Robustness of Neural Networks Based on MIP Optimization

Ramzi Ben Mhenni, Mohamed Ibn Khedher, Stéphane Canu

Robustness of Neural Networks Based on MIP Optimization

Ramzi Ben Mhenni[1], Mohamed Ibn Khedher[1] and Stéphane Canu[2]
[1]*IRT - SystemX, 8 Avenue de la Vauve, 91120 Palaiseau, France*
[2]*INSA Rouen Normandie, 685 Avenue de Universite 76800 Rouen, France*
{ramzi.ben-mhenni, mohamed.ibn-khedher}@irt-systemx.fr, stephane.canu@insa-rouen.fr

Keywords:
Neural network, Adversarial attack, Mixed Integer Programming, Branch-and-Bound algorithm.

Abstract:
Even though Deep Learning methods have demonstrated their efficiency, they do not currently provide the expected security guarantees. They are known to be vulnerable to adversarial attacks where malicious perturbed inputs lead to erroneous model outputs. The success of Deep Learning and its potential use in many safety-critical applications has motivated research on formal verification of Neural Network models. A possible way to find the minimal optimal perturbation that change the model decision (adversarial attack) is to transform the problem, with the help of binary variables and the classical *bigM* formulation, into a Mixed Integer Program (MIP). In this paper, we propose a global optimization approach to get the optimal perturbation using a dedicated branch-and-bound algorithm. A specific tree search strategy is built based on greedy forward selection algorithms. We show that each subproblem involved at a given node can be evaluated *via* a specific convex optimization problem with box constraints and without binary variables, for which an active-set algorithm is used. Our method is more efficient than the generic MIP solver Gurobi and the state-of-the-art method for MIPs such as MIPverify.

1 Introduction

Evaluating Robustness to adversarial examples is a very active research field in Deep Learning, which aims at finding an adversarial attack $\boldsymbol{a} \in \mathbb{R}^N$ *"–perturbed inputs vector that are very similar to some regular input but for which the output is radically different [Szegedy et al., 2014]–"*, approximating original data vector $\boldsymbol{x} \in \mathbb{R}^N$. This problem can be tackled through the minimization of the least-squares approximation error constrained by the change in the model decision [Carlini and Wagner, 2017].

$$\min_{\boldsymbol{a}} \; d(\boldsymbol{a}-\boldsymbol{x}) \;\; \texttt{s.t.} \begin{cases} \max_{i \neq y}(f_i(\boldsymbol{a})) > f_y(\boldsymbol{x}) \\ \boldsymbol{a} \in X_{valid} \end{cases}$$

where $d(-,-)$ denote a distance metric that measures the perceptual similarity between two input images and $y(\boldsymbol{x})$ is the true label of the input $\boldsymbol{x}$.

To evaluate the robustness of a neural network, several approaches are proposed in the state of the art, that can be grouped according to the formulation of the problem into: feasibility, optimization and reachability problems. A feasibility problem consists in converting the neural network to a feasibility problem for the existence of a counter-example [Katz et al., 2017, Ehlers, 2017, Bunel et al., 2018]. The reachability approach based consists in computing all the reachable set by the neural network and given the input dataset. Then, it checks if this set verify the desired constraints [Xiang et al., 2018, Gehr et al., 2018, Xiang et al., 2018]. Generally, the reachable dataset is computed by approximation. Finally, the optimization approaches consist in computing the maximum perturbation that can be applied to input data without changing the decision of the neural network [Tjeng et al., 2019, Lomuscio and Maganti, 2017]. Our approach lies in the optimization based approaches.

In this paper, we build a dedicated branch-and-bound algorithm for this problem. One key element in our work relies on showing that each node evaluation involved in the search tree can be performed through the optimization of con-

vex problem without binary variables and we build a specific tree-search exploration strategy. Section 2 describes the branch-and-bound algorithm principle, details our implementation strategy and links the node evaluation. Then, numerical results are given in Section 3, where the running time of the proposed implementation is compared to the MIP resolution with the Gurobi solver. A conclusion and directions for future work are finally given in Section 4.

1.1 Formulating Robustness as a Mixed Integer Program (MIP)

In this paper, we are focusing on Feed-Forward Neural Network where each neuron in a layer is connected with all the neurons in the previous layer. To simplify the process, we take a simple example of a network with one hidden layer. The problem can be written as follows:

$$\min_{\boldsymbol{a},\boldsymbol{h},\hat{\boldsymbol{h}}} \; d(\boldsymbol{a}-\boldsymbol{x}) \;\; \texttt{s.t.} \begin{cases} \boldsymbol{h} = \mathbf{W}\boldsymbol{a} + \boldsymbol{\beta}^{\boldsymbol{w}} \\ \hat{\boldsymbol{h}} = max(\boldsymbol{h}, 0) \\ \boldsymbol{s} = \mathbf{V}\hat{\boldsymbol{h}} + \boldsymbol{\beta}^{\boldsymbol{v}} \\ s_i \leqslant s_y \end{cases}$$

Formulating ReLU : Let $\hat{\boldsymbol{h}} = max(\boldsymbol{h}, 0)$ and $-M \leqslant \boldsymbol{h} \leqslant M$. There are three possibilities for the phase of the ReLU. If $\hat{\boldsymbol{h}} = 0$, we say that such a unit is stably inactive. Similarly, if $\boldsymbol{h} = \hat{\boldsymbol{h}}$, we say that such a unit is stably active. Otherwise, the unit is unstable. For unstable units, we introduce an indicator decision variable b witch indicates if the ReLU is active or not:

$$b_i = \begin{cases} 1 & \text{ReLu is active} \\ 0 & \text{ReLu not active} \end{cases}$$

Then, Evaluating Robustness problem is formulating as a Mixed Integer Program:

$$\mathcal{P} : \min_{\boldsymbol{a},\boldsymbol{h},\hat{\boldsymbol{h}},\boldsymbol{b}} \; d(\boldsymbol{a}-\boldsymbol{x}) \;\; \texttt{s.t.} \begin{cases} \boldsymbol{b} \in \{0,1\} \\ \boldsymbol{h} = \mathbf{W}\boldsymbol{a} + \boldsymbol{\beta}^{\boldsymbol{w}} \\ \hat{\boldsymbol{h}} \geqslant \boldsymbol{h} \; ; \hat{\boldsymbol{h}} \geqslant 0 \\ \hat{\boldsymbol{h}} \leqslant M\boldsymbol{b} \\ \hat{\boldsymbol{h}} \leqslant \boldsymbol{h} + M(1-\boldsymbol{b}) \\ \boldsymbol{s} = \mathbf{V}\hat{\boldsymbol{h}} + \boldsymbol{\beta}^{\boldsymbol{v}} \\ s_i \leqslant s_y \end{cases}$$

2 Branch-and-bound exploration

The branch-and-bound principle [Wolsey, 1998] relies on alternating between a *separation* step and an *evaluation* step. The first one consists in dividing a difficult problem into disjoint subproblems which are easier to solve, building a binary search tree. In our case, each separation corresponds to the decision: $b_{k_j} = 1$ or $b_{k_j} = 0$, for some variable b_{k_j} to be defined (see Figure 1). At node i, decisions have been made concerning

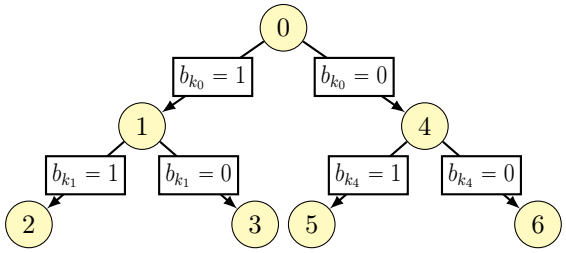


Figure 1: Separation step in a binary search tree: each node corresponding to the optimization problem $\mathcal{P}^{(n)}$, is divided into two children nodes obtaining by constraining one variable to be zero or non-zero.

the nullity of some variables: variables indexed by S^1 are non-zero, those indexed by S^0 are zero (and therefore are removed from the optimization problem) and decisions must still be made concerning the remaining undetermined variables, indexed by $\bar{\mathrm{S}}$.

The evaluation of node i of the search tree is based on the computation of a lower bound on $\mathcal{P}^{(i)}$, let say $z_\ell^{(i)}$ witch is obtained by the continuous Relaxation of the binary Variables:

$$\mathcal{P}^{\mathcal{R}^{(i)}} : \min_{\boldsymbol{a},\boldsymbol{h},\hat{\boldsymbol{h}},\boldsymbol{b}} \; d(\boldsymbol{a}-\boldsymbol{x}) \;\; \texttt{s.t.} \begin{cases} \boldsymbol{b} \in [0,1] \\ \boldsymbol{h} = \mathbf{W}\boldsymbol{a} + \boldsymbol{\beta}^{\boldsymbol{w}} \\ \hat{\boldsymbol{h}} \geqslant \boldsymbol{h} \; ; \hat{\boldsymbol{h}} \geqslant 0 \\ \hat{\boldsymbol{h}} \leqslant M_u\boldsymbol{b} \\ \hat{\boldsymbol{h}} \leqslant \boldsymbol{h} - M_l(1-\boldsymbol{b}) \\ \boldsymbol{s} = \mathbf{V}\hat{\boldsymbol{h}} + \boldsymbol{\beta}^{\boldsymbol{v}} \\ s_i \leqslant s_y \end{cases}$$

The continuous Relaxation $\mathcal{P}^{\mathcal{R}^{(i)}}$ will indicates if node i can contain an optimal solution. More precisely, let z_U denote the best known value of the objective function in $\mathcal{P}$ at a current step of the procedure—which is an upper bound on the optimal value. If $z_\ell^{(i)} \geqslant z_U$, then the node can be pruned. Otherwise, this node is separated into two subproblems according to some new decision: $b_{k_j} = 1$ or $b_{k_j} = 0$? The practical efficiency mostly depends on the tightness of the computed bounds (evaluation step) and on the branching and exploration strategies that are implemented (branching step).

2.1 Evaluation step

Lower bound and convex relaxation. At any node i of the search tree, a lower bound on $\mathcal{P}^{(i)}$ is obtained by solving $\mathcal{P}^{\mathcal{R}^{(i)}}$. Indeed, thanks to the box constraint $\|\boldsymbol{h}\|_\infty \leqslant M$ and convexity property, one has therefore the continuous relaxation $\mathcal{P}^{\mathcal{R}^{(i)}}$ is equivalent to $\mathcal{R}^{(i)}$.

$$\mathcal{P}^{\mathcal{R}^{(i)}} \Longleftrightarrow \mathcal{R}^{(i)}$$

with

$$\mathcal{R}^{(i)} : \min_{\boldsymbol{a},\boldsymbol{h},\hat{\boldsymbol{h}}} \; d(\boldsymbol{a}-\boldsymbol{x}) \;\; \texttt{s.t.} \begin{cases} \boldsymbol{h} = \mathbf{W}\boldsymbol{a} + \boldsymbol{\beta}^{\boldsymbol{w}} \\ \hat{\boldsymbol{h}} \geqslant \boldsymbol{h} \; ; \hat{\boldsymbol{h}} \geqslant 0 \\ \hat{\boldsymbol{h}} \leqslant \frac{\boldsymbol{h}+M}{2} \\ \boldsymbol{s} = \mathbf{V}\hat{\boldsymbol{h}} + \boldsymbol{\beta}^{\boldsymbol{v}} \\ s_i \leqslant s_y \end{cases}$$

Since both problems are defined on the same feasible domain, $\{\hat{\boldsymbol{h}} \geqslant \boldsymbol{h} \; ; \hat{\boldsymbol{h}} \geqslant 0 \; ; \hat{\boldsymbol{h}} \leqslant \frac{\boldsymbol{h}+M}{2}\}$ is a *convex relaxation* of the constraint $\hat{\boldsymbol{h}} = max(\boldsymbol{h}, 0)$ (see. figure 2). Let us remark that this well-known result (*the continuous, convex, relaxation of the ReLu*) is only valid under additional boundedness assumptions on the solution space, such as the box constraints that were introduced in problem P.

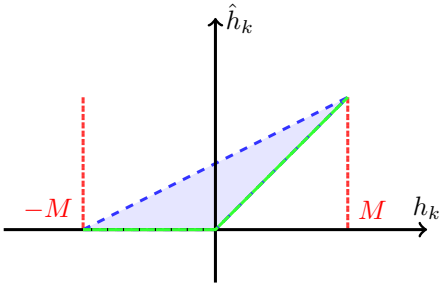


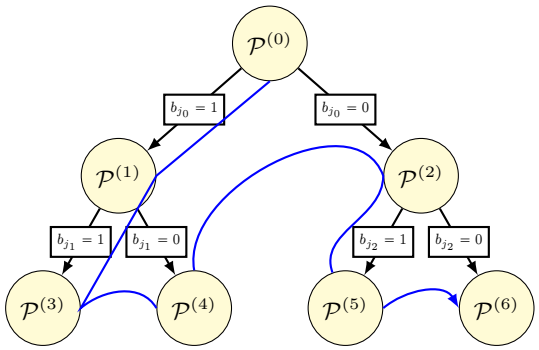
Figure 2: The tightest linear convex relaxation.

2.2 Branching rules and exploration strategy

The branching rule selects the index j of the variable which is used in order to subdivide problem $\mathcal{P}^{(i)}$ (see Figure 1). We propose to exploit the solution of $\mathcal{R}^{(i)}$, by selecting the variable with the highest absolute value in the minimizer:

$$j = \arg\max_{n \in \bar{\mathbb{S}}} \hat{h}_n^{(i)}.$$

This choice aims at selecting first the variables which are more likely to be nonzero at the optimal solution.



We use *depth-first search*, and our branching rule is based on selecting the binary variable, say b_i, with the highest value in the solution of the relaxed problem. We branch *up* first, that is, we first explore the branch corresponding to the decision $b_i = 1$. This strategy, similar to the principle of greedy forward selection algorithms [Mhenni et al., 2020], aims at activating first the most prominent nonzero variables in $x_{n_i} \neq 0$, therefore focusing on quickly finding satisfactory feasible solutions and subsequent upper bounds of good quality. Our proposed implementation is summarized in Algorithm 1, where L contains the queue of subproblems and $\hat{\boldsymbol{x}}$ denotes the best known solution along the exploration. The Branch-and-

0. **Initialization:** $L \leftarrow \{\mathcal{P}^{(0)}\}$; $z_U = +\infty$; $\hat{\boldsymbol{a}} := 0$.
1. **Optimality:** if $L = \varnothing$, then return the optimal solution $\hat{\boldsymbol{a}}$.
2. **Node selection:** choose a subproblem $i \in L$ by depth-first search and remove it from L.
3. **Node evaluation:** compute $z_\ell^{(i)}$.
4. **Pruning:**
   - If $z_\ell^{(i)} \geqslant z_U$, prune node i and return to step 1.
   - If $z_\ell^{(i)} < z_U$:
     - If $\boldsymbol{b}^{\mathcal{R}(i)} \in \{0,1\}$, then $z_U \leftarrow z_\ell^{(i)}$ and $\hat{\boldsymbol{a}} \leftarrow \boldsymbol{a}^{\mathcal{R}(i)}$. Prune node i and return to step 1.
5. **Branching:** subdivide node i by (2.2) and add the two subproblems to L.

Algorithm 1: Branch-and-bound algorithm for $\mathcal{P}$.

Bound algorithm converge to the global minimum in a finite number of steps. In the worst case, an exhaustive search is done (no node could be pruned).

						B&B$_{\text{HOME}}$		
MIP$_{\text{Gurobi}}$			MIP$_{\text{Verify}}$					
T	Nds	F	T	Nds	F	T	Nds	F
35	1800	4	27	-	4	7	1200	4

Table 1: Computational efficiency for robustness problems averaged over 100 instances. Computing time (T) number of explored nodes (Nds) and number of instances that did not terminate in 1 000 s (F).

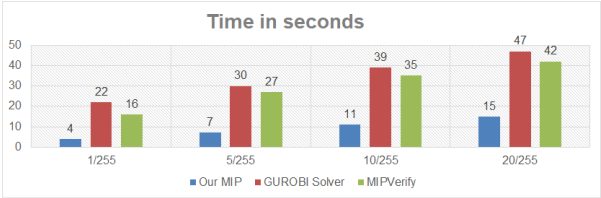


Figure 3: Computational Time for robustness problems averaged over 100 instances according to the intensity of the attack (maximum disturbance allowed in infinite norm)

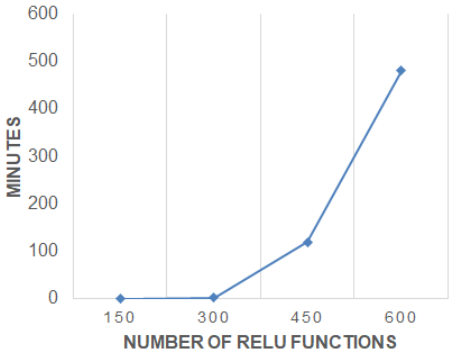


Figure 4: Computing time (Minutes) for robustness problems as a function of the number of ReLU in the model, average over 10 instances.

3 Performance Evaluation

We now evaluate the computational performance of our branch-and-bound strategy using Cplex MIP solver. We name this algorithm B&B$_{\text{HOME}}$. Computing times are compared with the Gurobi Mixed quadratic programming solver (named MIP$_{\text{Gurobi}}$) [1] and MIP$_{\text{Verify}}$[2]. All methods are run on a UNIX machine equipped with 32 Go RAM and with four Intel Core i7 central processing units clocked at 2.6 GHz. For each instance, the running time is limited to 1 000 s. Note that we only focus here on the computational efficiency of algorithms which are guaranteed to find the global optimum $\mathcal{P}$; due to the lack of space we do not compare the obtained solutions to that of standard, suboptimal methods.

In order to evaluate the behavior of our method regarding the complexity of the model, we have varied the number of hidden layers from 1 to 4 (i.e. from 150 to 600 activation ReLU functions). Results averaged over 100 instances of each problem are given in Table 1 and Figure 3. B&B$_{\text{HOME}}$ is much faster then MIP$_{\text{Gurobi}}$ and MIP$_{\text{Verify}}$ revealing the efficiency of our strategy. Most of all, we observe that the most important improvement achieved by B&B$_{\text{HOME}}$ is due to the efficiency of our continuous relaxation: the computing time per node with the proposed formulation is at least 4 times smaller than that of MIP$_{\text{Gurobi}}$. Even with this improvement, the results in Figure 4 show the limit of our approach especially when the number of ReLU in the model increases. We can see that the complexity increases exponentially and becomes unfeasible in a reasonable time for complex models.

4 Conclusion

In this paper, we proposed a branch-and-bound algorithm which is able to find exactly the optimal attack. We have shown that such problems could benefit from dedicated resolution methods. Our algorithm outperforms Gurobi, which is considered as one of the best MIP solvers. The proposed exploration strategy exploits the sparsity of the searched solution, by preferring the activation of nonzero variables in the decision tree, conjugated with depth-first search. Moreover, evaluation of each node by continuous relaxation was recast as a specific convex problem without binary variables. Following the same principle, further works may include the building of more efficient relaxations, involving Lagrangian relaxation and specific cutting planes [Wolsey, 1998] for may also improve the quality of lower bounds computed at each node. **But unfortunately even with this improvement, the results show the limit of our approach especially when the number of ReLU in the model increases and we are still far from using large indistrual models**.

[1] https://www.gurobi.com/

[2] https://vtjeng.com/MIPVerify.jl/latest/

REFERENCES

[Bunel et al., 2018] Bunel, R., Turkaslan, I., Torr, P. H., Kohli, P., and Kumar, M. P. (2018). A unified view of piecewise linear neural network verification. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 4795–4804, USA. Curran Associates Inc.

[Carlini and Wagner, 2017] Carlini, N. and Wagner, D. (2017). Towards Evaluating the Robustness of Neural Networks. Number: arXiv:1608.04644 arXiv:1608.04644 [cs].

[Ehlers, 2017] Ehlers, R. (2017). Formal verification of piece-wise linear feed-forward neural networks. *CoRR*, abs/1705.01320.

[Gehr et al., 2018] Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., and Vechev, M. (2018). Ai 2: Safety and robustness certification of neural networks with abstract interpretation. In *Security and Privacy (SP), 2018 IEEE Symposium on*.

[Katz et al., 2017] Katz, G., Barrett, C. W., Dill, D. L., Julian, K., and Kochenderfer, M. J. (2017). Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, pages 97–117.

[Lomuscio and Maganti, 2017] Lomuscio, A. and Maganti, L. (2017). An approach to reachability analysis for feed-forward relu neural networks. *CoRR*, abs/1706.07351.

[Mhenni et al., 2020] Mhenni, R. B., Bourguignon, S., and Idier, J. (2020). A greedy sparse approximation algorithm based on l1-norm selection rules. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5390–5394.

[Szegedy et al., 2014] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2014). Intriguing properties of neural networks. Number: arXiv:1312.6199 arXiv:1312.6199 [cs].

[Tjeng et al., 2019] Tjeng, V., Xiao, K. Y., and Tedrake, R. (2019). Evaluating robustness of neural networks with mixed integer programming. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

[Wolsey, 1998] Wolsey, L. A. (1998). *Integer Programming*. Wiley, New York, NY, USA.

[Xiang et al., 2018] Xiang, W., Tran, H., and Johnson, T. T. (2018). Output reachable set estimation and verification for multilayer neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11):5777–5783.

[Xiang et al., 2018] Xiang, W., Tran, H.-D., and Johnson, T. T. (2018). Reachable set computation and safety verification for neural networks with relu activations. *In Submission*.