means that a combination of the input variables is violating the constraint in these areas.

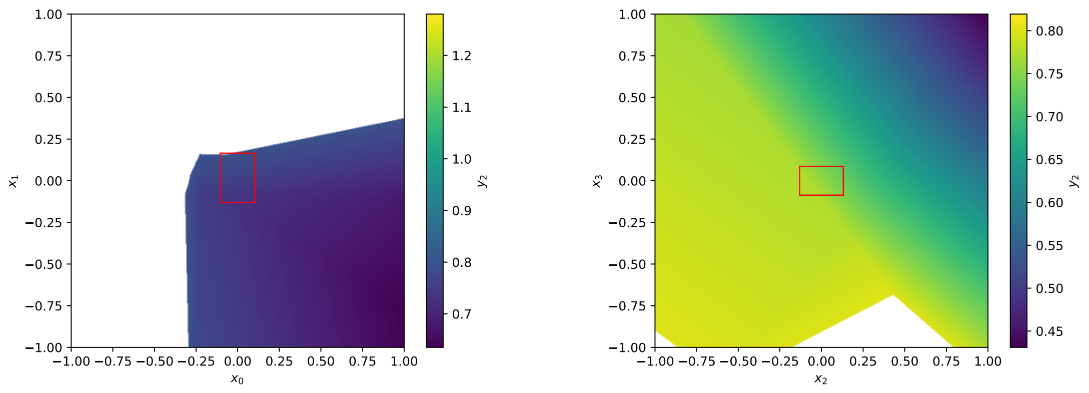

Figure 4: Different input variables and the networks response to the inputs(all inputs not shown are fixed at zero), here wite is used to signify the output constraint is not met in these regions. The red boxes show the values found by the algorithm.

5 Conclusion and Future Work

To conclude I presented a framework that is able to converge to Pareto optimal solution for finding input boxes, given output constraints on the DNN. The proposed algorithm is guaranteed to converge to these Pareto optimal solution by iteratively applying formal solving methods. The framework is based upon the formal guarantees and only requires that the solver algorithm is able to either find counter examples or a guarantee that no counter example exists.

The experiment shows that the technique fails to converge satisfyingly to nearly optimal solution. This mainly was caused by the Marabou algorithm taking longer and longer to run the more converged the bounds got.

Future work could focus on experimenting with more DNNs, since the DNN used for the experiment as shown in this paper turned out to be too complex to properly optimize. Experimenting with more DNNs might furthermore better highlight any potential strengths and weaknesses with this approach.

An additional improvement to the proposed algorithm would be to try it with other solvers. For instance, Henriksen and Lomuscio [7] developed a solver that uses adversarial search, and trades accuracy for correctness. One could for isntance use this method and assume the model to be adhering to the constraints when no counterexample is found within some timeout.

Another possible future improvement is to use more techniques from multiple objective optimization to find more preferable since the current technique might find boxes that are very long in one direction and quite short in other dimensions. Instead one might prefer input spaces that have the biggest multiplicative volume, in other words the solution that maximizes Equation 6. Future work could try to find better variable selection strategies to optimize these values.

$$\sum_{i=1}^{N} x_i \tag{6}$$

Finally, it should be possible to integrate the solver better with the proposed framework. The Marabou solver namely not only returns whether the constraints are met or not, but it also generates an assignment of variables that violate the constraints, these results could be used to update the optimistic bound more tightly. This technique should increase the performance of the framework since optimistic bounds are tightened faster, and it would also make the technique less reliant on the initialization of the optimistic bounds.

References

[1] C. Szegedy, W. Zaremba, I. Sutskever, *et al.*, *Intriguing properties of neural networks*, Feb. 2014. DOI: 10.48550/arXiv.1312.6199. arXiv: 1312.6199 [cs]. (visited on 03/21/2024).

[2] M. Bojarski, D. Del Testa, D. Dworakowski, *et al.*, *End to End Learning for Self-Driving Cars*, Apr. 2016. DOI: 10.48550/arXiv.1604.07316. arXiv: 1604.07316 [cs]. (visited on 03/22/2024).

[3] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks," in *Computer Aided Verification*, R. Majumdar and V. Kunčak, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2017, pp. 97–117, ISBN: 978-3-319-63387-9. DOI: 10.1007/978-3-319-63387-9_5.

[4] G. Katz, D. A. Huang, D. Ibeling, *et al.*, "The Marabou Framework for Verification and Analysis of Deep Neural Networks," in *Computer Aided Verification*, I. Dillig and S. Tasiran, Eds., vol. 11561, Cham: Springer International Publishing, 2019, pp. 443–452, ISBN: 978-3-030-25539-8 978-3-030-25540-4. DOI: 10.1007/978-3-030-25540-4_26. (visited on 03/21/2024).

[5] J. E. Stiglitz, "Pareto Optimality and Competition," *The Journal of Finance*, vol. 36, no. 2, pp. 235–251, 1981, ISSN: 1540-6261. DOI: 10.1111/j.1540-6261.1981.tb00437.x. (visited on 04/15/2024).

[6] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer, "Policy compression for aircraft collision avoidance systems," in *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, Sacramento, CA, USA: IEEE, Sep. 2016, pp. 1–10, ISBN: 978-1-5090-2523-7. DOI: 10.1109/DASC.2016.7778091. (visited on 04/17/2024).

[7] P. Henriksen and A. Lomuscio, "Efficient Neural Network Verification via Adaptive Refinement and Adversarial Search,"