Linear Program Powered Attack *

Ismaila SECK[1], Gaëlle LOOSLI[2], and Stéphane CANU[3]

[1]Normandie Univ, INSA Rouen, UNIROUEN, UNIHAVRE, LITIS.
Saint-Etienne-du-Rouvray, France, UCA
[2]PobRun, UCA
[3]Normandie Univ, INSA Rouen, UNIROUEN, UNIHAVRE, LITIS.
Saint-Etienne-du-Rouvray, France

Abstract

Finding the exact robust test error is a good way to assess neural networks, but it is a difficult task even on small networks and datasets like MNIST. On the one hand, comprehensive methods such as Mixed Integer Program (MIP) give exact robust test accuracy but are time-consuming. On the other hand, many popular attacks are fast but tend to perform poorly against robust networks and only provide a bound on the robust test error. The purpose of this paper is to present a fast and novel attack method called LiPPA, that gives better bounds than previous attacks. This method exploits the algebraic properties of networks with piecewise linear activation functions to partition the input space in such a way that for each subset of that partition, finding the local optimal adversarial example is done by solving a linear program. Switching from one subset to another is done using classic gradient-based attack tools. The empirical evidence reported on MNIST illustrates the interest of LiPPA over state-of-the-art fast attacks.

Mots-clef: Adversarial examples, adversarial attacks, adversarial robustness verification.

1 Introduction

Neural networks are very brittle in the sense that their high accuracy can rapidly drop to nearly 0 when maliciously crafted noises are added to the input images, which become adversarial examples [GSS15]. This kind of behavior of neural networks has a deterrent effect, making the use of neural network in safety critical environments, such as in self-driving cars or in medical imaging, a topic of controversy. Several defense mechanisms have been deployed from [PMW+16] which proved later to be inefficient when attacks get stronger. Complete verification methods such as Reluplex [KBD+17] and MIP (Mixed Integer Programming) formulation of the search of adversarial examples [TXT18] seem to be the ultimate approach able to give the exact robust test error [BIL+16]. However, those complete verifiers can take hours on some examples without being able neither to find an adversarial example nor to prove that there is no adversarial example within a given neighborhood of the original example. This leads sometimes to models for which there is a big gap between the lower and the upper bound of the robust test error.

To reduce the gap between those bounds, more powerful attacks, able to quickly find adversarial examples when they exist, are needed. Projected Gradient Descent [MMS+19], Carlini & Wagner attack [CW17] and the randomized gradient-free attack [CH18, CRH20] are among the strongest attacks to date but sometimes miss adversarial examples. The fastest implementation of MIP for adversarial robustness verification is MIPVerify [TXT18].

Our contributions

- We make the relationship between linear regions of the input space and binary combinations of the MIP formulation of the robustness verification problem more obvious, for a given model, via the introduction of a function **B** associating each input example with a binary combination,
- we experimentally prove that the number of linear regions around the original example is smaller than the exponential number of combinations allowed by the MIP formulation,
- we propose a white-box attack that exploits, given a model, the relationship between linear regions and binary values used in the MIP formulation. In

*Accepted at IJCNN 2021

a few seconds, our method is able to find adversarial examples that the MIP fails to detect given a much longer time.

2 Related work and background

Having complete access to a trained model (white-box setting) makes it easier to attack. This is because every piece of information about the model that is attacked can potentially be exploited to find vulnerabilities, thanks to gradient-based methods. The exploitation of that information also allows to accelerate the MIP verification, which is gradient-free. We present the related works through the point of view of information exploitation, making the difference between methods exploiting or not the gradient information.

2.1 Related work

Gradient-based attack methods such as Fast Gradient Sign Method [GSS15], Projected Gradient Descent [MMS+19] use accessible weights to construct adversarial examples, which are good against many neural networks, even with defense mechanisms [ACW18], so much so that PGD is commonly seen as a strong attack, if not the strongest First Order attack, when searching for adversarial examples with ℓ_∞ distortion. However, PGD can fail: some defense mechanisms manage to have the gradient point toward directions that are not harmful to the neural network and thus prevent gradient-based methods from increasing the error rate, giving thereby a *false sense of robustness* [ACW18], and when the minimal adversarial distortion is high, PGD fails to find adversarial examples, and it is even more patent when the allowed distortion is close to the minimal adversarial distortion [TXT18].

The search for adversarial examples can also be done using formal verification methods [BTT+18] that do not use the gradient. Especially when the neural networks can be represented as a piecewise linear function (when the ReLU activation is used), Satisfiability Modulo Theory (SAT) solvers can be used as in [KBD+17] as well as Mixed Integer Programming [TXT18].

Let us focus on the MIP formulation. Assuming that f represents a piece-wise linear function, let us say a neural network with ReLU activation, and $\mathbf{x}$ an example of the input space $\mathcal{X} = [0,1]^d$, belonging to the class y, a generic formulation of the search for adversarial examples is:

$$\begin{cases} \min\limits_{\mathbf{x}'} & \|\mathbf{x}-\mathbf{x}'\|_p \\ \text{s.t.} & \underset{i=1,\dots,c}{\text{argmax}} f_i(\mathbf{x}') \neq y, \\ & \mathbf{x}' \in [0,1]^d \ , \end{cases} \tag{1}$$

where p is usually 2 or ∞, f_i represents the $i-$th component of the output of the neural network and c the number of classes. To solve this optimization problem, the constraint $\underset{i=1,\dots,c}{\text{argmax}} f_i(\mathbf{x}') \neq y$ is linearized and each neuron is associated with a binary value, indicating if the neuron is active or not (Section 2.2). We generally want to check the existence of adversarial examples in a neighborhood of a clean example. In a restricted neighborhood, computing the lower and upper bounds on the pre-ReLU activation can show that some neurons are always active and some others are always inactive (whatever valid distortions are applied to the example). Those are called stable ReLUs [TXT18] and they do not require to be paired with a binary. Exploiting this information gives, to the best of our knowledge, the state-of-the-art verification method using an MIP formulation, as it reduces the number of binary values allowing the MIP to operate faster.

A valuable strength of the MIP is its ability to certify robustness, which is not the case for attacks. However, its main drawback is that it is time consuming, even when we simplify the problem to computing the robust test error [BIL+16]. It is, at least partially, due to the fact that the MIP not only gives an upper bound of the minimal adversarial distortion, the distance under which no modification can cause misclassification, but also gives a lower bound. And if enough time is given to the MIP, the upper bound and the lower bound of the minimal adversarial will converge to the same value, the minimal adversarial distortion. From the MIP, some efficient attacks can be derived by dropping the part of the MIP responsible for computing the lower bound. It is the case of the attacks [CH18, CRH20] which are gradient-free attacks that explore the neighborhood of clean examples and are able to find adversarial examples that are at minimal adversarial distortion distance. They do so by exploiting the fact that ReLU networks are piecewise affine functions [ABMM18] and solving the optimization 1 over each of the linear regions separately. Those regions are regions where the neural network can be represented by an affine function. This boils down to solving a linear program as shown in Section 2.2.

2.2 Background

Here we show the linear regions used in [CH18, CRH20] form equivalence classes in the input space as well as how they are related to the binary values of the MIP formulation, and an example of MIP formulation.

For simplicity, let us consider a multilayer perceptron with one hidden layer with e neurons, the classification function of that MLP can be written:

$$\begin{aligned} \hat{\mathbf{z}} &= W\mathbf{x} + \beta, \\ \mathbf{z} &= \max(\hat{\mathbf{z}}, 0), \\ f(\mathbf{x}) = \quad \mathbf{s} &= V\mathbf{z} + \alpha. \end{aligned} \tag{2}$$

We linearize the ReLU constraint, $\mathbf{z} = \max(\hat{\mathbf{z}}, 0)$ using a big-M formulation [WN99] as:

$$\mathbf{z}_i \geq 0, i = 1, \dots, e \tag{3}$$
$$\mathbf{z}_i \leq M_r\ \mathbf{b}_i, i = 1, \dots, e \tag{4}$$
$$\mathbf{z}_i \leq \hat{\mathbf{z}}_i + M_r(1 - \mathbf{b}_i), i = 1, \dots, e \tag{5}$$
$$\mathbf{z}_i \geq \hat{\mathbf{z}}_i, i = 1, \dots, e \tag{6}$$

with M_r such that $M_r \geq \max_{\mathbf{x}} |\hat{\mathbf{z}}_i|, \forall i = 1, \dots, e, \forall \mathbf{x} \in \mathcal{X}$ and $\forall i = 1, \dots, e, \mathbf{b}_i \in 0, 1$. Those constraints are such that for each $i = 1, \dots, e$:

- $\hat{\mathbf{z}}_i < 0 \Rightarrow \mathbf{b}_i = 0$, and then (3) and (4) imply $\mathbf{z}_i = 0$,
- $\hat{\mathbf{z}}_i > 0 \Rightarrow \mathbf{b}_i = 1$, and then (5) and (6) imply $\mathbf{z}_i = \hat{\mathbf{z}}_i$.

The value $\hat{\mathbf{z}}_i = 0$ can be associated with either 0 or 1, but in the following we will choose to associate it with 1. Let us now write an MIP formulation:

$$\left\{ \begin{array}{lll} \min & \|\mathbf{x} - \mathbf{x}'\|_p & \\ \text{s.t.} & \hat{\mathbf{z}} = W\mathbf{x}' + \beta, & \\ & \mathbf{z}_i \geq 0, & i = 1, \dots, e \\ & \mathbf{z}_i \leq M_r\mathbf{b}_i, & i = 1, \dots, e, \\ & \mathbf{z}_i \leq \hat{\mathbf{z}}_i + M_r(1 - \mathbf{b}_i), & i = 1, \dots, e \\ & \mathbf{z}_i \geq \hat{\mathbf{z}}_i, & i = 1, \dots, e \\ & \mathbf{s} = V\mathbf{z} + \alpha, & \\ & \mathbf{s}_t > \mathbf{s}_y. & \\ & \mathbf{x}' \in [0, 1]^d, & \\ & \mathbf{b} \in \{0, 1\}^e, & \\ & \mathbf{z} \in \mathbb{R}^e, & \\ & \hat{\mathbf{z}} \in \mathbb{R}^e, & \\ & \mathbf{s} \in \mathbb{R}^c, & \end{array} \right. \tag{7}$$

with big enough M_r. To simplify the formulation, the class t is targeted using the constraint $\mathbf{s}_t > \mathbf{s}_y$.

We derive a powerful attack that can potentially find the same solution as the MIP, but faster since it does not go through branching and bounding included in MIP solvers. The idea is to look for adversarial examples over linear regions and find a mechanism to select linear regions leading to good adversarial examples. For now, let us tackle the optimization over a linear region by using our function $\mathbf{B}$ introduced below.

Linear regions as equivalence classes We define function $\mathbf{B} : [0, 1]^d = \mathcal{X} \to \mathbf{B}(\mathcal{X}) \subseteq \{0, 1\}^e$ which associates to each input $\mathbf{x} \in \mathcal{X}$, $\mathbf{B} : \mathbf{B}(\mathbf{x} \in \mathcal{X}) = \mathbf{b} \in \{0, 1\}^e$, such that for $i = 1, \dots, e$:

$$\mathbf{b}_i = \left\{ \begin{array}{ll} 1, & \hat{\mathbf{z}}_i \geq 0, \\ 0, & \text{otherwise.} \end{array} \right. \tag{8}$$

This function $\mathbf{B}$ is surjective:

- for each $\mathbf{x} \in \mathcal{X}$, there exists a unique $\mathbf{b} \in \mathbf{B}(\mathcal{X})$ such that: $\mathbf{B}(\mathbf{x}) = \mathbf{b}$,
- for each $\mathbf{b} \in \mathbf{B}(\mathcal{X})$, there exists at least one $\mathbf{x}$ such that: $\mathbf{b} = \mathbf{B}(\mathbf{x})$.

Then, we can define the equivalence relation:

$$\mathcal{R} : \mathcal{R}(\mathbf{x}_\phi, \mathbf{x}_\omega) \iff \mathbf{B}(\mathbf{x}_\phi) = \mathbf{B}(\mathbf{x}_\omega). \tag{9}$$

In other words, $\mathcal{R}(\mathbf{x}_\phi, \mathbf{x}_\omega)$ means $\mathbf{x}_\phi$ and $\mathbf{x}_\omega$ have the same activations when going through f. It is used to create equivalence classes:

$$[\mathbf{x}] = \{\mathbf{x}' \in \mathcal{X} : \mathbf{B}(\mathbf{x}') = \mathbf{B}(\mathbf{x})\}. \tag{10}$$

And we know that the set of all equivalence classes forms a partition of the input space $\mathcal{X}$. In each equivalence class $[\mathbf{x}]$, the relationship between an input $\mathbf{x}' \in [\mathbf{x}]$ and its output $f(\mathbf{x}' \in [\mathbf{x}])$ is linear, i.e. there exist a matrix Γ and a vector γ such that: $f(\mathbf{x}' \in [\mathbf{x}]) = \Gamma\mathbf{x}' + \gamma$. Each equivalence class is a *linear region*. Let us see how those equivalence classes/linear regions and the function $\mathbf{B}$ can be integrated to the MIP formulation.

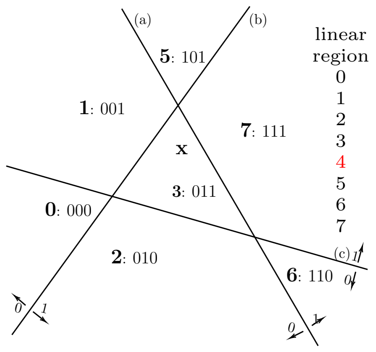
MIP and B together Once we find a linear region $[\mathbf{x}_\phi]$, we can look for adversarial examples in that region of the input space with the following optimization:

$$\left\{ \begin{array}{ll} \min & \|\mathbf{x} - \mathbf{x}'\|_p \\ \text{s.t.} & s = \Gamma_\phi\mathbf{x}' + \gamma_\phi, \\ & \mathbf{x}' \in [\mathbf{x}_\phi], \\ & \mathbf{s}_t > \mathbf{s}_y. \\ & \mathbf{s} \in \mathbb{R}^c, \end{array} \right. \tag{11}$$

with Γ_ϕ and γ_ϕ such that $f(\mathbf{x}' \in [\mathbf{x}_\phi]) = \Gamma_\phi\mathbf{x}' + \gamma_\phi$. Note that the optimization problem (11) does not have integers. Moreover, we do not have to determine the matrix Γ and the vector γ. In practice, we solve (11) by fixing the binary values in the MIP (7) to $\mathbf{B}([\mathbf{x}_\phi])$ to obtain the optimization problem $LP(\mathbf{B}(\mathbf{x}_\phi), t)$:

$$\left\{ \begin{array}{lll} \min & \|\mathbf{x} - \mathbf{x}'\| & \\ \text{s.t.} & \hat{\mathbf{z}} = W\mathbf{x}' + \beta, & \\ & \mathbf{z}_i \geq 0, & i = 1, \dots, e \\ & \mathbf{z}_i \leq M_r\mathbf{b}_i, & i = 1, \dots, e, \\ & \mathbf{z}_i \leq \hat{\mathbf{z}}_i + M_r(1 - \mathbf{b}_i), & i = 1, \dots, e \\ & \mathbf{z}_i \geq \hat{\mathbf{z}}_i, & i = 1, \dots, e \\ & \mathbf{s} = V\mathbf{z} + \alpha, & \\ & \mathbf{s}_t > \mathbf{s}_y, & \\ & \mathbf{x}' \in [0, 1]^d, & \\ & \underline{\mathbf{b} = \mathbf{B}(\mathbf{x}_\phi)}, & \\ & \mathbf{z} \in \mathbb{R}^e, & \\ & \hat{\mathbf{z}} \in \mathbb{R}^e, & \\ & \mathbf{s} \in \mathbb{R}^c, & \end{array} \right. \tag{12}$$

The optimization problems 11 and 12 are strictly equivalent since fixing $\mathbf{b}$ to $\mathbf{B}(\mathbf{x}_\phi)$ means that we are constraining the MIP to optimize only on the examples of the input space that have the same activations as $\mathbf{x}_\phi$ which is by construction $[\mathbf{x}_\phi] = \{\mathbf{x} \in \mathcal{X} : \mathbf{B}(\mathbf{x}) = \mathbf{B}(\mathbf{x}_\phi)\}$ as stated in 10. It is now obvious that there is a relationship between the binary values of the MIP and binary values in $\mathbf{B}(\mathcal{X})$. And as illustrated Figure 1, $\mathbf{B}(\mathcal{X})$ is included in the set of all possible combination, $\{0,1\}^e$ of the naive MIP 7. In that illustration, we also have an intuition about the fact that the real number of linear regions around a point is far less than the exponential number found by raising 2^e.



linear region	a	b	c	$a \leq b + c$
0	0	0	0	True
1	0	0	1	True
2	0	1	0	True
3	0	1	1	True
4	1	0	0	False
5	1	0	1	True
6	1	1	0	True
7	1	1	1	True

Figure 1: Correspondence between binary values and equivalence classes. $\mathbf{x}$ represents a point in a 2D input space, $(a), (b)$ and (c) are neurons, diving the plane into active/inactive parts. The 3-digit code abc is the binary encoding associated to each linear region of the input space: region 3 corresponds to the linear region where the neuron (a) has negative response, and the neuron b and c have positive responses: 3 is encoded as 011. The linear region number 4, encoded 100, does not have any antecedent since there can be no point in this setting verifying that. The constraint $a \leq b + c$ can represent this fact, as shown in the rightmost column of the table.

Considering the relations between linear regions, binary values given by $\mathbf{B}$ and the MIP formulation, a straightforward and unsophisticated idea is to enumerate the linear regions and solve a linear program over each to find adversarial examples in a neighborhood of a clean input. Clearly, this has a prohibitive computation time and we need an efficient method to explore only promising linear regions.

3 Faster robustness verification

In this section, we show how some fragments of the information given by using $\mathbf{B}$ have been used to speed up the MIP and that there is still unexploited information in 3.1. Then, in an attempt to further exploit the available information, we introduce our attack in 3.2.

3.1 Accelerating the MIP using B

Now we show the potential acceleration that can result from the exploitation of the information delivered by

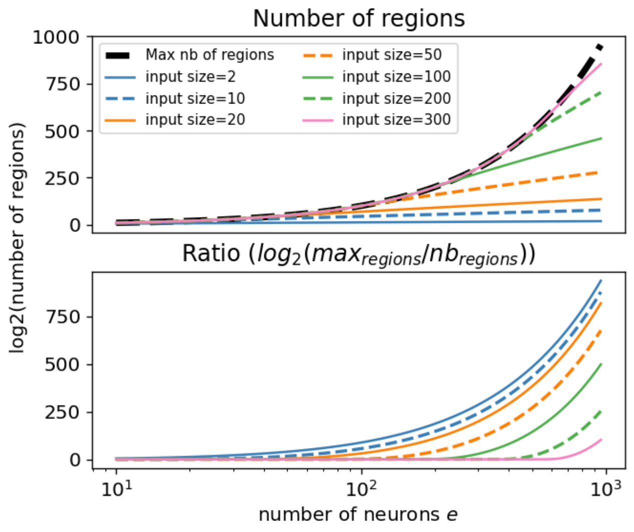


Figure 2: Number of binary combinations that can be modeled by the binary activations associated with a hidden layer. The black line is the maximum number of linear regions that this hidden layer can represent. Bottom: the same data represented as a ration between the black line and each of the others.

the function $\mathbf{B}$. We first show that the naive MIP exploits very little information in 3.1. Then we show how some information was integrated to the naive MIP by [TXT18] using stable ReLUs and how the stable ReLUs are linked to the function $\mathbf{B}$ in 3.1. Next, we show that there is still unexploited information beyond the stable ReLUs that could be harnessed to make the MIP more efficient.

Naive MIP is information oblivious Using $\mathbf{b} \in \{0,1\}^e$ in eq. 7 implicitly makes the assumption that the binary values in $\mathbf{b}$ are independent. As shown in Figure 1, this assumption does not hold as there are interactions between the different coordinates of the binary vector $\mathbf{b}$. In this simple example with only three neurons, when $b = 0$ and $c = 0$, then the only value that a can take is 0, fact we modeled with the constraint $a \leq b + c$. Then the set of binary combinations corresponding to linear regions is $\{0,1\}^3 \backslash (1,0,0)$, which makes 7: less than the cardinal of $\{0,1\}^3$, 8. The combinatorial dimension of the MIP can be drastically reduced if we manage to include such constraints.

The number of linear regions that can be represented by 20 neurons with a 2 dimensional input is 211, the ratio $\frac{211}{2^{20}} \approx 0.0002$. With a two-dimensional input, the ratio between the number of linear regions than can be represented by 30 neurons and 2^{30} is of the order of 10^{-7}, and this ratio gets smaller when the number of neurons is bigger, as we can see on Figure 2. Using the view of the input space as equivalency class/linear regions, and exploiting this information by restraining the branch and bound to the set of reachable binary combinations by adding some constraints, could lead to a speed-up of several orders of magnitude and make the problem of complete verification of neural networks and the search of mean minimal adversarial distortion less prohibitively time-consuming. However, it means finding and modeling those constraints, which is out of the scope of this paper. Fortunately, there is a simpler way to find useful information.

Function B and stable ReLUs Let $\mathbf{x}$ be an original example and $\mathcal{N}_{\mathbf{x}}$ be the neighborhood of $\mathbf{x}$ in which we are looking for adversarial examples. Let $\mathbf{B}(\mathcal{N}_{\mathbf{x}})$ be the set of all binary combinations corresponding to the linear regions covering $\mathcal{N}_{\mathbf{x}}$. Having binary values $\mathbf{b} \in \mathbf{B}(\mathcal{N}_{\mathbf{x}})$ instead of $\mathbf{b} \in \{0,1\}^e$ in eq. 7 would drastically decrease the branch and bound space and be much faster. [TXT18] already showed a correlation between the number of stable ReLUs and the solving time. The more stable ReLUs there are, the more the branch and bound tree is reduced, and the less time is spent to find the optimal solution and prove the optimality by several orders of magnitude. However, even doing so, there may remain a number of binary values that do not correspond to any linear region of the input space. The MIP will presumably spend time branching and bounding on binary combinations that do not have antecedents with respect to the function $\mathbf{B}$ and that will not produce any solution. The stable ReLUs are related to the set of binary combinations that can be obtained from the input space. The ReLUs corresponding to a neuron e_0 is stably equal to 0 (respectively 1) in $\mathcal{N}_{\mathbf{x}}$ if and only if for all $\mathbf{b} \in \mathbf{B}(\mathcal{N}_{\mathbf{x}})$, $\mathbf{b}_{e_0} = 0$ (respectively $\mathbf{b}_{e_0} = 1$). This shows that [TXT18] uses only a part of the information in $\mathbf{B}(\mathcal{N}_{\mathbf{x}})$.

Function B, beyond stable ReLUs As shown in Figure 1, there is information contained in $\mathbf{B}(\mathcal{N}_{\mathbf{x}})$ that is not captured by stable ReLUs. That is the case for any relationship existing between activations of neurons belonging to the same layer or to different layers. For example, in $\mathcal{N}_{\mathbf{x}}$, and in the one hidden layer illustration given earlier, we could have two neurons i_0 and i_1 such that $\hat{\mathbf{z}}_{i_0} < \hat{\mathbf{z}}_{i_1}$. This relationship is such that $\hat{\mathbf{z}}_{i_0} > 0 \Rightarrow \hat{\mathbf{z}}_{i_1} > 0$ it translates for the binaries associated with them into $\mathbf{b}_{i_0} = 1 \Rightarrow \mathbf{b}_{i_1} = 1$. The other way around, $\hat{\mathbf{z}}_{i_1} < 0 \Rightarrow \hat{\mathbf{z}}_{i_0} < 0$ which leads to $\mathbf{b}_{i_1} = 0 \Rightarrow \mathbf{b}_{i_1} = 0$. Those kinds of relationships may be hard to find if they are to be added to the MIP formulation. There may exist more complicated relationships between the activations that may prove to be difficult to determine a model to suit the MIP formulation. As a consequence, the number of linear regions is smaller than the theoretical exponential number of combinations even when taking into account the number of stable ReLUs.

We used the example of a single hidden layer, but the same reasoning can be applied to more complex networks including convolutional networks and other classical architectures for which MIP formulation can be used.

3.2 Linear Program Powered Attack

We have seen in Section 2 that there are two ways of exploiting the information when given full knowledge of the model parameters, a gradient-based way and a gradient-free way. Our method is going to use both ways. We use the information contained in the weights to see the input space as a partition such that on each subset of that partition, for a given target class, finding the best adversarial example is done by solving a linear program, as shown in Algorithm 1.

```
Data: $x$, $f$, params$_{stop}$, params$_{gs}$
Result: $\hat{x}$
[$\hat{x}$, $b$] ← initialisation($x$,$f$) [see 3.3] ;
while checkStopCriteria($x$,$\hat{x}$,params$_{stop}$) [see 3.4] do
    [$\hat{x}$, $b$] ← GSStep($\hat{x}$,$f$,$b$,params$_{gs}$) [see 3.2] ;
    [$\hat{x}$] ← LPStep($b$,$f$) [see 3.1]
end
```

Algorithm 1: LiPPA

Definition 3.1 (LPStep) *The exploitation step consists in solving eq.12 in the current linear region.*

Definition 3.2 (GSStep) *The exploration step can be achieved using various iterative gradient methods, initialized with the solution of the LPStep. Iterations are stopped as soon as the current point belongs to a new linear region or after a budget limit.*

Definition 3.3 (Initialisation) *The initialisation is called LiPPA$_0$: it is an LPStep inside the clean exemple linear region.*

Definition 3.4 (Stopping criteria) *There can be various stopping criteria:*

- *the current distance is below the targeted ϵ,*
- *time budget,*
- *step number budget,*
- *GSStep can not find another linear region.*

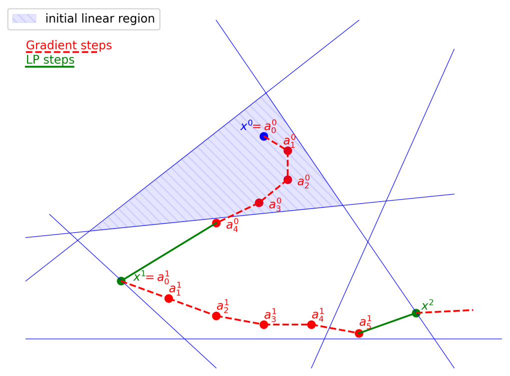


Figure 3: Illustration of the behavior of LiPPA.

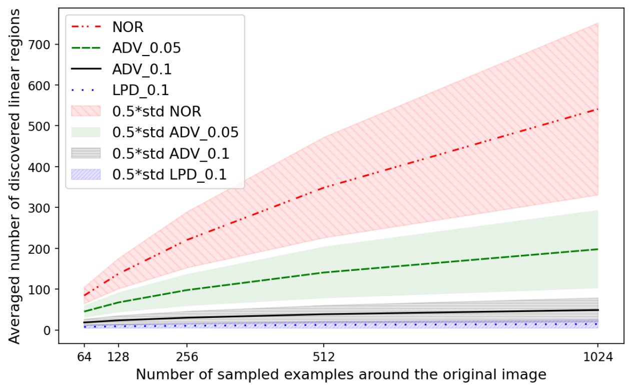


Figure 4: Evolution of the number of found linear regions averaged over the first 100 images of the MNIST test set with respect to the number of random samples around the original example, using MLP-B.

Our algorithm is similar to the attack of [CH18, CRH20], but there are some differences between our approach and theirs:

- we explicitly show that linear regions are equivalency classes,
- we explicitly show the link between the linear regions and the binary combinations in the MIP formulation (cf Figure 1),
- they compute, for each linear region $[\mathbf{x}]$ the matrix Γ and the offset γ such that $f(\mathbf{x}' \in [\mathbf{x}]) = \Gamma\mathbf{x}' + \gamma$, while this computation is not required in our approach,
- finally, they opted for a completely gradient-free approach selecting linear regions using a randomness, when our approach is hybrid since we use the gradient to select the linear regions.

4 Experimental results

Dataset and Models We used the MNIST dataset. Models are from [SYZ+19] and [WK18]. Architecture MLP-B is a multilayer perceptron with two 100-neurons hidden layers, MLP-C with one 1024-neurons hidden layer. CNN architecture has a first 16-filters convolutional, a second 32-filters layer, of 4×4 pixels each with a stride of two, then a 100-neurons hidden layer and the output layer. Model names are prefixed to show how they were trained: NOR for normal training with the cross-entropy loss function; ADV for adversarial training with examples generated by PGD; LP$_D$ are trained using [WK18]; MMR are trained using [CAH19]; MMR-ADV used both adversarial training with PGD and MMR. [1]

Threat model We test the robustness for ℓ_∞, although LiPPA could also be used for ℓ_1 and ℓ_2 without major changes. It is worth noting that for ℓ_2 the obtained optimization problem is a quadratic program. In LiPPA steps, the neurons' states (active or not) are known and are directly encoded, avoiding the use of ReLU big-M constraints. Let us denote i_- the indices i such that $\mathbf{b}_i = 0$ meaning the pre-ReLU activations are negative and i^+ the indices such that $\mathbf{b}_i = 1$ meaning the pre-ReLU activations are non-negative. The constraints ReLU constraints are then $\hat{\mathbf{z}}_i < 0, \mathbf{z}_i = 0, i \in i_-$ and $\hat{\mathbf{z}}_i \geq 0, \mathbf{z}_i = \hat{\mathbf{z}}_i, i \in i^+$.

4.1 Number of linear regions

Experimental setup We gave some hints allowing to understand that the effective number of linear regions can be much smaller than the exponential number of possible binary combinations even when taking into account the existence of stable ReLUs. Here we run an experiment to back that claim. First, for the original image, we generate an increasing number of points within a ball of infinite norm of 0.05 [2]. We count the number of uniquely discovered linear regions for MLP-B for each training setting. We average results over the first 100 images of MNIST test set. We finally sample 6144 random points around the clean example and this time count the number of unstable ReLUs found.

[1] All models are available online, at `https://github.com/Hadisalman/robust-verify-benchmark` for architecture MLP-B, at `https://github.com/max-andr/provable-robustness-max-linear-regions` for MLP-C and CNN.Those model can also be found at `https://anonymous.4open.science/r/fda53f2d-ee4f-40d6-a853-f9056513e82a/`, as well as our code.

[2] After the sampling a deformation random vector, some tricks are applied to fill the box around the input image better and maximize the chance of discovering more linear regions

Training	NOR	ADV-0.05	ADV-0.1	LP_D-0.1
n_U	24.54	14.89	7.94	4.32
2^{n_U}	$2.44\ 10^7$	$3.04\ 10^4$	246	20
linear regions	541	198	49	14

Table 1: Comparison between the number of linear found and the number of linear regions found across different training methods for the MLP-B architecture. The value next to the training method represent the value ℓ_∞ the model was trained to be robust to.

Results Figure 4 shows that, for the same architecture, the number of found linear regions is lower for robustified models. Linking with robustness measures, we can say that the more robust the network, the less linear regions are found. In table 1 shows the average number unstable ReLUs, n_U, the number of all possible binary combinations to n_U unstable ReLUs, and the number of linear regions found. We can see some substantial differences: for ADV-0.05, we have on average nearly 198 linear regions while the average number of unstable ReLUs found is 14.89, meaning on average there are $2^{14.89} > 3 \times 10^4$ possible binary combinations. We also observe that the number of found linear regions is roughly 4 times less for ADV-0.01. Likewise, the number of linear regions found for the LP_D is 3.5 times smaller than when trained with adversarial training for the same allowed perturbation.

Proving that the number of linear regions can be much smaller than the exponential number of all possible combinations in a neighborhood of an image is an encouraging first step. This step combined with a good mechanism to select the most promising linear regions/combinations are the heart of our approach.

4.2 LiPPA performance

Set-up details We used Basic Iterative Method [KGB17] in the GSStep, with $\epsilon_{BIM} = 0.5/256$, and a budget stopping criterion of 50 iterations. ϵ represents the ℓ_∞ perturbation for which we compute the lower bounds and upper bounds on the error rate. It is also used to train for robustness when the model is trained to be robust to adversarial attacks. The test error is the misclassification rate on the clean examples on the whole MNIST test set. The lower bound represents the best lower bound on the robust classification error found using PGD and MIP with a time limit of 3600s when the results are from [SYZ+19], and using PGD, the gradient-free attacks of [CH18, CRH20] and an MIP with a time limit of 120s when from [CAH19].

Results In Table 2, note that when the lower bound is equal to the upper bound, MIP has converged before the timeout for each example (has found an adversarial or has proven there is none). Overwise, it means that there are some examples around which no adversarial example was found, but at the same time there was no proof of robustness around those examples, the time limit was reached by the MIP. Our attack does better than the previously reported lower bounds on NOR-MLP-C, ADV-MLP-C, and ADV-CNN and is not far from the others. Even though the reported results for NOR-MLP-B and ADV-MLP-B are on the whole test while the results of LiPPA are only on the first 1000 samples, it seems that LiPPA outperforms the attacks used to compute that lower bound and the MIP which were given an hour.

Network (ϵ)	Test error	Lower bound	LiPPA	Upper bound
NOR MLP-B (0.05)	2.05	53.37*[SYZ+19]	**63.30**	94.04*
NOR MLP-C (0.1)	1.44	93[CAH19]	**94.3**	100
ADV MLP-B (0.1)	3.33	16.25*[SYZ+19]	**20.40**	34.3*
ADV MLP-C (0.1)	0.92	10[CAH19]	**11.90**	99
ADV CNN (0.1)	0.82	3[CAH19]	**4.50**	100
LP_D MLP-B (0.2)	15.72	36.33*[SYZ+19]	38.3^+	36.33*
MMR MLP-C (0.1)	2.11	22.5[CAH19]	21.30	24.9
MMR CNN (0.1)	1.65	6[CAH19]	5.80	6
MMR-ADV MLP-C (0.1)	2.04	14[CAH19]	13.40	14.1
MMR-ADV CNN (0.1)	1.19	3.6[CAH19]	3.20	3.6

Table 2: Results on the first 1000 test examples. Starred results*: computed on the whole test set. **Bold results**: beating previous lower bound. Lower bounds are the best reported in their respective papers. The upper bound are the corresponding ones. $^+$: The upper bound is computed on the whole test set while only the first 1000 samples were used in our experiment.

Observation of runtime The average running time of LiPPA compares favorably to MIP but can not compete with gradient methods running on GPU. However, we observed an interesting behavior (see Table 3): if LiPPA is successful and finds an adversarial example, it finds it really fast. If LiPPA runs more than a few seconds, it is a good prediction that it will fail. Knowing this, the time budget seems a good strategy for the stopping criterion.

5 Discussion and conclusions

We propose a new attack, LiPPA, that alternates exploitation and exploration steps. In the exploration step, LiPPA takes advantage of gradient-based information by following the gradient, not to find adversar-

Model	Failed (in sec)	Successful (in sec)
ADV-CNN	31.10	0.64
MMR-CNN	36.93	2.45
MMR-ADV-CNN	29.87	0.45
NOR-MLP-B	15.90	0.42
ADV-MLP-B	16.52	0.25
NOR-MLP-B	16.24	0.25

Table 3: Average runtime for failed and successful LiPPA search.

ial examples but to explore another promising linear region that could lead to a better adversarial example. And during the exploitation step, LiPPA benefits from gradient-free information by solving a linear program in the linear region proposed by the exploration step.

Doing that, LiPPA is able to achieve better lower bounds on the robust test error getting ahead of previous methods including MIP with time limits going up to one hour, as well as PGD, establishing itself as a new state-of-the-art attack. We show that the input space of neural networks with piecewise linear activation, among them, the ReLU, can be partitioned into equivalence classes and that the state-of-the-art MIP verifier uses a part of the information contained in that partitioning to be faster than concurrent work. We show that there is still unexploited information in the verification with respect to the linear regions of the input space.

Future work includes finding better ways to propose linear regions in which to look for adversarial examples as well as looking for a way to discover constraints to add to the MIP formulation to focus only on the binary values corresponding the linear regions of the input space to accelerate the robustness verification.

References

[ABMM18] Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding Deep Neural Networks with Rectified Linear Units. In *International Conference on Learning Representations*, February 2018. 2

[ACW18] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. In *Proceedings of the 35th International Conference on Machine Learning*, 2018. July 2018. arXiv: 1802.00420. 2

[BIL+16] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. Measuring Neural Net Robustness with Constraints. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2613–2621. Curran Associates, Inc., 2016. 1, 2

[BTT+18] Rudy R Bunel, Ilker Turkaslan, Philip Torr, Pushmeet Kohli, and Pawan K Mudigonda. A Unified View of Piecewise Linear Neural Network Verification. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 4790–4799. Curran Associates, Inc., 2018. 2

[CAH19] Francesco Croce, Maksym Andriushchenko, and Matthias Hein. Provable Robustness of ReLU networks via Maximization of Linear Regions. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2057–2066. PMLR, April 2019. ISSN: 2640-3498. 6, 7

[CH18] Francesco Croce and Matthias Hein. A randomized gradient-free attack on ReLU networks. *arXiv:1811.11493 [cs, stat]*, November 2018. arXiv: 1811.11493. 1, 2, 6, 7

[CRH20] Francesco Croce, Jonas Rauber, and Matthias Hein. Scaling up the Randomized Gradient-Free Adversarial Attack Reveals Overestimation of Robustness Using Established Attacks. *International Journal of Computer Vision*, 128(4):1028–1046, April 2020. 1, 2, 6, 7

[CW17] Nicholas Carlini and David Wagner. Towards Evaluating the Robustness of Neural Networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57, May 2017. ISSN: 2375-1207. 1

[GSS15] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. *International Conference on Learning Representation*, 1050:20, 2015. 1, 2

[KBD+17] Guy Katz, Clark Barrett, David Dill, Kyle Julian, and Mykel Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. *arXiv:1702.01135 [cs]*, May 2017. arXiv: 1702.01135. 1, 2

[KGB17] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv:1607.02533 [cs, stat]*, February 2017. arXiv: 1607.02533. 7

[MMS+19] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. *arXiv:1706.06083 [cs, stat]*, September 2019. arXiv: 1706.06083. 1, 2

[PMW+16] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks. *arXiv:1511.04508 [cs, stat]*, March 2016. arXiv: 1511.04508. 1

[SYZ+19] Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. A Convex Relaxation Barrier to Tight Robustness Verification of Neural Networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 9832–9842. Curran Associates, Inc., 2019. 6, 7

[TXT18] Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. Evaluating Robustness of Neural Networks with Mixed Integer Programming. September 2018. 1, 2, 4, 5

[WK18] Eric Wong and Zico Kolter. Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope. In *International Conference on Machine Learning*, pages 5283–5292, 2018. 6

[WN99] Laurence A. Wolsey and George L. Nemhauser. *Integer and Combinatorial Optimization*. John Wiley & Sons, July 1999. Google-Books-ID: vvm4DwAAQBAJ. 3