Sparse and invisible adversarial attacks using MIP Optimization

Ramzi Ben Mhenni, Mohamed Ibn Khedher, Stéphane Canu

ARTICLE TEMPLATE

Sparse and invisible adversarial attacks using MIP Optimization

Ramzi Ben Mhenni[a], Mohamed Ibn Khedher[a] and Stéphane Canu[b]

[a] IRT SystemX, 2 boulevard Thomas Gobert, 91120 Palaiseau, France
[b] INSA Rouen Normandie, 685 Avenue de Universite 76800 Rouen, France

ARTICLE HISTORY
Compiled March 25, 2024

ABSTRACT
Deep Learning methods are known to be vulnerable to adversarial attacks where malicious perturbed inputs lead to erroneous model outputs. From a safety perspective, highly sparse adversarial attacks are particularly dangerous. On the other hand the pixel wise perturbations of sparse attacks are typically large and thus can be potentially detected. Adversarial attacks during training have been early on proposed as a potential defense, now known as **adversarial training**. Previous research has shown that ℓ_0-norm has good sparsity but is challenging to solve. We propose a new technique to craft adversarial examples aiming at minimizing ℓ_1 distance to the original image regularized by the **Total variation** (TV) function. This will favor the change of pixels in the region of high variation making the attacks almost invisible. 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). By formulating the problem as an MIP, we can ensure that the solution is the globally optimal one, meaning that we can guarantee that the attack is both sparse and invisible. In this paper, we propose a global optimization approach to get the optimal sparse invisible perturbation using a 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.

KEYWORDS
Neural network; Sparse and invisible adversarial attacks; Mixed Integer Programming.

1. Introduction

Cutting-edge neural networks are susceptible to adversarial attacks, which involve manipulating input images to produce significantly different outputs while maintaining a close similarity to the original images [4, 26, 33]. Adversarial attacks can be classified as either white-box or black-box, depending on the attacker's level of knowledge of the target model. In white-box attacks, the attacker has access to the model during the attack, whereas in black-box attacks, the attacker can only query the output of the classifier or confidence scores of all classes. To address this problem, researchers have proposed minimizing a distance metric that measures the perceptual similarity between two input images.

A non-trivial adversarial attack is meant to be small and "imperceptible" in certain sense [24, 35]: in this manner, it has little impact on the semantic meaning of

CONTACT A. N. Author. Email: ramzi.ben-mhenni@irt-systemx.fr

the entry. In the most common pixel-level additive attack setting, the standard measure of attack magnitude is by the ℓ_p-norm of perturbations, with $p = 0, 1, 2,$ or ∞. Typically the attacks try to find points on or close to the decision boundary, where the distance is measured in the pixels space, most often wrt the ℓ_2-norm and the ℓ_∞ minimize the confidence in the correct class in some ϵ-ball around the original image [5]. These distances can be used to constrain the maximum perturbation size, which is useful for ensuring the attack is not too aggressive. In the other hand, we find the ℓ_1-norm and the ℓ_0-"norm". Contrary to the first ones, witch tend to bring changes in the image of low value but with a high density, the ℓ_1-norm and the ℓ_0-"norm" tend to create sparsity in the solution but with a generally very high value [22, 35]. Several studies have investigated the vulnerability of neural networks to adversarial sparse attacks and proposed various defense mechanisms. Nguyen et al. (2015) explored the ℓ_0-norm attack, which aims to minimize the number of perturbed pixels. Zhu et al. (2021) studied sparse adversarial attacks and introduced a sparsefool algorithm for generating adversarial examples with sparse perturbations. These studies contribute to the understanding of adversarial attacks and the development of defense strategies.

In this paper we are dealing specifically with sparse adversarial attacks, that is we want to modify the smallest amount of pixels in order to change the decision. Our idea is the use of regularization techniques, such as total variation (TV), to encourage sparse perturbation and making the perturbations less noticeable to human observers. By combining the ℓ_1 distance minimization with TV regularization, we can selectively alter pixels in regions of high variation, effectively camouflaging the adversarial modifications. Section 2 describes the problem of adversarial attack and how it can be formulated as a Mixed Integer Program (MIP). Section 3 describes the branch-and-bound algorithm principle, details our implementation strategy and links the node evaluation. Then, numerical results are given in Section 4, 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 5.

2. Formulating Adversarial Attacks as a Mixed Integer Program

A feed-forward neural (FNN) network with ReLu (Rectified Linear Unit) is a type of artificial neural network where information flows only in one direction, from the input layer to the output layer. The input layer receives the initial input, which is then propagated through one or more hidden layers, each consisting of several nodes or neurons. These neurons perform a linear transformation on their inputs and then apply a non-linear activation function, such as the ReLu function, to produce their output. The ReLu activation function is defined as follows:

$$f(x) = max(0, x)$$

where x is the input to the neuron. It returns the maximum of 0 and the input value x. This function has the advantage of being computationally efficient and provides a sparse activation. Overall, feed-forward neural networks with ReLu have become a popular choice for many machine learning tasks due to their ability to learn complex non-linear relationships between inputs and outputs.

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:

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

The decision variables of the problem include $\boldsymbol{a}$, $\boldsymbol{h}$, and $\hat{\boldsymbol{h}}$: $\boldsymbol{a}$ represents the perturbation vector, $\boldsymbol{h}$ represents the input to the ReLU activation function, and $\hat{\boldsymbol{h}}$

represents the output of the ReLU function. The constraints in the problem include: The output of the ReLU function $\hat{\boldsymbol{h}}$ is greater than or equal to $\boldsymbol{h}$, and greater than or equal to 0. The output of the target class s_y is greater than or equal to the output of all other classes s_i. The objective function of the problem is to minimize the distance $d(\boldsymbol{a}-\boldsymbol{x})$ between the original input vector and the perturbed input vector.

2.1. Formulating ReLU

Let $\hat{\boldsymbol{h}} = \max(\boldsymbol{h}, 0)$, where $\boldsymbol{h}$ is a vector of inputs to the ReLU and $-M \leqslant \boldsymbol{h} \leqslant M$. In this context, M represents a constant value that bounds the range of inputs. There are three possibilities for the phase of the ReLU. If $\hat{\boldsymbol{h}} = 0$, we say that such a unit is stably inactive, meaning that the output of the unit is always zero. Similarly, if $\boldsymbol{h} = \hat{\boldsymbol{h}}$, we say that such a unit is stably active, meaning that the output is always equal to the input. Otherwise, the unit is unstable and may switch between being active or inactive depending on small perturbations in the input. To deal with the instability of these units, we introduce an indicator decision variable b_i, which indicates whether the ReLU is active or not. Specifically, b_i takes on a value of 1 if the ReLU is active, and 0 if it is inactive. This formulation is referred to as Binary ReLU.

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

Then, evaluating the robustness of a neural network with ReLU activation functions can be formulated as a Mixed Integer Program (MIP). The goal of this problem is to minimize the distance between the original input and the perturbed input while ensuring that the output of the network remains the same. The MIP involves optimizing over the input perturbation and the binary variables b_i, subject to constraints that ensure the perturbation is within a certain distance from the original input and that the output of the network remains the same.

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

The MIP formulation presented is aimed at evaluating the robustness of a neural network with ReLU activation functions. The problem, denoted by $\mathcal{P}$, involves minimizing the distance $d(\boldsymbol{a}-\boldsymbol{x})$ between the original input vector $\boldsymbol{x}$ and the perturbed input vector $\boldsymbol{a}$, subject to several constraints. The decision variables $\boldsymbol{b}$ represents the binary decision variable indicating whether a ReLU unit is active or not. The constraints in the problem include: The binary variable $\boldsymbol{b}$ must take on a value of either 0 or 1. The output of the ReLU function $\hat{\boldsymbol{h}}$ is bounded by $M\boldsymbol{b}$ from above and by $\boldsymbol{h} + M(1-\boldsymbol{b})$ from below.

2.2. Distance metric for evaluating adversarial attacks

The choice of the distance is very important, both for the quality of the solution as well as for the resolution time of the MIP. In the literature, the standard measure of attack magnitude is by the ℓ_p-norm of perturbations, with $p = 0, 1, 2,$ or ∞. A vector that is sparse in the strictest sense of the term means that only a few of its coefficients have non-zero values. Given this definition, it is clear that the ℓ_0 norm,

which can also be seen as the limit of the "norm" ℓ_p [1] as p tends to zero:

$$||\boldsymbol{x}||_0 = \lim_{p\to 0} \|\boldsymbol{x}\|_p^p = \lim_{p\to 0} \sum_{i=1}^{n} |x_i|^p, \tag{1}$$

is the most appropriate function for measuring sparsity. The discrete nature of the ℓ_0 norm often makes the problem computationally combinatorial and very difficult, which has led to several alternative sparsity measures. It has been shown that replacing the ℓ_0 norm with a function of the form $\sum_{i=1}^{N} \varphi(|x_i|)$ produces sparse solutions, i.e., solutions that contain zero values, if and only if φ is strictly increasing at 0 [23]. This is the case for the ℓ_p norm with $0 < p \leqslant 1$. While the case $p < 1$, which generates difficult optimization problems because they are non-convex, has been studied in several works[17, 34], The ℓ_1 norm ($|\boldsymbol{x}|_1 := \sum_i |x_i|$) remains the most commonly used and easiest-to-consider sparsity measure due to its convex nature (see Figure 1).

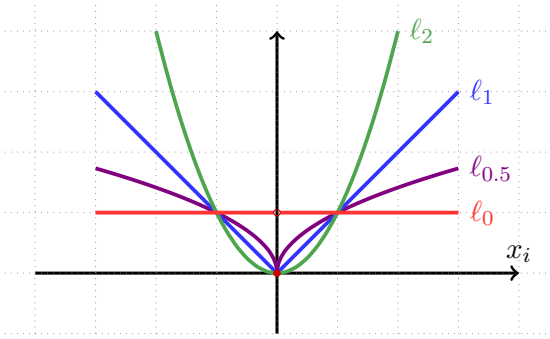

Figure 1. Functions of the scalar variable $\varphi(x_i)$ involved in the definition of various ℓ_p norms, with $\|\boldsymbol{x}\|_p^p = \sum_i \varphi(|x_i|)$, for $p = 0$, 0.5, 1 and 2.

2.2.1. MIPs Reformulation of the ℓ_p-norm

- The "norm" ℓ_0 is not a true norm as it does not satisfy the triangle inequality. It is defined as the number of non-zero elements in a vector or matrix. However, it can be reformulated exactly by adding binary variables $t \in 0, 1$ to the model. Let $\mathbf{x}$ be a vector, and t_i be a binary variable that takes the value 1 if $x_i \neq 0$ and 0 otherwise. Then, the ℓ_0 norm of $\mathbf{x}$ can be expressed as follows:

$$\|\mathbf{x}\|_0 = \sum_{i=1}^{n} t_i$$

  Using this formulation, the problem of minimizing the ℓ_0 norm of $\mathbf{x}$ subject to linear constraints can be converted into a MIP problem, which can be solved using a Branch-and-Bound method. However, the ℓ_0 norm is known to be non-convex and NP-hard, which makes it difficult to solve for large-scale problems.
- The ℓ_1 norm of a vector $\mathbf{x}$, denoted by $||\mathbf{x}||_1$, is the sum of the absolute values of its elements. Mathematically, it can be expressed as:

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|$$

  The ℓ_1 norm is a convex function and is easier to minimize than the ℓ_0 norm. It is also a good proxy for the ℓ_0 norm in many applications, because it encourages sparsity in a similar way. In fact, minimizing the ℓ_1 norm subject to linear constraints is a standard convex optimization

[1] The term "norm" ℓ_p for $0 < p < 1$ is also an abuse of language: this function, not satisfying the triangular inequality, is only a quasi-norm.

problem that can be solved efficiently using various optimization algorithms. To reformulate the ℓ_1 norm for use in a MIP solver, we can introduce additional variables and constraints. Let $\mathbf{x}$ be a vector of decision variables that we want to minimize the ℓ_1 norm of, subject to some linear constraints. We can introduce a vector of non-negative variables $\mathbf{t}$ such that $t_i = |x_i|$ for $i = 1, \ldots, n$. Then, we can express the ℓ_1 norm as the sum of the variables t_i, i.e., $\|\mathbf{x}\|_1 = \sum_{i=1}^{n} t_i$. To ensure that $t_i = |x_i|$, we can introduce the following constraints: $\{t_i \geqslant x_i, \quad t_i \geqslant -x_i, \quad i = 1, \ldots, n\}$ These constraints ensure that t_i is greater than or equal to the absolute value of x_i, and that t_i is also greater than or equal to the negative of x_i. Therefore, t_i must be equal to $|x_i|$. Finally, we can add these new variables and constraints to our existing linear program and solve it using a mixed-integer linear programming solver.

- The ℓ_∞ norm of a vector $\mathbf{x}$, denoted by $\|\mathbf{x}\|_\infty$, is the maximum absolute value of its elements. Mathematically, it can be expressed as:

$$\|\mathbf{x}\|_\infty = \max_{i=1}^{n} |x_i|$$

To reformulate the ℓ_∞ norm, we can introduce a scalar variable t that represents the maximum absolute value of the elements of $\mathbf{x}$. Then, we can express the ℓ_∞ norm as follows:

$$\|\mathbf{x}\|_\infty = t$$

To ensure that t is equal to the maximum absolute value of the elements of $\mathbf{x}$, we can introduce the following constraints: $\{t \geqslant x_i, \quad t \geqslant -x_i, \quad i = 1, \ldots, n\}$ These constraints ensure that t is greater than or equal to each element of $\mathbf{x}$ and its negation. Therefore, t must be equal to the maximum absolute value of the elements of $\mathbf{x}$. Finally, we can add these new variables and constraints to our existing linear program and solve it using a mixed-integer linear programming solver. By minimizing the variable t subject to the linear constraints and the additional constraints, we will obtain a solution that minimizes the ℓ_∞ norm of the vector $\mathbf{x}$.

2.2.2. Total Variation (TV)

The Total Variation (TV) function is commonly used in image processing and is defined as the sum of the absolute differences between neighboring pixels (spatial variability). It was initially proposed for image denoising and reconstruction [25]. To calculate the total variation, the absolute differences of adjacent pixel values in the input images are summed. This provides an estimate of the noise present in the images, which can be employed as a loss function in optimization to reduce noise in images.

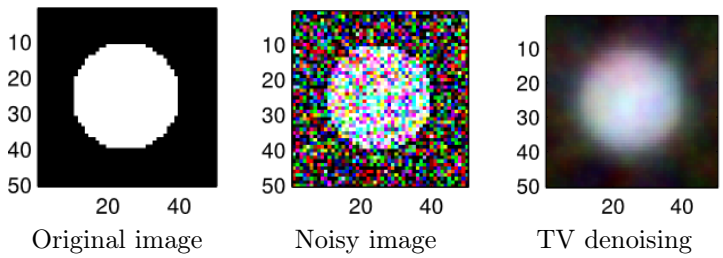

Figure 2. Total Variation denoising example

The Noisy image in Figure 2 has a lot of noise in it, making it hard to see the details of the objects in the image. To denoise the image we can use the total variation denoising. As you can see, the denoised image is much clearer and easier to see. The edges and details of the objects in the image are preserved, while the noise has been removed. Mathematically, for an image represented by a matrix $\mathbf{X} = (x_{ij})_{i=1,\ldots,m,j=1,\ldots,n}$, the TV function $TV(\mathbf{X})$ can be expressed as:

$$TV(\mathbf{X}) = \sum_{i=1}^{m-1} \sum_{j=1}^{n-1} \sqrt{(x_{i+1,j} - x_{i,j})^2 + (x_{i,j+1} - x_{i,j})^2}.$$

To reformulate the TV function for use in a MIP solver, we can introduce additional variables and constraints. We can introduce a set of non-negative variables $l^h_{i,j}$ (respectively $l^v_{i,j}$) that represent the absolute difference between horizontal (respectively vertical) neighboring pixels, scaled by the binary variables $y_{i,j,k}$. Specifically, we have: $l^v_{i,j} = x_{i,j} - x_{i-1,j}$ and $l^h_{i,j} = x_{i,j} - x_{i+1,j}$. We can express the TV function as the sum of the variables ${l^h}_{i,j}$ and ${l^v}_{i,j}$ subject to the linear constraints and the additional constraints. Specifically, we have $TV(\mathbf{X}) = \sum_{i=1}^{m-1} \sum_{j=1}^{n-1} \sqrt{{l^h}^2_{i,j} + {l^v}^2_{i,j}}$.

3. Branch-and-bound exploration

Separation and evaluation step. The branch-and-bound principle [29] 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 3.2). At node i, decisions have been made concerning 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 step is crucial in the Branch-and-Bound method, as it allows the algorithm to efficiently prune the search space and focus on promising candidate solutions. By discarding solutions that are guaranteed to be worse than the current best solution, the algorithm can avoid exploring large portions of the search space, leading to significant computational savings. For our example, 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. In continuous relaxation, the integer constraints on the decision variables are relaxed to allow them to take on any real value within a specified range. This transforms the discrete optimization problem into a continuous optimization problem. For example, if the decision variable $\boldsymbol{b}$ can only take on integer values, we relax this constraint to allow $\boldsymbol{b}$ to take on any real value between 0 and 1. Continuous relaxation can be written as follows:

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

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).

3.1. 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 $\mathcal{R}^{(i)}$.

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

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, the set of constraints $\{\hat{\boldsymbol{h}} \geqslant \boldsymbol{h} \; ; \hat{\boldsymbol{h}} \geqslant 0 \; ; \hat{\boldsymbol{h}} \leqslant \frac{\boldsymbol{h}+M}{2}\}$ is a *convex relaxation* of the constraint $\hat{\boldsymbol{h}} = max(\boldsymbol{h}, 0)$ (see. figure 3). 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. By

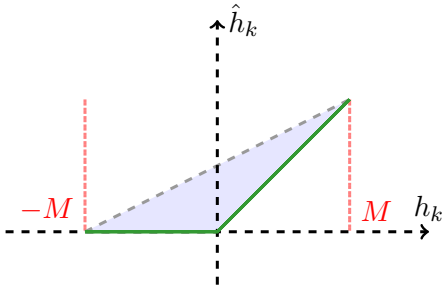

Figure 3. The tightest linear convex relaxation of the ReLU function under additional boundedness assumptions on the solution space.

using the convex relaxation of the ReLU function, one can benefit from the convexity of the optimization problem and apply efficient convex optimization algorithms to find the optimal solution.

3.2. Branching rules and exploration strategy

The branching rules and exploration strategy determine how the search space is partitioned into smaller subproblems and how these subproblems are explored. The branching rules and exploration strategy are crucial components of the algorithm and can significantly impact its performance. The branching rule selects the index j of the variable which is used in order to subdivide problem $\mathcal{P}^{(i)}$ (see Figure 3.2). At each node of the search tree, the algorithm selects a variable to branch on and creates two subproblems by adding a constraint that fixes the selected variable to a specific value. 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. The exploration strategy determines how the algorithm explores the search space and selects which subproblems to explore next. 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 $b_i = 1$.

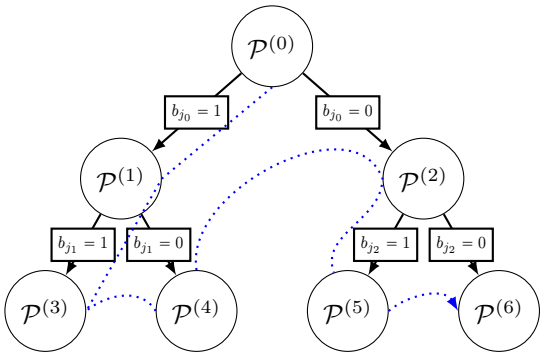

Figure 4. Exploration strategy : Depth-first search. Each node corresponding to the optimization problem $\mathcal{P}^{(n)}$, is divided into two children nodes obtaining by constraining one variable to be zero or non-zero.

This strategy, similar to the principle of greedy forward selection algorithms [21], 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 $\widehat{\boldsymbol{x}}$ denotes the best known solution along the exploration. The Branch-and-Bound algorithm converge to the global minimum

(1) **Initialization:** $L \leftarrow \{\mathcal{P}^{(0)}\}$; $z_U = +\infty$; $\widehat{\boldsymbol{a}} := 0$.

(2) **Optimality:** if $L = \varnothing$, then return the optimal solution $\widehat{\boldsymbol{a}}$.

(3) **Node selection:** choose a subproblem $i \in L$ by depth-first search and remove it from L.

(4) **Node evaluation:** compute $z_\ell^{(i)}$.

(5) **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 $\widehat{\boldsymbol{a}} \leftarrow \boldsymbol{a}^{\mathcal{R}(i)}$. Prune node i and return to step 1.

(6) **Branching:** subdivide node i by (3.2) and add the two subproblems to L.

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

in a finite number of steps. In the worst case, an exhaustive search is done.

4. Performance Evaluation

	Model	Type	units	Layers	Activation Function
CIFAR10	4×100	fully connected	140	4	ReLU
	6×100	fully connected	610	6	ReLU
	9×200	fully connected	1,810	9	ReLU
	ConvSmall	convolutional	4,852	3	ReLU
	ConvMed	convolutional	7,144	3	ReLU
	MaxPool	convolutional	53,938	9	ReLU
MNIST	3×50	fully connected	110	3	ReLU
	3×100	fully connected	210	3	ReLU
	6×100	fully connected	510	6	ReLU
	9×200	fully connected	1,610	9	ReLU
	ConvSmall	convolutional	3,604	3	ReLU
	ConvMed	convolutional	5,704	3	ReLU
	MaxPool	convolutional	13,798	9	ReLU

Table 1. Neural networks used to analyze the MNIST and CIFAR10 datasets

We tested our approach on two image datasets: MNIST [7] and CIFAR [Krizhevsky et al.]. The original version MNIST dataset contains 60,000 grayscale images of handwritten digits, with a resolution of 28x28 pixels, and it includes 10,000 additional images for testing. The digits in the images are white, and they appear on a black background. CIFAR-10 is a commonly used benchmark dataset that consists of 60,000 color images in 10 classes, with 6,000 images per class. The images are 32x32x3 pixels in size and cover a wide range of objects, such as animals, vehicles, and household items.In our experiments, we used neural networks to analyze the MNIST and CIFAR10 datasets, as shown in Table 1. Our evaluation included architectures with a maximum of 53K hidden units. We used both adversarially trained networks, which are designed to withstand adversarial attacks, and undefended networks.

To evaluate our approach, we selected the first 100 images from the test set of each dataset. However, we only considered the images that were correctly classified by the neural network from these 100 images.

4.1. Visualization

To illustrate the difference between a sparse and non-sparse adversarial attack, we show three example images in Figure 5. The first image shows a non-sparse attack generated using the MIP-ℓ_2 attack, while the second image shows a sparse attack generated using the MIP-ℓ_1 attack, and the third one shows a sparse invisible attack using our proposed MIP-ℓ_2-TV attack. In the non-sparse attack on the top, the perturbations are spread out across the entire image, affecting a large number of pixels. This attack is unlikely to happen for real applications. On the other hand, in the sparse attack in the middle, the perturbations are concentrated in a small area of the image, affecting only a few pixels. The MIP-ℓ_1 is easy to detect, as the changes are more noticeable and the image may appear visibly distorted. Finally, the MIP-ℓ_1-TV attack on the bottom is both sparse and invisible, as it affects only a few pixels and minimizes the total variation of the perturbations, making them imperceptible to the human eye.

These examples highlight the importance of sparsity in adversarial attacks, as it can greatly increase the stealthiness and effectiveness of the attack. By minimizing the total variation of the perturbations, we can generate highly sparse adversarial attacks that are almost invisible, making them a significant threat to the security of deep neural networks.

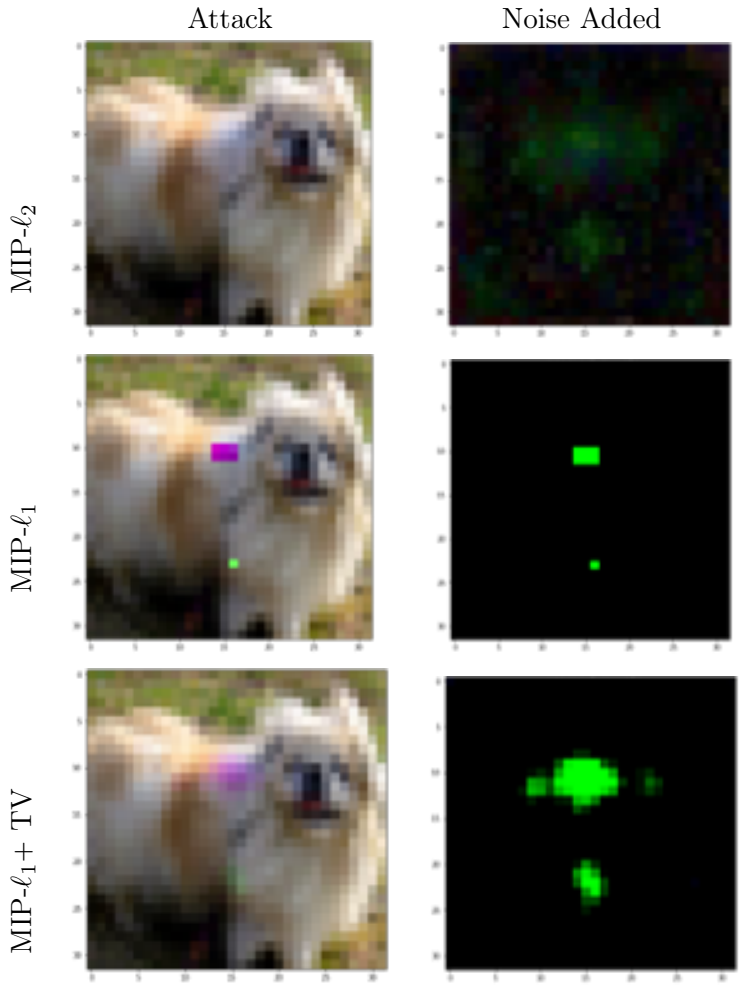

Figure 5. Example images of non-sparse adversarial attack (MIP-ℓ_2), sparse adversarial attack (MIP-ℓ_1) and sparse invisible attack (MIP-ℓ_1-TV).

4.2. Evaluation of solution quality

We evaluated the performance of each method on the standard image classification dataset CIFAR-10. We measured the success rate of the attacks in terms of the percentage of images that were misclassified by the target model. Additionally, we also evaluated the sparsity and invisibility of the attacks[2], as these are important criteria for the safety and security of deep neural networks. We compared the effectiveness of our proposed technique for crafting adversarial examples with several other state-of-the-art methods. Specifically, we compared our approach with the following techniques:
FGSM: Fast Gradient Sign Method is a popular white-box attack that perturbs the input image in the direction of its gradient with respect to the loss function.
PGD: Projected Gradient Descent Attack is another white-box attack that aims to identify the adversarial examples by iteratively perturbing the input image in the direction that maximizes the loss function, subject to a small perturbation budget.
C&W: Carlini and Wagner's attack is a state-of-the-art optimization-based approach that finds the minimal perturbation that causes the image to be misclassified,

[2]**Attack's invisibility** : it is important to note that no single metric or method can capture the invisibility of an attack perfectly. The perception of visibility can be influenced by various factors, such as the complexity of the image, the size and location of the perturbations, and the background of the image. Therefore, a combination of different metrics and human evaluation are used to obtain a more accurate assessment of the attack's invisibility.

optimizing a non-convex objective function that measures the distance between the original image and the perturbed image.

Our experiments showed that our proposed technique achieved a higher success rate than FGSM, PGD, and C&W. However, our technique was significantly more sparse than the other methods, with a smaller percentage of pixels perturbed. This indicates that our approach is more focused on identifying the most important pixels to modify, rather than perturbing the image indiscriminately. Moreover, our technique was almost completely invisible, with the perturbations being imperceptible to the human eye. This is an important characteristic for real-world applications, as it means that the attacks are less likely to be detected and can therefore pose a greater threat to the security of deep neural networks.

Table 2 summarizes the results of our experiments, showing the success rate, sparsity, and invisibility of each method. To calculate the success rate, we first generate adversarial examples, and then evaluate the accuracy of the targeted model on the perturbed examples. If the accuracy drops significantly compared to the accuracy on the original examples, then the attack is considered successful. The success rate is calculated as the percentage of test examples that are misclassified by the model after applying the attack. As can be seen, our proposed MIP-ℓ_1-TV technique achieved the highest sparsity and invisibility compared to GSM, PGD, and C&W, while maintaining a best success rate as it's a global optimization approach. These results demonstrate the potential of our approach for improving the security and safety of deep neural networks against adversarial attacks.

Method	**Success rate (%)**	**Sparsity (%)**	**Invisibility (%)**
FGSM	56	03	34
PGD	35	06	55
CW	61	11	85
MIP-ℓ_1	88	92	05
MIP-ℓ_1-TV	88	85	88

Table 2. Comparison of success rate, sparsity, and invisibility of different adversarial attack methods.

Compared to these methods, the proposed MIP-ℓ_1-TV technique offer several advantages. First, the MIP-based techniques can generate highly sparse and almost invisible adversarial attacks, which can be particularly dangerous from a security perspective. Second, the MIP-based techniques provide a principled optimization framework for generating adversarial examples, which allows for better control over the attack parameters and the generation process. Finally, the MIP-based techniques offer a global optimization approach, which can ensure the optimality of the generated adversarial examples, unlike other methods such as PGD and FGSM, which are susceptible to local optima.

4.3. Empirical Time Cost

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$_{\text{Gurobi}}$) [3] and MIP$_{\text{Verify}}$[4]. 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; due to the lack of space we do not compare the obtained solutions to that of standard, suboptimal methods.

To provide a comprehensive comparison between the different algorithms presented in this article, we employ performance profiles, a powerful tool for visualizing and analyzing algorithm performance. Performance profiles allow us to compare the

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

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

performance of multiple algorithms across a range of problem instances. In our evaluation, the performance profiles are constructed based on the computation times of each algorithm for a set of problem instances. The computation time is limited to 1,000 seconds for each instance. The performance profile depicted in Figure 6 pro-

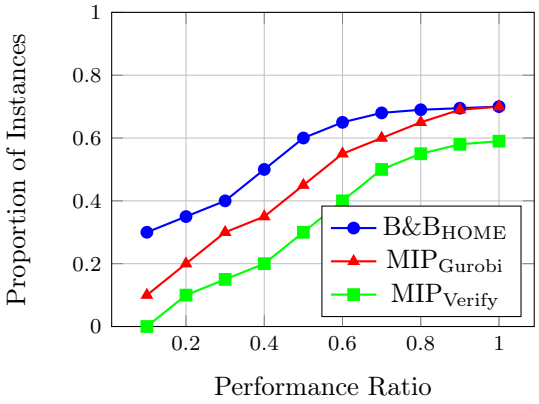

Figure 6. Performance Profile of B&B$_{\text{HOME}}$, MIP$_{\text{Gurobi}}$, and MIP$_{\text{Verify}}$. Computation time is limited to 1,000 seconds for each instance.

vides valuable insights into the relative performance of the algorithms B&B$_{\text{HOME}}$, MIP$_{\text{Gurobi}}$, and MIP$_{\text{Verify}}$. The graph reveals the proportion of instances for which each algorithm achieved a certain performance ratio, represented by the x-axis. We observe that B&B$_{\text{HOME}}$ consistently outperforms both MIP$_{\text{Gurobi}}$ and MIP$_{\text{Verify}}$ across the entire range of performance ratios. This indicates that B&B$_{\text{HOME}}$ achieves better computational efficiency and finds solutions closer to the global optimum within the given time limit. The higher proportion of instances where B&B$_{\text{HOME}}$ has a lower performance ratio demonstrates its superior performance compared to the other algorithms. On the other hand, MIP$_{\text{Gurobi}}$ and MIP$_{\text{Verify}}$ exhibit slightly lower performance ratios, indicating comparatively longer computation times and potentially suboptimal solutions. However, it is worth noting that MIP$_{\text{Verify}}$ performs better than MIP$_{\text{Gurobi}}$, as evidenced by its higher proportion of instances with lower performance ratios. This suggests that MIP$_{\text{Verify}}$ is more efficient in finding good-quality solutions compared to MIP$_{\text{Gurobi}}$. Moreover, the performance profiles highlight the scalability challenges faced by all algorithms as the complexity of the problem instances increases. The computation times for all algorithms increase rapidly with larger and more complex models, indicating the inherent difficulty of the problem. However, even under these challenging conditions, B&B$_{\text{HOME}}$ demonstrates a notable advantage over the other algorithms.

In order to evaluate the behavior of our method regarding the complexity of the model, we have varied the number of hidden layers. Results averaged over 100 instances of each problem are given in Table 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 7 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. The complexity of a MIP problem depends on the number of decision variables, constraints, and the type of problem formulation. One approach to formulating the adversarial attack problem as an MIP is to use binary variables. However, the number of possible combinations of binary variables grows exponentially with the number of ReLU. This leads to a combinatorial explosion in the number of feasible solutions, which makes it challenging to find the optimal solution using standard optimization techniques. As a result, specialized algorithms such as branch-and-bound methods are often required to solve large-scale MIP problems efficiently. However, even with these algorithms, the computational cost can become prohibitively expensive as the size of the problem increases, which limits the scalability of MIP-based approaches for adversarial attacks.

	CIFAR10	B&B$_{\text{HOME}}$			MIP$_{\text{Gurobi}}$			MIP$_{\text{Verify}}$		
		T	Nds	F	T	Nds	F	T	Nds	F
norm L2	4x100	**90.8**	1,125	0	150.9	2,879	0	120,2	1,987	0
	6x100	**840.5**	7,343	42	885	7,855	48	872.8	7,256	48
	ConvSmall	**802**	6,955	47	850	7,102	47	-	-	-
	ConvMed	1,000	-	50	1,000	-	50	-	-	-
	MaxPool	1,000	-	50	1,000	-	50	-	-	-
norm L1	4x100	289.7	12,759	1	426.8	18,245	14	470.2	16,975	6
	6x100	**481.5**	3,743	5	609.2	3,512	31	672.8	3,856	17
	ConvSmall	**601**	256	32	645	301	35	-	-	-
	ConvMed	**980**	1,350	49	1000	1,546	50	-	-	-
	MaxPool	1,000	-	50	1,000	-	50	-	-	-
L1+TV	4x100	325.2	14,222	6	483.4	20,885	11	-	-	-
	6x100	**809.2**	44,743	37	881.5	53,512	41	-	-	-
	ConvSmall	**860**	2,498	47	945	3,767	47	-	-	-
	ConvMed	1,000	-	50	1,000	-	50	-	-	-
	MaxPool	1,000	-	50	1,000	-	50	-	-	-

	MNIST	T	Nds	F	T	Nds	F	T	Nds	F
norm L2	3x50	**55.2**	57,932	0	74,3	96,876	0	**52.7**	10,122	0
	6x100	**87.3**	91,003	9	110.8	120,483	9	101	98,478	9
	ConvSmall	**328.4**	12,393	28	442,2	39,888	33	-	-	-
	ConvMed	**995**	42,676	49	1,000	-	50	-	-	-
	MaxPool	1,000	-	50	1,000	-	50	-	-	-
norm L1	3x50	**25.3**	9,988	0	30.1	10,983	0	39.9	14,451	0
	6x100	**38.8**	13,819	0	52.4	21,332	0	40.1	16,383	0
	ConvSmall	**220**	10,223	20	375,2	22,378	28	-	-	-
	ConvMed	**925**	82,339	49	1,000	-	50	-	-	-
	MaxPool	-	-	50		-	50	-	-	-
norm L2	3x50	**62.7**	10,122	0	83.5	13,326	0	-	-	-
	6x100	**98.2**	102,30	30	128	123,84	38	-	-	-
	ConvSmall	**377**	14,762	28	517,3	58,020	33	-	-	-
	ConvMed	**998**	51,736	49	1,000	-	50	-	-	-
	MaxPool	1,000	-	50	1,000	-	50	-	-	-

Table 3. Computational efficiency for robustness problems averaged over 50 instances. Computing time (T), number of explored nodes (Nds), and number of instances that did not terminate in 1,000 s (F).

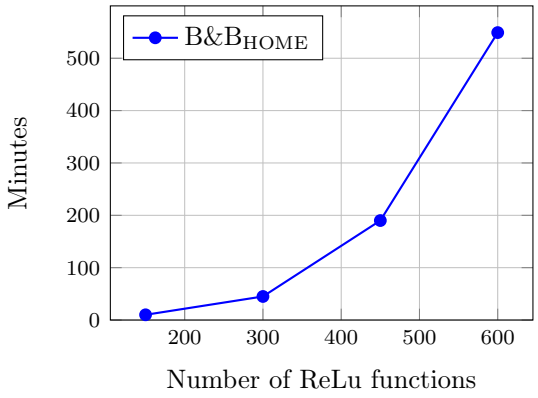

Figure 7. Computing time (Minutes) for robutness problems as a function of the number of ReLU in the model, average over 10 instances.

5. Conclusion

In this paper, we have proposed a new method for generating adversarial examples that are both highly sparse and almost invisible. Our approach is based on Mixed Integer Programming (MIP), which provides a principled optimization framework for generating adversarial examples with controlled sparsity and visibility. We have demonstrated the effectiveness of our method by comparing it with other state-of-the-art adversarial attack methods. Our results show that our MIP-based techniques can generate highly sparse and almost invisible adversarial examples that are capable of fooling deep learning models with high success rates. One of the key advantages of our method is that it provides a global optimization approach, which can ensure the optimality of the generated adversarial examples. Overall, our proposed method represents an important step towards the development of more robust and secure deep learning models that are less vulnerable to adversarial attacks. Future work could investigate the application of our method to other domains and tasks, as well as the development of more advanced defense mechanisms that can mitigate the impact of adversarial attacks. In addition to proposing a new method for generating adversarial examples using MIP, we have also developed a branch-and-bound algorithm to solve the resulting optimization problem. Our algorithm is designed to efficiently explore the combinatorial space of binary variables and find the optimal solution to the MIP problem. The key advantage of our algorithm is that it provides a principled and efficient approach to solving the MIP relaxation problem.

While our approach has demonstrated promising results in generating sparse and invisible adversarial examples using MIP, there are still limitations to the scalability of the method. Specifically, as the number of ReLU units in the neural network model increases, the complexity of the optimization problem grows exponentially, making it increasingly difficult to find the optimal solution within a reasonable amount of time. Despite these challenges, we believe that the approach proposed in this article represents a significant step forward in the field of adversarial attacks, and that it has the potential to be applied in a wide range of practical applications in machine learning and computer vision. By continuing to develop new algorithms and techniques for generating robust and secure models, we can help to ensure that the benefits of deep learning can be realized in a safe and reliable manner.

References

[1] Akhtar, N. and Mian, A. (2018). Threat of adversarial attacks on deep learning in computer vision: A survey. *CoRR*, abs/1801.00553.

[2] Bastani, O., Ioannou, Y., Lampropoulos, L., Vytiniotis, D., Nori, A. V., and Criminisi, A. (2016). Measuring neural net robustness with constraints. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 2621–

2629, USA. Curran Associates Inc.

[3] 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.

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

[5] Carlini, N. and Wagner, D. (2018). Audio adversarial examples: Targeted attacks on speech-to-text. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 1–7.

[6] Chakraborty, A., Alam, M., Dey, V., Chattopadhyay, A., and Mukhopadhyay, D. (2018). Adversarial attacks and defences: A survey. *CoRR*, abs/1810.00069.

[7] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142.

[8] Dvijotham, K., Stanforth, R., Gowal, S., Mann, T. A., and Kohli, P. (2018). A dual approach to scalable verification of deep networks. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, pages 550–559.

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

[10] 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*.

[11] Goodfellow, I. J., Shlens, J., and Szegedy, C. (2015). Explaining and harnessing adversarial examples. *ICLR*, 1412.6572v3.

[12] Jang, U., Wu, X., and Jha, S. (2017). Objective metrics and gradient descent algorithms for adversarial examples in machine learning. *ACSAC2017*, 262-277.

[13] 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.

[14] Kolter, J. Z. and Wong, E. (2017). Provable defenses against adversarial examples via the convex outer adversarial polytope. *CoRR*, abs/1711.00851.

[Krizhevsky et al.] Krizhevsky, A., Nair, V., and Hinton, G. Cifar-10 (canadian institute for advanced research).

[16] Kurabin, A., Goodfellow, I. J., and Bengio, S. (2017). Adversarial examples in the physical world. *ICLR*, 1607.02533v4.

[17] Lai, M. and Wang, J. (2011). An unconstrained ℓ_q minimization with $q \leqslant 1$ for sparse solution of underdetermined linear systems. *SIAM Journal on Optimization*, 21(1):82–101.

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

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

[20] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2017). Towards deep learning models resistant to adversarial attacks.

[21] 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.

[22] Modas, A., Moosavi-Dezfooli, S.-M., and Frossard, P. (2019). Sparsefool: a few pixels make a big difference. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9087–9096.

[23] Moulin, P. and Liu, J. (1999). Analysis of multiresolution image denoising schemes using generalized Gaussian and complexity priors. *IEEE Transactions on Information Theory*, 45(3):909—919.

[24] Nguyen, A., Yosinski, J., and Clune, J. (2015). Deep neural networks are easily fooled:

High confidence predictions for unrecognizable images. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436.

[25] Rudin, L. I., Osher, S., and Fatemi, E. (1992). Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena*, 60(1-4):259–268.

[26] 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].

[27] Tjeng, V. and Tedrake, R. (2017). Verifying neural networks with mixed integer programming. *CoRR*, abs/1711.07356.

[28] 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.

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

[30] Wong, E. and Kolter, J. Z. (2018). Provable defenses against adversarial examples via the convex outer adversarial polytope. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 5283–5292. PMLR.

[31] 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.

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

[33] Xiao, C., Li, B., Zhu, J.-Y., He, W., Liu, M., and Song, D. (2018). Generating adversarial examples with adversarial networks. *arXiv preprint arXiv:1801.02610*.

[34] Xu, Z., Zhang, H., Wang, Y., Chang, X., and Liang, Y. (2010). L1/2 regularization. *Science China Information Sciences*, 53(6):1159–1169.

[35] Zhu, M., Chen, T., and Wang, Z. (2021). Sparse and imperceptible adversarial attack via a homotopy algorithm.

[0]