Exemples adversaires et régions linéaires

Ismaila Seck

Normandie Université

THÈSE

Pour obtenir le diplôme de doctorat

Spécialité Informatique

Préparée au sein de INSA Rouen Normandie

Adversarial examples and linear regions

Présentée et soutenue par

Ismaila SECK

Thèse soutenue publiquement le 26 Octobre 2021 devant le jury composé de		
Stéphane CANU	Professeur à l'Université de Rouen Normandie	Directeur de thèse
Gaëlle LOOSLI	Maîtresse de conférence à l'Université Clermont-Auvergne	Encadrante de thèse
Vincent BARRA	Professeur à l'Université Clermont-Auvergne	Rapporteur de thèse
Fabrice MERIAUDEAU	Professeur à l'Université de Bourgogne	Rapporteur de thèse
Moustapha CISSE	Chercheur et Directeur de centre de recherche Google AI à Accra	Examinateur de thèse
Samia AINOUZ	Professeure à l'Université de Rouen Normandie	Examinatrice de thèse

Thèse dirigée par Stéphane Canu et Gaëlle Loosli

Acknowledgments

It is with great pleasure that I thank everyone who played a role in helping me achieve this milestone. Whether mentioned or not, I am thankful to everyone who has taught anything or helped in any way.

I want to thank my supervisor Stéphane Canu for accepting me as Ph.D. student. I want to thank my supervisor Gaëlle Loosli for the opportunities she gave me to discover the academic research world, giving me the opportunity to attend the *Conférence d'Apprentissage, CAp 2017*, a national conference, as well as the *European Symposium on Artificial Neural Networks* during that internship. She also helped me enroll in my Ph.D. program. Their support has been pivotal throughout this very special part of my life as Ph.D. student.

I would like to thank the jury members of my Ph.D. defense: Samia Ainouz, Vincent Barra, Fabrice Mériaudeau, and Moustapha Cissé. I thank them for their time, their presence, the insightful questions that showed their interest to my work. It was an honor for me to defend my thesis in front of them. I would like to thank Alain Rakotomamonjy and Christian Wolf for the part their role as member of my thesis monitoring committee.

I would like to thank my office colleagues : Cyprien Ruffino,Rachel Blin, Linlin Jia, Djamila Boukhelil, Flavi. I also want to thank Benjamin Deguerre, Jean-Baptiste Louvet, Mathieu, Anis, Ahmad and Souleymane Moussa for our discussions, especially at lunch time. I also want to thank Soufiane Belharbi for the feedback he kindly accepted to give.

I would like to thank Gilles Gasso for his very wise advices and his support during the lockdown. I would like to thank Benoit Gaüzère, Clement Chatelain, and Romain Hérault for their help during my teaching as a Teaching Assistant.

I would like to thank Sandra Hague and Brigitte Diarra for their help in all matters in their area of expertise.

I would like to thank Philippe Vaslin, Vanel Siyou, Danh Nguyen and Jocelyn De Goër for their welcome during the time I spent at LIMOS as well as the LIMOS team.

I would like to thank Seydina Ndiaye from the Virtual University of Senegal (UVS) and the UVS staff. I also want to thank the students I met there, for their kind words of encouragements and their interest in my work.

I dedicate this work to my family for making every necessary effort and even more, instilling in me values making me who I am.

I am grateful for the wonderful experience and very unique journey that my Ph.D. was.

Table of Contents

List of Figures

List of Tables

List of Symbols

Input space

$\mathscr{X}$	the input space of some classifier, generally a subspace of $\mathbb{R}^d$
$\mathbf{x} \in \mathscr{X}$	an input example belonging to the input space, belongs to $\mathscr{X}$
$\mathbf{x}'$	an adversarial example belonging to the input space, belongs to $\mathscr{X}$
$\mathscr{B}_{\epsilon,p}(\mathbf{x})$	$\{\mathbf{x}' \in \mathscr{X} : \lVert\mathbf{x}' - \mathbf{x}\rVert_p \leq \epsilon\}$, the ℓ_p ball of radius ϵ around $\mathbf{x}$.
δ	an (additive) adversarial noise, $\delta = \mathbf{x}' - \mathbf{x}$

Output space

$\mathscr{C}$	the set of all classes, usually {1,2,..., C}
C	the number of classes
$c \in \mathscr{C}$	a class of $\mathscr{C}$
$t \in \mathscr{C}$	the target class used by an attacker
$\mathscr{Y}$	the output space of some classifier, usually $\mathbb{R}^{\mathrm{C}}$
$\widehat{\mathbf{y}}$	the output of classifier, belongs to $\mathbb{R}^{\mathrm{C}}$
$\widehat{\mathbf{y}}^{(i)}$	the prediction for the i-th sample
$\widehat{c}$	the class predicted by a classifier, for example argmax($\widehat{\mathbf{y}}$)
$\widehat{\mathbf{y}}_j$	the j-th component of the prediction $\widehat{\mathbf{y}}$
onehot	the onehot encoding scheme, the following function $\mathscr{C} \rightarrow \{0,1\}^{\mathrm{C}}$ such that onehot(c)$_c$=1 and onehot$_j$=0 for $j \neq c$

Dataset

N	the number of samples in a dataset
$\mathscr{D}$	a dataset, $\left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right)_{i \in \{1,\ldots,\mathrm{N}\}} \in (\mathscr{X}, \mathscr{Y})$
$\mathbf{x}^{(i)}$	the i-th sample of a dataset
$\mathbf{y}^{(i)}$	the ground truth of the i-th sample of the dataset in a onehot format unless stated otherwise
$\mathbf{x}, \mathbf{y}$	an input and its ground truth label in a onehot format. We used this notation instead of using the supercript when there is not ambiguity; $\mathbf{y} = onehot(c)$
$\mathbf{y}_j$	the j-th component of the prediction $\mathbf{y}$

Neural Networks

f	a representation of a neural network or another model, $f(\mathbf{x}) = \left(f_1(\mathbf{x}), \ldots, f_{\mathrm{C}}(\mathbf{x})\right) = \widehat{\mathbf{y}} \in \mathscr{Y}$

D_f	from $\mathscr{Y} \to \mathscr{C}$, for example, $\mathrm{argmax}_i\, f_i$ for classification.
$\widehat{f}$	a representation of neural network in which ReLU constraints are relaxed
$\mathscr{L}$	defines a loss function, usually the cross-entropy
$\mathbf{B}_f(\mathbf{x})$	function mapping the input $\mathbf{x}$ to the binary value corresponding to its linear region according the activations of the network f. Noted $\mathbf{B}(\mathbf{x})$ when there is no ambiguity.
$\mathbf{b}$	a vector of binary values
$\mathrm{ReLU}(x)$	the Rectified Linear Unit activation which returns x when $x \geq 0$ and 0 otherwise

Optimization

$\mathscr{P}$	an optimization problem
$\widehat{\mathbf{x}}$	the incumbent solution of a MIP problem, the current best feasible solution
$\widehat{\mathbf{x}}'$	a feasible solution of a MIP problem
β_{T}	the branching factor, the maximum number of children at a node a search tree (of a branch-and-bound algorithm)
δ_{T}	the search depth of a search tree (of a branch-and-bound algorithm)
$f_{\mathscr{P}}$	the objective function of the optimization problem $\mathscr{P}$
$\mathscr{I}$	the set of unstable ReLUs
$\mathscr{I}^-$	the set of stably inactive ReLUs
$\mathscr{I}^+$	the set of stably active ReLUs
$\mathscr{T}$	a theory in the Satisfiability Modulo Theories.

Miscellaneous

$\mathrm{sgn}(x)$	the sign function applied to x or element-wisely if x is not a scalar

Introduction in french

L'intelligence artificielle en général, et l'apprentissage profond en particulier, sont utilisés dans les voitures autonomes, la médecine, la prévention des fraudes, la prédiction des tremblements de terre, le traitement du langage naturel, les jeux, etc. Certains de ces domaines sont des environnements critiques pour la sécurité, des environnements dans lesquels la sécurité est d'un intérêt primordial et où la défaillance des modèles d'apprentissage profond peut causer des dommages importants, c'est le cas des voitures autonomes et de la médecine. Par exemple, en ce qui concerne les voitures autonomes, la classification erronée d'un panneau d'arrêt comme une limitation de vitesse peut être très dangereuse. Et de manière intrigante, il est possible, en modifiant malicieusement un panneau stop avec des étiquettes ou des patchs, de tromper les classifieurs d'apprentissage profond pour qu'ils prédisent une limitation de vitesse au lieu d'un panneau stop, alors qu'un humain reconnaîtrait toujours un panneau stop. Ces images soigneusement modifiées sont appelées exemples adversaires. Ils apparaissent dans la classification d'images ainsi que dans d'autres domaines, comme le montre le chapitre consacré aux exemples adversaires. L'existence de tels exemples adversaires peut avoir un effet dissuasif sur l'utilisation de l'apprentissage profond, en particulier dans les environnements critiques pour la sécurité. Il n'y a pas encore de consensus sur la raison de l'existence des exemples adversaires. Cependant, leur existence soulève un certain nombre de questions, allant jusqu'à remettre en cause la croyance bien établie selon laquelle les modèles d'apprentissage profond identifient des concepts de plus en plus complexes à mesure que l'on passe de la couche d'entrée vers la couche de sortie. Certaines exigences sont nécessaires pour rendre davantage populaire l'utilisation de l'apprentissage profond en particulier et de l'IA en général, comme le stipule les *lignes directrices éthiques pour une IA digne de confiance* de l'Union européenne : "Les systèmes d'IA doivent être plus fiables, suffisamment sûrs pour résister à la fois aux attaques manifestes et aux tentatives plus subtiles de manipulation des données." Il soulève une série de besoins auxquels doivent répondre les algorithmes d'apprentissage profond pour encourager leur utilisation à l'avenir :

- Transparence : les utilisateurs doivent savoir que des modèles d'apprentissage profond sont utilisés et connaître les forces et les limites de ces modèles.
- Explicabilité/interprétabilité, les décisions des modèles d'apprentissage profond doivent être interprétables. Les décisions des modèles doivent pouvoir être expliquées aux humains. Les humains doivent être en mesure de connaître le raisonnement qui sous-tend les décisions du modèle.
- Les modèles doivent être précis et leurs performances reproductibles, ou du moins afficher des résultats comparables lorsqu'ils sont exécutés sur les mêmes données. Ils doivent également être fiables, leurs décisions devant rester correctes même lorsque les données ont été malicieusement manipulées pour les tromper.

Comme il est très facile de tromper les modèles lorsqu'ils ne sont pas entraînés pour être robustes aux attaques adverses, ce qui remet en question leur fiabilité, des mesures ont été prises pour résoudre ce problème. Des méthodes d'entraînement qui rendent les réseaux neuronaux moins susceptibles d'être trompés par des exemples adversaires sont en cours de développement, allant de l'idée de générer des exemples adversaires et de s'entraîner sur ces derniers, à des procédures plus sophistiquées. Ces procédures sont appelées "défense". Par opposition aux procédures utilisées pour trouver des exemples adversaires, qui sont appelées attaques. Une course aux armements entre les attaques adversaires et les défenses adversaires s'est produite. Chaque fois qu'une défense semblait être robuste face aux attaques existantes au moment de la publication, ses vulnérabilités étaient exposées quelque temps plus tard par une nouvelle attaque. Et finalement, cette nouvelle attaque échouera contre une défense plus récente. Cette situation montre que, si l'évaluation des attaques contre les défenses peut suffire à évaluer l'efficacité des attaques, ce n'est pas le cas pour les défenses. En effet, une défense peut être très efficace contre une famille

d'attaques alors qu'elle ne rend pas vraiment plus robustes les modèles, mais utilise plutôt des astuces qui rendent difficile pour une certaine famille d'attaques la recherche des exemples adversaires. Dès lors, comment évaluer l'efficacité des méthodes de défense adversaires si l'utilisation d'attaques ne suffit pas ? Des méthodes de vérification formelle sont utilisées. Elles consistent à formuler la recherche d'exemples adversaires comme un problème d'optimisation ou comme un problème dans un autre cadre mathématique qui est capable de trouver les exemples adversaires que les autres attaques peinent à trouver. Les vérifications formelles sont très puissantes, mais elles ont un inconvénient majeur, elles prennent généralement beaucoup de temps.

Cette thèse se concentre sur la fiabilité des modèles. Ses contributions portent sur les mécanismes de défense et les moyens de mesurer plus précisément la robustesse des modèles à un moindre coût en termes de temps de calcul.

Structure de la thèse: Cette thèse est composée de deux parties : *Pré-requis et outils* (*background and tools*) et *nos contributions* (*our contributions*). La première partie est composée de deux chapitres : Outils d'optimisation et Exemples adversaires : un tour d'horizon.

Dans le chapitre *Outils d'optimisation*, nous présentons des méthodes telles que les programmes linéaires qui sont utilisées dans certaines méthodes de vérification ainsi que dans certaines de défenses contre les exemples adversaires. Nous présentons ensuite la méthode Branch-and-Bound qui peut être utilisée pour résoudre certains problèmes qui ne peuvent pas être résolus par des programmes linéaires. Nous présentons ensuite la Satisfiabilité Modulo Théories (SMT) qui est une généralisation du problème de la satisfaction booléenne, le problème SAT. Nous présentons également les programmes linéaires entiers (ILP, Integer Linear program) ainsi que les programmes mixtes entiers (MIP, Mixed Integer Program) pour résoudre les programmes linéaires comme les problèmes d'optimisation avec des valeurs discrètes. MIP et SMT utilisent une sorte de Branch-and-Bound et sont utilisés pour vérifier formellement la robustesse des réseaux de neurones. Nous présentons également l'algorithme de Différence de fonctions convexe (DCA, Difference of Convex algorithm) qui peut être utilisé pour essayer de trouver des exemples adversaires qui sont aussi proches des exemples originaux que ceux trouvés par les méthodes de vérification formelle tout en étant moins coûteux en temps. L'algorithme DCA est basé sur une reformulation de la formulation MIP qui relaxe les contraintes entières tout en pénalisant la solution qui ne sont pas entières. Il s'est avéré efficace sur certains problèmes MIP classiques où il parvient à trouver des solutions optimales.

Le deuxième chapitre commence par des illustrations montrant certains domaines où l'apprentissage profond est utilisé. Ensuite, un problème général de classification est posé, suivi de la sous-section des exemples adversaires. Dans cette sous-section, une terminologie sur les exemples adversaires est donnée ainsi que quelques hypothèses expliquant leur existence. Ensuite, les attaques adversaires sont présentées, montrant les différents types de perturbations adversaires et la manière dont elles sont mesurées, donnant une taxonomie des attaques, suivie de quelques attaques tirées de la longue liste d'attaques existantes. Ces attaques sont choisies en fonction de leur popularité, de leur originalité ou de leur efficacité. Nous présentons ensuite les défenses adversaires en partant d'une formulation de la classification en présence d'exemples adversaires, une taxonomie des défenses, puis nous divisons les défenses en deux parties : les défenses empiriques qui sont les plus courantes et les *défenses prouvables* qui viennent avec une preuve plus formelle de robustesse. Après cela, nous introduisons les méthodes de vérification formelle, puis nous parlons des limites théoriques de la robustesse aux exemples adversaires. Enfin, nous présentons quelques outils à utiliser pour travailler sur les exemples adversaires.

Dans la deuxième partie, nous présentons nos contributions en trois chapitres, un chapitre présentant un mécanisme de défense (chapitre trois), et deux chapitres dans lesquels nous présentons des attaques qui sont dérivées d'une formulation MIP de la vérification de la robustesse. Le chapitre trois vise à produire un réseau plus robuste en combinant l'entraînement adversaire avec l'idée de double rétro-propagation, d'où le nom, L_1*-norm double backpropagation adversarial defense*. La double rétropropagation est le fait de pénaliser le gradient de la sortie par rapport à

l'entrée pour obtenir des modèles qui généralisent mieux.

Après avoir travaillé sur un mécanisme de défense, nous nous intéressons aux moyens de mesurer efficacement sa robustesse. Le MIP, étant la formulation de vérification de la robustesse la plus populaire en raison de sa précision, sert de point de départ à partir duquel nous essayons de trouver des moyens de le rendre plus efficace, de le rendre moins chronophage. Nous utilisons une approche de relaxation et de pénalisation, en relaxant les contraintes d'intégrité qui rendent le MIP lent et en introduisant une pénalisation pour obtenir l'intégrité en utilisant l'algorithme de différence de fonctions convexes (DCA) au chapitre quatre. Assez rapidement, cette approche a atteint sa limite, cependant elle nous a conduit à une autre attaque, également dérivée de la vérification du MIP et qui est très efficace, *Linear Program Powered Attack* (LiPPA) qui est notre dernier chapitre, le chapitre cinq.

Le chapitre cinq présente une nouvelle attaque, LiPPA, qui exploite les propriétés des réseaux neurones avec des activations ReLUs. En effet, l'espace d'entrée de ces réseaux de neurones peut être partitionné astucieusement de sorte que l'exemple adversaire le plus proche d'un exemple valide puisse être retrouvé juste en résolvant un programme linéaire. Dans ce chapitre, nous faisons aussi le lien entre cette partition et les variables binaires qui sont utilisées dans la formulation MIP de la vérification de la robustesse des réseaux de ces neurones. Ce chapitre ouvre la porte à de nombreuses perspectives quant à l'accélération des méthodes de vérification.

Introduction

> " *AI systems need to be more reliable, secure enough to be resilient against both overt attacks and more subtle attempts to manipulate data.* "
>
> EU's ethics guidelines for trustworthy AI

Artificial Intelligence in general, and especially Deep Learning, are used in self-driving cars, medicine, fraud prevention, earthquake prediction, natural language processing, game playing, etc. Some of these areas are safety-critical environments, environments in which safety is of paramount interest and where the failure of deep learning models can cause significant damage, such as self-driving cars and medicine. For example, regarding self-driving cars, the misclassification of a stop sign as a speed limitation can be very dangerous. And intriguingly, it is possible by maliciously modifying a stop sign with tags or with patches, to fool deep learning classifiers to predict a speed limitation instead of a stop sign while a human would still recognize a stop sign. Those carefully modified images are called adversarial examples. They occur in image classification as well as in other domains as it is shown in the adversarial example overview chapter. The existence of such adversarial examples can have a deterrent effect on the use of deep learning, particularly in safety-critical environments. There is not yet a consensus on the reason of the existence of adversarial examples. However, it raises a number of questions, going as far as questioning the well-established belief that deep learning models identify increasingly complex concepts as we go from the input layer to the output layer. Some requirements are necessary to popularize even more the use of deep learning in particular and AI in general as stated in the European Union's *ethics guidelines for trustworthy AI* : "AI systems need to be more reliable, secure enough to be resilient against both overt attacks and more subtle attempts to manipulate data." It raises a series of needs that need to be addressed by deep learning algorithms to encourage its use in the future:

- Transparency, the users should know that deep learning models are used, and be aware of the strengths and limitations of the models.
- Explainability/interpretability, the decisions of the deep learning models are required to be interpretable. Models' decisions have to be explainable to humans. Humans should be able to know the reasoning behind the decisions of the model.
- Trustworthiness/reliability, models should be accurate and their performance reproducible, at least they should display comparable results when run on the same data. They should also be reliable, their decision should stay correct even when the data have been maliciously manipulated to fool them.

Since it is very easy to fool models when they are not trained to be robust to adversarial attacks, questioning their reliability, steps have been taken to address this issue. Training methods that make neural networks less prone to be fooled by adversarial examples are being developed,

going from the idea of generating adversarial examples and training on them, to more sophisticated procedures. Those procedures are called defenses. As opposed to the procedures used to find adversarial examples that are called attacks. An arms race between adversarial attacks and adversarial defenses occurred. Whenever a defense seemed to be robust to existing attacks at publication time, its vulnerabilities were exposed some time later by a new attack. And eventually, that new attack will fail against a newer defense. This situation shows that, if evaluating attacks against defenses can be enough to assess the effectiveness of attacks, it is not the case for defenses. In fact, a defense may be very effective against a family of attacks while it does not truly make more robust models, but rather uses tricks that make it hard for a certain family of attacks to find adversarial examples. So, how can we assess the effectiveness of adversarial defense methods if using attacks is not enough? Formal verification methods are used. They consist of formulating the search of adversarial examples as an optimization problem or as a problem in another mathematical framework that is able to find the adversarial examples that the attacks struggle to find. The formal verifications are very powerful, but they have a main drawback, they are generally time-consuming.

This thesis focuses on the reliability of the models. Its contributions are on the defense mechanisms and ways to measure more accurately the robustness of the models.

Structure of the thesis: This thesis is made of two parts : *Background and tools* and *our contributions*. The first part is made of two chapters : *Optimization tools* and *Adversarial examples: an overview*.

In the chapter *Optimization tools*, we present methods such as linear programs that are used in some verification methods as well as in some defenses. Then we present the Branch-and-Bound method that can be used to solve some problems that cannot be handled by linear programs. We then present Satisfiability Modulo Theories (SMT) which are a generalization of the Boolean satisfying problem. We also present Integer Linear Programs (ILP) as well as Mixed Integer Programs (MIP) to solve Linear Programs like optimization problems with discrete values. MIP and SMT use some kind of Branch-and-Bound and are used to formally verify the robustness of neural networks. We also present Difference of Convex Algorithms (DCA) that can be used to try to find adversarial examples that are as close to original examples as those found by the formal verification methods while being less time-consuming. DCA is based on a reformulation of the MIP formulation that relaxes the integer constraints while penalizing the non-integer solutions. It has proven effective on some classical MIP problems where it manages to find solutions that are optimal.

The second chapter starts with illustrations showing some domains where deep learning is used. Then a general classification problem is set, followed by the adversarial examples subsection. In this subsection, some terminology about adversarial examples is given as well as some hypotheses explaining their existence. Then adversarial attacks are presented, showing the different kinds of adversarial perturbations and of they are measured, giving a taxonomy of attacks, followed by some attacks from the long list of existing attacks. Those attacks are chosen according to their wide use, their originality or their effectiveness. Then we present adversarial defenses starting with a formulation of adversarial classification, a taxonomy of defenses, we then divide the defenses into two parts the empirical defenses that are most common and provable defenses that come with a more formal proof of robustness. After that, formal verification methods are introduced. Then, we talk about the theoretical limits on adversarial robustness and finally we present some tools to use when it comes to adversarial examples.

In the second part, we present our contributions in three chapters, a chapter presenting a defense mechanism (Chapter 3), and two chapters in which we present attacks that are derived from a MIP formulation of the robustness verification. Chapter 3 aims at producing more robust networks by combining the adversarial training with the double backpropagation idea, hence the name, L_1*-norm double backpropagation adversarial defense*. Double-backpropagation is the fact of penalizing the gradient of the output with respect to the input to obtain models that generalize better. After working on a defense mechanism, we get interested in the ways to measure efficiently

its robustness. The MIP being the most popular robustness verification formulation due to its accuracy serves as starting point as we try to find ways to make it run faster. We use a relaxation and penalization approach, relaxing the integrity constraints that make the MIP slow and introduce a penalization to obtain the integrity using Difference of Convex Algorithm (DCA) in Chapter 4. This approach reached its limit quickly enough, however it led us to another attack, also derived the MIP verification and that is very efficient, Linear Program Powered Attack (LiPPA) that is our last chapter, Chapter 5.

In chapter 5, LiPPA is presented, a new attack which exploits the properties of neural networks with ReLU activations. Indeed, the input space of these neural networks can be partitioned in a clever way allowing to find in regions of this partition the closest adversary example just by solving a linear program. In this chapter, we also relate this partition to the binary variables that are used in the MIP formulation of the robustness verification of these neural networks. This chapter opens the door to many perspectives on the acceleration of verification methods.

Publications

International conferences:

- **Linear Program Powered Attack**. Ismaila Seck, Gaëlle Loosli, and Stephane Canu. International Joint Conference on Neural Networks (IJCNN),2021.
- L_1**-norm double backpropagation adversarial defense**. Ismaila Seck, Gaëlle Loosli, and Stephane Canu. European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN). 2019.

French conferences:

- **Linear Program Powered Attack**. Ismaila Seck, Gaëlle Loosli, and Stephane Canu. Conférence d'Apprentissage 2021.
- L_1**-norme double rétropropagation**.Ismaila Seck, Gaëlle Loosli, and Stephane Canu. Conférence d'Apprentissage 2019.
- **Generative Adversarial Networks and Cerema AWP database**. Ismaila Seck and Gaëlle Loosli. European Network for Business and Industrial Statistics 2018.
- **Baselines and a datasheet for the Cerema AWP dataset**. Ismaïla Seck, Khouloud Dahmane, Pierre Duthon, and Gaëlle Loosli. Conférence d'Apprentissage 2018.

Part I

Background and tools

Background and tools: Introduction

In the part, we are going to present optimization tools used in adversarial example field to verify the robustness of neural networks or to find adversarial examples. This part is made of two chapters, the first one dealing with the optimization tools and the second one presenting adversarial examples.

Chapter 1

Optimization tools

Contents

1.1 Introduction

In this chapter, we present some tools sufficient to enable a good comprehension of the methods used throughout this manuscript. Methods presented here are Linear Programs (LPs), Mixed Integer Programs (MIPs), Satisfiability Modulo Theories (SMT), and Difference of Convex Algorithm. The MIPs and SMT are then seen as a special case of the Branch-and-Bound (BB), that is also presented.

1.2 Linear Programs

1.2.1 Definition and illustration

Definition 1.1 (Linear Programs)**.** *Linear Programs are optimization problems where the objective function and the constraints are linear with respect to its variables that are continuous.*

LPs are usually of the form, or can be rewritten to fit the form [Wolsey and Nemhauser, 1999]:

$$\begin{cases} \max\limits_{\mathbf{x}} & \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} \leq \mathbf{b}, \\ & \mathbf{x} \geq 0, \end{cases} \tag{1.1}$$

with $\mathbf{x}$ the variables of the optimization problem, $\mathbf{c}$ and $\mathbf{b}$ are vectors, and $\mathbf{A}$ is a given matrix. The function that associates $\mathbf{x}$ to $\mathbf{c}^\top\mathbf{x}$ is the objective function. The constraints $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ and $\mathbf{x} \geq 0$ define the set of feasible solutions which is a convex polytope. Let us give an illustration of how a real life problem can be modeled as a linear program to maximize a profit.

Example 1.1 (Example of linear program)**.** *A farmer produces two types of strawberry yogurt using strawberries, milk, and sugar. To produce one kilogram of type 1 yogurt, which is sugar free, two kilograms of strawberries and one kilogram of milk are used. To produce one kilogram of the type 2, one kilogram of strawberries, two kilograms of milk, and one kilogram of sugar are used. The type 1 yogurt is sold 4€ and the type 2, 5€. The farmer has 800 kg of strawberries, 700 kg of milk, and 300 kg of sugar.*

Let x_1 and x_2 be, respectively, the quantity in kilograms of type 1 and type 2. Which quantity should the farmer produce to maximize the selling price?

The problem is modeled in a linear program:

- *The selling prize, the objective function, is $4x_1 + 5x_2$.*
- *The constraint on the available quantity of strawberries: $2x_1 + x_2 \leq 800$.*
- *The constraint on the available quantity of milk: $x_1 + 2x_2 \leq 700$.*
- *The constraint on the available quantity of sugar: $x_2 \leq 300$.*
- *Constraints on production: positivity constraints: $x_1, x_2 \geq 0$.*

The problem is written:

$$\begin{cases} \max\limits_{x_1,x_2} & 4x_1 + 5x_2 \\ s.t. & 2x_1 + x_2 \leq 800 \\ & x_1 + 2x_2 \leq 700 \\ & x_2 \leq 300 \\ & x_1, x_2 \geq 0 \end{cases} \tag{1.2}$$

The canonical form can be obtained identifying: $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, $\mathbf{c}^\top = \begin{bmatrix} 4 & 5 \end{bmatrix}$, $\mathbf{A} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \\ 0 & 1 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} 800 \\ 700 \\ 300 \end{bmatrix}$.

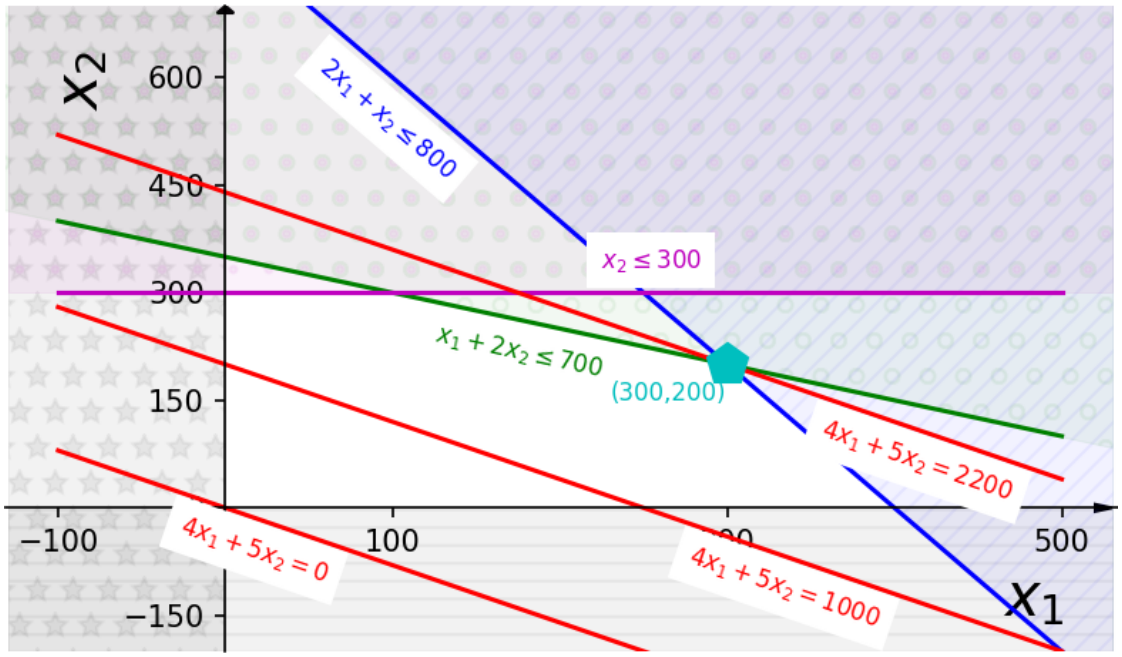


Figure 1.1 – Graphical Resolution of a Linear Program. The constraints are represented by the corresponding line and the region not respecting them are hatched. The objective function is represented in red. Sliding its line in the direction maximizing its value leads to the optimal solution: (300, 200).

1.2.2 Dual a Linear Program

Considering that Problem (1.1) to be primal, the dual optimization problem problem associated is of the form:

$$\begin{cases} \min\limits_{\mathbf{u}} & \mathbf{b}^\top \mathbf{u} \\ \text{s.t.} & \mathbf{A}^\top \mathbf{u} \geq \mathbf{c}, \\ & \mathbf{u} \geq 0, \end{cases} \tag{1.3}$$

Nevertheless, since the dual of the dual is the primal, it is not essential which one is called primal or dual.

Proposition 1.1 (Weak Duality)**.** *If* $\mathbf{x}$ *is a solution of the primal (Problem* (1.1)*) and* $\mathbf{u}$ *is a solution of the dual (Problem* (1.3)*), then we have* $\mathbf{c}^\top \mathbf{x} \leq \mathbf{b}^\top \mathbf{u}$.

The primal-dual relation allows to have bounds. At the optimum (if it exists), this inequality of the Weak Duality proposition is satisfied as an equality as stated by the following theorem.

Theorem 1. *If* (1.1) *and* (1.3) *have finite solutions* $\mathbf{x}^*$ *and* $\mathbf{u}^*$*, then* $\mathbf{c}^\top \mathbf{x}^* = \mathbf{b}^\top \mathbf{u}^*$.

This theorem can be used to prove that an optimal solution is found.

1.2.3 Resolution of Linear Programs

There are several ways to solve a linear program. A linear program can have a graphical solution. In that case, the polytope of feasible solutions is drawn and the optimal solution is found by a sliding the line representing the objective function in the direction maximizing the profit until the optimal solution is reached. Usually, the optimal solution is a vertex, a corner of the feasible polytope. An illustration is given in Figure 1.1. We can see in that figure, the lines represent the constraints and the polytope formed by the feasible region. We can also see the line representing the reward that is slided in the direction that maximizes the reward until the optimal point is reached.

It is not always possible to have a graphical solution due the dimension of the variables $\mathbf{x}$ which can reach several thousands for some problems. In that case, the most popular methods to solve linear programs are interior-point methods and simplex algorithms.

Interior-point methods One strong point of the interior-point method is that they are polynomial for some class of optimization problems, in particular linear programs. They are iterative methods that find a better solution at each iteration within the polytope. They are competitive with the simplex methods.

Simplex algorithms The simplex is the most popular method to solve linear programs. This method has been preferred to interior-point methods' implementations that were not yet efficient. It constructs a series of feasible solutions by visiting the vertices of the feasible polytope. In practice, The simplex method is very efficient, however, the worst-case complexity of the simplex is exponential time.

Example of optimization problem that are not Linear Programs. All problems cannot fit the linear program canonical formulation, i.e., they are not linear programs. For example, some constraints could have polynomial with respect to the variables $\mathbf{x}$, in the case the polynomials of degree two, we get a quadratic program. Or we could get a variable taking values in several intervals that are not contiguous. Another case is when we have functions that are piecewise linear in the constraints or as the objective function. Some variables could belong to a discrete set, for example $\mathbb{N}$, in that case, Branch-and-Bound can be used to solve the optimization problem.

1.3 Branch-and-Bound

Branch-and-bound (BB) algorithms have been successfully used to find exact solutions for a wide range of optimization problems. BB uses a tree search strategy to implicitly list all possible solutions to a particular problem, using trimming rules to exclude regions of the search space that cannot lead to a better solution[Morrison et al., 2016]. Let us define an minimization problem as $\mathscr{P} = (\mathrm{X}, f_{\mathscr{P}})$, where:

- X, the search space, is a set of valid solutions to the problem,
- $f_{\mathscr{P}} : \mathrm{X} \to \mathbb{R}$ is the objective function that is minimized.

The goal is to find an optimal solution $\mathbf{x}^* \in \underset{\mathbf{x}\in\mathrm{X}}{\operatorname{argmin}} f_{\mathscr{P}}(\mathbf{x})$. To solve $\mathscr{P}$, BB iteratively builds a search tree T of subsets of the search space or subproblems. The best feasible solution, the incumbent solution, $\widehat{\mathbf{x}} \in \mathrm{X}$, is stored. At each iteration, the algorithm selects a new subproblem $\mathrm{S} \subseteq \mathrm{X}$ to explore from a list L of unexplored subproblems. If a candidate incumbent solution $\widehat{\mathbf{x}}' \in \mathrm{S}$ can be found with a better objective value than $\widehat{\mathbf{x}}$, i.e. $f_{\mathscr{P}}(\widehat{\mathbf{x}}') < f_{\mathscr{P}}(\widehat{\mathbf{x}})$, the incumbent solution is updated. On the other hand, if it can be proven that no solution in S has a better objective value than $\widehat{\mathbf{x}}$, i.e. $\forall \mathbf{x} \in \mathrm{S}, f_{\mathscr{P}}(\mathbf{x}) \geq f_{\mathscr{P}}(\widehat{\mathbf{x}})$, the subproblem is pruned.

Otherwise, child subproblems are generated by partitioning S into an exhaustive set of subproblems $\mathrm{S}_1, \mathrm{S}_2$, etc which are then inserted into T, the search tree. Once no unexplored subproblem remains, the best incumbent solution is returned. Since subproblems are only pruned if they contain no solution better than $\widehat{\mathbf{x}}$, it must be the case that $\widehat{\mathbf{x}} \in \underset{\mathbf{x}\in\mathrm{X}}{\operatorname{argmin}} f(\mathbf{x})$. A pseudocode for the common BB algorithm is given in algorithm 1.

Branching Branching refers to the fact that at a given node the search space is divided into smaller regions. During the branching, no feasible solution should be lost since it could mean missing the optimal solution. The branching rules used in practice give a guarantee on keeping all feasible solutions. If we were to apply only branching, we would branch on all nodes containing more than one solution, until all leaves are reached. The use of bounding helps to avoid such a prohibitive task.

Bounding The part that allows to discard a search region is the bounding and helps reduce the time spent branching on nodes that will not give a better solution. At a given node, a bound of

Algorithm 1 : Branch-and-Bound(X, $f_{\mathscr{P}}$)

```
Input: X and f_P
Result : x̂ and a lower bound on the best solution
Set L = {X}
while L ≠ ∅ do
    Select a subproblem S from L to explore
    if a solution x̂' ∈ {x ∈ S : f_P(x) < f_P(x̂)} then
        Set x̂ = x̂'
    end
    if S cannot be pruned then
        Partition S into S_1, S_2, ...
        Insert S_1, S_2, ... into L
    end
    Remove S from L
end
Return x̂
```

the objective function is evaluated. And if after evaluation we note that from that a better solution cannot be found from that node, then there is no need to continue branching on that node. Computing good bounds can help reduce importantly the time spent branching. However, the computing time of the bounds should be very low since the evaluation of bounds are repeated a large number of times.

BB and termination The algorithm 1 is guaranteed to terminate if X is finite and the partitioning procedure creates child subproblems $S_1, S_2, \ldots$ that are proper subsets of S at each subproblem S. Let us recall the definition of a proper set: a set S_1 is a proper subset of S, noted $S_1 \subsetneq S$, if S_1 is a subset of S but S_1 is not equal to S. In other words, there is at least one element of S which does not belong to S_1. The order in which the subsets are selected depends on the search strategy that is discussed in Section 1.3.1.

BB worst-case time complexity Remark that the set X and all subproblems are usually given implicitly. That is, given an element $\mathbf{x}$, membership in a subproblem can be verified without difficulty, and it is not necessary to know all the members of X or its subproblems to partition them. The complexity of BB algorithm is linked to two factors:

- the branching factor β_T of the search tree, the maximum number of children generated at any node of the tree,
- the search depth δ_T of the search tree, the length of the longest path from the root of T to a leaf, a node with no children.

BB algorithms operate in $\mathscr{O}(M\beta_T^{\delta_T})$ worst-case running time, where M is a bound on the needed time to explore a subproblem. This worst-case time complexity can be prohibitive even for some toy problems. For example, in our case where binary values are used, if we do a binary branching, $\beta_T = 2$ and $\delta_T = e$ is the number of ReLUs in the model (or the number of unstable ReLUs, see the end of the paragraph 5.4.1), which is $\mathscr{O}(M\,2^e)$. Nevertheless, the pruning happening during optimization can drastically improve the algorithm performance.

BB phases There are two significant phases of all BB algorithms, the search phase and the verification phase, as shown in figure 1.2. The search phase is the phase in which the algorithm has not yet found an optimal solution. The verification phase is the phase in which the incumbent solution is optimal but there are still unexplored subproblems in the tree that cannot be pruned. The incumbent solution can not be proven to be optimal until no unexplored subproblems remain. Moreover, the delimitation between the search phase and the verification phase is not known until the algorithm terminates. In the case that BB has completed the verification phase, it is said

to have produced a certificate of optimality. This certificate of optimality and the guarantee it gives are the main reason MIP are used to find the minimal adversarial distortion and adversarial verification.

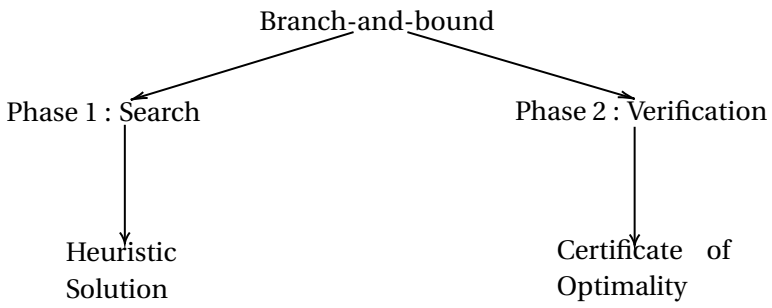


Figure 1.2 – Phases of the branch-and-bound algorithm [Morrison et al., 2016].

Reasons to improve BB search phase performance There are two main considerations to improve the BB algorithm during the search phase:

- First, if the algorithm terminates without producing a certificate of optimality, for example, the time limit is reached, the incumbent solution can still be returned as an heuristic solution. In our case, an heuristic solution would correspond to an adversarial example, and its distance to the original example can be satisfying even though it is not guaranteed that the adversarial example is at the minimal adversarial distortion distance. In that case, the search phase of the BB can be seen as an attack.
- Secondly, finding an optimal solution earlier, the search phase has a direct impact on the size of the search tree and therefore on the required time to certify the optimality since all nodes with bounds greater than the optimum value are pruned. That is intuitively the reasoning behind best-first search strategy [Dechter and Pearl, 1985] that is shown to explore the fewest number of subproblems of any search strategy.

Our attack can be seen as improvement of the search phase of the BB search phase. With our attacks, it is possible to initialize the BB to an optimal or near-optimal solution.

1.3.1 Search strategies

The exploration order of unexplored subproblems of T is determined by the search strategy. There mainly two options using a priori rules, for example Depth-first search, or using adaptive rules, for example best-first search. We discuss the common search strategies along with their strengths and weaknesses.

Depth-first search The Depth-first search (DFS) strategy is a search strategy that can be implemented by maintaining the list of unexplored subproblems L as a stack. The algorithm removes the top item from the stack to take the next subproblem to explore, and when children are generated as a result of branching, they are inserted on the top of L. Thus, the next subproblem that is explored is the most recently generated subproblem, last in first out. Experience seems to indicate that feasible solutions are more likely to be found deep in search trees than at nodes near the root. Depth first search is the default option in most commercial codes [Wolsey and Nemhauser, 1999].

The use of depth-first search strategy has two drawbacks:

- The first drawback is that naive implementations of DFS do not use any information about the problem structure or bounds, as a result, the search process can spend a long time exploring regions of the search space that cannot produce a better solution than the current one. Another similar phenomenon is the fact that different regions of the search space fail for the same reasons, for example, one branching constraint leading to infeasibility, but the algorithm must explore many subproblems before finding infeasibility.

- The second drawback is observed when the the search tree is excessively unbalanced. That is, if some optimal solutions are close to the root, but there exist paths in the search tree T that do not lead to an optimal solution. DFS can unfortunately explore numerous long and bad paths before it explores a path leading to an optimal solution, although this computation time often could be avoided via pruning if the search strategy instead chooses to explore a short optimal path first.

A myriad of variants to the DFS exist that try to get around those flaws.

Breadth-first search Breadth-first search (BrFS) is the opposite of DFS for the reason that it is implemented with a queue or the first-in-first-out data structure. With BrFS, all subproblems that are at the same depth are explored before exploring any deeper subproblems.

One advantage of BrFS is that optimal solutions that are close to the root are discovered quickly. Therefore, it produces good performance on unbalanced search trees. However, since complete solutions are generally at larger depths, BrFS is usually unable to exploit pruning rules that compare to the current incumbent solution. For this reason, the memory requirements for BrFS are quite high, and it is generally not used in a BB context. Two exceptions are in the presence of dominance relations, and a good heuristic finds a good incumbent solution that allows pruning. We talk about the dominance relation between two subproblems S_1 and S_2 if for any feasible solution from S_2 or its descendants, it is possible to find a solution of S_1 or its descendants that is at least as good.

Best-first search In settings where sufficient memory is accessible to store the complete unexplored search tree, the best-first search (BFS) strategy is usually used. This strategy uses a heuristic measure-of-best function μ that computes a value $\mu(S)$ for each unexplored subproblem, and selects the one minimizing μ as the subsequent subproblem to explore. There are many choices for the measure-of-best function, one common choice may be a bound on the quality of the most effective solution in the subproblem. Best-first search offers a variety of serious advantages over DFS. It is often able to find good solutions earlier within the search process because it is not tied to exploring one specific branch of the tree before the other.

1.3.2 Lower bound and Branch-and-bound

The ability to increase the lower bound to quickly match the optimal solution is crucial if we want to find the exact solution of a problem without the algorithm spending hours and hours. Two aspects of the BB affect the lower bound, the pruning rules, and the branching strategy. Several pruning rules exist, but they are often problem specific and are focused on integer variables, binary variables in our case. The most natural approach to prune is to compute a lower bound on the objective function value for each subproblem and use this to prune subproblems having a lower bound that is not better than the incumbent solution value. Some aspects of the problem are relaxed to compute lower bounds. For instance, in our verification problem, lower bounds are computed by relaxing the binary constraints and allowing values between 0 and 1 instead of being either 0 or 1. Relaxing the integrality constraints is a straightforward way of computing lower bounds, even though it frequently leads to very loose lower bounds. And the speed at which the lower bounds increase depends on the search strategy and the pruning rules. And illustration of the search strategy is given in Figure 1.3.

According to the illustration in Figure 1.3, the search could end after 8 steps for DFS, 9 steps for BrFS, and 6 steps for the BFS. At the same time, if the initial incumbent solution was 9, at most 2 iterations would have been taken before termination of the search, and several branching would not have been necessary. This also illustrates the importance of having a heuristic able of quickly finding a good solution, as it is the case when we use LiPPA to find adversarial examples that could be optimal without having to go through branching and bounding since it exploits previously unused information. Having an efficient way to compute tight lower bounds for the BB can also speed up the verification phase a lot.

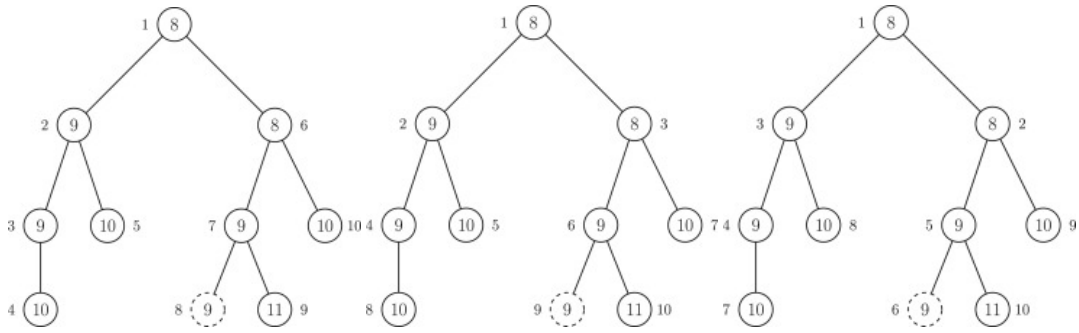


Figure 1.3 – Illustration of the search strategies and the evolution of the lower bound. On the left we have an illustration of the DFS, in the middle we have an illustration of the BrFS, and on the right an illustration of the BFS. The dashed subproblem is optimal, the numbers inside the nodes are subproblem lower bounds, and numbers outside the nodes indicate the exploration order. The algorithm starts with an incumbent solution of value 10. BFS uses the lower bound as the measure-of-best, with ties broken arbitrarily. Note that this requires that a subproblem's lower bound is computed prior to inserting it into the list of unexplored subproblems [Morrison et al., 2016].

In this section, we have seen that the MIP used as a verification method can be assimilated to the execution of a BB algorithm. We have seen that the BB has two phases, a phase in which the optimal solution is searched and the verification phase where the optimal solution is found but the certificate of optimality is not yet found. Two intuitive ways of improving the performance of MIP are having better heuristics and pair them with a tighter lower bound approximation of the objective function. Our algorithm LiPPA fits well in the first way since it is derived from the MIP formulation of minimal adversarial distortion and exploits information that are not directly available to the MIP to potentially find the optimal solution, thus leading to pruning considerable regions of the search space.

1.4 Satisfiability Modulo Theories

SMT is a generalization of SAT problem to more complex theories than the boolean one. It is therefore intuitive to start by stating the SAT problems and how they are solved in practice. More information are available in [Biere et al., 2009].

1.4.1 (Boolean) satisfiability problem

Reminder on Boolean algebra We will start with a reminder on Boolean algebra.A Boolean variable is a variable that can take the values True or F$alse$, also noted 1 and 0. There are three basic operations used in Boolean algebra, all other operations can be expressed using only those three:

- the conjunction, AND noted $\wedge$ such that:

$$x_1 \wedge x_2 \begin{cases} 1, & \text{if } x_1 = x_2 = 1 \\ 0, & \text{otherwise} \end{cases} \tag{1.4}$$

- the disjonction, OR noted $\vee$ such that:

$$x_1 \vee x_2 \begin{cases} 0, & \text{if } x_1 = x_2 = 0 \\ 1, & \text{otherwise} \end{cases} \tag{1.5}$$

- the negation, NOT noted $\neg$ such that:

$$\neg x \begin{cases} 1, & \text{if } x = 0 \\ 0, & \text{if } x = 1 \end{cases} \tag{1.6}$$

Table 1.1 – Truth tables of $\wedge$,$\vee$ and $\neg$.

x_1	x_2	$x_1 \wedge x_2$	$x_1 \vee x_2$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

x	$\neg x$
0	1
1	0

The truth tables of those basic operations can be found in Table 1.1.

The basic operations can also be expressed with ordinary arithmetical operations (addition and multiplication) or with minimum and maximum functions:

$$\begin{aligned} x_1 \wedge x_2 = &\quad x_1 x_2 = \min(x_1, x_2) \\ x_1 \vee x_2 = &\quad x_1 + x_2 - x_1 x_2 = \max(x_1, x_2) \\ \neg x = &\quad 1 - x \end{aligned} \tag{1.7}$$

Definition 1.2 (Boolean formula)**.** *A Boolean formula is made of Boolean variables, Boolean operators, and parenthesis and takes the value* True *or* False *(respectively 1 or 0) when its boolean variables are assigned to* True *or* False*.*

Example 1.2 (Example of a boolean formula)**.** $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

Each one of the sequences in that example is a clause, a disjunction of Boolean variables. A Boolean variable is called a positive literal and its negation is called a negative literal.

Definition 1.3. *A formula is said to be in conjunctive normal form when it is a conjunction of clauses or a single clause.*

The Boolean formula in Example 1.2 is in conjunctive normal form. This form will be used to deal with satisfiability problems since every Boolean formula can be developed into an equivalent conjunctive normal form.

Definition 1.4 (SAT)**.** *A Boolean satisfiability problem, also called propositional satisfiability problem or SAT, is a problem of determining, given a Boolean formula (and its variables), whether or not there is a way to assign the variables to* True *or* False *such that the formula is evaluated as* True*.*

Definition 1.5 (n-SAT)**.** *When a formula is a conjunction of clauses of at most* n *literals, finding if it is satisfiable or not is known to be an* n*-SAT problem.*

SAT is the first problem that was shown to be NP-Complete[1]. A SAT problem can be solved using BB algorithm as shown in the following example.

Example 1.3 (SAT and BB)**.** *Let* $\mathrm{S}(x_1, x_2, x_3, x_4)$*, defined as follow, be the formula we want to satisfy:*

$$\begin{aligned} \mathrm{S}(x_1, x_2, x_3, x_4) = &\underbrace{(x_1 \vee x_2 \vee x_3)}_{\mathrm{C}_1} \wedge \underbrace{(\neg x_1 \vee \neg x_3 \vee x_4)}_{\mathrm{C}_2} \wedge \underbrace{(x_2 \vee \neg x_3 \vee x_4)}_{\mathrm{C}_3} \wedge \\ &\underbrace{(\neg x_1 \vee \neg x_2 \vee x_3)}_{\mathrm{C}_4} \wedge \underbrace{(\neg x_1 \vee \neg x_3 \vee \neg x_4)}_{\mathrm{C}_5} \wedge \underbrace{(x_1 \vee \neg x_2 \vee \neg x_3)}_{\mathrm{C}_6} \end{aligned} \tag{1.8}$$

Let us solve this SAT problem using BB as illustrated in Figure 1.4 with Depth-First-Search as a branching strategy. We branched on the value of x_1 *to obtain* $\mathrm{S}(0, x_2, x_3, x_4)$ *and* $\mathrm{S}(1, x_2, x_3, x_4)$*. The assignation* $x_1 = 0$*, mean the clause* $\mathrm{C}_2, \mathrm{C}_4$ *and* C_5 *are satisfied, and can be ignored(since* $1 \wedge x = x$*), so:*

$$\mathrm{S}(0, x_2, x_3, x_4) = \underbrace{(x_2 \vee x_3)}_{\mathrm{C}_1|x_1=0} \wedge \underbrace{(x_2 \vee \neg x_3 \vee x_4)}_{\mathrm{C}_3|x_1=0} \wedge \underbrace{(\neg x_2 \vee \neg x_3)}_{\mathrm{C}_6|x_1=0}. \tag{1.9}$$

[1] The particular case 2-SAT is can be solved in polynomial time.

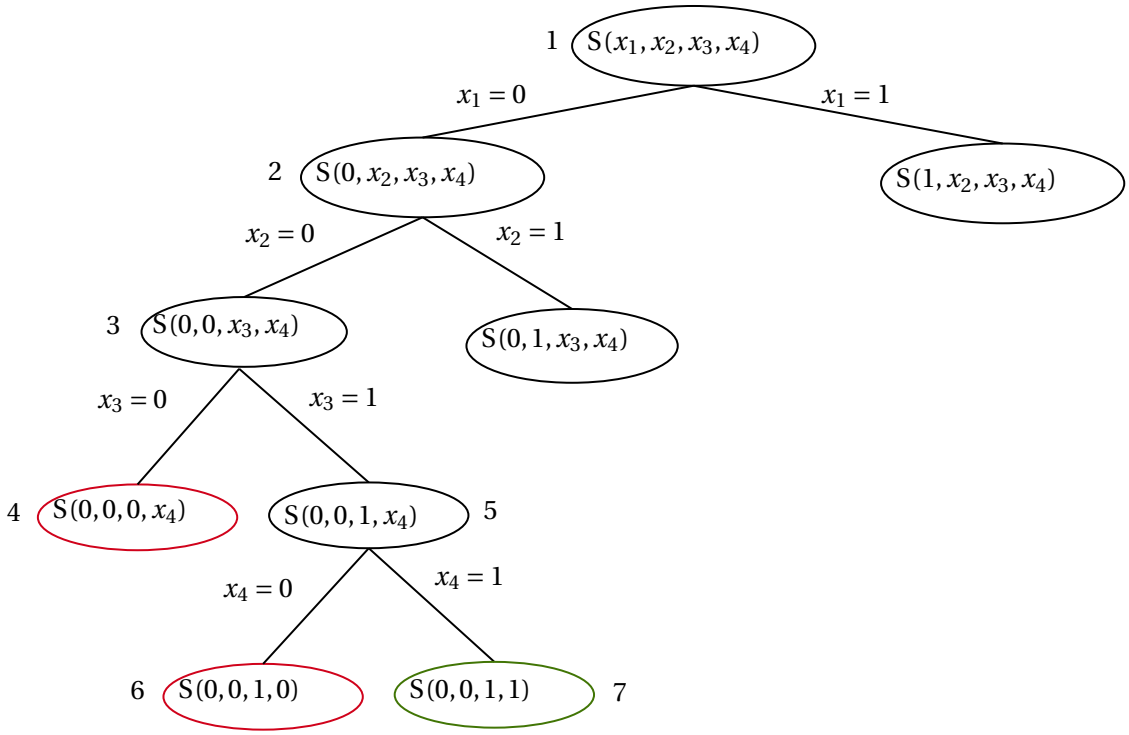


Figure 1.4 – Illustration of the use of BB to solve a SAT problem.The number near the nodes represents the visiting order in the search tree. The red nodes mean that we found a combination that does not satisfy our formula. The green node represents a solution satisfying our formula.

From $S(0, x_2, x_3, x_4)$ *and branching on the value of* x_2*, we get* $S(0, 0, x_3, x_4)$ *and* $S(0, 1, x_3, x_4)$*. When* $x_2 = 0$*, the clause* C_6 *is satisfied and can be ignored:*

$$S(0, 0, x_3, x_4) = \underbrace{x_3}_{C_1|x_1=0,x_2=0} \wedge \underbrace{(\neg x_3 \vee x_4)}_{C_3|x_1=0,x_2=0} . \tag{1.10}$$

From $S(0, 0, x_3, x_4)$*, we see that if* $x_3 = 0$*, no matter what the value of* x_4 *is,* S *will be unsatisfied, and then is not necessary to branch on the value of* x_4*. Then if* $x_3 = 1$*,* C_1 *is satisfied and we get:*

$$S(0, 0, 1, x_4) = \underbrace{x_4}_{C_3|x_1=0,x_2=0,x_3=1} . \tag{1.11}$$

Then assigning $x_4 = 0$*, will lead to* $S(0, 0, 1, 0) = 0$*, and assigning* $x_4 = 1$*, would give* $S(0, 0, 1, 1) = 1$ *proving that* $S(x_1, x_2, x_3, x_4)$ *is satisfiable.*

Another solution could have been found using Breath-First-Search at the node $S(0, 1, 0, x_4) = 1$.

Note that there are other methods to solve a SAT problem and can be related one way or another to branch-and-bound and search trees. That is the case of DPLL algorithm (used in the verification method [Katz et al., 2017]) and also Conflict-Driven Clause Learning (CDCL) methods.

The satisfiability problem has been formulated with Boolean variables. It can be extended to a more rich set using some constraints, for example, as discussed in 1.4.2.

1.4.2 Satisfiability Modulo Theories

Definition 1.6 (What is a theory?)**.** *A theory* $\mathcal{T}$ *on a set of symbols* Σ *is a set of formulas on the symbols of* Σ*.* Σ *is called signature.*

Example 1.4 (Equational theory)**.** *With the equation theory with* $\Sigma = \{=\}$*. The inequality, since* $x_1 \neq x_2$ *can be reformulated as* $\neg(x_1 = x_2)$*. Here is an example of the formula of the equational theory:* $x_1 + x_2 = 17 \wedge x_1 - x_2 = 5 \wedge x_1 \neq 13$

Decision procedure SMT relies on decision procedures. Decision procedures are methods used to decide on the satisfiability of formulas for elementary theories such as equational theory. Those procedures must satisfy some properties: incrementality, backtrackability, production of explications, and constraint implications.

Incrementality It should be possible to successively call the decision procedure on a set of constraints $c_0 \subset c_1 \subset \cdots \subset c_k$ in such a way that for a given set of constraints c_i it is possible to reuse the computation done for c_{i-1}, instead of starting from scratch.

Backtrackability The data structure used should allow to backtrack efficiently to an anterior state of the algorithm without having to reinitialize the computations.

Production of explanations When a set c of constraints is unsatisfiable, the decision procedure should produce a subset $\mathscr{U}$ of inconsistent constraints. The subset is referred to as the *unsat core*, and shows why the set c is unsatisfiable. That set should be as small as possible since its complementary could be added to the constraints while looking for feasible solutions as illustrated in Example 1.5. For example, if $c = x < y, y < z, z < 10, x < z, x > 15, a < b$, then the set $\mathscr{U}$ should be: $z < 10, x < z, x > 15$.

Example 1.5. *Using the equational theory presented above, here is a formula:*

$$\mathrm{S}(a, b, x_1, x_2, x_3) = \neg(a = b) \wedge (x_1 = a \vee x_1 = b) \wedge (x_2 = a \vee x_2 = b) \wedge (x_3 = a \vee x_3 = b) \wedge \neg(x_1 = x_2) \quad (1.12)$$

Finding a solution satisfying of S *using the classical branching on the tree can take some time while a few steps suffice when other search methods are simultaneously deployed. In this example, we use CDCL, which we consider to be a heuristic that goes through the search tree to reach leaves that potentially contain a solution instead of using tradition search strategies. In Figure 1.5, we have an illustration of how a SMT problem is linked to a corresponding SAT problem. The SMT problem from Step 1 is transformed into a SAT problem in Step 2 via* T2B*, which is solved and a solution is given in Step 3. That solution is transformed back to the original form of the SMT problem in Step 4 via the function* B2T*. Then a theory solver T Solve is used to evaluate if the proposed solution satisfies the formula at Step 5 and the solution does not satisfy our formula. The explanation, the core unsat, is given in Step 6 and converted in SAT in Step 7,* $\mathscr{U} = \mathrm{P}_2 \wedge \mathrm{P}_4 \wedge \neg \mathrm{P}_8$*. In Step 8,* $neg(\mathscr{U})$ *is added to the initial. This addition can be seen as forcing to backtrack to a node that does not violate the added constraint. Then the new Boolean formula is solved in step 9, and transformed back in Step 10. The evaluation at Step 11 shows that the problem is satisfied by the assignation in step 10.*

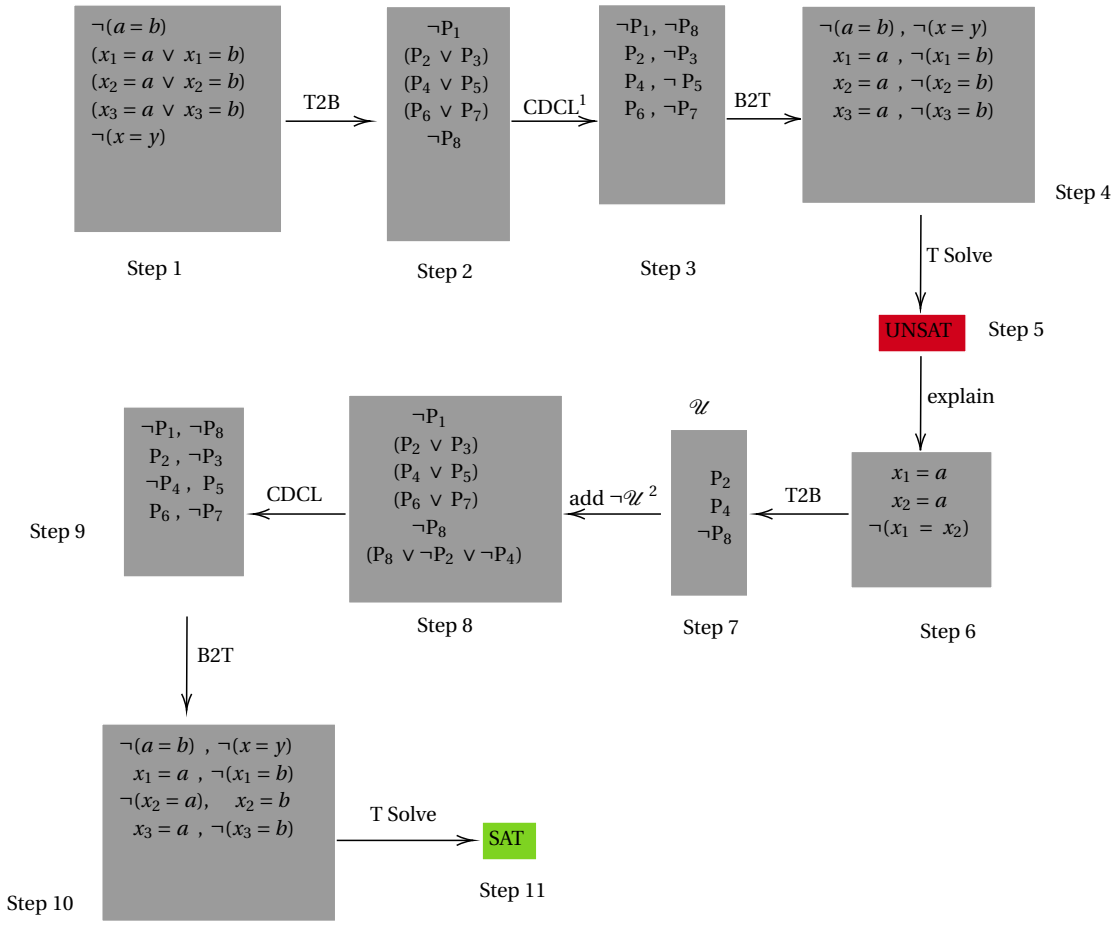


Figure 1.5 – Illustration of the resolution of an SMT problem. We use T2B to convert the formula from its theory to Boolean variables to obtain a SAT problem, CDCL is used to solve the SAT problem. Then B2T is used to reconvert the SAT assignation to the corresponding SMT one. A solution of the SAT is not always a solution of the SMT as shown at Step 5.To check if a solution of the SAT problem satisfies the STM one, the function *T Solve* is use. In Step 8, The negation of the core unsat $\mathscr{U}$ is then added to force the SAT to respect some new constraint that were not initially encoded in Step 2. And this time, the solution given by the SAT solver also satisfies the initial formula.

1.5 Integer Linear Programs and Mixed Integer Programs

1.5.1 Definition and illustration

Integer Linear Programs (ILPs) are like LPs but instead of optimizing over continuous variables, the optimization is done over a discrete set, for example $\mathbb{N}$. Integer programming is NP-complete. It has several applications, for example, in production planning and Traveling Salesman Problem.

$$\begin{cases} \max\limits_{\mathbf{x}} & \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} \leq \mathbf{b}, \\ & \mathbf{x} \geq 0, \\ & \mathbf{x} \in \mathbb{N}^n \end{cases} \tag{1.13}$$

In some problems, we can have some integer variables and other continuous ones:

$$\begin{cases} \max\limits_{\mathbf{x}} & \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} \leq \mathbf{b}, \\ & \mathbf{x} \geq 0, \\ & \mathbf{x} \in \mathbb{N}^{n_1} \times \mathbb{R}^{n_2} \end{cases} \tag{1.14}$$

with n_1 the number of integer variables and n_2 the number of continuous variables.

Example 1.6. *We can take the same Example 1.1 and add the constraint that the number of kilograms of yogurt is an integer. We would then obtain the following:*

$$\begin{cases} \max\limits_{x_1,x_2} & 4x_1 + 5x_2 & \\ s.t. & 2x_1 + x_2 & \leq 800 \\ & x_1 + 2x_2 & \leq 700 \\ & x_2 & \leq 300 \\ & x_1, x_2 & \geq 0 \\ & x_1, x_2 & \in \mathbb{N} \end{cases} \tag{1.15}$$

We just added the constraint: $x_1, x_2 \in \mathbb{N}$, and it was enough to transform the Linear Program (1.2) *into an Integer Linear Program. We can also want x_1 to be an integer while we allow x_2 to stay continuous to obtain a Mixed Integer Program.*

1.5.2 Solving an ILP or a MIP

To solve ILPs and MIPs, Branch-and-Bound is commonly used with integer relaxation as the evaluation method to compute the bounds. After relaxation, the problem becomes a Linear Program that can be solved to obtain a bound on the objective function. Bounds that can be used to discard some regions of the search space as discussed in Section 1.3.

Branching for ILP/MIP After relaxation, there are chances that real values are found for variables that should be integers. For example, we found $x_1 = 3.4$ where x_1 is supposed to be an integer at the final solution. Then the branching on the value of x_1 consists in splitting the search region into two regions. Since x_1 is supposed to be an integer, we either have $x_1 \leq 3$ or $x_1 \geq 4$. The splitting used for ILPs/MIPs is such that if we want to split according to a variable x whose value is not an integer yet, we split the search region into two part, the part $x \leq \lfloor x \rfloor$ and the part $x \geq \lfloor x \rfloor + 1$.

Bounding for ILP/MIP The resolution of the LP obtained after relaxation gives a bound on the optimal value of the objective function. At each node, a bound is computed. The bounds get tighter and tighter when the resolution progresses and reaches deeper nodes.

Example 1.7 (illustration of the Branch-and-Bound)**.** *Let us illustrate some iterations of the Branch-and-Bound applied to an ILP.*

$$\begin{cases} \max\limits_{x_1,x_2} & 2x_1+3x_2 \\ s.t. & 3x_1+2x_2 \leq 17 \\ & -x_1+7x_2 \leq 23 \\ & x_1,x_2 \geq 0 \\ & x_1,x_2 \in \mathbb{N} \end{cases} \tag{1.16}$$

A graphical illustration of the BB applied to (1.16) *is seen in Figure 1.6. We relax the integrity constraints and solve the linear program we obtain which will be the evaluation of the root (node* 1*). We get as a solution* $(\frac{73}{23}, \frac{86}{23})$*, let us say* $(3.17, 3.74)$*, giving* 17.57 *as the first bound on the objective function. For the sake of argument, we branch on the value of* x_1*. We then have a left child (node* 2*) with the addition of the constraint* $x_1 \leq 3$ *and a right child (node 3) with the addition of the constraint* $x_1 \geq 4$*. Note that this preserve all solutions since it only eliminates the values of* x_1 *between 3 and 4 not included. Then we solve the subproblem at the node corresponding to the left child. We find approximately* $(3, 3.72)$ *with a bound of* 17.15*. We see that there is already a slight improvement of the bound. Solving the subproblem of the right child, we get* $(4, 2.5)$ *and the bound* 15.5 *which is a greater improvement over the previous best bound. Then splitting on the left side we find a solution that gives the value 15 for the objective function. This solution is the optimal solution for our problem. We can exploit the information that the objective function should be an integer and deduce from the best bound* 15.5 *a better bound* 15*. If the bound found at node 3 was greater than 16, we would have to branch at that node to make sure that there does not exist a better solution than the one given at node 4.*

In practice, we could have used the equation $3x_1 + 2x_2 \leq 17$ *to derive bounds on* x_1 *and* x_2*, we would get* $x_1 \leq 5 = \lfloor \frac{17}{3} \rfloor$ *and* $x_2 \leq 3 = \lfloor \frac{17}{5} \rfloor$*. This process is called a cutting-plane method known as Gomory's cut. For the interested reader, more details are available in [Wolsey and Nemhauser, 1999].*

1.5.3 Optimality proof and optimality gap

Proving that a solution is optimal is not an easy task for ILPs and MIPs. For some problems, in practice, the optimization does not run until the optimality is proven, an optimality gap is instead.

Definition 1.7 (Optimality Gap)**.** *The optimality gap measures the gap between the incumbent, the best known solution, and the best bound. Let* $\widehat{\mathbf{x}}$ *be the incumbent solution, and* ζ *the best bound on the objective function, the gap is defined as:*

$$\begin{cases} \dfrac{|\widehat{\mathbf{x}} - \zeta|}{|\widehat{\mathbf{x}}|} & , \textit{ if } \quad \widehat{\mathbf{x}} \neq 0 \\ 0 & , \textit{ if } \quad \widehat{\mathbf{x}} = \zeta = 0 \\ \infty & , \textit{ if } \quad \widehat{\mathbf{x}} = 0 \textit{ and } \zeta \neq 0 \end{cases} \tag{1.17}$$

It is necessary to have an optimality gap of 0 to claim the optimality has been proven.

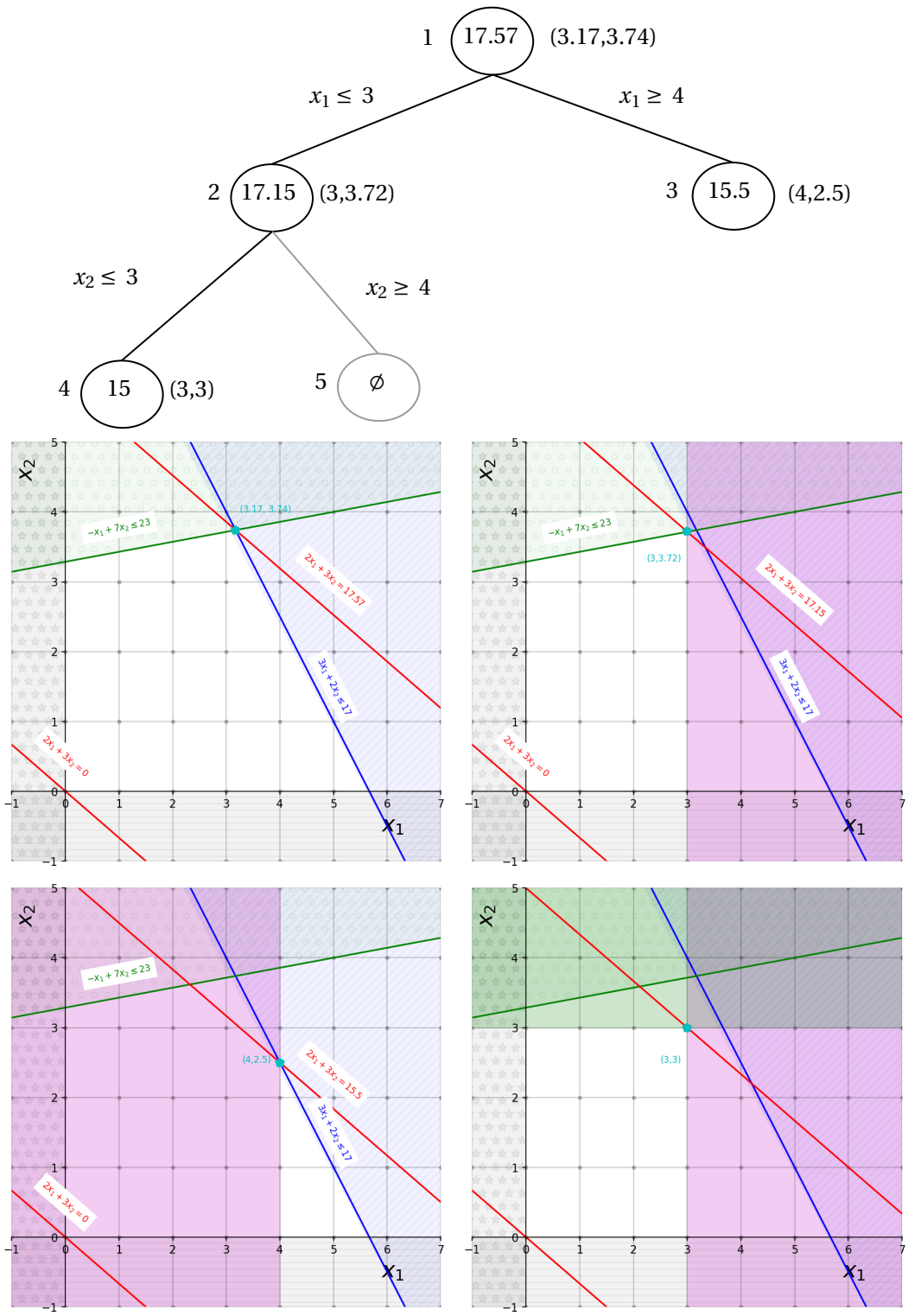


Figure 1.6 – The BB tree and a graphical illustration of the first branching. The value written on the left of each node is the order, we use a Breadth-First-Search. The value in the node is the upper bound found by the LP at that node and the value on the right of the node is the coordinate that achieves the bound. Then from left to right and top to bottom, we have the graphical illustrations of the Root problem (node 1), its left child (node 2), its right child (node 3), and the optimal solution (node 4). The coordinate shown are the coordinate of the point solution of the relaxation. At the root we have an upper bound of 17.57, which mean the objective value is bounded by 17 since $2x_1 + 3x_2$ should be an integer. The right child improves that bound to $15 = \lfloor 15.5 \rfloor$, with 15.5 the bound found at node 3. Then The node 4 gives the optimal solution.

1.6 Difference of Convex Algorithm

During this thesis, we had to deal with mixed integer programs. One way to solve this type of problem is to consider their continuous relation and recast the resulting nonconvex optimization problem as a DC program [Niu and Pham Dinh, 2008].

The difference of convex algorithm (DCA) has been designed for finding a local solution to this kind of nonconvex optimization problems when the cost function to be minimized is DC (this problem is also called a DC program), that is when it can be expressed as the difference of two convex functions. We just give hereafter a brief introduction to DC programming. For more details, see for instance [Horst and Thoai, 1999, Niu, 2010].

Definition 1.8 (DC function)**.** *Let* $\mathrm{J}_1(\mathbf{x}), \mathrm{J}_2(\mathbf{x})$ *two real valued convex functions on some Euclidian space,* $\mathrm{J}(\mathbf{x})$ *is a DC function if it can be expressed as* $\mathrm{J}(\mathbf{x}) = \mathrm{J}_1(\mathbf{x}) - \mathrm{J}_2(\mathbf{x})$.

Example 1.8. *The function* $\mathrm{J}(x) = x(1-x)$ *is a DC function since* $\mathrm{J}(x) = \mathrm{J}_1(x) - \mathrm{J}_2(x)$ *with* $\mathrm{J}_1(x) = x$ *and* $\mathrm{J}_2(x) = x^2$.

Indeed, the DC decomposition of a given DC function J is not unique since, for any convex function $g(\mathbf{x})$, we have $\mathrm{J}(\mathbf{x}) = (\mathrm{J}_1(\mathbf{x}) + g(\mathbf{x})) - (\mathrm{J}_2(\mathbf{x}) + g(\mathbf{x}))$.

Based on DC functions, we can define DC programs as follows.

Definition 1.9 (DC Program)**.** *Let* J *be a DC function over* $\mathbb{R}^d$ *and* $\mathscr{X}$ *a closed convex subset of* $\mathbb{R}^d$. *Then, the following optimization problem*

$$\min_{\mathbf{x}\in\mathscr{X}} \mathrm{J}(\mathbf{x}),$$

is called a DC Program.

Example 1.9. *The optimization problem* $\min_{x\in[0,2]} x(1-x)$, *is a DC program. It admits a global solution* ($x = 2$) *and a suboptimal local minimum* ($x = 0$).

Note that many non-convex optimization problems under constraints,including MIP , can be transformed into DC Programs [Niu and Pham Dinh, 2008].

An interesting property of DC functions is that they can have a convex majorization. Assume $\mathrm{J}_2(\mathbf{x})$ is differentiable and $\nabla_{\mathbf{x}}\mathrm{J}_2(\mathbf{x}_t)$ denotes its gradient at $\mathbf{x}_t$. A convex majorization function of $\mathrm{J}(\mathbf{x}) = \mathrm{J}_1(\mathbf{x}) - \mathrm{J}_2(\mathbf{x})$ at $\mathbf{x}_t$ is

$$\mathrm{J}(\mathbf{x}) \leq \mathrm{J}_1(\mathbf{x}) - \mathrm{J}_2(\mathbf{x}_t) - (\mathbf{x}-\mathbf{x}_t)^\top \nabla_{\mathbf{x}}\mathrm{J}_2(\mathbf{x}_t).$$

The DC Algorithm (DCA) is a general procedure based on this convex majoration and design to minimize a DC function. It proceeds by successive convex relaxations. At each iteration t, we can define the convex majoration function

$$\mathrm{J}_c(\mathbf{x}) = \mathrm{J}_1(\mathbf{x}) - \mathrm{J}_2(\mathbf{x}_t) - (\mathbf{x}-\mathbf{x}_t)^\top \nabla_{\mathbf{x}}\mathrm{J}_2(\mathbf{x}_t),$$

to be minimized. The Algorithm 3 presents the pseudo code of DCA for solving the DC program $\min_{\mathbf{x}\in\mathscr{X}} \mathrm{J}(\mathbf{x}) = \mathrm{J}_1(\mathbf{x}) - \mathrm{J}_2(\mathbf{x})$. It has been proven that DCA is a well defined and convergent algorithm.

In practice, the convergence conditions $|(g-h)(x_{t+1}) - (g-h)(x_t)| \leq \varepsilon$ or $\|x_{t+1} - x_t\| \leq \varepsilon$ are often used, with a user defined value of ε. Note also that, although there is no analytic result yet to justify this, according to the DC Programming literature, the local optimization approach often yields the global optimum for some problems. The main difficulty for DCA implementation is to find a relevant DC decomposition.

Algorithm 2 : DCA(J)

Input: $J_1(\mathbf{x})$, $J_2(\mathbf{x})$ and $\mathscr{X}$
Result : $\mathbf{x}^\star \in \mathscr{X}$ a local minimizer of $J(\mathbf{x})$ over $\mathscr{X}$
Set $t = 0$, initialize $\mathbf{x}_t \in \mathrm{dom} J_1 \cap \mathscr{X}$
while *not convergence* **do**
  Solve $\mathbf{x}_{t+1} = \mathrm{argmin}_{\mathbf{x}\in\mathscr{X}} J_c(\mathbf{x}) = J_1(\mathbf{x}) - \mathbf{x}^\top \nabla_{\mathbf{x}} J_2(\mathbf{x}_t)$
  $t = t + 1$
end
Return $\mathbf{x}_t$

1.7 Conclusion

In this chapter, we have presented methods that are used in the verification of the robustness of neural networks except DCA. We started with Linear Programs.

We then talked about the Branch-and-Bound that can be used to deal with some optimization problems that Linear Programs cannot handle. Then we saw the Satisfiability Modulo Theories that is a generalization of the (Boolean) Satisfiability problem.

After we saw the Integer Linear Programs and Mixed Integer Programs, to handle problems that are very similar to Linear Programs but that have values that should be integers. We also how to solve those problems with Branch-and-Bound using Linear Programs to evaluate the bounds.

Finally, we talk about Difference of Convex Algorithms that we will use to find adversarial examples.

Chapter 2

Adversarial examples: an overview

Contents

Deep learning, the state-of-the-art Deep learning is now used in many domains as the state-of-the-art tool. Neural networks became the go-to when it comes to several tasks such as image classification [Krizhevsky et al., 2017]. They even made possible things that were considered impossible not long ago. Deep learning is used on self-driving cars, in medicine, in fraud prevention, in earthquake prediction, in natural language processing and in game playing just to name a few (see for instance [Sengupta et al., 2020] for a detailed review and associated references).

Illustrations of its use on self-driving cars In autonomous vehicles, deep learning can be used in several ways. It can be used to drive the car by controlling the speed and direction of the car using reinforcement learning [Bansal et al., 2018]. It can also be used to recognize traffic signs in order to respect the highway code, the stops, the priorities, the speed limitation. Another use of deep learning would be putting bounding boxes and tracking other vehicles [Shafiee et al., 2020] in order to avoid collision as illustrated in Figure 2.1. It can be used also to monitor the human driver himself.

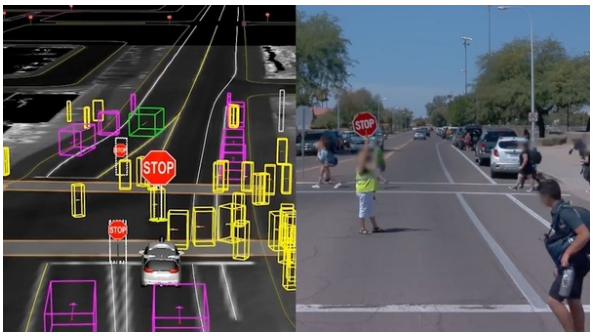


Figure 2.1 – Illustration of an automatic perception (left) of a real road scene (right). Image from Waymo's video showing autonomous driving technology interacting with crowded school crossing[1].

Illustrations of its use in Medicine Deep learning is used in medical imaging [Suzuki, 2017], to provide diagnostics [Bakator and Radosav, 2018, Esteva et al., 2021] (as illustrated Figure 2.2) and in surgery [Hashimoto et al., 2018]. It could be used either to act autonomously or as an assistant to human professionals.

Existence of adversarial examples However, neural networks are very brittle in the sense that, their high accuracy can rapidly drop to nearly zero when maliciously crafted noise is added to the input, which becomes an adversarial example, as shown in Figure 2.3 [Goodfellow et al., 2015].

The use of adversarial examples in safety-critical environments This kind of behavior of neural networks has a deterrent effect making their use in safety-critical environments such as in aviation safety (see Example 1.1), self-driving car (see Example 1.2) or in medical imaging (see Figure 2.4 for an illustration) a topic of controversy, rightly or wrongly.

In fact, in order to use deep learning in such environments, some guarantees are needed in terms of reliability, explainability of the decision-making and interpretability.

Example 2.1 (The ACAS Xu safety system)**.** *Designed to avoid mid-air collisions between aircrafts, the ACAS Xu system issues turn advisory outputs - strong left, weak left, conflict-free, weak right, or strong right - given the input values for relative positions, headings, and speeds. Originally created as a multi-gigabyte lookup table of rules, the system has been compiled using a series of 45 neural networks. Inputs and outputs of the system are illustrated Figure 2.5. The safety of this critical*

[1] `youtube.com/watch?v=Vu8gmFhiGko`
[2] `openai.com/blog/adversarial-example-research/`

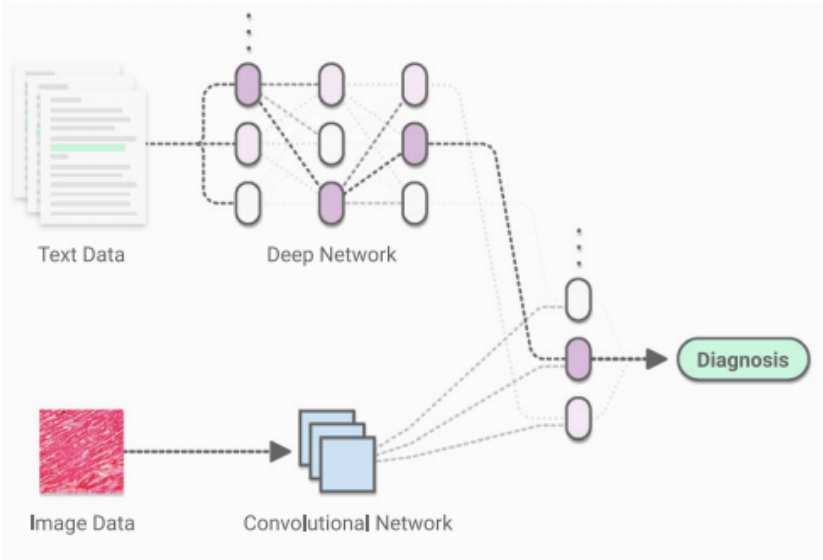


Figure 2.2 – A multimodal discriminative model, trained on image data with convolutional networks and non-image data, with the appropriately chosen deep network [Esteva et al., 2021].

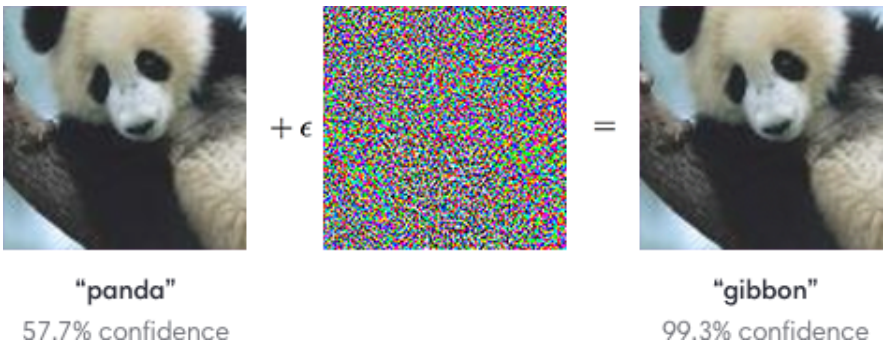


Figure 2.3 – The famous Panda's picture popularized by [Goodfellow et al., 2015] seminal paper on adversarial examples. The image on the left is a clean image recognized by a deep network as a panda, on the center some noise calculated to make the resulting image (on the right) be recognized as a gibbon with high confidence. Taken from OpenAI's blog[2].

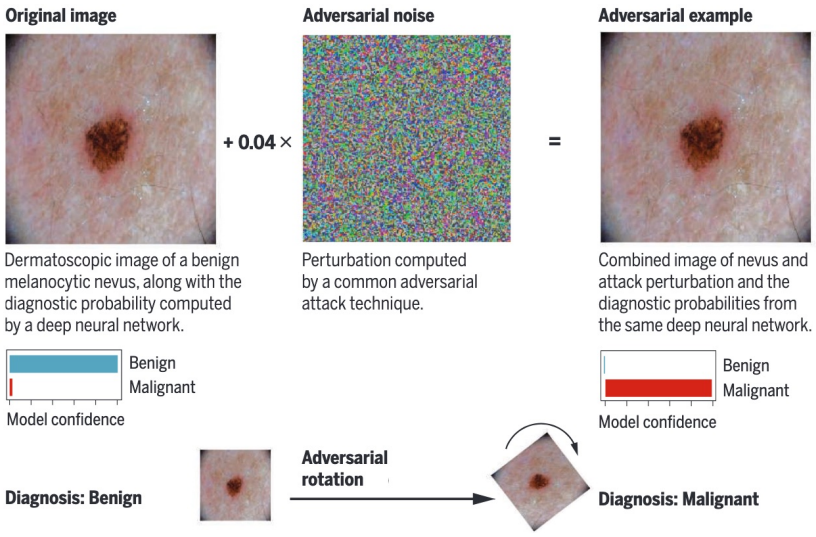


Figure 2.4 – Illustration of two different adversarial attacks on a medical diagnostic system based on deep learning image processing. Adapted from [Finlayson et al., 2019].

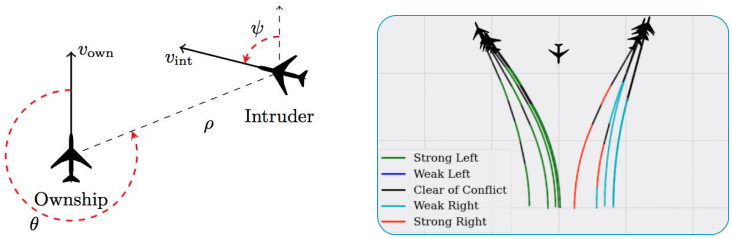


Figure 2.5 – Illustration of the input (left) and output (right) of the ACAS Xu system. Pictures are from [Katz et al., 2017] were you can find more details.

system clearly depends on the reliability of these neural networks. Originally introduced by [Katz et al., 2017], this example has been used in the 2020 competition for neural network verification[3].

Example 2.2 (Self driving cars)**.** *The new generation of advanced driving assistance systems (ADASs) and autopilots of semi or fully autonomous cars include neural networks (see for instance [Grigorescu et al., 2020]). These systems have been proven to be vulnerable to attacks, as illustrated in Figure 2.6. See [Qayyum et al., 2020] for a review on this topic.*

Figure 2.6 – Speed limit road sign designed to fool Tesla's Autopilot[4] (left) and projected images that cause the ADAS to fail [Nassi et al., 2020] (right).

The necessity of defense mechanisms and some defense mechanisms Several defense mechanisms have been deployed since [Papernot et al., 2016c] which prove later to be inefficient when attacks get stronger. Those stronger attacks may be completely new. Sometimes, it is enough to allow more attempts to existing attacks to expose the weakness of the defenses.

Measuring the robustness of DNN In that context, complete verification methods such as Reluplex [Katz et al., 2017] and MIP (Mixed Integer Programming) formulations of the search of adversarial examples [Tjeng et al., 2018] seem to be the ultimate judge able to give the exact robust test error of a neural network [Bastani et al., 2016]. However,those complete verifiers can spend, on some examples, up to an hour without being able neither to find an adversarial example nor to prove that there is no adversarial within a given neighborhood of the original example. This leads sometimes to have models for which robustness is not guaranteed and no adversarial example is found. That sometimes justifies the use of attacks to measure the robustness of classifiers even though attacks can miss some adversarial examples. Even though using attacks it is not possible to prove that there are no adversarial examples in a region.

Although, as we have just seen, adversarial examples exist for all types of applications, in our work we mainly focus on adversarial examples in computer vision, specifically for image classification. Accordingly, the remainder of this chapter is organized as follows. We begin by introducing the general classification problem (Section 2.1). Then, we give a general presentation of adversarial examples and reasons that explain their existence in Section 2.2. Section 2.3 is dedicated to the description of some popular attacks used to create adversarial examples as well as their taxonomy and related threat models. Next, we study the defenses in Section 2.4 and their taxonomy

[3] `sites.google.com/view/vnn20/vnncomp`

[4] `mcafee.com/blogs/other-blogs/mcafee-labs/model-hacking-adas-to-pave-safer-roads-for-autonomous-vehicles/`

by distinguishing between empirical and provable defenses. Then, Section 2.5 is devoted to the formal verification methods which guarantee, theoretically and under defined threat models, the robustness of deep neural network models. Section 2.6 is dedicated to the theoretical limits on adversarial robustness and Section 2.7 is dedicated to the tools that can be used to work on adversarial examples.

2.1 General classification problem

As said above, Deep learning is the go-to pick for more and more tasks, among which (image) classification that will be considered in the following lines. Let f_θ represent a classifier with parameter θ, and specifically a deep neural network with θ representing the weights, going from the input space $\mathscr{X}$ to the output space $\mathscr{Y}$. Let us define the loss function $\mathscr{L}$ used to compute the correctness of the prediction of the classifier compared to the ground truth. The loss functions used are mostly continuous and differentiable almost everywhere, allowing to use a gradient descent algorithm. As an illustration, for images, we usually have $\mathscr{X} = [0,1]^d$ where d is the number of pixels of the images, the output space $\mathscr{Y} = \mathbb{R}^{\mathrm{C}}$ or $\mathscr{Y} = [0,1]^{\mathrm{C}}$ with C the number of classes, depending on the use or not of the softmax. The loss function is usually the cross-entropy loss written for the ground truth $\mathbf{y}$ and the softmaxed $\widehat{\mathbf{y}} = f_\theta(\mathbf{x})$,

$$\mathscr{L}(\mathbf{y}, \widehat{\mathbf{y}}) = -\sum_{j=1}^{\mathrm{C}} \mathbf{y}_j \log(\widehat{\mathbf{y}}_j). \tag{2.1}$$

The true goal of the learning of the classifiers is to find the value of θ that minimizes:

$$\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim \mathrm{D}}[\mathscr{L}(\mathbf{y}, \widehat{\mathbf{y}})], \tag{2.2}$$

with D the joint probability distribution of the input examples paired with their labels. However, in practice, we do not have access to the distribution D but we have some realizations of it, for example N realizations, $\mathscr{D} = (\mathbf{x}^{(i)}, \mathbf{y}^{(i)})_{i=1}^{\mathrm{N}}$ (the training set). Using the available data, model is trained by minimizing the empirical risk:

$$\frac{1}{\mathrm{N}} \sum_{i=1}^{\mathrm{N}} \mathscr{L}(\mathbf{y}^{(i)}, \widehat{\mathbf{y}}^{(i)}). \tag{2.3}$$

Once a model is trained, its performance is measured on the test set to have a sense of its generalization performance. Deep learning models are excellent at this task as shown by empirical results on Imagenet [Deng et al., 2009] achieving performance comparable to the human level [Krizhevsky et al., 2017]. However, [Szegedy et al., 2014] showed that the good performance of deep neural networks can rapidly go up in smoke if the input images were to be slightly modified to fool the classifiers. Those misleading images are adversarial examples. Their discovery raises new challenges to tackle not only in classification but in most tasks that can be performed by deep learning, arising by the way some questions such as the reason of existence that will be dealt with in the next section after we briefly define what adversarial examples are.

2.2 Adversarial examples

In this section, we define and illustrate some common terms used in the adversarial example field in Subsection 2.2.1. After that, we look at possible explanations of the existence of adversarial examples in Subsection 2.2.2. Most definitions and explanations can suit different domains ranging from simple classification, multilabel classification, to image segmentation, and several other domains.

2.2.1 Definitions and illustrations

Adversarial examples Adversarial examples are introduced in [Goodfellow et al., 2017] as '*inputs to machine learning models that an attacker has intentionally designed to cause the model to make a mistake*'. This definition is suitable since it is extensive enough to take into account several perturbations, threat models, attackers and can even be convenient in domains different from the image classification that we will focus on later.

We will now give a more formal definition of adversarial examples while attempting to cover the maximum number of machine learning situations. Let:

- $\mathbf{x} \in \mathscr{X}$ be a clean example (an unmodified image of the dataset $\mathscr{D}$),
- $f : \mathscr{X} \to \mathscr{Y}$, a model trained to perform a task, it could be image segmentation, speech-to-text or simply image classification (reference to parameter θ has been omitted to alleviate the notation),
- $\mathscr{Y}'$ the decision of the model based on its output, for example, for classification, it is the class space different from the output space $\mathscr{Y}$ which would be a C dimension vector space with C the number of classes.
- $\mathrm{D}_f(\mathbf{x}) : \mathscr{Y} \to \mathscr{Y}'$, the decision of the model f for the input $\mathbf{x}$,
- $\mathbf{x}' = transform(\mathbf{x}) \in \mathscr{X}$, a transformation of $\mathbf{x}$ aiming at causing the model to make a mistake,
- $\|\mathbf{x} - \mathbf{x}'\|_{\mathscr{X}} \geq 0$[5], measuring the distortion between $\mathbf{x}$ and its transformed version $\mathbf{x}'$,
- $\|\mathrm{D}_f(\mathbf{x}) - \mathrm{D}_f(\mathbf{x}')\|_{\mathscr{Y}'} \geq 0$, measuring the difference between $\mathrm{D}_f(\mathbf{x})$ and $\mathrm{D}_f(\mathbf{x}')$,
- $\eta_{\mathscr{X}}$, a threshold such that if $\|\mathbf{x} - \mathbf{x}'\|_{\mathscr{X}} > \eta_{\mathscr{X}}$, $\mathbf{x}'$ is too far,too different from $\mathbf{x}$ to be considered as being an adversarial example,
- $\eta_{\mathscr{Y}'}$, a threshold such that if $\|\mathrm{D}_f(\mathbf{x}) - \mathrm{D}_f(\mathbf{x}')\|_{\mathscr{Y}'} > \eta_{\mathscr{Y}'}$ the decision taken at $\mathbf{x}$ is different of the decision taken from the decision taken at $\mathbf{x}'$ in such a way that if one is correct the other is not when $\mathbf{x}$ and $\mathbf{x}'$ are close enough.

Given these two positive thresholds $\eta_{\mathscr{X}}$ and $\eta_{\mathscr{Y}'}$, if $\mathrm{D}_f(\mathbf{x})$, the decision of the model at $\mathbf{x}$ is correct, then $\mathbf{x}'$ is an adversarial example of the model f near the input $\mathbf{x}$ when:

$$\|\mathbf{x} - \mathbf{x}'\|_{\mathscr{X}} \leq \eta_{\mathscr{X}} \text{ and } \|\mathrm{D}_f(\mathbf{x}) - \mathrm{D}_f(\mathbf{x}')\|_{\mathscr{Y}'} \geq \eta_{\mathscr{Y}'}. \tag{2.4}$$

This definition suits several domains such as speech-to-text, text comprehension, image segmentation, classification, multilabel classification, etc. It does, however, assume that the adversarial examples are crafted around a known clean example which is not always the case. Generative Adversarial Networks [Goodfellow et al., 2014] have been used to generate adversarial examples that are, at least a priori, not paired with any input of the dataset [Fang et al., 2019].

Nevertheless, most of the illustrations that we are going to use can be seen as classification problems:

- multilabel classification on images can be seen as a single image classification with respect to each set of antagonist labels,
- image segmentation can be seen as a classification problem, where each pixel is classified,
- object detection can be seen as a classification over all possible square regions on an image, to say if they are interesting or not.

We can therefore give a formal definition most suited to classification. In that case, $\mathrm{D}_f(\mathbf{x}) = \mathrm{argmax}_i\, f_i(\mathbf{x}) \in \mathscr{Y}'$ where the space $\mathscr{Y}'$ is the set of classes $\mathscr{C} \subset \mathbb{N}$ and equation 2.4 in the definition of adversarial examples becomes:

$$\|\mathbf{x} - \mathbf{x}'\|_{\mathscr{X}} \leq \eta_{\mathscr{X}} \text{ and } \mathrm{D}_f(\mathbf{x}) \neq \mathrm{D}_f(\mathbf{x}'), \tag{2.5}$$

making the threshold $\eta_{\mathscr{Y}'}$ useless.

[5] when $\mathscr{X}$ is identifiable to normed vector space then a norm can be used to measure the distortion. If necessary, it can be replaced by any other dissimilarity measure as is the case for example when $\mathbf{x}$ is a text.

Adversarial perturbation An adversarial perturbation is the modification that is operated on a clean image. Depending on how the attack is performed, a perturbation can be additive, geometric, or functional [Laidlaw and Feizi, 2019]. It is additive when it is the addition of an adversarial noise computed, for example, using the gradient of the loss with respect to the inputs, geometric when some geometric perturbations such as rotation or translation are used. For an image, functional perturbations could be the modification of brightness, contrast, or other characteristics of the image. Adversarial noise is not random noise: random noise with small magnitude is usually harmless to classifiers. All sorts of perturbations can be performed as far as the inputs' classification by humans stays the same for the clean example and the manipulated one.

Attacker/adversary The agent who crafts the adversarial examples is called the attacker or the adversary. It can be someone trying to get around a malware detector [Grosse et al., 2017]. It can be someone trying to hack a finger print sensor [Arthur, 2013], someone trying to cause misidentification of a facial recognition system [Sharif et al., 2016], or someone trying to make the vision system of a self-driving car misclassify traffic signs [Eykholt et al., 2018]. In the following, it will mostly refer to the person crafting adversarial examples to fool the classifiers.

Transferability of adversarial examples refers to the fact that an adversarial example created to fool a model A, usually using information about that model, can also fool a model B or even generalize to several models. For example, a model created from a deep neural network can fool another deep model or an SVM [Papernot et al., 2016a]. Transferability will be further explored in Section 2.3 when we discuss the adversarial attacks.

We now add two more related definitions useful to Subsection 2.2.2 which will give some insights explaining the existence of adversarial examples.

Useful features Given an input space, any feature that is predictive relatively to the task we want to perform is a useful feature. For example, for a classification task, any feature that is highly correlated with a class is a useful feature since it can help to predict the class of an input. It can be an object, a background, or anything else that a human can see and that is meaningful to a human, for example, the presence of a smile. It can also be something humans do not pay attention to or simply do not perceive. Taking the example of image classification, it can be something like an invisible tag that is highly correlated with a class.

Robust and non-robust features We are only interested in useful features, so we assume the features are useful to discuss their robustness or not. A feature will be considered robust if, given a task and a set of perturbations, there is no perturbation that will change the initial correlation between the feature and a class in such a way that it creates a new misleading correlation. Typically, in image classification and the set of small ℓ_∞-norm perturbations, every feature a human can use to classify a picture can be accepted as a robust feature. Robust features may exist even beyond human perception, we can think again of the invisible tag example. On the contrary, given a dataset and a set of perturbations, nonrobust features are features whose correlation to a class can be altered to create a correlation to another class, creating a deceiving correlation. Non-robust features are related to adversarial examples according to some authors [Ilyas et al., 2019]. As robust features, they can be perceptible or not.

2.2.2 Source/Cause of adversarial examples

Several hypotheses have been proposed to explain the existence of adversarial examples. We will explore the arguments of some of those hypotheses as well as the arguments going against them. We will start by studying the hypothesis based on dimensionality and architecture via the *linearity hypothesis*. Then we explore the hypothesis that adversarial examples are possible because of the

data via the *non-robust feature hypothesis*. And finally, we study the effect of the learning algorithm through the *evolutionary stalling* lens.

The linearity hypothesis The linearity hypothesis is backed up by various papers with different arguments. Although initially, the existence of adversarial seemed to be linked to the high nonlinearity of deep neural networks, that idea became quickly out of favor to the benefit of the linear hypothesis that dragged more attention. The paper [Goodfellow et al., 2015] popularized the linearity conjecture and strongly linked it to the dimensionality of the input examples, in particular in the case the adversarial noise is measured with ℓ_∞-norm. Let $\mathbf{x}'$ be an adversarial example such that $\mathbf{x}' = \mathbf{x} + \delta$ with $\|\delta\|_\infty = \epsilon$. Let $\mathbf{w}$ be a weight vector. We have:

$$\mathbf{w}^\top \mathbf{x}' = \mathbf{w}^\top \mathbf{x} + \mathbf{w}^\top \delta. \tag{2.6}$$

The adversarial perturbation δ cause an increase of $\mathbf{w}^\top \delta$ compared to activation on the original example $\mathbf{x}$. If we maximize the increase regarding the ℓ_∞-norm and obtain $\delta = \epsilon \operatorname{sgn}(\mathbf{w})$, then the increase will be $\epsilon \sum_{i=1}^{d} |\mathbf{w}| = \epsilon \|\mathbf{w}\|_1$ with d the dimension of $\mathbf{x}$. Let m be the average magnitude of $\mathbf{w}$, then $\epsilon \sum_{i=1}^{d} |\mathbf{w}| = \epsilon\, d\, m$. With the explicit apparition of the input dimension d, we can infer that the adversarial vulnerability of neural networks increases with the input dimension [Simon-Gabriel et al., 2019]. In this latter paper, this intuition is backed up and extended. The authors showed more formally, with theoretical and empirical evidence, that many feedforward neural networks are increasingly vulnerable to ℓ_p-norm attacks with growing input dimensions, confirming the relationship between dimension and vulnerability of classical training.

Despite the popularity of the linearity hypothesis and the evidence provided, there is little skepticism about its ability to fully explain the existence of adversarial examples. The linearity hypothesis is unconvincing according to [Tanay and Griffin, 2016]. The authors argue that "small perturbations do not provoke changes in activation that grow linearly with the dimensionality of the problem when they are considered relatively to the activations themselves". They highlighted the fact that while $\mathbf{w}^\top \delta$ grows linearly with the input dimension so does $\mathbf{w}^\top \mathbf{x}$ the activation of the clean example $\mathbf{x}$, assuming that the weight and pixel distributions remain the same. As a consequence, the quotient between the two quantities, $\frac{\mathbf{w}^\top \delta}{\mathbf{w}^\top \mathbf{x}}$ would remain constant. They support their claim by performing linear classification with a linear binary SVM to classify 3s and 7s from MNIST and another one with a scaled-up version of the same images to have 200×200 images. Irrespective of the size of the images, the weights vector found by linear SVM looks very similar to the one found by logistic regression in [Goodfellow et al., 2015] when visualized. The two SVM had the same error rate. And noting ϵ_{28} and ϵ_{200} the value of the infinite norm of the perturbation $\|\delta\|_\infty$ needed to fool the SVMs on 99% of the corresponding test set, they obtained $\frac{\epsilon_{200}}{200} \approx \frac{\epsilon_{28}}{28}$. The outcome of this experiment is that "the dimensionality argument does not hold: high dimensional problems are not necessarily more prone to the phenomenon of adversarial examples". In another experiment, they show that linear behavior alone is not sufficient to cause adversarial examples using toy examples.

Another work, related this time to the linearity of the activation function, presents interesting results [Krotov and Hopfield, 2018] using Dense Associative Memory models [Krotov and Hopfield, 2016]. One of the many differences between Dense Associative Memory models and common deep neural networks is the rectified polynomials activation F_n which replace the Rectified Linear Unit. We can have, for example:

$$\mathrm{F}_n(x) = \begin{cases} x^n, \text{ if } x \geq 0 \\ 0\,, \text{ if } x < 0. \end{cases} \tag{2.7}$$

This nonlinearity can make the model more robust but does not suffice by itself. The linearity that we talk about here is according to the activation functions and not according to the input and its dimensionality. We will talk more about Dense Associative Method when we talk about defending

against adversarial examples in Section 2.4. It has been observed in [Krotov and Hopfield, 2018] that the success rate of transferring adversarial examples from models with activation with a lower degree is lower than the opposite. It means that if a model with activation F_{n_1} is used to create adversarial examples, and another model is trained using F_{n_2} with $n_1 < n_2$, transferring adversarial example from the first model to the second will have a low success rate compared to if position were switched. In the case of $n = 1$, we get the standard ReLU activation, meaning that adversarial examples created on a usual deep neural network architecture are harmless to Dense Associative Methods when the parameter n is high enough.

The non-robust feature hypothesis In the paper [Ilyas et al., 2019], it is said that "Adversarial vulnerability is a direct result of sensitivity to well-generalizing features in the data." Since the goal of the training of deep neural networks and other classifiers is only to have good accuracy, they can rely on any feature they find useful to do so, without distinguishing whether they are robust or not. And the conjecture here is that the existence of imperceptible nonrobust features used by the model is a cause, not necessarily the only one, of some adversarial vulnerability that is exploited by attackers to fool the models. They run an experiment on a restricted version of ImageNet, which we call $\mathscr{D}$.

In that experiment, they first train a model f_1 using $\mathscr{D}$. They used that model f_1 to create a new version of training data in $\mathscr{D}$ by creating adversarial examples and pairing them randomly with a target class to obtain the new pairs $(\mathbf{x}', t)$ that compose the training set $\hat{\mathscr{D}}_{rand}$. Then they train a new model f_2 on $\hat{\mathscr{D}}_{rand}$. That dataset would be mislabeled for a human because the robust features we rely on stay unchanged. The reported accuracies of f_1 and f_2 on the test set of $\mathscr{D}$ are, respectively, 96.6% and 87.9%.

The fact that the model f_2 manages to have a high accuracy while trained on a mislabeled data shows that non- robust features are used by the model and that those nonrobust features are enough to achieve relatively high accuracy. The same experience but with a deterministic target instead of a random target this time also has 64.4% accuracy on the test set $\mathscr{D}$, comforting the findings.

We can infer that there are links between those nonrobust features and the intriguing properties found in [Engstrom et al., 2019a]. In that paper, given two images $\mathbf{x}_{1a}$ and $\mathbf{x}_2$ from two different classes, they show that is easy to construct another image $\mathbf{x}_{1b}$ indistinguishable from $\mathbf{x}_{1a}$ such that $\mathbf{x}_{1b}$ and $\mathbf{x}_2$ have almost identical internal representation with respect to a deep neural network. Since the models rely on nonrobust features, their internal representation may heavily be dependent on them, and given a pair of images, one can without difficulty modify those features on one image in order, aligning them with the nonrobust features present on the other image.

Along the same line of thought, showing that classifiers do not rely on the same feature than humans, there is [Geirhos et al., 2019]. The findings of [Geirhos et al., 2019] are deep learning models do not rely on the shape as much as humans do and rely instead on "texture", highlighting the difference between the features used by humans for the classification task. One admissible argument is that when instead of presenting real images, the corresponding images when passed through an edge detector were presented to both humans and machine learning models, the machine learning models suffered an important drop in accuracy compared to the human subjects of the experiment. Another experiment, designed to have an opposition between texture and shape also leads to the conclusion that humans rely more on shape for classification while deep learning models are biased toward texture.

Evolutionary stalling At the beginning of the training of deep neural networks, samples are not well classified and contribute to the loss and therefore to readjusting the weights, in other words, to moving decision boundaries. During training, samples start to be increasingly well classified, contributing less and less to the loss [Rozsa et al., 2018]. As a consequence, the samples get stuck very close to the decision boundaries, making it easy for attackers to fool the models. Evolutionary stalling can also be linked to the presence of nonrobust features that the neural network may rely

on and that can be manipulated to mislead the models.

Other explanation There are other explanations that are worth mentioning. The existence of adversarial examples is "due to an inherent uncertainty that neural networks have about their predictions" according to [Cubuk et al., 2017]. Facts such as low flexibility of the networks [Fawzi et al., 2016a], flatness of decision boundaries [Fawzi et al., 2016b] and large local curvature of the decision boundaries [Moosavi-Dezfooli et al., 2017b] have also been linked to the existence of adversarial examples.

About transferability In [Tabacof and Valle, 2016] the authors empirically demonstrated that adversarial examples appear in large regions in the pixel space by adding noise to adversarial images and reporting the number of noisy images that were classified back to the correct class. The authors of [Tramèr et al., 2017] found that adversarial examples span a continuous subspace of large dimensionality. They see the overlapping of those adversarial subspaces across different classifiers as a cause of the transferability of adversarial examples across those classifiers. This same idea is a plausible reason for the existence of universal adversarial perturbations that are discussed in Section 2.3.

There are many hypotheses trying to explain the existence of adversarial examples, but there is not yet a consensus on the reasons for the existence of adversarial examples. Furthermore, all hypotheses do not perfectly align with each other. There are even some hypotheses such as the "linearity hypothesis" that were refuted with convincing experimental proofs.

2.3 Adversarial attacks

We have seen that adversarial attacks are a threat to the success of deep learning and to their use in safety-critical domains. This section aims to present the main attacks producing adversarial examples. To this end, following the Equation (2.5), we begin by describing the most common distortion measures that are also perturbation measures. Then we introduce characteristics to classify the attacks that we use to present a selection of the most popular attacks in the literature.

2.3.1 Perturbations and their measures

Ideally, we would use distortion measures that align perfectly with humans' vision systems. However, finding such a measure is a very difficult problem since humans' decision on images is invariant to a broad range of transformations that can fool classifiers. For that reason, more simple measures are commonly used, such as norm-based measures. More recently, more sophisticated perturbations have been proposed [Wong et al., 2019], showing that there is still a lot to discover in this relatively young field. We start with norm-based perturbations which are widely used, then we take a look at other perturbations that draw less attention.

2.3.1.1 Norm based perturbations

Data used in machine learning are usually identifiable to a vector. For that reason, using Euclidean distance and other vector norms is straightforward. Even though those norms do not perfectly align with human perception [Sen et al., 2019]. Robustness to such perturbations does not imply robustness in a broader sense, but the lack of robustness to norm-based perturbations hints at vulnerabilities to the attacks that a model may face when deployed.

ℓ_0-norm The attackers use ℓ_0 pseudo norm that counts the number of nonzero components of a perturbation vector when they are interested in finding adversarial examples while modifying just a few pixels of the original image. It is possible to create an adversarial example by modifying just one pixel [Su et al., 2019]. The only constraint with ℓ_0-norm perturbations is that they have a valid

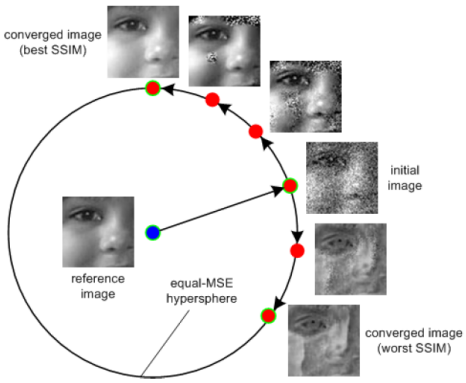


Figure 2.7 – Image from [Wang and Bovik, 2009] showing equidistant images from an original one can be drastically different from a reference image and between them depending on the The structural similarity (SSIM) index.

range of pixel values to produce a valid adversarial example. Usually, this kind of perturbation is perceivable on the image but does not alter the content since only a few pixels are modified. The Jacobian-based Saliency Map Attack (JSMA) [Papernot et al., 2016b] described in Subsection 2.3.3 is an attack used to produce such sparse adversarial perturbations.

ℓ_1-norm Adversarial perturbations with ℓ_1-norm have not been thoroughly studied. They usually serve as convex surrogates with ℓ_0 norm [Chen et al., 2018]. Its convexity and its sparsity are two nice properties of this norm together with its linear nature[6]. If while looking for a perturbation with minimal ℓ_1-norm we find a perturbation with only one nonzero value, then we find at the same time a one-pixel perturbation. The best ℓ_1-norm known attack is given by [Chen et al., 2018].

ℓ_2-norm attacks This norm is popular even though it does not necessarily align with the human perception as shown in Figure 2.7. Using this norm on the MNIST dataset, it is also possible for instance to create an image of a 7 near the image of a 3, making it difficult to call the created 7 an adversarial example if it is actually a 7. Another drawback is that this norm does not facilitate the computation of bounds on pre-ReLU activations[7], compared to others. Using ℓ_2-norm and interval arithmetic give vacuous bounds on the activations even on the first layer. The ℓ_1-norm perturbation also suffers from this drawback. Carlini & Wagner attack [Carlini and Wagner, 2017b] is one of the most powerful $\ell_2 - norm$ attacks. The randomized Gradient-Free attacks [Croce and Hein, 2018, Croce et al., 2020] are also very powerful.

ℓ_∞-norm The infinite norm is the most popular of this batch. One reason is that it seems to align better with human perception than the other norm-based perturbation measures. It also has the advantage of giving tight bounds on the activations of the first layer using interval arithmetic. Projected Gradient Descent (PGD) [Madry et al., 2019] is among the most powerful ℓ_∞ attacks.

We can also find in the literature perturbations that respect simultaneous constraints of several norms. We can find an example of that in [Croce and Hein, 2019] where ℓ_0-norm is combined to ℓ_∞-norm to constraint the perturbations. That combination allowed them to find sparse perturbations that are less perceivable compared to the ones found using only ℓ_0-norm constraint.

[6] The minimization of a ℓ_1-norm is a linear program.

[7] more details about the pre-ReLU activation bounds on page 59, paragraph MIPVerify

Other Researchers have started looking at perturbations and measures beyond the vector norms. Simple geometric transformations such as translation or rotation are enough to fool classifiers as observed in [Engstrom et al., 2019b]. It is also sometimes sufficient to change the brightness, or the contrast of an image for classifiers to make mistakes [Yang et al., 2021]. We can also find more complex perturbations such as the Wasserstein Attack [Wong et al., 2019] using Sinkhorn projections. Trace-norm adversarial attacks [Kazemi et al., 2020] use the nuclear norm, the sum of the matrix singular values of the perturbation image matrix. Such perturbations are found by looking for adversarial perturbations with the minimal nuclear norm instead of minimal ℓ_p-norm.

2.3.2 Taxonomy of attacks

Since adversarial examples drew the attention of the community, a multitude of ways to build them, namely, adversarial attacks, have been designed. Those adversarial attacks can be classified according to parameters such as the goal of the attack, the level of knowledge used and some other specificities of the attack.

2.3.2.1 Classification according to the goal of the attack

There are mainly three goals that an attacker can pursue: reducing the confidence of the prediction, misleading the classifier into making a wrong prediction, or misleading the classifier into predicting a specific class.

Confidence reduction Here, the goal is to reduce the "confidence" of the neural network. It does not necessarily mean changing the classification, having a lower confidence for the correct class is the objective.

Misclassification The goal here is to trick the classifier into making a wrong prediction, whatever the wrong prediction may be. For example, the attacker may want to make a classification system used on a self-driving car to misclassify a stop sign into any other road sign. These are untargeted attacks as opposed to targeted attacks.

Targeted Misclassification The goal here is to trick the model into predicting a class chosen by the attacker. Taking the same example as in simple misclassification, an attacker can be more malicious and try to fool the classification system into classifying a stop sign into a priority road sign, which can be harmful to passengers.

2.3.2.2 Classification according to the knowledge of the model

Attacks can also be classified according to the level of knowledge that the attackers have about the attacked model. There are mainly two settings, white-box attacks, where the attacker has access to all available information, and black-box attacks, where attackers are only allowed to query the model. Gray-box setting is when the attackers have limited information about the model they want to attack.

White-box attacks In white-box setting, the attackers have full access to the network, they have access to the architecture, the parameters, the training data, with all available information. Exploiting that information allows attackers to find very subtle adversarial perturbations by using, for instance, the gradient of the loss function with respect to the input of the model.

Black-box attacks To prevent the attack, an intuitive idea is to restrict the access of the model. The model will then be considered as a black box, taking an input image and giving its class. However, even that is not enough to prevent the model from being successfully attacked. To attack a model in such a setting, the first generation of black-box attacks creates a model that they train

to mimic the behavior of the hidden model. They submit examples of their own dataset to the hidden model and train their surrogate model to have the same prediction. Once the surrogate consistently predicts the same class as the hidden model, white-box methods can be used to craft adversarial examples for the surrogate and count on the transferability of those adversarial examples to fool the hidden model. The new generation of adversarial does not go through all that gymnastic to fool the hidden model. They just query the hidden model and use some mechanisms to find adversarial perturbations and to progressively improve the quality of the adversarial perturbations [Brendel et al., 2018]. In that setting, the number of queries needed to find reasonable adversarial examples is taken into account to compare the efficiency of black-box attacks.

Gray-box attacks If knowledge about the architecture or (non-exclusive) about the training data are given to the attacker, then we talk about gray-box setting. The same algorithms as in black-box can be used while trying to exploit as much as possible the available information. It is maybe that fact that led some authors of the community to consider gray-box settings as part of black-box settings.

2.3.2.3 Specificities of the attack

An adversarial attack can produce perturbations that are untargeted or targeted, universal or image specific, adaptive or nonadaptive, one-step or multistep, can be performed during training time or at test time among other specificities of adversarial attacks.

Untargeted/Targeted Untargeted attacks are attacks performed with the goal of misleading a model to make a mistake without targeting the class to be predicted a priori. Targeted attacks are when not only we want to fool the model but we want it to predict a particular class.

Image specific/universal An attack can be image specific, meaning that given an image, the attacker tries to find a transformation of the image such that the transformed image is similar to the original one but that is misclassified. Most attacks in image classification are image-specific since the transformations/perturbations are different for each image. There exist also universal attacks. Unlike image-specific attacks, given a model, universal attacks aim at finding a single perturbation that can be added to any image to obtain an adversarial image with a high probability of fooling the model. Those universal adversarial perturbations were also shown to transfer between different models [Moosavi-Dezfooli et al., 2017a].

Non adaptive/Adaptive Most of the attacks used to challenge the robustness of neural networks are nonadaptive in the sense that they are not designed to exploit the weaknesses of defense mechanisms but rather rely on common ways to find adversarial examples. However, some defense mechanisms can neutralize the familiar attacks. To expose the vulnerability of these defenses, adaptive attacks have been created. We can find in [Tramer et al., 2020] examples of defenses that seem robust since they resist popular attacks and that have been defeated using adaptive attacks.

One step/multistep Attacks are either one-shot, meaning they perform only one step, or multistep, they perform several steps of the procedure they use to find adversarial attacks. Most single-step adversarial attacks can be looped in an iterative way to obtain a multistep attack. Iterative versions are more powerful than single step versions.For example, the Fast Gradient Sign Method (see Definition 2.1) uses a single gradient step to find adversarial examples, so it is a one-step attack. When several iterative gradient steps are used to obtain a Basic Iterative Method, we get a multistep attack.

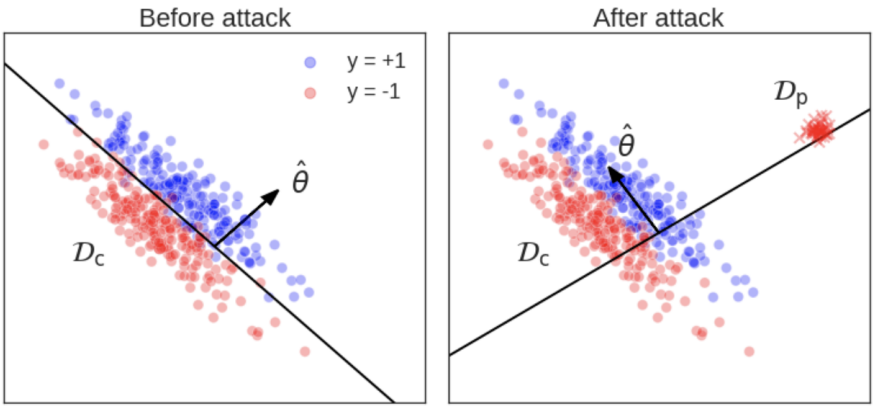


Figure 2.8 – Illustration of the effect of poisonous data on the decision boundary of a classifier [Koh et al., 2018].

Test time attacks/poisoning attacks Most of the attacks in the literature are test time attacks. They are performed once the models are trained and do not intervene during training. Poisoning attacks on the contrary, interfere with the training. There are two ways that poisoning attacks can interfere with training, increasing the error rate by moving the decision boundary to poisonous data or creating a backdoor. By modifying, adding, or deleting some data from the training set, the attackers can alter the decision boundary of the model and increase the error rate. It is illustrated in Figure 2.8, where we can see that the decision boundary obtained after the introduction of poisonous data is incorrect. Even though the poisonous data in the example can easily be identified as outliers, it is enough to prove the effect of data poisoning. More subtle ways of poisoning data exist [Koh et al., 2018]. Another target of the poisoning attack is the creation of a backdoor. In that case, the model has a normal behavior when a trigger is not present on the input. The trigger plays the role of the signal sent to the model by the attacker. When the trigger is detected, then the model would behave differently. For example, the provider of the spam detector can create a backdoor activated by a particular sentence, such that any spam containing that sentence will not be detected.

2.3.3 Adversarial attacks

Attacks presented here are essentially for image classification unless stated otherwise. The vast majority of attacks also rely on the gradient of the loss function, the cross-entropy loss, or another loss designed to enhance the fooling rate.

L-BFGS The paper [Szegedy et al., 2014] put the adversarial in the spotlight in the image classification field. They found that small (additive) adversarial noises are enough to fool the classifiers. Given a sample $\mathbf{x}$, they look for an adversarial example $\mathbf{x}'$:

$$\begin{cases} \min\limits_{\mathbf{x}' \in [0,1]^d} & \|\mathbf{x} - \mathbf{x}'\|_2^2 \\ \text{s.t.} & \mathrm{D}_f(\mathbf{x}') = t\,, \end{cases} \tag{2.8}$$

with $\mathrm{D}_f(\mathbf{x}') = \operatorname{argmax}_i f_i(\mathbf{x}')$, the decision of the network, t, a target class different of the true label associated with $\mathbf{x}$. This problem, being difficult to solve directly, is reformulated as followed:

$$\min_{\mathbf{x}' \in [0,1]^d} \quad c\,\|\mathbf{x} - \mathbf{x}'\|_2^2 + \mathscr{L}(t, f(\mathbf{x}'))\,, \tag{2.9}$$

where $\mathscr{L}$ is the cross-entropy loss function and $c > 0$ is a hyperparameter usually found with line search. The problem (2.9) is then solved using L-BFGS. If the objective function ($\mathbf{x}' \to \|\mathbf{x} - \mathbf{x}'\|_2^2 + \mathscr{L}(t, f(\mathbf{x}'))$) is convex, then (2.9) gives an exact solution, but this is generally not the case for the popular neural networks used in deep learning due to nonlinear activation functions. Since this optimization is a bit slow, some faster ways to find adversarial examples were created soon after.

FGSM and iterative variants Fast Gradient Sign Method [Goodfellow et al., 2015] was a popular attack until the apparition of Projected Gradient Descent [Madry et al., 2019] among other variants of FGSM.

Definition 2.1. *FGSM, in its original version, is a single step attack used to compute an adversarial attack with ℓ_∞-norm constraints. Given a model f and a sample $\mathbf{x}$ associated with the class $\mathbf{y}$, FGSM compute an adversarial noise δ:*

$$\delta = \varepsilon \; sgn\left(\nabla_{\mathbf{x}} \mathscr{L}(\mathbf{y}, f(\mathbf{x}))\right). \tag{2.10}$$

The computation of (2.10) is fast thanks to the use of GPUs. FGSM exploits the local linearity of models to find adversarial examples. There are some trivial variants of FGSM:

- Random Step-FGSM [Tramèr et al., 2020] where a small random step is taken before computing the gradient of the loss.
- Least Likely Class Method [Kurakin et al., 2017] where the least likely class is targeted. Let us define $\mathbf{y}_{\mathrm{LL}} = \mathrm{argmin}_i \, f_i(\mathbf{x})$, then the associated adversarial example $\mathbf{x}'_{\mathrm{LL}}$ is defined as follows:

$$\mathbf{x}'_{\mathrm{LL}} = \mathbf{x} - \varepsilon \; \mathrm{sgn}\left(\nabla_{\mathbf{x}} \mathscr{L}(\mathbf{y}_{\mathrm{LL}}, f(\mathbf{x}))\right). \tag{2.11}$$

  Here, we use a minus to go in the direction that will decrease $\mathscr{L}(\mathbf{y}_{\mathrm{LL}}, f(\mathbf{x}))$, i.e. increasing the chance of $\mathbf{x}'$ to be recognized as belonging to the class $\mathbf{y}_{\mathrm{LL}}$.
- Basic iterative method **(BIM)** / projected gradient descent **(PGD)**, which is applying several steps of FGSM (it can also be applied to its variants):

$$\mathbf{x}'_0 = \mathbf{x}, \quad \mathbf{x}'_{i+1} = \mathrm{C}lip_{\mathscr{X},\varepsilon}\{\mathbf{x}'_i + \alpha_{bim} \mathrm{sgn}\left(\nabla_{\mathbf{x}} \mathscr{L}(\mathbf{y}, f(\mathbf{x}'_i))\right)\}, \tag{2.12}$$

  where $\mathrm{C}lip_{\mathscr{X},\varepsilon}$ ensure that a valid adversarial example is found at each step and that is within distance ε of the original example. The positive scalar α_{bim} is another parameter, such that $n_{step}\,\alpha_{bim}$ is equal to the value ε, with n_{step} the number of iterative steps. Sticking to this definition and allowing the $n_{step}\,\alpha_{bim}$ to be much greater than ε, we obtain the popular Projected Gradient Descent (PGD) [Madry et al., 2019] which uses $\mathrm{C}lip_{\mathscr{X},\varepsilon}$ to project back examples that go outside of the desired region.
- There are other variants such as, for example, the variant using momentum [Dong et al., 2018].

JSMA The Jacobian-based Saliency Map Attack (JSMA) [Papernot et al., 2016b] is an attack aiming at fooling classifiers while modifying the fewest pixels, so an attack under ℓ_0-norm. This attack is based on targeting the pixels that are more susceptible of causing a model to fail and modifying them to fool the model. They use the Jacobian matrix of the logit outputs (pre-softmax) $f(\mathbf{x})$, $\nabla_{\mathbf{x}} f(\mathbf{x})$ to select the pixel to be modified. Once that pixel is modified, they repeat the same process on the obtained image, recomputing the Jacobian matrix, and selecting the most harmful pixel. That process is repeated until the total number of pixel modifications allowed is attained or an adversarial example is found.

One pixel attack After JSMA proved that it was possible to fool a network by modifying a few pixels of the input image, [Su et al., 2019] showed that it was possible to go further by fooling the network by altering as little as one pixel. They used the Differential Evolution concept [Das and Suganthan, 2011] belonging to the general class of evolutionary algorithms. A strength of this approach compared to other evolutionary algorithms is that it keeps the diversity of the population. Differential Evolution does not exploit gradient-based information and thus does not need the objective function that it optimizes to be differentiable. In practice, for a given example, they create a population in $\mathbb{R}^5$, each vector containing the abscissa and the ordinate, then the RGB values of the selected candidates. After that, the elements are randomly modified creating children. Each child competes with its parent and the fittest is conserved for the next generation. One pixel attack is very successful at finding an adversarial example by modifying a single pixel with the model having high confidence of the wrong prediction.

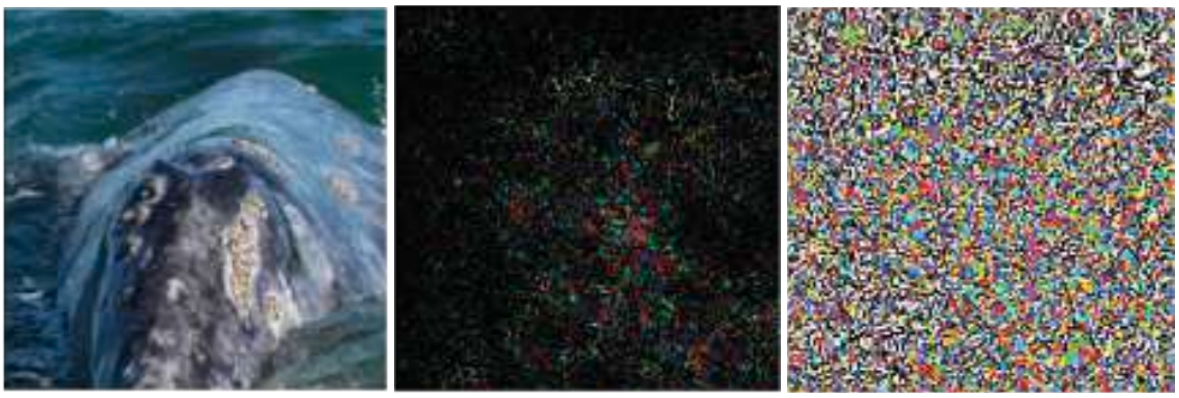


Figure 2.9 – Comparison between the noise perturbations produced by DeepFool and FGSM. Left: the clean image classified as a whale; middle: perturbations produced using DeepFool; right: perturbations produced using FGSM. The perturbations cause the model to predict a turtle instead of a whale.

Carlini & Wagner The Carlini & Wagner attack [Carlini and Wagner, 2017b] is a gradient based attack that can be used to generate ℓ_2, ℓ_∞ and ℓ_0 norm adversarial perturbations. It is nevertheless mostly used with ℓ_2 perturbation, for which it is among the strongest attacks. The Carlini & Wagner attack can be related to [Szegedy et al., 2014], which it improves by using some tricks to use the Adam optimizer that does not naturally support box constraints. It also uses a loss that is different from the one used during training for attack models. It was used to defeat the distillation defense [Papernot et al., 2016c] which had so far resisted to the available attacks.

DeepFool Another algorithm named DeepFool was proposed in [Moosavi-Dezfooli et al., 2016] to compute iteratively adversarial perturbations with small norms. Deepfool takes the original example as an initialization and assumes the linearity of the decision boundaries around it. It then computes, using the linear boundary assumption, the minimal distortion needed to reach the decision boundary. Then, from the new point, the same operation is repeated until an adversarial example is found. The perturbations are accumulated at each step. In practice, the adversarial perturbations produced using DeepFool are less noisy than those produced using FGSM (or its versions) as shown in Figure 2.9. The reason is that it is not necessary for the DeepFool perturbation to modify every pixel with the same magnitude as it is done using FGSM for instance. Deepfool is also used to create *universal adversarial perturbations* that we will see now.

Universal Adversarial Perturbation All attacks cited previously create a perturbation for each given clean example. What about the existence of a single adversarial noise that can be added to any given image to fool the classifier with high probability? Such an adversarial perturbation was proposed in [Moosavi-Dezfooli et al., 2017a] under the name of "universal adversarial perturbation" as opposed to image-specific perturbations. For a given model, to create such a perturbation, they select a certain number of images, for example, 2000 examples, and create for those examples a unique adversarial perturbation that fools the model on them. That universal adversarial perturbation will then generalize with high probability to other examples. It has also been shown that those universal adversarial perturbations transfer across different models. In practice, they initialize the perturbation δ_u with a 0 matrix of the convenient shape. Then for each example, if the perturbation does not fool the image, an image-specific adversarial perturbation δ is computed to fool the model. Once such a perturbation is found, it is added to δ_u which becomes: $\delta_u = \delta_u + \delta$. The universal adversarial perturbation is also projected to the desired region at some points during execution. Other universal adversarial attacks can be found in [Chaubey et al., 2020] along with some defenses against them.

Adversarial examples generated with Generative models or auto-encoder Auto-encoders can be used to generate adversarial examples as shown in [Baluja and Fischer, 2017]. Their approach,

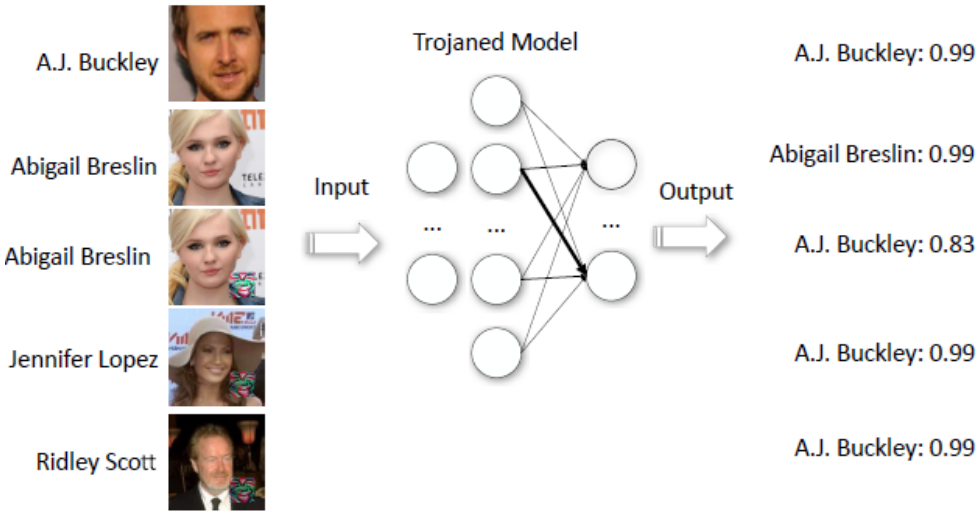


Figure 2.10 – Illustration of a trojaning attack. We can see the colored trigger near the lower right corner on the last three images. Image taken from [Liu et al., 2017].

called Adversarial Transformation Networks, is to train an auto-encoder taking clean images as input that provides as output an adversarial image against a network or a set of networks. The loss used to train the auto-encoder encapsulates two parts. The first part is the classical reconstruction loss that encourages the encoder to produce an output image similar to the input, and the second part encourages to produce an image that is misclassified by the classification network or a set of networks. Once Adversarial Transformation Networks are fully trained, they can be used to generate adversarial examples in a black-box setting. A similar attack, called AdvGAN, has been proposed more recently by [Xiao et al., 2019a].

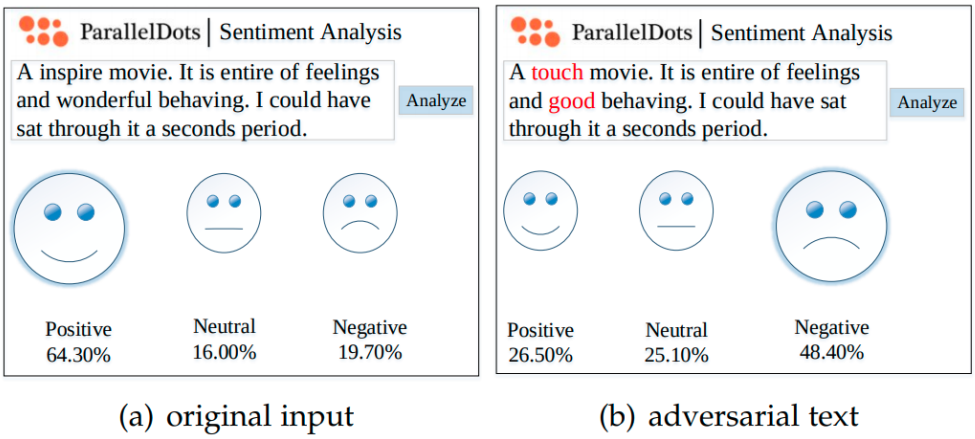
Trojaning attacks, adversarial patches and eyeglasses A trojaning attack hides a certain function in the networks that will be activated by a certain trigger. The trojaned networks would perform as expected when the trigger is not detected, and would act differently when the trigger is detected as illustrated in figure 2.10. We can see [Liu et al., 2017] as an example of poisoning attack. Given a model, they select some neurons that have high influence over the classification, the neurons that are the easiest to manipulate. The neurons with the highest sum of absolute value of incoming weights. For those neurons, they create trojan triggers by maximizing the activation of those neurons. They create a dataset by reverse-engineering the activations of the neurons of the model, i.e., maximizing the activation of those neurons and storing the images that maximize their activation to form a new dataset. And for each element of that dataset, they create a trojaned version, then they retrain the model such that it behaves normally when the trigger is not present and makes a wrong prediction when a trigger is on the image.

Similarly, in [Brown et al., 2018], the authors present an attack that they named Adversarial Patch, a way to craft "universal, robust, targeted adversarial image patches in the real world". The patches created are like stickers that can be put on any image to change the classification.

The same kind of patches has been proposed to attack facial recognition systems [Sharif et al., 2016] (a real-world attack). They consist of eyeglasses that one can wear to *evade being recognized or to impersonate another individual.*

This type of attack can be very damaging, especially considering how common it is to download pretrained models to fine-tune them.

Beyond simple image classification Most of the attacks presented here are relative to image classification, in the following paragraphs, we are presenting attacks on some other domains as illustrations of the existence of adversarial on other domains. The goal is to show that adversarial



(a) original input (b) adversarial text

Figure 2.11 – Example of adversarial examples in natural language processing. The attack was performed on ParallelDots's sentiment analysis tool , a popular text analysis system. Image taken from [Wang et al., 2021].

examples are observed well beyond classification.

Adversarial attacks on Speech-to-text Voice assistants that process sound to execute commands are also prone to adversarial examples. Carlini and Wagner show that it is possible to attack algorithms used in automatic recognition that are employed by voice assistants [Carlini and Wagner, 2018]. They produce sounds that are similar but are associated with very different texts[8].

Attacks on natural language processing There are also adversarial examples on natural language processus [Wang et al., 2021] as illustrated with Figure 2.11. By modifying slightly a text, while maintaining the same idea, it is possible to mislead the models to have a wrong output.

Attacks on autoencoders and Generative Models Adversarial examples for autoencoders were studied in [Tabacof et al., 2016]. Their attack was able to modify the input image leading the autoencoder to recreate a completely different image instead of the input image. To create adversarial examples, they manipulate the internal representation of the adversarial image to make it similar to that of the target image. Other methods were proposed in [Kos et al., 2017] to attack variational autoencoder (VAE) and VAE-GAN.

Attacks on deep reinforcement Learning Deep reinforcement learning is used to efficiently train a computer to play games against humans. Computers powered with AI are able to beat the best human in some games, namely, AlphaGO [Holcomb et al., 2018]. However, models trained with deep reinforcement learning are not free from adversarial vulnerabilities as illustrated in Figure 2.12. In that image, we can see the normal behavior of the model taking the right action on the clean image. Then we can see it will take a wrong action due to adversarial manipulation of the image. FGSM was used to create an adversarial attack on the model trained using reinforcement learning techniques [Huang et al., 2017]. Several other attacks on deep reinforcement learning can be found in [Chen, 2019].

Attacks on semantic segmentation and object detection Viewing the segmentation and object detection tasks as "classifying multiple targets in an image" [Akhtar and Mian, 2018], [Xie et al., 2017] compute adversarial examples for image segmentation and detection. The target is described as "a pixel or receptive field in segmentation, and object proposal in detection". With

[8]Audio files and their transcriptions are available at `https://nicholas.carlini.com/code/audio_adversarial_examples`

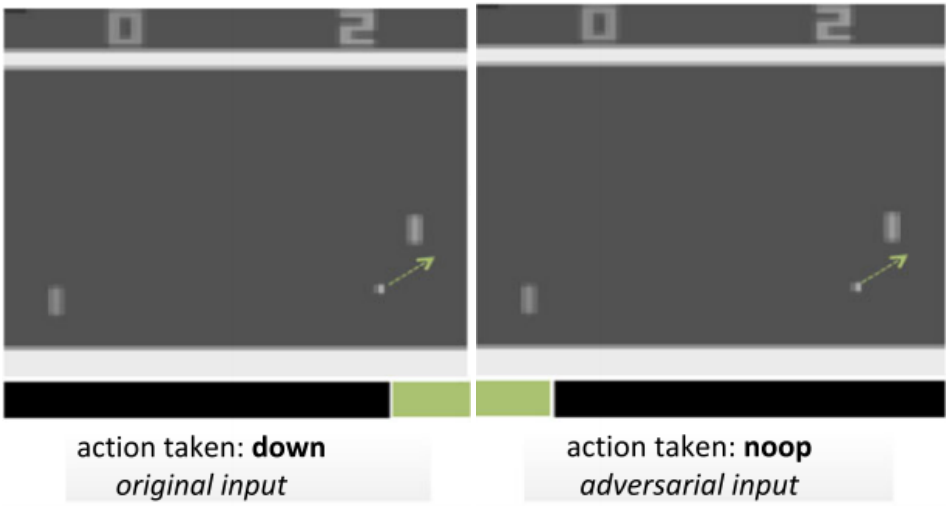


Figure 2.12 – Example of adversarial attack on Deep Reinforcement Learning. On the left image, the normal behavior of the model, we can see it takes the right decision. On the right, an adversarial image where a wrong decision is taken. The attack on the image was done using FGSM.

this viewpoint, they proposed "Dense Adversary Generation" that optimizes a loss function over a set of pixels to generate adversarial examples. Illustrations are given in Figure 2.13.

Transition/Summary Although deep learning algorithms are widely used, it is quite easy to fool them due to their brittleness. There is a large body of work showing ways to attack models in image classification. Table 2.1 summarize them with some of their attributes. Nevertheless, the challenge that adversarial examples poses goes well beyond image classification. Adversarial examples arise in several domains with different data, structured or not. They arise in domains like such as image segmentation, object detection, speech-to-text, natural language processing...

Table 2.1 – Summary of the attributes of diverse attacking methods. The column 'Perturbation' indicates the p-norm of the perturbations for which the attack is the most efficient. In the last column , the strength (higher for more stars) is based on our practice of the attack. This Table is adapted from Table 1 in [Akhtar and Mian, 2018].

Method	Type	Targeted or Not	Specific/Universal	Perturbation	Learning	Strength
L-BFGS	White box	Targeted	Image specific	ℓ_∞	One shot	$\star\star$
FGSM	White box	Targeted	Image specific	ℓ_∞	One shot	$\star\star$
PGD	White box	Non targeted	Image specific	ℓ_∞	Iterative	$\star\star\star$
JSMA	White box	Targeted	Image specific	ℓ_0	Iterative	$\star\star$
One-pixel	Black box	Non targeted	Image specific	ℓ_0	Iterative	$\star$
C&W attacks	White box	Targeted	Image specific	ℓ_2	Iterative	$\star\star\star$
DeepFool	White box	Non targeted	Image specific	ℓ_∞	Iterative	$\star\star\star$
Uni. perturbations	White box	Non targeted	Universal	ℓ_∞	Iterative	$\star\star$
Auto-Encoder	White box	Targeted	Image specific	ℓ_2	One shot	$\star$

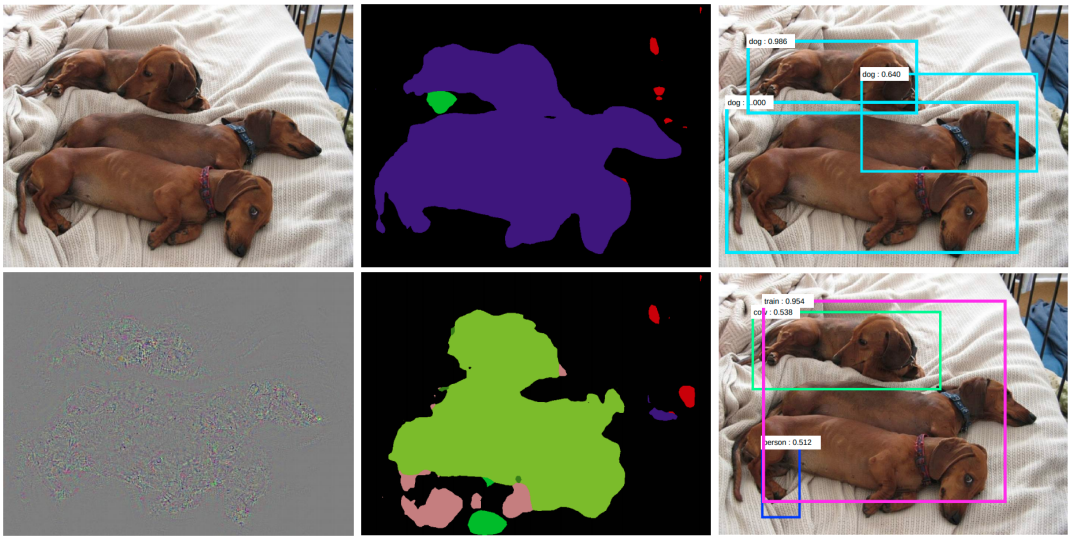


Figure 2.13 – Adversarial example illustration for image segmentation and detection from [Xie et al., 2017]. On the first row from left to right, we have a clean example, its segmentation by [Long et al., 2015] and the results of its segmentation by [Ren et al., 2016]. On the second row, the adversarial perturbations multiplied by 10 to be visible, the segmentation of the adversarial image obtained and its detection. In the segmentation, the color purple represents the class dog and the light green represents the class person.

2.4 Adversarial defenses

Now that we have presented adversarial examples and different attacks to generate them, there remains legitimate questions. How to prevent them? How to defend against them? In this section, we review methods for training neural networks to be robust to adversarial attacks under some threat models. There are three main ways to defend against adversarial examples. The first way is via robust optimization. The second way is to detect adversarial examples. The third way is by using gradient masking[Silva and Najafirad, 2020]. But first, let us start by formulating the adversarial classification challenge.

A formulation of adversarial classification In contrast to normal classification (introduced in the Section 2.1) which aims at classifying the input sample, adversarial classification intended to guarantee that no input near that input sample is classified differently. Since the classification loss measures the correctness of the classification, adversarial classification aims at having the worst case loss around an input to be low, to guarantee the correctness of the decision taken by the classification around that input. Hence, in adversarial training, instead of minimizing the empirical loss defined in Equation (2.3) as it is done in normal classification, the following quantity is minimized with respect to f:

$$\frac{1}{\mathrm{N}}\sum_{i=1}^{\mathrm{N}} \max_{\|\mathbf{x}^{(i)}-\mathbf{x}\|_{\mathscr{X}}\leq\varepsilon} \mathscr{L}(\mathbf{y}^{(i)}, f(\mathbf{x})), \tag{2.13}$$

with ε a positive scalar controlling the "volume" of the desired region of robustness. This is the quantity that adversarial defenses (those relying on robust optimization) optimize, while other defense strategies rely on some tricks that are mostly ineffective [Carlini et al., 2019].

2.4.1 Taxonomy of defenses

Following a broad categorization proposed in [Akhtar and Mian, 2018], we can classify defense mechanisms in three categories that can overlap:

- defenses that modify the data. Adversarial training is a good example of data modification to achieve robustness. A less typical method to train robust models is using "robustified"

version of the dataset as proposed in [Ilyas et al., 2019] where nonrobust features were withdrawn from the training images. The data modification can also occur at the test time, where input images are preprocessed with the aim of defending against adversarial examples.
- defenses that modify the model itself. Defensive distillation [Papernot et al., 2016c] that will be discussed is an example of that.
- defenses that use another model to defend against adversarial examples, using, for example, a GAN or an Auto-encoder to preprocess the input data so that it is well classified.

Adversarial defenses can also be categorized according to the fact that they are deterministic or not, empirical or provable, that they have a linear behavior, or they are highly nonlinear.

Deterministic vs Stochastic Most of the defenses we are going to present are deterministic. However, there are some defenses that are stochastic, such as Stochastic Activation Pruning [Dhillon et al., 2018] where a random subset of activations are pruned and the others are scaled up to compensate. Authors claim that this method gives robustness to the models and increases accuracy.

Empirical vs provable Defenses are generally empirical. However, there are some provable defenses such as [Wong and Kolter, 2018] and [Raghunathan et al., 2020] that use mathematical tools to prove the robustness of their models.

linear vs highly nonlinear When we talk about the linearity of a model, we also include the piecewise linear model. ReLU being the most popular activation function, the models are generally piecewise linear. Models with activations such as sigmoid or hyperbolic tangent, can be put in the same group since they are not highly nonlinear. When we refer to highly nonlinear models, we think about models using a rectified polynomial unit with degree going up to 30 as in [Krotov and Hopfield, 2018].

We can also categorize the defenses according to the fact that they rely on robust optimization, on adversarial detection or gradient masking.

Robust optimization Defenses based on robust optimization are those based on approaches that improve the robustness of the model by using regularization, certification bounds, or adversarial examples during adversarial training. It is for example the case of adversarial training where adversarial examples are used during training, and all the defense mechanisms minimizing 2.13.

Adversarial example detection Another approach is to recognize the inputs given to the classifier when they are adversarial. If adversarial examples are recognized, they can be ignored, therefore no need to classify them.

Gradient masking Another defense mechanism consists in denying the attacker access to meaningful gradients to use to create attacks. This can be done by using functions that are not differentiable. In that case, it is not possible to compute the gradient. Some tricks can also be used to obfuscate the gradients as in k-WTA [Xiao et al., 2019b], illustrated Figure 2.14.

The taxonomy allows to compare different defense mechanisms. As it could be unfair to compare deterministic mechanisms to stochastic mechanisms, for example. In the following, we present defenses that cover the different aspects stated above. As for the attacks, we present the defenses also based on their popularity, their efficiency, their originality, the goal being to give an overview of the existing defenses.

2.4.2 Empirical defenses

The first defenses against adversarial examples that emerged were empirical defenses. Defensive Distillation [Papernot et al., 2016c] is ones of them. It seemed robust at first, but more powerful attacks showed it was not.

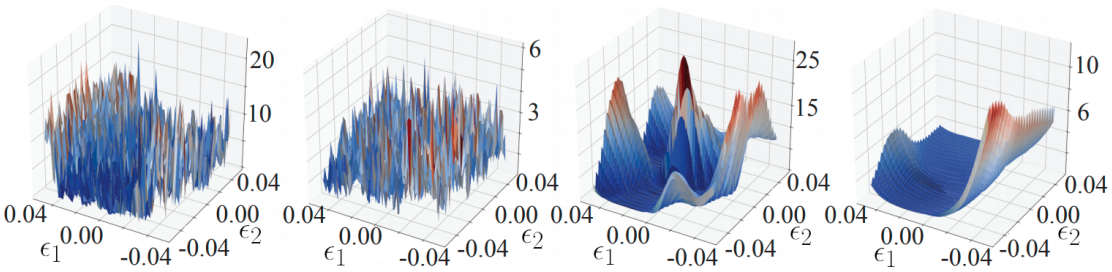


Figure 2.14 – Example of gradient obfuscation. From left to right, we have a visualization of the loss of Resnet18 model trained with k-WTA [Xiao et al., 2019b] only, another adversarially trained, a Resnet18 trained with ReLU activations, and the last one trained by adversarial training. We can see that the model trained with k-WTA has a loss landscape that is very rugged compared to the one of the model trained with ReLU activations. We can also observe that adversarial training has a smoothing effect on the loss landscape.

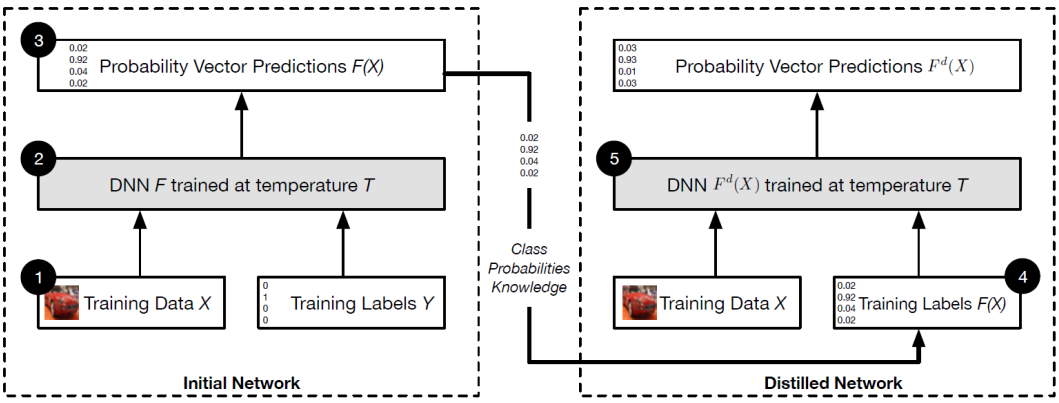


Figure 2.15 – On the left, the training of the "teacher" network and on the right, the training of the "student" network, the distilled one trained with the outputs of the "teacher" network. Image taken from [Papernot et al., 2016c].

Defensive distillation In [Hinton et al., 2015], the concept of distillation is introduced as a training method allowing to transfer knowledge from a network with a lot of parameters to another one with fewer parameters. This concept was later used in [Papernot et al., 2016c] to defend against adversarial attacks. How does defensive distillation work? A first neural network is trained with a modified softmax function in the final layer, introducing a constant $T^\circ \geq 1$, with $f(\mathbf{x})$ the pre-softmax output for the input $\mathbf{x}$ and C is the number of classes:

$$\text{softmax}_{T^\circ}(f(\mathbf{x})) = \frac{\exp\left(f_i(\mathbf{x})/T^\circ\right)}{\sum_{c=1}^{C} \exp\left(f_c(\mathbf{x})/T^\circ\right)}. \tag{2.14}$$

Once this network is trained, it is considered as the "teacher" and its logits (pre-softmax activation) are going to be used as training labels for a second network called the "student", as illustrated in Figure 2.15. Typically, higher temperatures are used for the student network since it makes the gradient vanish. Defensive distillation has the effect of flattening the softmax while the parameter T° grows. It becomes more obvious, when we take the limit $T^\circ \to \infty$ of the softmax, since, for any input $\mathbf{x}$, we obtain the same output $\frac{1}{C}$ for all classes. And having a constant output means the gradient can not be exploited to craft adversarial examples. Then, we can say that defensive distillation uses gradient obfuscation to defend against attacks. It gave a false sense of robustness until it was defeated by the Carlini & Wagner attack [Carlini and Wagner, 2017b] which showed that distilled models are as vulnerable as undistilled models. Adversarial examples that fool the undistilled models also fool the distilled ones.

Adversarial training and its variants The idea that using adversarial examples to train models to be robust against adversarial modification emerged very early. The core idea of adversarial training is to generate adversarial examples on the fly and to train on them. But the generation of adversarial examples was too time consuming and prevented people from creating adversarial examples on the fly to robustify the models. This problem was solved using FGSM [Goodfellow et al., 2015]. FGSM was used to prove experimentally the effectiveness of adversarial training. More recently, PGD, that we presented as an iterative variant of FGSM, has been found to be preferable to FGSM for creating adversarial examples. Even though training with PGD can take more time, experimentally, models trained with PGD are more robust than the one trained with FGSM.

Given an adversarial example $\mathbf{x}'$ near a clean example $\mathbf{x}$, the robust model is trained by minimizing the adversarial training loss, a convex combination of the loss on the clean data and the loss on a corresponding adversarial example:

$$\mathscr{L}_{adv} = \alpha\mathscr{L}\left(f(\mathbf{x}), \mathbf{y}\right) + (1-\alpha)\mathscr{L}\left(f(\mathbf{x}'), \mathbf{y}\right), \tag{2.15}$$

where $\alpha \in]0,1]$ balances the "importance" given to the loss at $\mathbf{x}$ and $\mathbf{x}'$. The value $\alpha = 0$ means training only on adversarial samples, and the $\alpha = 1$, normal training without adversaries. The value $\alpha = 0.5$ seems to work well in practice even though other values could give better results. This approach is generally used with PGD as said previously, but the idea is generic enough to be used with any other way of generating adversarial examples. Adversarial training has a smoothing effect on the loss function as shown in Figure 2.14. There exist also a theoretical motivation of adversarial training coming from the robust optimization field [Tsiligkaridis and Roberts, 2021, Shaham et al., 2018]. Adversarial training is the most popular, it has also been proven to be relatively efficient against attacks.

Stochastic Activation Pruning It is also possible to introduce randomness in the "structure" of the network to defend against adversarial attacks. It is that idea that is used in [Dhillon et al., 2018] with Stochastic Activation Pruning (SAP), where some activations are randomly pruned by being set to 0. This approach has similarities with dropout at test time [Srivastava et al., 2014]. A difference between the latter and SAP is that, while the pruning probability is uniform for dropout, for SAP it depends on the magnitude of the activations. The survival probability of an activation, i.e. the probability of not being pruned, is:

$$p_j^i = \frac{|\widehat{\mathbf{z}}_j^i|}{\sum_{k=1}^{\mathrm{K}^i} |\widehat{\mathbf{z}}_k^i|}, \tag{2.16}$$

where $\widehat{\mathbf{z}}_j^i$ is the the activation of the j-th neuron of the i-th layer and K^i is the number of activations on the i-th layer.

We can see that neurons with smaller activation magnitudes have less chance of surviving, meaning they are more likely to be pruned than neurons with higher activation magnitudes. The magnitudes of the surviving neurons are rescaled so that the pruned model behaves like the "natural" one on clean samples while being more robust against adversarial attacks. The stochasticity of the model is believed to make it more robust against attacks because the attacker has more trouble accessing the gradient. In other words, SAP relies on gradient masking and is an example of a defense modifying the model. SAP was empirically proved effective in increasing the robustness, for example, of [van Hasselt et al., 2015] in reinforcement learning. However, SAP has been shown to be vulnerable to an iterative attack [Athalye et al., 2018a] relying on gradients computed using *Expectation Over Transformation* [Athalye et al., 2018b].

Thermometer encoding As seen previously, one of the reasons that was thought to cause the existence of adversarial examples was the linearity of the model (*cf* the linearity hypothesis introduced page 38). The idea that breaking that linearity should lead to a more robust model is exploited in [Buckman et al., 2018]. They propose to quantize the inputs and then use *thermometer*

encoding that works as follows. Let q be the quantization function. Suppose $x \in [0,1]$ a real number, let us say the value of pixel for instance, $q_l(x)$ returns the smallest value k such that $x < \frac{k}{l}$, with l a positive integer and $k \in \{1,...,l\}$. As illustration, if $x = 0.55$, and $l = 10$ $q_{10}(0.55) = 6$. This quantization allows to erase the effect of some small perturbations, for example, $q_{10}(0.51) = q_{10}(0.55) = q_{10}(0.59) = 6$. Then thermometer encoding of a real number x and $k \in \{1,...,l\}$ is:

$$\mathrm{TE}_k(x) = \mathrm{TE}_k\left(q_l(x)\right) = \begin{cases} 1 & \text{if } k > q_l(x), \\ 0 & \text{otherwise}. \end{cases} \tag{2.17}$$

For example, using the example value as in the previous example TE(0.55) = [0000011111]. The TE representation is nonlinear and non-differentiable, making it impossible for the attacker to directly have a gradient to exploit to create adversarial. Combining this representation with adversarial training leads to models that are robust to PGD attack because this method uses gradient masking via a modification of the input. However, TE encoding alone is not sufficient since [Athalye et al., 2018a] showed that under strong enough attack, the accuracy of the model drops to 0 (or nearly 0) , meaning TE does not imply a real improvement alone. They also showed that combining TE and adversarial training can lead to less robust models than using adversarial training alone.

Parseval Network Another way to train a more robust neural network is through regularization. Parseval Networks [Cisse et al., 2017] are an example of regularization that aims at training more robust networks. The idea developed in this paper is to train the neural network in a way that allows to control the difference of the loss at two points $\mathbf{x}$ and $\mathbf{x}'$ by the difference of their norm multiplied by a constant, the Lipschitz constant. A relation such as:

$$\forall \mathbf{x}, \mathbf{x}' \in \mathscr{X}, \forall \mathbf{y} \in \mathscr{Y}, |\mathscr{L}(f(\mathbf{x}), \mathbf{y}) - \mathscr{L}(f(\mathbf{x}'), \mathbf{y})| \leq \lambda_p \|\mathbf{x} - \mathbf{x}'\|_p, \tag{2.18}$$

with λ_p, the Lipschitz constant for the p-norm associated with $\mathscr{L}(f(.),.)$. Parseval networks are trained to have a small of λ_p. Seeing networks as a composition of functions, they have to ensure that each function of the neural network is Lipschitz function with a small Lipschitz constant. The training is designed to maintain the spectral norm of the weight matrices to 1. For a weight matrix $\mathrm{W} \in \mathbb{R}^{d_{in} \times d_{out}}$ with $d_{out} \leq d_{in}$, Parseval regularization maintains $\mathrm{W}^\top \mathrm{W} \approx \mathrm{I}_{d_{out}}$, with $\mathrm{I}_{d_{out}}$ the identity matrix of $\mathbb{R}^{d_{out} \times d_{out}}$. Combining Parseval Networks with adversarial training seems to give a robust model. Among other benefits of Parseval Networks, they were shown to converge faster than the corresponding vanilla version and they are believed to make a better use of the capacity of the networks. Parseval networks have resisted FGSM, however, more powerful methods such as PGD should break it.

Miscellaneous Detecting adversarial examples is by itself a hard task as shown in [Carlini and Wagner, 2017a]. We did not talk about the adversarial example detection in detail for several reasons. The first reason is that their effectiveness is not proven, to the best of our knowledge. In [Carlini and Wagner, 2017a], all ten detection systems that were studied were successfully bypassed even if it required the use of a particular loss. Sometimes, even attackers that are oblivious to the detection mechanism manage to fool the detection system. Researchers are yet to produce a detection system that is robust. And it seems to be a very difficult task since usually the models used can be targeted and fooled.

Using ensembles of models is also not enough to be robust to adversarial attacks since directions that fool simultaneously several models can be found as shown by *universal adversarial perturbation* [Moosavi-Dezfooli et al., 2017a]. Preprocessings like blurring, cropping, compressing are also useless, especially when the attacker is aware of the preprocessings used. And even in the case when the preprocessings are randomly chosen, it is still possible to attack successfully the model by using for instance the *Expectation Over Transformation* [Athalye et al., 2018b].

Many of the empirical defenses have been broken after being found to be effective. This calls for a more rigorous evaluation of the robustness of neural networks, as we will see in the Section

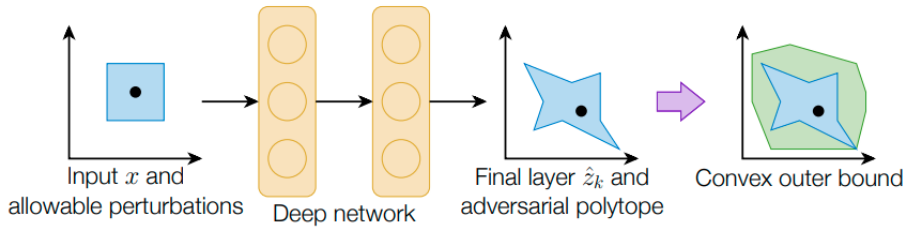


Figure 2.16 – Conceptual illustration of the (non-convex) adversarial polytope, and an outer convex bound [Wong and Kolter, 2018].

2.5. Some defense mechanisms were designed to give theoretical guarantees or empirical guarantees that can be more or less easily proven by the methods which will be presented in the Section 2.5. They are called the provable defenses.

2.4.3 Provable defenses

Defending against adversarial is a very difficult task. Several defenses published at top conferences have been proven ineffective and been completely broken for the most part by either using existing attacks at publication time or by creating new ones more suited for the task of breaching that defense. Hence, the needed of provable defenses that can be proven to be efficient. Those provable neural networks are usually piecewise to allow verification.

Provable defenses via convex outer adversarial polytope The first provable defenses to be published in image classification are [Wong and Kolter, 2018] and [Raghunathan et al., 2020]. Those two methods use an estimation of the upper bound of the adversarial loss to train provably robust models. We will focus on [Wong and Kolter, 2018], which is more representative of the approach than [Raghunathan et al., 2020], which were designed for smaller networks. [Wong and Kolter, 2018] also has another version [Wong et al., 2018] that scales their defense to larger networks.

The main idea of [Wong and Kolter, 2018] is the computation of a convex outer approximation of the set of activations that can be reached by a norm-bounded perturbation. Ideally, the set of activations is directly computed and used, but this set is generally not convex and is difficult to obtain. Then the authors developed a robust optimization method minimizing the worst case over the outer approximation. In Figure 2.16, we have an illustration showing the set of activations reachable by a norm-bounded perturbation around a clean input and an outer convex polytope. The worst case loss over this outer region is minimized using a linear program during training. At the test time, all examples around which there are adversarial examples are successfully found. However, due to the fact that an upper bound of the worst case loss is used, some robust examples can be wrongly flagged as having adversarial examples around them. Simply said, if there are adversarial examples around the sample, this method is aware of their presence, but sometimes it wrongly says that they are adversarial examples. Even more simply, if this method says an example is robust, then it is robust, and if it says there are adversarial examples, it can sometimes be wrong.

Provable Robustness of ReLU networks via Maximization of Linear Regions In [Croce et al., 2019], the provable robustness problem was discussed in a very interesting way. Working with ReLU activation, the functions f, representing the neural networks they use, are piecewise linear. Then any input $\mathbf{x}$ can be affected to a linear region. Their idea to promote robustness and provability is to maximize the linear regions containing the training points. If the linear regions around the points are large enough, for each point, proving robustness within a p-norm ball contained in that linear region or finding an adversarial example can be done by solving a few linear programs. To explain the idea more clearly, given an input $\mathbf{x}$, let us use the two distances they defined:

- $d_{\mathrm{D}}(\mathbf{x})$, the distance between the input $\mathbf{x}$ and the decision boundary,
- $d_{\mathrm{B}}(\mathbf{x})$, the distance between the input $\mathbf{x}$ and the border of the linear region containing $\mathbf{x}$.

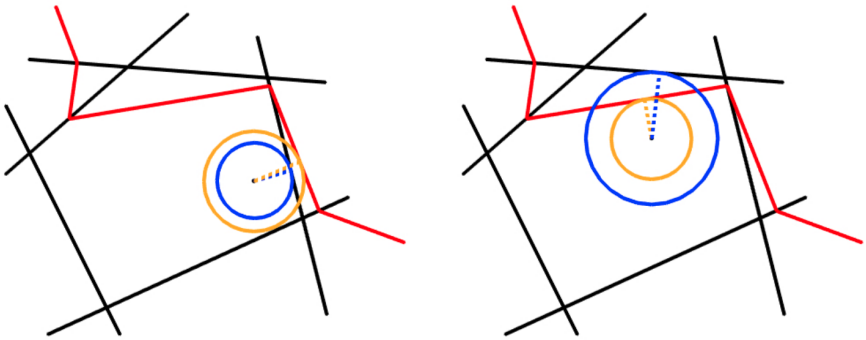


Figure 2.17 – **Left:** the input $\mathbf{x}$ is closer to the border of the linear region containing $\mathbf{x}$ (black) than to the decision boundary (red). In this case the smallest perturbation that leads to a change of the decision lies outside the linear region of $\mathbf{x}$. **Right:** the input $\mathbf{x}$ is closer to the decision boundary than to the border of the linear region containing $\mathbf{x}$, so that the projection of the point onto the decision hyperplane provides the adversarial example with smallest norm (Figure taken from [Croce et al., 2019]).

An illustration is given in Figure 2.17, in which we can see two possible configurations: $d_{\mathrm{D}}(\mathbf{x}) \leq d_{\mathrm{B}}(\mathbf{x})$ or $d_{\mathrm{B}}(\mathbf{x}) \leq d_{\mathrm{D}}(\mathbf{x})$. This figure gives also an intuition about the following theorem:

Theorem 2. *Explaining consequences of each setting:*

1. *if $d_{\mathrm{B}}(\mathbf{x}) \leq d_{\mathrm{D}}(\mathbf{x})$, then $d_{\mathrm{B}}(\mathbf{x})$ is a lower bound on the minimal ℓ_p-norm of the perturbation necessary to change the class*
2. *if $d_{\mathrm{D}}(\mathbf{x}) \leq d_{\mathrm{B}}(\mathbf{x})$, then $d_{\mathrm{D}}(\mathbf{x})$ is equal to the minimal adversarial ℓ_p-norm perturbation necessary to change class.*

In other words, if we restrict to the linear region of the original input and compute the $d_{\mathrm{B}}(\mathbf{x})$ and the minimal adversarial distortion in that linear region, there are two possibilities:

- either $\mathbf{x}$ is closer to the border of the its linear region, in that case $d_{\mathrm{B}}(\mathbf{x})$ is a lower bound of the distance $d_{\mathrm{D}}(\mathbf{x})$, which corresponds to the left image in Figure 2.17,
- or $\mathbf{x}$ is closer to the decision boundary than the border, and in that case, the projection of $\mathbf{x}$ on the decision boundary give the minimal adversarial distortion $d_{\mathrm{D}}(\mathbf{x})$, the case represented on the right image of Figure 2.17.

We see now how interesting it is to be the case corresponding to the point 2 of Theorem 2. The maximization of the linear regions done by [Croce et al., 2019] aims at putting all training points in that situation, increasing the chances of test points to be in that same situation, and proving robustness or vulnerability without having to solve time-consuming MIPs.

Nevertheless, we sometimes cannot escape the formal verification methods because even the provable methods are not sufficient to prove robustness or vulnerability.

2.5 Formal verification of robustness to adversarial examples

This introduction is adapted from [Vechev, 2020] lecture on mathematical certification of neural networks where the following quote from Dijkstra is recalled: "program testing can be used to show the presence of bugs, but never their absence". We can adapt that quote to adversarial attacks by saying that adversarial attacks can be used to show the presence of adversarial examples but never their absence. Adversarial attacks can be identified to "program testing" in the sense that, if they find an adversarial example which can be identified to a "bug", then it is find, but it is not possible to certify the absence of an adversarial example using attacks. To prove the robustness, it is necessary to use mathematical certification. Let us give a general formulation of formal verification [Vechev, 2020]. Given:

- a neural network f,
- a property over the inputs Φ, named precondition. In our case, Φ generally represents a region of the input space, for example given an input $\mathbf{x}$, the intersection between a p-norm ball of radius ϵ around $\mathbf{x}$ and the input space $\mathscr{X}$, $\mathscr{B}_{\epsilon,p}(\mathbf{x})$.
- a property over the outputs Ψ, named postcondition. In our case, Ψ generally represents the right class prediction.
- a property (P), such that for every input $\mathbf{x}$ satisfying the precondition, the output of the neural network $f(\mathbf{x})$ satisfies the postcondition.

$$(\mathrm{P}) : \forall \mathbf{x}' \in \mathscr{X}, \{\mathbf{x}' \vDash \Phi\} \implies \{f(\mathbf{x}') \vDash \Psi\}. \tag{2.19}$$

  In our case, the robustness property that means for every input belonging to a region near a clear example, the output of the neural network is correct.

Formal verification aims at either giving

- either proves the property (P) holds.
- or returns an input $\mathbf{x}$ such that the property (P) does not hold, showing that:

$$\exists \mathbf{x}' \vDash \Phi \wedge \neg\{f(\mathbf{x}') \vDash \Psi\}\,. \tag{2.20}$$

  It means it returns an input satisfying the precondition but does not satisfy the postcondition.

Formal verifications have several characteristics: they can be sound or unsound, complete or incomplete.

Soundness A verification method is sound if when a property is violated, when the verification method terminates, the method always states that the property is violated. Generally, we refer to sound methods as certification methods.

Unsoundness A method is unsound if when a property is violated, the method could potentially terminate stating the property is satisfied. We can see the attacks as an unsound verification method, in the sense that they can terminate without finding adversarial examples even though they are the domain in which they search.

Complete A method is called complete if it is able to prove the property holds when it holds. It is the case of MIP [Tjeng et al., 2018] or [Katz et al., 2017] which are able to prove the robustness of a piecewise linear classifier on some regions. Complete methods are usually time-consuming and do not yet scale to very deep neural networks.

Incomplete A method is incomplete if it is not guaranteed to prove a property when the property effectively holds. It is the case of the verification method used in [Wong and Kolter, 2018] that uses an upper bound on the loss. This method can flag a region potentially containing an adversarial example while there is no adversarial example in that region.

Regarding verification methods, we generally use sound methods. Those sound methods are generally either incomplete and scalable or complete but do not scale up very well. Incomplete methods are usually a relaxation of some complete method allowing to speed up the computation. Let us start by presenting the complete methods, next we present incomplete ones.

2.5.1 Complete Verification methods

The complete and sound methods are presented hereafter. These are the methods that are able to optimally give the minimal adversarial distortion. We will take a look at the MIP formulation in general and the formulation of [Tjeng et al., 2018] in particular, and to ReLUPlex [Katz et al., 2017].

Classic formulations of the robustness verification problem Let $f : [0,1]^d \to \mathbb{R}^d$ represent a neural network's classification function. And let $\mathcal{N}_{\mathbf{x}}$ a convex neighborhood of $\mathbf{x} \in \mathcal{X}$ associated to the class c.

There are two ways to formulate the robustness problem using a Mixed Integer Linear Program. Either we minimize the difference of the outputs for two close inputs, or we look for the closest input modifying the output.

The optimization over the output of the neural networks to prove or deny robustness around the input $\mathbf{x}$ is written:

$$\Delta = \min_{\mathbf{x}' \in \mathcal{N}_{\mathbf{x}}, i \neq c} f_c(\mathbf{x}') - f_i(\mathbf{x}'), \tag{2.21}$$

where $f_i(\mathbf{x}')$ is the output related to i-th class for the input $\mathbf{x}'$ of the neural network. With this formulation, given a convex neighborhood $\mathcal{N}_{\mathbf{x}}$, we know that the neural is robust on $\mathcal{N}_{\mathbf{x}}$ if and only if $\Delta > 0$. $\Delta > 0$ means that for any points $\mathbf{x}' \in \mathcal{N}_{\mathbf{x}}$, we always have $f_c(\mathbf{x}') - f_i(\mathbf{x}') \geq \Delta > 0, \forall i \neq c$ since by definition $\Delta = \min_{\mathbf{x}' \in \mathcal{N}_{\mathbf{x}}, i \neq c} f_c(\mathbf{x}') - f_i(\mathbf{x}')$ is the minimum value that $f_c(\mathbf{x}') - f_i(\mathbf{x}')$ can take. This leads to:

$$\forall\, \mathbf{x}' \in \mathcal{N}_{\mathbf{x}}, \forall\, i \neq c, f_c(\mathbf{x}') > f_i(\mathbf{x}'). \tag{2.22}$$

Meaning any point in $\mathcal{N}_{\mathbf{x}}$ belongs to the class y. If on the contrary $\Delta \leq 0$, it means:

$$\exists\, \mathbf{x}' \in \mathcal{N}_{\mathbf{x}}, \exists\, i \neq c, f_c(\mathbf{x}') \leq f_i(\mathbf{x}'). \tag{2.23}$$

Then the neural network is not robust since there exists a point in $\mathcal{N}_{\mathbf{x}}$ which is not classified to c.

The second way is to minimize the distance between the original example $\mathbf{x}$ and the adversarial examples in $\mathcal{N}_{\mathbf{x}}$. It is formulated as follows:

$$\begin{cases} \min\limits_{\mathbf{x}' \in \mathcal{N}_{\mathbf{x}}} & \|\mathbf{x} - \mathbf{x}'\|_p \\ \text{s.t.} & \max\limits_{i=\{1,\ldots,C\} \setminus c} f_i(\mathbf{x}') > f_c(\mathbf{x}'). \end{cases} \tag{2.24}$$

where p is 1, 2 or ∞.

If the Problem 2.24 is infeasible, then the decision of the classifier does not change in $\mathcal{N}_{\mathbf{x}}$ then we have Equation 2.22. Otherwise, a solution is found and it is enough to prove that the classifier is not robust on $\mathcal{N}_{\mathbf{x}}$.

Problems 2.21 and 2.24 give the same result regarding computing the robust test error. However, we prefer formulation 2.24 because it can give the minimal adversarial distortion in addition to just proving the vulnerability in a region.

MILP formulation of ReLU Apart from the ReLU or other piecewise linear activation such as Maxout [Goodfellow et al., 2013], the other components of neural networks are usually linear expressions and can be directly integrated in solvers. This is not the case of ReLU : ReLU is not a linear expression, but since it is piecewise linear, some tricks can be used to obtain a linearized expression of ReLU. Obtaining a linear expression of ReLU requires the introduction of binary values, one binary value for each ReLU activation, and the use of a majoring constant, the big-M constant. Let $\hat{\mathbf{z}}$ be a pre-ReLU activation on a layer containing e neurons and $\mathbf{z} = \mathrm{R}e\mathrm{LU}(\hat{\mathbf{z}})$, then a linearization can be written as follows [Wolsey and Nemhauser, 1999]:

$$\mathbf{z}_i \geq 0, \qquad i = 1, \ldots, e \tag{2.25}$$

$$\mathbf{z}_i \leq \mathrm{M}_r\, \mathbf{b}_i, \qquad i = 1, \ldots, e \tag{2.26}$$

$$\mathbf{z}_i \leq \hat{\mathbf{z}}_i + \mathrm{M}_r(1 - \mathbf{b}_i), \qquad i = 1, \ldots, e \tag{2.27}$$

$$\mathbf{z}_i \geq \hat{\mathbf{z}}_i, \qquad i = 1, \ldots, e \tag{2.28}$$

$$\mathbf{b}_i \in \{0, 1\}, \qquad i = 1, \ldots, e \tag{2.29}$$

with M_r such that $\mathrm{M}_r \geq \max|\hat{\mathbf{z}}_i|, \forall i = 1, \ldots, e$.

Proof. We can remark that the constraints act in pairs to fix the value of $\mathbf{z}_i$ according to the value of $\mathbf{b}_i$. The pair, when $\mathbf{b}_i = 0$, the pair 2.25 and 2.26 forces $\mathbf{z}_i = 0$ while the pair 2.27 and 2.28 forces $\mathbf{z}_i = \widehat{\mathbf{z}}_i$ when $\mathbf{b}_i = 1$.The case $\widehat{\mathbf{z}}_i = 0$, is a particular case that can be satisfied with the ReLU considered active, $\mathbf{b}_i = 1$ or inactive, $\mathbf{b}_i = 0$.

Let us show that this formulation is such that, for a fixed i, $\{\widehat{\mathbf{z}}_i \leq 0\} \iff \{\mathbf{z}_i = 0\}$, and $\{\widehat{\mathbf{z}}_i \geq 0\} \iff \{\mathbf{z}_i = \widehat{\mathbf{z}}_i\}$:

- inactive case: $\{\widehat{\mathbf{z}}_i \leq 0\} \iff \{\mathbf{z}_i = 0\}$. Let us suppose that $\widehat{\mathbf{z}}_i < 0$ and prove that $\mathbf{z} = 0$.
  - Let us suppose $\mathbf{b}_i = 1$. Then we have $\mathbf{z}_i = \widehat{\mathbf{z}}_i < 0$[9], then the constraints 2.25 and 2.27 are unsatisfiable. Thus, we cannot have $\widehat{\mathbf{z}}_i < 0$ and $\mathbf{b}_i = 1$.
  - The other possibility is $\mathbf{b}_i = 0$. In that case, we have $\mathbf{z} = 0$ and all constraints are satisfied.

    We demonstrated that $\{\widehat{\mathbf{z}}_i < 0\} \implies \{\mathbf{z}_i = 0\}$. Let us now suppose we have $\mathbf{z}_i = 0$ and show that $\widehat{\mathbf{z}}_i \leq 0$. Having $\mathbf{z}_i = 0$, let us suppose $\widehat{\mathbf{z}}_i$ positive, then inequation 2.28 is infeasible, meaning that it is impossible to have $\mathbf{z}_i = 0$ and $\widehat{\mathbf{z}}_i > 0$, so $\{\mathbf{z}_i = 0\} \implies \{\widehat{\mathbf{z}}_i \leq 0\}$.
- the same reasoning is applied to prove $\{\widehat{\mathbf{z}}_i \geq 0\} \iff \{\mathbf{z}_i = \widehat{\mathbf{z}}_i\}$.

□

The formulation of the ReLUs and how they are handled are the main differences between the methods used to prove the robustness of neural networks. Unfortunately, the introduction of binary values makes the robustness verification problem NP-complete. Given a neural network, the total number of combinations is 2 to the power of the number of ReLUs, which is already prohibitive for small size networks. The most efficient robustness verification methods aim at mitigating the exponential number of possible combinations by extracting information from the neural networks.

Reluplex Reluplex is a technique for solving linear programs with ReLU constraints by extending the simplex method to handle ReLUs, hence the name Reluplex [Katz et al., 2017]. Reluplex does not require branching in advance as it would be the case for the branch and bound algorithm, for example. The ReLU constraints are incrementally satisfied with slack variables, basic variables, and nonbasic solutions as in a normal simplex. In practice, branching may be used but only when it is necessary. In their experiment, that was used on around 10% of the total number of ReLUs. Reluplex was then the state-of-the-art verification system. To have an efficient implementation of reluplex, bound tightening should be used to derive bounds on the activations to get stable ReLUs. This will in turn reduce the number of needed branching. Reluplex is based on Satisfiability Modulo Theory (SMT). Reluplex is sound under real arithmetic (unsound under floating point arithmetic [Singh et al., 2019]) and complete and has been used to verify the airborne collision avoidance system (ACAS Xu) [Julian et al., 2016]. However, Reluplex is reported to produce an "error" on some instances in [Dutta et al., 2017], the guess was that it was due to the use of floating point arithmetic. Note that there exist other SMT based verifiers such as Planet [Ehlers, 2017].

MIPVerify Associated with [Tjeng et al., 2018], MIPVerify is the state-of-the-art verifier using MIP. MIPVerify exploits the feedforward structure of neural networks to compute bounds on the pre-ReLU activations of the networks. This information is used to reduce the solve time. Having tight bounds allows to discover that some ReLUs are always active or always inactive, thus allowing to decrease the number of binary values needed for the optimization. The authors of [Tjeng et al., 2018] introduced asymmetric bounds, instead of using the same big-M value, they used distinct

[9] the case $\widehat{\mathbf{z}}_i = 0$ is excluded for simplicity

values representing the lower bounds and the upper bounds. Using the setting as in 2.25-2.29 , the set of constraints linearizing the ReLU becomes :

$$\mathbf{z}_i \geq 0, \qquad i \in \mathscr{I} \tag{2.30}$$

$$\mathbf{z}_i \leq \mathrm{U}_i\, \mathbf{b}_i, \qquad i \in \mathscr{I} \tag{2.31}$$

$$\mathbf{z}_i \leq \hat{\mathbf{z}}_i - \mathrm{L}_i(1-\mathbf{b}_i), \; i \in \mathscr{I} \tag{2.32}$$

$$\mathbf{z}_i \geq \hat{\mathbf{z}}_i, \qquad i \in \mathscr{I} \tag{2.33}$$

$$\mathbf{b}_i \in \{0,1\}, \qquad i \in \mathscr{I}, \tag{2.34}$$

with $\mathscr{I}$, the set of unstable ReLUs, meaning ReLUs such that $\mathrm{L}_i \leq \widehat{\mathbf{z}}_i \leq \mathrm{U}_i$ with $\mathrm{L}_i < 0$ and $\mathrm{U}_i > 0$. If $\mathrm{L}_i \geq 0$ then i-th ReLUs is stably active then $\mathbf{z}_i = 0$; if $\mathrm{U}_i \leq 0$ then the neuron is stably inactive then $\mathbf{z}_i = \widehat{\mathbf{z}}_i$. Note that it is possible to go back to the naive symmetric formulation by replacing U_i and $(-\mathrm{L}_i)$ by $\mathrm{M} = \max(\mathrm{U}_i, -\mathrm{L}_i)$. MIPVerify was able to achieve an improvement of several orders of magnitude with respect to a naive implementation of MIP. MIPVerify is also between two and three orders of magnitude faster than the state-of-the-art Satisfiability Modulo Theory (SMT) based verifier: Reluplex [Katz et al., 2017].

2.5.2 Incomplete Verification Methods

A unified view of LP-relaxed verifiers is presented [Salman et al., 2019] gathering all available LP-relaxation methods used to verify the robustness of neural networks [Zhang et al., 2018, Wong and Kolter, 2018] among others. They compared the methods between them and showed the existence "inherent barrier to tight verification" for the large class of methods captured by their work. We will present the certification method used in [Wong and Kolter, 2018] and DeepPoly [Singh et al., 2019] which is using abstract interpretation.

LP-Relaxed Dual The paper [Wong and Kolter, 2018] that proposed the first provably robust model of with more than 3 layers uses a verification method in order to robustify the model and also prove robustness. They use the convex relaxation of ReLU with the minimum area in the input-output plane as illustrated in the Figure 2.19-(a). This relaxation allows to find a convex outer polytope on the set of reachable output of the neural network. Let us explicit the optimization used to certify robustness of a model, $\mathbf{x}$ being the original example, c its class, $\mathscr{Z}_\varepsilon(\mathbf{x})$ the convex outer polytope of the reachable output for input in $\mathscr{B}_{\varepsilon,p}(\mathbf{x})$, $\hat{f}$ is obtained by relaxation of the ReLU:

$$\begin{cases} \min\limits_{\mathbf{x}' \in \mathscr{B}_{\varepsilon,p}(\mathbf{x})} & \hat{f}_c(\mathbf{x}') - \hat{f}_t(\mathbf{x}') \\ \text{s.t.} & \hat{f}(\mathbf{x}') \in \mathscr{Z}_\varepsilon(\mathbf{x}), \end{cases} \tag{2.35}$$

for a given class $t \neq c$. The optimization problem 2.35 is a Linear Program. Its objective function is such that if at the end of the optimization it is positive, then the neural network is robust on $\mathscr{B}_{\varepsilon,p}(\mathbf{x})$. For efficiency reasons, instead of solving directly the optimization problem with constraints represented by the relaxation, the dual problem is solved. They derived a way to find efficiently a solution for the dual in one backward pass through a modified version of the neural network. And since any solution of the dual solution is a lower bound of the objective function of the Primal 2.35, the optimization is done on the dual to obtain a positive lower bound guaranteeing the robustness on $\mathscr{B}_{\varepsilon,p}(\mathbf{x})$.

DeepPoly In [Singh et al., 2019], a new approach that takes ℓ_p-norm perturbations as well as other more complex transformations such as brightening/darkening or rotation was proposed, under the name of DeepPoly. DeepPoly can also certify neural networks with nonlinear activations like sigmoid or tanh. DeepPoly uses an abstract interpretation replacing each pixel by an interval. This interval contains the value that the pixel can take and is in practice represented by the lower bound and the upper bound of the pixel. For example, if 0.1 ℓ_∞-norm perturbation is allowed and a pixel of value 0.5 will be replaced by the interval [0.4,0.6]. For each image, a lower

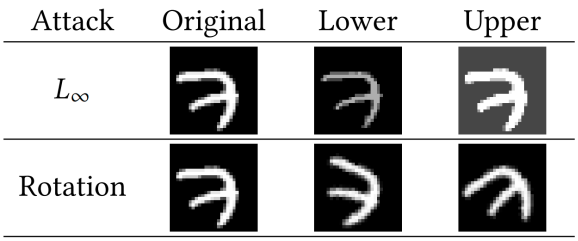


Figure 2.18 – Example of attacks on the MNIST dataset [Singh et al., 2019]

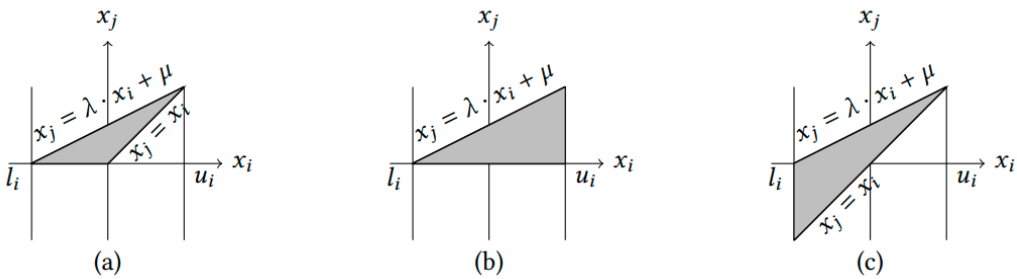


Figure 2.19 – Convex approximations for the ReLU function: (a) shows the convex approximation with the minimum area in the input-output plane in gray, (b) and (c) show the two convex approximations proposed in [Singh et al., 2019], with $\lambda = \frac{u_i}{u_i - l_i}$ and $\mu = \frac{-l_i\, u_i}{u_i - l_i}$

and upper images are produced with respect to the allowed perturbation, and all possible perturbations "lie" between the lower and upper image, see Figure 2.18. For the ℓ_∞-norm perturbation, the lower image is the initial image obtained when all pixels take their minimal value, the upper image is obtained when all pixels take their maximum value. For the rotation, the lower image is the image obtained when the initial image is rotated to achieve the maximum allowed angle clockwise, and the upper image is obtained by rotating the initial image anticlockwise until the maximum allowed rotation is reached. DeepPoly uses two linear relaxation of the ReLU formulation choosing at each step the one giving the most tight bounds, the one that has the lower area, corresponding to the filled region in Figure 2.19. This use of relaxations and the fact that the expressions of the intermediate value (values on the hidden layers) are expressed with respect to the input variables only before computing the bounds, allows to have tighter bounds than previous methods, that generally use the bounds coming from the preceding layer. Although DeepPoly is incomplete for scalability purposes, it can be made complete by iteratively refining the input.

My experience To find adversarial examples, besides our own implementations, we used Cleverhans. To verify the robustness of adversarial examples, we used MIPVerify. It has the advantage of being a complete method and allows to find the minimal adversarial distortion. ERAN is another interesting framework, an incomplete verifier that can be enough to prove robustness in a region. It can also be used recursively to be a complete method. A more detailed review of adversarial example and robustness related existing frameworks will be given in Section 2.7.

Summary and transition More information about the verification methods can be found in [Huang et al., 2020] and in the included references. In [Bunel et al., 2018], a unified view of several verification methods is presented, linking them to the branch and bound algorithms. We have seen that proving the robustness property of a neural network is an NP-complete problem. We have also seen some information used to make it easier to prove robustness using bound tightening and asymmetric bounds. We can now look at some theoretical limits on adversarial robustness.

2.6 Theoretical Limits on Adversarial Robustness

Many works (see, for instance, [Shafahi et al., 2018, Pinot et al., 2019, Bhagoji et al., 2019, Dohmatob, 2019, Yin et al., 2019, Javanmard et al., 2020] and the included references, just to name a few) aim at establishing theoretical results on adversary examples to answer fundamental questions such as:

- are adversary examples inevitable?
- is it possible to simultaneously achieve good performance of clean examples and robustness to adversarial examples?
- does achieving robustness require more data compared to general classification?

To prove that adversarial examples are inevitable, Ali Shafahi and its co-authors [Shafahi et al., 2018] propose a theoretical analysis of adversarial examples and identify bounds on the vulnerability of a classifier to adverse attacks.

Among the discussed examples in this article, the one dealing with the search for adversarial examples in a subset of the unit cube in high dimension, is particularly enlightening. They consider the case where the class of an example is defined by a bounded density function over $[0,1]^d$, the unit hypercube of dimension d and where u denotes its upper bound. They also assume that this class occupies less than half the cube. They prove, in Equation (5), that when sampling a random point $\mathbf{x}$ from this class distribution, with probability at least

$$1 - u\,\frac{\exp^{-2\pi\varepsilon^2}}{2\pi\varepsilon},$$

either this point is misclassified or has an adversarial example $\mathbf{x}'$ such that $\|\mathbf{x}-\mathbf{x}'\|_2 \leq \varepsilon$. From this inequality, they infer that:

- there is no relation between the dimension of the input space and the adversarial susceptibility,
- the quantity that matters is u, the upper bound on the class density.

They emphasize the fact that the more a data set is concentrated, the greater the bound u on its density is and the more the probability of finding an adversarial example is small. Thus, since the MNIST dataset is much more concentrated than CIFAR 10, itself more concentrated than ImageNet, the corresponding upper bound u on their densities is getting smaller and smaller, so that it is easier to find an adversarial example for ImageNet than for CIFAR 10 and easier to find one for CIFAR 10 than for MNIST. They conclude by noticing that *the question whether adversarial examples are inevitable is an ill-posed one. Clearly, any classification problem has a fundamental limit on robustness to adversarial attacks that cannot be escaped by any classifier.*

In [Tsipras et al., 2019], to show the possible antagonism between low standard error and low robust error, they built a classification task and a classifier on it, with a very small standard error and large adversarial robustness error. To this end, they considered a toy binary classification task where data's labels y were sampled from a Rademacher distribution and input feature $\mathbf{x}$ sampled as follows:

$$\mathbf{x}_1 = \begin{cases} y, & \text{with probability } p \\ -y, & \text{with probability } 1-p \end{cases} \quad \text{and} \quad \mathbf{x}_2,\ldots,\mathbf{x}_p \overset{i.i.d.}{\sim} \mathcal{N}(\nu y, 1), \text{ with } y \in \{-1,1\},$$

where ν is a fixed scalar. In this setting, it can be seen that, for ν of order $1/\sqrt{d}$, a simple linear classifier $f(\mathbf{x}) = \mathrm{sgn}(w^\top \mathbf{x})$ with $w = (0, \frac{1}{d}, \ldots, \frac{1}{d})$ attains low standard error (<0.01). The probability of making the right prediction is:

$$\mathrm{P}r[f(\mathbf{x}) = y] = \mathrm{P}r\left[\mathrm{sgn}(w^\top \mathbf{x}) = y\right] = \mathrm{P}r\left[\frac{y}{d}\sum_{i=1}^{d}\mathcal{N}(\nu y, 1) > 0\right] = \mathrm{P}r[\mathcal{N}(\nu, \frac{1}{d}) > 0], \tag{2.36}$$

which is 0.99 when $\nu \geq 3/\sqrt{d}$, meaning a standard error of 0.1. Note that this probability does not depend on the value p, p could be 0.5, it would not change the standard accuracy. However, for

$\varepsilon \geq 2\nu$, its adversarial robust error is close to 1, since the probability of predicting correction under attack is upper bounded by a value close to 0:

$$\min_{\delta \leq 2\nu} \mathrm{P}r[\mathrm{sgn}(\mathbf{x}+\delta) = y] \leq \mathrm{P}r[\mathcal{N}(\nu, \frac{1}{d}) - 2\nu > 0] = \mathrm{P}r[\mathcal{N}(-\nu, \frac{1}{d}) > 0] = 1 - \mathrm{P}r[\mathcal{N}(\nu, \frac{1}{d}) > 0]. \quad (2.37)$$

This illustrates the fact that, to achieve robustness, weakly correlated features should be avoided. They also gave a theorem that shows that on their toy dataset, high standard accuracy and high robust accuracy cannot be simultaneously achieved. As an illustration, let us say $p = 0.75$, then in order to achieve more than 0.75 standard accuracy, a classifier must rely on non-robust features (all features except $\mathbf{x}_1$), and then those features can be slightly modified to cause misclassification as shown above.

Classifier-dependent upper bounds on the adversarial robustness error were provided with strong assumptions on both the data and the classifier, for instance, through the so-called "*Strong No Free Lunch" Theorem*" using powerful tools from geometric probability theory [Dohmatob, 2019], proving that, given those assumptions, adversarial robustness is impossible. In the same paper, [Dohmatob, 2019] also provided universal upper bounds on the adversarial robustness error by generalizing the seminal work of [Bhagoji et al., 2019] who analyzed adversarial attacks as optimal transport.

These limits remain theoretical. The fact that our brain is able, most of the time, to be robust to these attacks is a proof of the existence of some defense mechanism. The nature of the whole theoretical framework on adversarial attacks and robustness has to be revisited in the light of this remark *just like in coding theory where a rethinking of the constraints on a channel leads to the Shannon limit to be improved* [Dohmatob, 2019].

2.7 Tools for adversarial examples and robustness

Adversarial attacks, defenses, and verification methods are active research fields and luckily, there are some frameworks available to allow using them. The most used frameworks for deep learning are Tensorflow and torch [Nguyen et al., 2019]. These frameworks mostly harness the computing power of GPUs along with other technical tricks to allow the training of deep neural networks that would have been impossible to train in the 50s. In this section, we are going to suppose the reader is familiar with those standard deep learning frameworks and focus on the frameworks that are developed on top of them and are related to adversarial attacks, defenses and verification methods. We are going to point out some frameworks that are useful with respect to adversarial examples, mostly in image classification.

Cleverhans Cleverhans[10] is the first library related to adversarial. The goal of this platform is to give a reference implementation of attacks against models. It can make it easier to compare the effectiveness of a defense, for example. Cleverhans can be used with Jax, Pytorch, or TensorFlow 2. One of the strong points of Cleverhans is its community and its contributors, among them some big names in the field.

RobustBench RobustBench[11] provides a benchmark of defenses that respect some criteria to allow a fair comparison between them and eliminate some defenses known to be based on gradient masking. They maintain a leaderboard and have a collection of the most robust models. They mainly take into account the robustness to ℓ_p-norm perturbation since they are the most popular however they plan to extend comparison beyond ℓ_p-norm perturbation.

Adversarial Robustness Toolbox (ART) All popular machine learning frameworks are supported by ART[12]. They also claim to support all data types, audio and video, for example, and machine

[10] `https://github.com/cleverhans-lab/cleverhans`
[11] `https://robustbench.github.io/`
[12] `https://github.com/Trusted-AI/adversarial-robustness-toolbox`

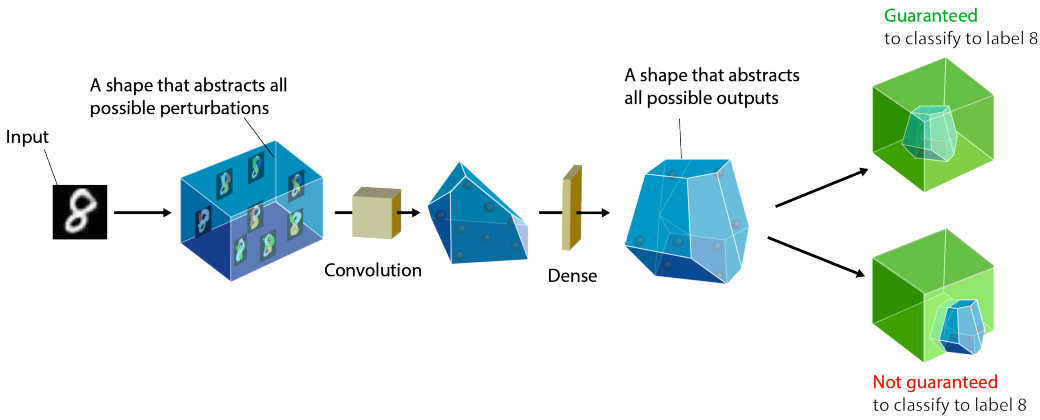


Figure 2.20 – Illustration of robustness verification via abstract interpretation, image taken from ERAN's github.

learning tasks citing object detection, speech recognition. ART is developed with Python. They provide tools to developers and researchers to defend and evaluate their models against several kinds of attacks, among which poisoning.

Marabou Queries about neural network properties, typically robustness, can be answered with Marabou[13] [Katz et al., 2019]. Marabou is based on SMT and transforms the queries into constraints satisfaction problems. Besides the robustness queries of neural networks, it also takes reachability queries, meaning given an input region, can some output be reached.

ERAN A group of researchers at ETH Zurich developed the ETH Robustness Analyzer for Neural Networks (ERAN[14]). ERAN is based on abstract interpretation for complete and incomplete verification of models running on MNIST, CIFAR-10 and ACAS Xu. Figure 2.20 illustrates the verification process performed by ERAN. It supports linear and nonlinear activation functions. It also supports less common geometric transformations and vector field transformations. They approach to the best of our knowledge, the only one that can certify beyond ℓ_p-norm perturbations. They also made available a set of defended and undefended networks.

Foolbox Foolbox[15] is a Python library that allows to run adversarial attacks against deep learning models built on top of EagerPy to work natively with Tensorflow, Pytorch, and Jax. More information about foolbox can be found in [Rauber et al., 2018, Rauber et al., 2020].

MIPVerify The paper [Tjeng et al., 2018] was accompanied with a verification framework coded in Julia, MIPVerify[16]. It verifies, using a MIP formulation, the robustness of neural networks to ℓ_1, ℓ_2 and ℓ_∞ adversarial perturbations. Using MIP means they can verify networks with piecewise linear activations but cannot verify networks with sigmoid or tanh activations. And for a network with piecewise linear activation, they are able to find the minimal adversarial distortion given enough time.

TextAttack *TextAttack*[17] *is a Python framework for adversarial attacks, data augmentation, and model training in Natural Language Processing.* They gather several attacks on text models like

[13] `https://github.com/NeuralNetworkVerification/Marabou`
[14] `https://github.com/eth-sri/eran`
[15] `https://github.com/bethgelab/foolbox`
[16] `https://github.com/vtjeng/MIPVerify.jl`
[17] `https://github.com/QData/TextAttack`

Textbugger [Li et al., 2019]. This attack is able to change the classification of a text from a negative comment to a positive one among other uses.

2.8 Conclusion

In this chapter, we gave an informal definition of adversarial examples. We formalized that definition except for the case of adversarial examples generated from scratch. Remarking that most of the problems we are dealing with can be seen as classification problems and that this thesis is focused on adversarial examples in the field of image classification, we precised this formal definition in the case of classification. We gave several illustrations of adversarial examples in other domains such as speech-to-text and natural language processing. We discussed the conjectures explaining the existence of adversarial examples, among them the linearity hypothesis, evolutionary stalling. We then presented a standard classification problem whose aim is just to classify input points without searching to ensure that the points around those inputs will be classified the same way. Then we went on to talk about adversarial attacks and the ways to create adversarial examples. We introduced different threat models including white-box and black-box attacks, the nature of adversarial perturbations, additive, geometric and functional perturbations, for example. We gave several examples of perturbations in image classification and other fields such as semantic segmentation or object detection. Then we followed up with adversarial defenses that can be based on robust optimization, adversarial example detection, or gradient masking. Arguably, defense mechanisms that do not rely on robust optimization are just subterfuges to make it harder for attacks to find adversarial robust but do not lead to really robust models. Most of those defenses are defeated by adaptive attacks. Once the models are trained to be robust, it is important to have approximation of the robust test error, then using attacks is not enough since attacks cannot guarantee the absence of adversarial examples. There is then a necessity to use sound verification methods, whether complete or incomplete, depending on the goal, the exact robust test error, its lower bound, or a tighter upper bound than the ones given by simple attacks. We then studied some theoretical aspects of adversarial robustness. Theoretically, in some specific settings, adversarial examples are inevitable, and in others it is impossible to achieve simultaneously good standard accuracy and good robust accuracy. However, the fact that human beings are not prone to adversarial examples that fool deep learning models shows it should be possible to design vision systems robust to such perturbations. It also shows neural networks are not mimicking yet the human vision system.

Here are some points to bear in mind for a better understanding of the contributions

1. Adversarial training has a regularizing effect on the loss landscape that it flattens and smoothes. Bearing that in mind could help understand better the Chapter 3.
2. The robustness verification is an open problem. The attacks can miss adversarial examples and give a false sense of robustness, and the formal verification methods are very time-consuming.
3. The MIP formulation of the ReLU networks introduces binary values. Those binary values will be discussed further in Chapter 4 and Chapter 5.

Part II

Contributions

Our contributions: Introduction

After getting interested in adversarial examples, we produced an defense against them in [Seck et al., 2019]. In that paper, we propose a penalization inspired from the double backpropagation. That penalization combined with adversarial training allows to control locally the Lipschitz constant around the training points. This is allowed by directly penalizing the gradients of the logits instead of the gradient of the loss.

After that, we get interested formal verification of the robustness of adversarial with Mixed Integer Programming (MIP). And due to the fact that formal MIPs are usually time-consuming, we made an attempt to use DCA as a way to find a good solution of the MIP more quickly using Difference of Convex Algorithms (DCA). DCA has previously been used to find good solutions for some classical MIP problems. It was sometimes able of finding the optimal solution. Unfortunately, the studies were not fruitful. However, the bright side is that it led us to our main contribution.

It was while studying the reasons why DCA was not as successful as expected, that the idea of Linear Program Powered Attack (LiPPA) emerged. In that contribution, we bridge attacks and verifications through the definition linear region. In this paper we also show that the theoretical number of possible combination is far greater than the actual number of possible combinations due the geometry of the verification problem in the image space. A simpler illustration is given in 2D to give an intuition about that fact.

Chapter 3

L_1-norm double backpropagation adversarial defense

Contents

3.1 Introduction

In the previous chapter, we saw that how important it is to have reliable and robust models, models that can resist adversarial attacks. To obtain such models, one idea is to have a constant output over a region around known training points. For a differentiable function, that means an arbitrarily chosen norm of the output's gradient with respect to the input should be 0 or at least as small as possible over that region. Our claim is that by using adversarial training and by penalizing the gradient's norm of the output with respect to the input, the robustness of the model can be improved. The L_1-norm is chosen and this choice will be theoretically motivated by calculus.

3.2 Related work

Adversarial Training The adversarial examples were popularized by [Szegedy et al., 2014], and the principle of adversarial training was introduced at the same time. Adversarial training consists in augmenting the dataset with potentially adversarial points. However, it was impractical, since the method used to generate adversarial samples, L-BFGS, was too slow. Adversarial training became more convenient to use with the introduction of Fast Gradient Sign Method (FGSM) [Goodfellow et al., 2015] (see Definition 2.1), which is much faster and generates adversarial examples with a good success rate. Moreover, using a first-order expansion of Taylor series, adversarial training can be seen as an ℓ_1-norm penalty of the derivative of the loss with respect to the inputs [Simon-Gabriel et al., 2019].

Adversarial training, regularization and Double Backpropagation Note that *regularization functional which penalizes derivatives of the resulting classifier function are not typically used in deep learning* [Hein and Andriushchenko, 2017]. The idea of penalizing the gradient of the output with respect to the input was introduced by [Drucker and Lecun, 1992] under the name *double backpropagation* and later used in [Hochreiter and Schmidhuber, 1995] to find large connected regions of the error function called flat minima. The ultimate goal was the improvement of the generalization of their models, which differs slightly from our goal here. In *double backpropagation*, the ℓ_2-norm of the loss is penalized. Using the energy loss function, it was stressed that the penalization of the loss would have little to no effect when the classification is good. To balance out that effect, the multiplicative parameter of the penalization ought to be large enough.

Double Backpropagation and adversarial examples The difference between double backpropagation and our gradient penalization is that we penalize the ℓ_1-norm of the gradient of each output with respect to the input while, in backpropagation, the ℓ_2-norm of the loss is penalized. Hence, we should not have the problem that occurs when the penalization term is multiplied by a small error vector. Nevertheless, this might not be a problem when using a different loss function. Although empirical evidence shows double backpropagation's efficiency to enhance generalization, it is insufficient to defend against adversarial examples. That is what [Papernot et al., 2016c] highlights saying *limiting sensitivity to infinitesimal perturbation [e.g., using double backpropagation][Drucker and Lecun, 1992] only provides constraints very near training examples, so it does not solve the adversarial perturbation problem.* However, there are evidence that coupling gradient penalty with adversarial training increases the robustness.

Parseval networks As said in the previous paragraph, most regularizations are applied to the loss function. Parseval Networks [Cisse et al., 2017] introduced a mathematically motivated training method that enables to control the fluctuation of the output with respect to the fluctuation of the input using a Lipschitz constraint. Their method is time consuming.

3.3 Defensive gradient penalty

It has been shown that maliciously crafted examples can fool the deep learning classifiers and lead them to misclassify those examples with high confidence. In addition to the adversarial training, a gradient penalization is proposed here to make the model more robust. Let $\mathbf{x}$ be an input image stored in a vector belonging to the input space $\mathscr{X} = [0,1]^d$, where d is the dimension of $\mathbf{x}$ and let $\mathbf{y}$ be the target, a one-hot label, associated with $\mathbf{x}$. Let f, a differentiable function, $f : \mathscr{X} \to \mathbb{R}^{\mathrm{C}}$ where C is the number of classes, such that the decision function $\mathrm{D}_f(x) = argmax(f(x))$ represents our classifier. And let $\mathscr{L}(\mathbf{y}, f(\mathbf{x}))$ be a common differentiable loss function.

Further intuition about the regularization effect of adversarial training The regularization effect of adversarial was empirically shown (see Figure 2.14), here we try to give another intuitive explanation of this effect. For a given input $\mathbf{x}$, a perturbation direction $\mathbf{v}$ and a positive scalar ε, the first-order approximation of the transfer function i-th component, f_i, of the neural network is:

$$f_i(\mathbf{x}+\varepsilon\mathbf{v}) = f_i(\mathbf{x}) + \varepsilon\mathbf{v}^{\mathrm{T}}\nabla_{\mathbf{x}} f_i(\mathbf{x}) + \circ(\varepsilon\|\mathbf{v}\|). \tag{3.1}$$

The absolute difference between the output of f_i at $\mathbf{x}$ and $\mathbf{x}+\varepsilon\mathbf{v}$ is approximated by:

$$|f_i(\mathbf{x}+\varepsilon\mathbf{v}) - f_i(\mathbf{x})| \approx \varepsilon|\mathbf{v}^{\mathrm{T}}\nabla_{\mathbf{x}} f_i(\mathbf{x})|. \tag{3.2}$$

From Equation 3.2 we can have an intuition about the regularization effect of adversarial training, where we want the model f to have a correct output at $\mathbf{x}$ and at $\mathbf{x}+\varepsilon\mathbf{v}$. We are going to give a more formal relation between the absolute difference between two input points and the gradient.

Theorem 3 (Hölder's inequality)**.** *For any pair (p,q), such that $\frac{1}{p}+\frac{1}{q}=1$, and for any $\mathbf{u},\mathbf{v}\in\mathbb{R}^d$, the following holds:*

$$|\mathbf{u}^\top\mathbf{v}| \le \|\mathbf{u}\|_p\|\mathbf{v}\|_q. \tag{3.3}$$

Formal relation between the input difference and the gradients Combining the right hand side of (3.2) with the Hölder's inequality gives the following:

$$|\mathbf{v}^{\mathrm{T}}\nabla_{\mathbf{x}} f_i(\mathbf{x})| \le \|\mathbf{v}\|_\infty\|\nabla_{\mathbf{x}} f_i(\mathbf{x})\|_1. \tag{3.4}$$

Considering the FGSM attack [Goodfellow et al., 2015], we have $\mathbf{v} = \mathrm{sgn}(\nabla_{\mathbf{x}}\mathscr{L}(\mathbf{y}, f(\mathbf{x})))$, we have $\mathbf{v} = \{\pm 1\}^d$, and $\|\mathbf{v}\|_\infty = 1$. This gives the intuition that finding weights that minimize the sensitivity to infinitesimal perturbations of the input can be done by minimizing the ℓ_1-norm of the gradient of each component of the function f with respect to the input. Our approach is therefore to penalize the ℓ_1-norm of the gradient of each coordinate $f_i(\mathbf{x})$ not only at the original point but also at the potentially generated adversarial points. If we manage to have 0 as the ℓ_1-norm of those gradients at two points that are more or less close, then the output of the classifier is almost constant along the segment joining those two points. Indeed we have:

$$0 \le |f_i(\mathbf{x}+\varepsilon\mathbf{v}) - f_i(\mathbf{x})| \le \varepsilon \sup_{t\in[0,\varepsilon]} \|\nabla_{\mathbf{x}} f_i(\mathbf{x}+t\mathbf{v})\|_1 . \tag{3.5}$$

Proof. From the mean value theorem applied on $g(t) = f_i(\mathbf{x}+t\mathbf{v})$ we have:

$$\exists t_0 \in]0,\varepsilon[: f_i(\mathbf{x}+\varepsilon\mathbf{v}) - f_i(\mathbf{x}) = \varepsilon\mathbf{v}^\top\nabla_{\mathbf{x}} f_i(\mathbf{x}+t_0\mathbf{v}). \tag{3.6}$$

We can then take the absolute values and apply Hölder's inequality which gives

$$|f_i(\mathbf{x}+\varepsilon\mathbf{v}) - f_i(\mathbf{x})| = \varepsilon|\mathbf{v}^\top\nabla_{\mathbf{x}} f_i(\mathbf{x}+t_0\mathbf{v})| \le \varepsilon\|\mathbf{v}\|_\infty\|\nabla_{\mathbf{x}} f_i(\mathbf{x}+t_0\mathbf{v})\|_1. \tag{3.7}$$

We remind that $\|\mathbf{v}\|_\infty = 1$, after that, we can see that:

$$\|\nabla_{\mathbf{x}} f_i(\mathbf{x}+t_0\mathbf{v})\|_1 \le \sup_{t\in]0,\varepsilon[} \|\nabla_{\mathbf{x}} f_i(\mathbf{x}+t\mathbf{v})\|_1, \tag{3.8}$$

and finally

$$0 \leq |f_i(\mathbf{x}+\varepsilon\mathbf{v}) - f_i(\mathbf{x})| \leq \varepsilon \sup_{t\in[0,\varepsilon]} \|\nabla_{\mathbf{x}} f_i(\mathbf{x}+t\mathbf{v})\|_1.$$

□

Comparison with Parseval Networks Like in Parseval Networks [Cisse et al., 2017], the main idea is to control the fluctuations of the outputs with respect to the fluctuations of the inputs. In Parseval Networks, the sensitivity on the whole input domain is controlled via a Lipschitz constant. Instead of controlling the sensitivity on the whole input domain, what we do is to control it only around the training inputs with the regularization that will be introduced in the following lines.

Strengthening the regularization via explicit penalization and adversarial training We give an intuition about the regularization effect of adversarial training as well as the experimental proof. To increase the regularization effect, an explicit penalization of the ℓ_1-norm is added to the loss function. Hence, there is an analogy to [Drucker and Lecun, 1992], which penalizes the ℓ_2-norm of the gradient of the loss, while in this paper the sum of the ℓ_1-norm of the outputs $(f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_C(\mathbf{x}))$ with respect to the input is directly penalized. However, it is very hard to make the derivatives very small around an input while penalizing only at one point, the clean input. It is also ineffective against adversarial examples.

Better results are obtained when we penalize the gradients on the training set and on the adversarial example near the training set. Doing so, each training point is associated with an adversarial example, as in classical adversarial training, and we penalize the gradient at both these points. Let $\mathbf{x}$ be a point of the training set and $\mathbf{x}'$ the adversarial example computed from $\mathbf{x}$ using the FGSM (or any of its variants such as PGD). Let us assume that we train the classifier for long enough so that $\|\nabla_{\mathbf{x}} f_i(\mathbf{x})\|_1 \approx 0$ and $\|\nabla_{\mathbf{x}} f_i(\mathbf{x}')\|_1 \approx 0$ for all $i = 1, \ldots, c$. Our conjecture is that the regularization effect induced by adversarial training reinforced by the penalization maintains the gradients' norm small between $\mathbf{x}$ and $\mathbf{x}'$. Since this process is repeated, the gradients' norm should remain small around the clean inputs. The ideal training progress is presented in figure 3.1. See section 3.5.2 for comparison of the results between classical adversarial training and the new variant introduced in this paper. The loss function used is:

$$\begin{aligned}
\mathscr{L}_{gp}(\mathbf{y}, f(\mathbf{x})) &= \mathscr{L}(\mathbf{y}, f(\mathbf{x})) + \lambda \sum_{i=1}^{C} \|\nabla_{\mathbf{x}} f_i(\mathbf{x})\|_1 \\
&= \mathscr{L}(\mathbf{y}, f(\mathbf{x})) + \lambda \sum_{i=1}^{C} \sum_{j=1}^{d} |\mathrm{J}_{ij}| \\
&= \mathscr{L}(\mathbf{y}, f(\mathbf{x})) + \lambda \|\mathrm{J}\|_{1,1},
\end{aligned} \tag{3.9}$$

where d denotes the dimension of the input image, C, the number of neurons on the output layers of the neural network *i.e.* the number of classes, λ, the penalization parameter, $\mathrm{J}_{ij} = \frac{\partial f_i(\mathbf{x})}{\partial x_j}$, the Jacobian matrix of f and $\|.\|_{1,1}$, the entry wise L_1-norm.

3.4 Why not just penalize the gradient of the loss?

Expression of the gradient of the loss with respect to the gradients of the f_i The cross-entropy loss is generally used to train image classification models. If f represents the pre-softmax output, the cross-entropy loss is written:

$$\mathscr{L}(\mathbf{y}, f(\mathbf{x})) = -\sum_{j=1}^{C} \mathbf{y}_j \log\left(\frac{\exp(f_j(\mathbf{x}))}{\sum_{i=1}^{C} \exp(f_i(\mathbf{x}))}\right). \tag{3.10}$$

The vector $\mathbf{y}$ is generally a one-hot vector of an integer $c \in \{1, \dots, C\}$ (in the case of distillation, it could be different from a one-hot vector), the c-th element is 1 and all others are 0. We can then rewrite the loss function:

$$\mathscr{L}(\mathbf{y}, f(\mathbf{x})) = -\log\left(\frac{\exp(f_c(\mathbf{x}))}{\sum_{i=1}^{C}\exp(f_i(\mathbf{x}))}\right) = -f_c(\mathbf{x}) + \log\left(\sum_{i=1}^{C}\exp(f_i(\mathbf{x}))\right). \tag{3.11}$$

We take the gradient of the loss:

$$\begin{aligned}
\nabla_{\mathbf{x}}\mathscr{L}(\mathbf{y}, f(\mathbf{x})) &= -\nabla_{\mathbf{x}} f_c(\mathbf{x}) + \nabla_{\mathbf{x}}\log\left(\sum_{i=1}^{C}\exp(f_i(\mathbf{x}))\right) \\
&= -\nabla_{\mathbf{x}} f_c(\mathbf{x}) + \frac{\nabla_{\mathbf{x}}\left(\sum_{i=1}^{C}\exp(f_i(\mathbf{x}))\right)}{\sum_{i=1}^{C}\exp(f_i(\mathbf{x}))} \\
&= -\nabla_{\mathbf{x}} f_c(\mathbf{x}) + \frac{\sum_{i=1}^{C}\nabla_{\mathbf{x}}\exp(f_i(\mathbf{x}))}{\sum_{i=1}^{C}\exp(f_i(\mathbf{x}))} \\
&= -\nabla_{\mathbf{x}} f_c(\mathbf{x}) + \frac{\sum_{i=1}^{C}\left(\nabla_{\mathbf{x}} f_i(\mathbf{x})\right)\exp(f_i(\mathbf{x}))}{\sum_{i=1}^{C}\exp(f_i(\mathbf{x}))} \\
&= -\nabla_{\mathbf{x}} f_c(\mathbf{x}) + \sum_{i=1}^{C}\left(\nabla_{\mathbf{x}} f_i(\mathbf{x})\right)\frac{\exp(f_i(\mathbf{x}))}{\sum_{j=1}^{C}\exp(f_j(\mathbf{x}))}.
\end{aligned} \tag{3.12}$$

Let us note h: $h_i(\mathbf{x}) = \frac{\exp(f_i(\mathbf{x}))}{\sum_{j=1}^{C}\exp(f_j(\mathbf{x}))} \in]0,1[$, the softmax applied to $f(\mathbf{x})$. Then the loss is written:

$$\nabla_{\mathbf{x}}\mathscr{L}(\mathbf{y}, f(\mathbf{x})) = (h_c(\mathbf{x}) - 1)\,\nabla_{\mathbf{x}} f_c(\mathbf{x}) + \sum_{i=1; i\neq c}^{C} h_i(\mathbf{x})\nabla_{\mathbf{x}} f_i(\mathbf{x}). \tag{3.13}$$

Relation between penalizing the loss and penalizing the logit As we shown by Equation (3.13), the gradient of the loss is at each point, a linear combination of the gradients of the $f_i, i \in \{1, \dots, C\}$. let us define the function $\alpha(\mathbf{x}) : \alpha_i(\mathbf{x}) = h_i(\mathbf{x}), i \in \{1, \dots, C\}\backslash\{c\}$ and $\alpha_c(\mathbf{x}) = h_c(\mathbf{x}) - 1$, then we have the following condensed rewriting:

$$\nabla_{\mathbf{x}}\mathscr{L}(\mathbf{y}, f(\mathbf{x})) = \sum_{i=1}^{C}\alpha_i(\mathbf{x})\nabla_{\mathbf{x}} f_i(\mathbf{x}). \tag{3.14}$$

We can take the norm of the loss and have a bound on it:
Theoretically, penalizing directly f should be more effective than penalizing the loss

$$\|\nabla_{\mathbf{x}}\mathscr{L}(\mathbf{y}, f(\mathbf{x}))\|_1 = \|\sum_{i=1}^{C}\alpha_i(\mathbf{x})\nabla_{\mathbf{x}} f_i(\mathbf{x})\|_1 \leq \sum_{i=1}^{C}|\alpha_i(\mathbf{x})|\|\nabla_{\mathbf{x}} f_i(\mathbf{x})\|_1 . \tag{3.15}$$

Note that $|\alpha_i(\mathbf{x})| < 1$, we finally have:

$$\|\nabla_{\mathbf{x}}\mathscr{L}(\mathbf{y}, f(\mathbf{x}))\|_1 \leq \sum_{i=1}^{C}\|\nabla_{\mathbf{x}} f_i(\mathbf{x})\|_1 = \|J\|_{1,1} . \tag{3.16}$$

The gradient of the loss and the sign of $\alpha(\mathbf{x})$ From Equation 3.15, we see that, for the cross-entropy loss, the gradient of the loss is a linear combination of the gradients of the logits. Thus, theoretically, the gradient of the loss can have a small norm while the gradients of the logits have large norms. Even more extreme, nothing prevents the the loss to have a null gradient while the gradients of the logits have great values that cancel out during linear combination. We can also remark that $\alpha_c(\mathbf{x}) < 0$ while the others, $\alpha_i(\mathbf{x}), i \neq c$, are positive. This is not counter-intuitive, since to increase the loss function, we should follow directions decreasing the logit of the correct class, and increase the logit of the wrong classes.

The gradient of the loss and the order of magnitude of $\alpha(\mathbf{x})$ At the end of the training, the "confidences" are usually high for the prediction. Meaning the values of $\alpha(\mathbf{x})$ are very small and very close to 0. For the correct class, we are usually near 1, meaning, $\alpha_c(\mathbf{x}) = h_c(\mathbf{x}) - 1$ is nearly 0 (and is negative). We also have

$$\sum_{i=1;i\neq c}^{C} \alpha_i(\mathbf{x}) = \sum_{i=1;i\neq c}^{C} h_i(\mathbf{x}) = 1 - h_c(\mathbf{x}),$$

and $0 < h_i(\mathbf{x}) < 1$, meaning all those for $i \neq c$, $h_i(\mathbf{x})$ is nearly 0. Thus, generally, after training, $\alpha_i(\mathbf{x})$ are almost 0. This can reduce the effect a penalization. In an extreme case where the $\alpha_i(\mathbf{x}) = 0$, we would get a 0 gradient for the loss even if the gradients of the logits are not 0.

Another reason to penalize the gradient of the logits and not the gradient of the loss The gradient of the loss being a linear combination of the gradient of the logits, controlling the gradient of the logits can help to control the gradient of the loss (see Equation 3.16) and have a better regularization. In contrast, controlling the gradient of the loss function does not give any guaranty that the gradients of the logits are small. Furthermore, some defense mechanisms can use some tricks to hide the gradient of the loss as it was the case with defensive distillation [Papernot et al., 2016c]. However, their trick did not seem to affect the logits even though the losses were smoothed, since the Carlini & Wagner attack were able to fool their defense when they used the logits to compute a new loss that they used to craft adversarial examples.

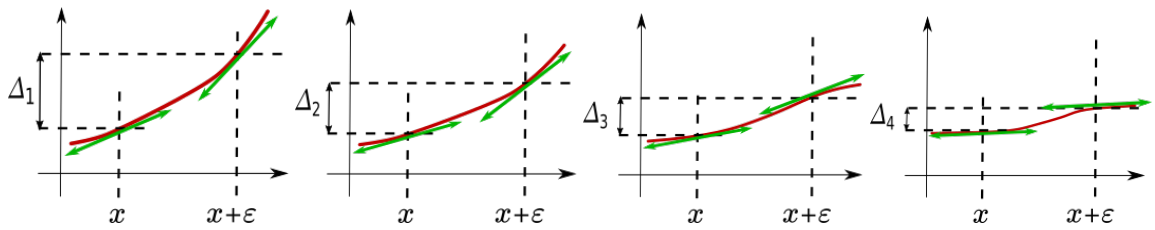


Figure 3.1 – Figure showing the ideal progression of the training for one training point x, and associated adversarial points. At first, the difference $\Delta = f(x+\varepsilon) - f(x)$ of ordinates and gradients norm at x and $x+\varepsilon$ are high. During the course of the training, Δ decreases, making the value at x and $x+\varepsilon$ closer. The gradients' norm also decrease. In the end, we have almost the same value for f at those points, and the gradients are nearly 0. In those conditions, we have $\Delta_1 > \Delta_2 > \Delta_3 > \Delta_4 \approx 0$, and the variation between x and $x+\varepsilon$ is then very small.

	model A	model B	model C
1st layer	conv(64,8,2)	conv(128,3,1)	FC(512)
2nd layer	conv(128,6,2)	conv(64,3,2)	FC(256)
3rd layer	conv(128,5,1)	FC(128)	FC(128)
4th layer	FC(10)	FC(10)	FC(10)
Total params	710,218	1,460,938	567,434

Table 3.1 – Description of the models used in this paper: conv(nf, k, s) represents a convolutional layer with nf filters, of size $k \times k$ applied with a stride s. FC(nn) represents a fully connected layer with nn neurons. All activations are ReLU except the output layer. The numbers in the last row represent the number of parameters.

3.5 Experimental Results

All experiments here are built upon 3 models (referred as A, B, and C for simplicity) that are described in Table 3.1. The first experiment is our proof of concept. It shows that the penalization helps the defense. Since alone it is still not enough to propose a robust model, the second experiment explores the coupling with adversarial training.

3.5.1 Penalization effect

The goal of this experiment is to show that penalizing the gradients improves the accuracy of the predictions against adversarial examples generated using FGSM. The more we penalize, the more the effect is visible. Then we will have to test with more powerful attacks, we will test the performance against FGSM.

Experimental setup We attack the model using the FGSM in a white-box setup in which the attacker has access to the parameters of the model and to the test set. The value $\varepsilon = 0.3$ is chosen as the intensity of the pixel-wise perturbation allowed to the attacker, as it is often the case for the data used in this experiment. Several values of λ are used, allowing to have a notion of the influence of that hyper-parameter on the performance on adversarial samples. The MNIST dataset and three different architectures are used. The training and testing split of Keras is used, and the training set is then split into two groups, one of 55000 is used to train the models, and the remaining 5000 are used as a validation set. Models are trained until their accuracy on the validation set stops increasing (difference of accuracy between 2 epochs less than 0.0001) after 10 epochs, or until 100 epochs of training with clean data is achieved. This process is repeated 10 times, the mean value and the standard deviation are recorded in Table 3.2. The program always terminated before 100 hundred epochs were reached, around the 20th epoch.

λ		50	25	10	1	0.1
A	clean	99.37 ± 0.04	99.36 ± 0.03	99.38 ± 0.04	99.35 ± 0.03	99.35 ± 0.06
	adv	45.09 ± 5.36	44.96 ± 4.44	46 ± 10.16	28.83 ± 9.34	22.64 ± 5.02
B	clean	98.99 ± 0.05	98.99 ± 0.05	98.98 ± 0.07	98.94 ± 0.07	98.90 ± 0.05
	adv	6.35 ± 3.258	4.69 ± 1.91	3.69 ± 0.73	3.00 ± 1.14	3.17 ± 0.82
C	clean	98.63 ± 0.11	98.67 ± 0.05	98.57 ± 0.06	98.55 ± 0.05	98.60 ± 0.08
	adv	29.92 ± 14.24	25.76 ± 11.96	18.24 ± 9.97	19.99 ± 5.63	13.66 ± 6.64

Table 3.2 – Accuracy on clean test and adversarial test point of model. We observe that the more we penalize, the more robust the model becomes. However, while the gain is significant, it's clearly not enough to call it a robust model.

3.5.2 Coupling with adversarial training

ε		0.05		0.1		0.2		0.3	
		clean	adv	clean	adv	clean	adv	clean	adv
A	adv_train	99.36 ±0.05	98.57 ±0.11	99.34 ±0.06	97.50 ±0.26	99.27 ±0.03	96.39 ±0.37	99.30 ±0.06	95.72 ±0.19
	adv_train_gp	99.38 ±0.028	**98.73** ±0.072	99.35 ±0.035	97.45 ±0.30	99.25 ±0.061	96.63 ±0.30	99.26 ±0.06	**97.60** ±0.34
B	adv_train	98.94 ±0.079	98.37 ±0.48	99.07 ±0.03	98.22 ±0.61	98.96 ±0.05	98.55 ±0.26	98.85 ±0.05	98.67 ±0.07
	adv_train_gp	99.05 ±0.05	98.18 ±0.15	99.06 ±0.05	97,64 ±1.04	98.95 ±0.02	98.30 ±0.49	98.84 ±0.07	97.80 ±1.11
C	adv_train	98.98 ±0.06	96.26 ±0.22	98.68 ±0.06	93.56 ±0.47	98.59 ±0.07	93.24 ±0.48	98.48 ±0.1	95.52 ±0.74
	adv_train_gp	98.79 ±0.04	96.36 ±0.11	98.87 ±0.07	**94.56** ±0.16	98.69 ±0.09	93.73 ±0.75	98.42 ±0.08	**96.62** ±0.51

Table 3.3 – Table showing the performance of models A, B and C, trained with adversarial training and adversarial training plus gradient penalty ($\lambda = 10$) We can see that, the adv_gp performs better for model A and $\varepsilon = 0.3$, and is sensibly equal for other values of ε. The large standard deviation values might be an indication that the training was still in progress when 100th epochs was reached.

The previous experiments show that our penalization has an interesting effect on the robustness of the models. Now we explore the coupling with adversarial training, and the hope is that the penalization has the same impact to obtain more robust models than adversarial training alone. We train the same model using adversarial training with the classical loss function and with the gradient penalty in addition to that loss. We train for one hundred epochs. The adversarial training consists in adding adversarial examples to the original training set to make the models more robust. Typically, the FGSM is used to generate those adversarial examples due to its practicality. Therefore, it will be used in the following experiments with clipping to make sure that adversarial examples remain in $\mathscr{X}$. During training, for each image of a batch, an adversarial image is generated and added to the batch with the same target as the original image associated. A label smoothing of 0.1 is also used for this experiment.

Table 3.3 shows results for $\lambda = 10$. It turns out that the performances are not as impressive as they were without adversarial training, even though in some settings, the penalization helps. One reason could be that λ is not high enough. However, for computing time reasons, we could not conduct the same set of experiments with different values at the same level of care. Hence, we propose some partial results to argue our point in the Table 3.4 where we see that increasing λ provides better results until the penalization is too strong. We note also that we increased the number of epochs up to 150.

λ	0	10	50	100	200	1000
clean	98.48 ±0.10	98.42 ±0.08	98.49 ±0.06	98.59 ±0.10	98.60 ±0.11	98.62 ±0.07
adv	95.52 ±0.74	96.62 ±0.51	**97.25** ±0.05	96.88 ±0.51	96.71 ±0.19	94.65 ±0.79

Table 3.4 – For model C, we fixed $\varepsilon = 0.3$ and different λ, on the same setting as previous experiment. $\lambda = 0$ refers to the adversarial learning alone. We observe improvement when λ increases, until it fails when too high.

[0]The loss function used in practice is not exactly the one proposed in (3.9). Instead of $\|J\|_{1,1}$, $\sum_{j=1}^{d} |\sum_{i=1}^{c} \frac{\partial f_i(\mathbf{x})}{\partial x_j}|$ is used due to time constraints.

3.6 Conclusion

In this chapter, we propose to improve deep neural networks' robustness to adversarial examples by adding a penalization term. We show that penalizing the gradients of the output with respect to the input, or in other words, the Jacobian, can have an effect defending against adversarial but does not suffice. Coupled with adversarial training to give L_1-norm double backpropopagation adversarial defense, we provide good hints that it is a good approach. Once we have trained a model using our method and want to measure its robustness in the most precise way. It means using a MIP since it is the state-of-the-art verification method. However, the MIP is time-consuming. For that reason, we take a look at a promising method, DCA, that can potentially achieve the same performance as the MIP but much faster.

Chapter 4

Difference of Convex algorithm

Contents

4.1 Difference of convex functions and difference of convex algorithms

Finding the adversarial example which minimizes the distance with an original example is done using Mixed Integer Problems (MIPs) formulation of the search problem. The MIPs are time-consuming even though there have been some improvements of their efficiency. However, those improvements are still not enough and the runtime of the MIP is still important. In our attempt to find a way to obtain the optimal solution without having to run the time-consuming MIP, Difference of convex and Difference of Convex Algorithms (DCA) seemed to be promising. In fact, DCA has been been successfully used to find good solutions to MIP problems, often finding the optimal solution much faster than done by classical MIP formulation [Niu, 2010]. Having said that, we did not manage to find satisfying results using DCA. We will therefore present a series of experiments highlighting the behavior of DCA, or let us say our implementation of DCA, on our problem. The last experiment leads to a plausible reason explaining why DCA may fail on the robustness verification problem.

MIP formulation The MIP formulation 2.24 will be used to compute, given an original example, the closest adversarial example. For simplicity, we take here the example of a multilayer perceptron with e neurons that is described for an input $\mathbf{x}$ by the following operations:

$$\begin{aligned} \hat{\mathbf{z}} &= \mathrm{W}\mathbf{x} + \beta, \\ \mathbf{z} &= \max(\hat{\mathbf{z}}, 0), \\ f(\mathbf{x}) = \mathbf{s} &= \mathrm{V}\mathbf{z} + \alpha; \end{aligned} \tag{4.1}$$

The operation $\mathbf{z} = \max(\hat{\mathbf{z}}, 0)$ is nonlinear, hence we introduce new binary variables $\mathbf{b}_i$, $i = 1, \dots, e$ as shown by its reformulation 2.25-2.29. Finally, given an original example $\mathbf{x}$ associated with the class y, the obtained MIP optimization problem is:

$$\begin{cases} \min\limits_{\substack{\mathbf{x}' \in [0,1]^d \\ b \in \{0,1\}^e}} & \|\mathbf{x} - \mathbf{x}'\| \\ \text{s.t.} & \hat{\mathbf{z}} = \mathrm{W}\mathbf{x}' + \beta \\ & \mathbf{z}_i \geq 0, & i = 1, \dots, e \\ & \mathbf{z}_i \leq \mathrm{M}_r \mathbf{b}_i, & i = 1, \dots, e \\ & \mathbf{z}_i \leq \hat{\mathbf{z}}_i + \mathrm{M}_r (1 - \mathbf{b}_i), & i = 1, \dots, e \\ & \mathbf{z}_i \geq \hat{\mathbf{z}}_i, & i = 1, \dots, e \\ & \mathbf{s} = \mathrm{V}\mathbf{z} + \alpha \\ & \max\limits_{i = \{1, \dots, c\} \setminus y} \mathbf{s}_i > \mathbf{s}_y \\ & \mathbf{b}_i \in \{0, 1\}, & i = 1, \dots, e \end{cases} \tag{4.2}$$

We simplify the optimization problem by replacing the constraint: $\max\limits_{i = \{1, \dots, c\} \setminus y} \mathbf{s}_i > \mathbf{s}_y$ by $s_t > s_y$, with t our target class. We then obtain:

$$\begin{cases} \min\limits_{\substack{\mathbf{x}' \in [0,1]^d \\ b \in \{0,1\}^e}} & \|\mathbf{x} - \mathbf{x}'\| \\ \text{s.t.} & \hat{\mathbf{z}} = \mathrm{W}\mathbf{x}' + \beta \\ & \mathbf{z}_i \geq 0, & i = 1, \dots, e \\ & \mathbf{z}_i \leq \mathrm{M}_r \mathbf{b}_i, & i = 1, \dots, e \\ & \mathbf{z}_i \leq \hat{\mathbf{z}}_i + \mathrm{M}_r (1 - \mathbf{b}_i), & i = 1, \dots, e \\ & \mathbf{z}_i \geq \hat{\mathbf{z}}_i, & i = 1, \dots, e \\ & \mathbf{s} = \mathrm{V}\mathbf{z} + \alpha \\ & \mathbf{s}_t > \mathbf{s}_y \\ & \mathbf{b}_i \in \{0, 1\}, & i = 1, \dots, e \end{cases} \tag{4.3}$$

This will allow us to focus only on the binaries $\mathbf{b}_1, \dots, \mathbf{b}_e$. Instead of directly solving (4.3), we will try to solve it using DCA.

DCA The main reason that makes the MIP slow is the branching on the binary values. With DCA, we will relax the binary constraints but still manage to have a binary result with a penalty. We are showing a way to get rid of the binary constraints on b_i, $i = 1,\dots,e$ by combining a relaxation of the binary constraint and a penalization of $\mathbf{b}_i(1-\mathbf{b}_i)$, $i = 1,\dots,e$ to push the optimization toward having a binary b without explicitly declaring them as binaries. The problem is then reformulated as follows:

$$\begin{cases} \min & \|\mathbf{x}-\mathbf{x}'\| + \lambda \sum_{i=1}^{e} \mathbf{b}_i(1-\mathbf{b}_i) \\ \text{s.t.} & \hat{\mathbf{z}} = \mathrm{W}\mathbf{x}' + \beta, \\ & \mathbf{z}_i \geq 0, & i = 1,\dots,e \\ & \mathbf{z}_i \leq \mathrm{M}_r \mathbf{b}_i, & i = 1,\dots,e, \\ & \mathbf{z}_i \leq \hat{\mathbf{z}}_i + \mathrm{M}_r(1-\mathbf{b}_i), & i = 1,\dots,e \\ & \mathbf{z}_i \geq \hat{\mathbf{z}}_i, & i = 1,\dots,e \\ & \mathbf{s} = \mathrm{V}\mathbf{z} + \alpha, \\ & \mathbf{s}_k > \mathbf{s}_y \\ & \mathbf{x}' \in [0,1]^d, \\ & \mathbf{b} \in [0,1]^e, \\ & \mathbf{z} \in \mathbb{R}^e, \\ & \hat{\mathbf{z}} \in \mathbb{R}^e, \\ & \mathbf{s} \in \mathbb{R}^{\mathrm{C}}. \end{cases} \tag{4.4}$$

with λ the parameter controlling how hard we penalize the solution for not having a binary values for $\mathbf{b}$.

We can now apply the idea of DCA to the problem by linearizing the terms $-\mathbf{b}_i^2, i = 1,\dots,e$.

That means that, given the initialization $\mathbf{b}^0$, we build a sequence $\{\mathbf{b}^k, k \in \mathbb{N}\}$ (see algorithm 3). At each step of the DCA, the following optimization problem, DCAStep($\mathbf{b}^k, f$) , is solved with a fixed value for $\mathbf{b}^k$:

$$\begin{cases} \min & \|\mathbf{x}-\mathbf{x}'\| + \lambda \sum_{i=1}^{e} (\mathbf{b}_i - 2\mathbf{b}_i^k \mathbf{b}_i) \\ \text{s.t.} & \hat{\mathbf{z}} = \mathrm{W}\mathbf{x}' + \beta, \\ & \mathbf{z}_i \geq 0, & i = 1,\dots,e \\ & \mathbf{z}_i \leq \mathrm{M}_r \mathbf{b}_i, & i = 1,\dots,e, \\ & \mathbf{z}_i \leq \hat{\mathbf{z}}_i + \mathrm{M}_r(1-\mathbf{b}_i), & i = 1,\dots,e \\ & \mathbf{z}_i \geq \hat{\mathbf{z}}_i, & i = 1,\dots,e \\ & \mathbf{s} = \mathrm{V}\mathbf{z} + \alpha, \\ & \mathbf{s}_t > \mathbf{s}_y, \\ & \mathbf{x}' \in [0,1]^d, \\ & \mathbf{b} \in [0,1]^e, \\ & \mathbf{z} \in \mathbb{R}^e, \\ & \hat{\mathbf{z}} \in \mathbb{R}^e, \\ & \mathbf{s} \in \mathbb{R}^{\mathrm{C}}. \end{cases} \tag{4.5}$$

Algorithm 3 : DCA

Data : $\mathbf{x}'$, f, params_{stop}
Result : $\mathbf{x}', \mathbf{b}^k$
$[\mathbf{x}', \mathbf{b}] \leftarrow$ initialization($\mathbf{x}, f$) ;
k=0
$\mathbf{b}^k = \mathbf{b}$
while *checkStopCriteria(*params_{stop}*)* **do**
  $[\mathbf{x}', \mathbf{b}] \leftarrow$ DCAStep($\mathbf{b}^k, f$) (4.5)
  $k \leftarrow k+1$
  $\mathbf{b}^k \leftarrow \mathbf{b}$
end

Comments on DCA for robustness verification Given a neural network, DCA needs an initialization for the value **b**. We will see some initializations in the following section. The used stopping criteria is the stability of the objective function. We stop when the objective function has very little to no change using the criteria $\frac{|obj_{new}-obj_{old}|}{|1+obj_{old}|} \leq 10^{-8}$. We also stop DCA if after 30 iterations it has not converged in the presented results. However, we observed that different stopping criteria do not change the results. The DCA algorithm can return **b** that is not binary. We study the effect of such non binary **b** on the results given by DCA.

What happens when b is not binary? Let us suppose we have $\widehat{\mathbf{z}}_i \leq 0$, then $\mathbf{b}_i$ should be equal to 0. If instead of having $\mathbf{b}_i = 0$, we have $\mathbf{b}_i = 0.01$, then the constraints $0 \leq \mathbf{z}_i \leq \mathbf{b}_i \mathrm{M}_r$ that should constraint $\mathbf{z}_i$ to be equal to 0 only give $0 \leq \mathbf{z}_i \leq 0.01\mathrm{M}_r$. In that case, $\mathbf{z}_i$ can be different from $\max(0, \widehat{\mathbf{z}}_i)$ since nothing prevents it from being $0.01\mathrm{M}_r$. The same reasoning holds when $\widehat{\mathbf{z}}_i \geq 0$ and that instead of having $\mathbf{b}_i = 1$ we have $\mathbf{b}_i = 0.99$. Then the constraints $\widehat{\mathbf{z}}_i \leq \mathbf{z}_i \leq \widehat{\mathbf{z}}_i + \mathrm{M}_r(1-\mathbf{b}_i)$ that should constraint $\mathbf{z}_i = \widehat{\mathbf{z}}_i$ when $\mathbf{b}_i = 1$ only give $\widehat{\mathbf{z}}_i \leq \mathbf{z}_i \leq \widehat{\mathbf{z}}_i + 0.01\mathrm{M}_r$. And nothing prevents $\mathbf{z}_i$ from being equal to $\widehat{\mathbf{z}}_i + 0.01\mathrm{M}_r$. Simply said, when **b** is not binary, instead of having $\mathbf{z}_i = \max(0, \widehat{\mathbf{z}}_i)$ as done by the ReLU activation, we get $\mathbf{z}_i \geq \max(0, \widehat{\mathbf{z}}_i)$ with a gap $\mathbf{z}_i - \max(0, \widehat{\mathbf{z}}_i)$ less or equal to either $\mathbf{b}_i \mathrm{M}_r$ or $(1-\mathbf{b}_i)\mathrm{M}_r$ according to the case. That, combined to the sensitivity of the neural networks, leads to the fact that results given by DCA do not match the true forward pass of the neural networks when after resolution **b** is not binary. Hence, the proposed examples may or may not be real adversarial examples.

4.2 DCA struggling with our problem

DCA was a promizing way to find near optimal adversarial examples, however it quickly hits its limit as it is shown by the following experiment.

Experimental setup: We are going to show the results of 4 experiments based on the architecture of a Multilayer Perceptron (MLP) with one hidden layer with 300 neurons, trained on MNIST. To run the optimization, we have to give an initialization $\mathbf{b}^0$ to DCA, and also give a value to the parameter λ.

4.2.1 Experiment 1: the impact of λ, with fixed initialization

The goal of this experiment is to study the impact of the value of λ on the output of DCA. We fix a value of λ and run DCA until the stopping criteria is satisfied. We initialize $\mathbf{b}^0$ using the activations of the original sample **x** which is, we do a forward pass of **x** to our network and retrieve the values of the activations i.e. for $i = 1, \ldots, e$:

$$\widetilde{\mathbf{b}}_i^0 = \begin{cases} 1 & \text{if} \quad \hat{\mathbf{z}}_i \geq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{4.6}$$

We will pay attention to the respect of the ReLU constraints which are met when $\mathbf{b}^k$ returned by DCA is either 0 or 1. Otherwise there is a gap between the value $\mathbf{z}$ found by DCA and the value $\max(0,\widehat{\mathbf{z}})$. As a consequence, when we pass the $\mathbf{x}'$ given by DCA to the neural network, the output layer of the neural network and the value found for $\mathbf{s}$ do not match. It means that the result of the optimization is incorrect since for the same input, the output found by the optimization is different from the actual output corresponding to the input. The optimization is very sensitive to the value of $\mathbf{b}$.

For a given example, we plot the difference between $\mathbf{z}_i$ and $\max(0,\widehat{\mathbf{z}}_i)$ and also the value $\mathbf{b}_i(1-\mathbf{b}_i)$ for $i = 1,\ldots,300$ after convergence of DCA for different values of λ (see Figure 4.1). In that figure, we see that when the value of λ is small, there is a gap between the values $\mathbf{z}_i$ and $\max(0,\widehat{\mathbf{z}}_i)$ making the output of DCA irrelevant. For $\lambda = 10^{-1}$ and $\lambda = 1$, we see that for some neurons, the associated $\mathbf{b}_i$ is such that $\mathbf{b}_i(1-\mathbf{b}_i)$ is almost 0.25 which is the maximum achievable by $\mathbf{b}_i(1-\mathbf{b}_i)$ when $\mathbf{b}_i$ is between 0 and 1. And it means that $\mathbf{b}_i$ is near 0.5. For those neurons, we also see a considerable gap between $\mathbf{z}_i$ and $\max(0,\widehat{\mathbf{z}}_i)$, making the result of DCA irrelevant. When the value of λ increases, this behavior is more and more penalized so for λ ranging from 10 to 10^6 the values of $\mathbf{b}_i(1-\mathbf{b}_i)$ get smaller and smaller, meaning the $\mathbf{b}_i$ tends to be either 0 or 1. However, there were still a gap between the output of the final layer of the network given by DCA when compared to the forward pass of the neural network. It is only for $\lambda = 10^7$ and greater values that we get the results of DCA become valid, i.e., they match the output of the forward pass of the neural network.

From Figure 4.1, we conclude that, when λ is not big enough the constraints of the ReLU, $\mathbf{z}_i = \max(0,\hat{\mathbf{z}}_i), i = 1,\ldots,e$ are not met, leading to an invalid result. And when the value λ is big enough, we have a binary value, however that value obtained is the same used to initialize the DCA.

4.2.2 Experiment 2: the impact of λ, initialization based on $\hat{\mathbf{z}}_i$

In the conclusion of the first experiment, we notice that for a large enough value of λ, DCA gives a correct output without changing the initialization value of $\mathbf{b}_i$. In this experiment, we will use a different initialization to have further knowledge of the behavior of DCA regarding the values of $\mathbf{b}_i$. We define the rounding operation, the operation that, given an float x, returns the integer that is closest to that integer noted $\lfloor x \rceil$. We use a slightly different approach. Instead of initializing $\mathbf{b}^0$, we use the following initialization:

$$\mathbf{b}_i^0 = \begin{cases} \text{random number between 0.5 and 1(excluded), if } \hat{\mathbf{z}}_i \geq 0, \\ \text{random number between 0 and 0.5(excluded), otherwise.} \end{cases} \tag{4.7}$$

Doing so, we can generate several random numbers which are rounded to $\widetilde{\mathbf{b}}^0$. We see that when λ is not great enough, we have invalid results as in experiment 1 (§4.2.1). For big enough λ, we observe that, at the end of the optimization the values obtained by $\mathbf{b}$ are equal to the round value of the initial $\mathbf{b}^0$, which are equal to $\widetilde{\mathbf{b}}^0$.

Given an initialization $\mathbf{b}^0$, we are tempted to conclude that whenever the solution is valid, DCA is leaning toward the round value of $\mathbf{b} = \lfloor \mathbf{b}^0 \rceil$, we refer to experiment 3 (§4.2.3) to validate this hypothesis. Figure 4.2 shows an example of an adversarial example found using DCA.

4.2.3 Experiment 3: the impact of λ, with random initialization

In this experiment, we are going to initialize $\mathbf{b}^0$ with a completely random value, and then run DCA. Here again, when λ is not big enough we have small value for $\|\mathbf{x}-\mathbf{x}'\|_2^2$ and big value for $\sum_{i=1}^{e} \mathbf{b}_i(1-\mathbf{b}_i)$, meaning $\mathbf{b}$ is far from being binary.We see also that for big enough λ, the final value for $\mathbf{b}$ is (close to) $\lfloor \mathbf{b}^0 \rceil$, and the found adversarial examples are far from the original example, that is not surprising since the values were chosen randomly (see Figure 4.3). Building on the results of experiment 1 (§4.2.1), experiment 2 (§4.2.2), and experiment 3 (§4.2.3), we can conclude that to get a valid result, it is necessary to have a large value for λ, and that in that case, DCA is driven to a

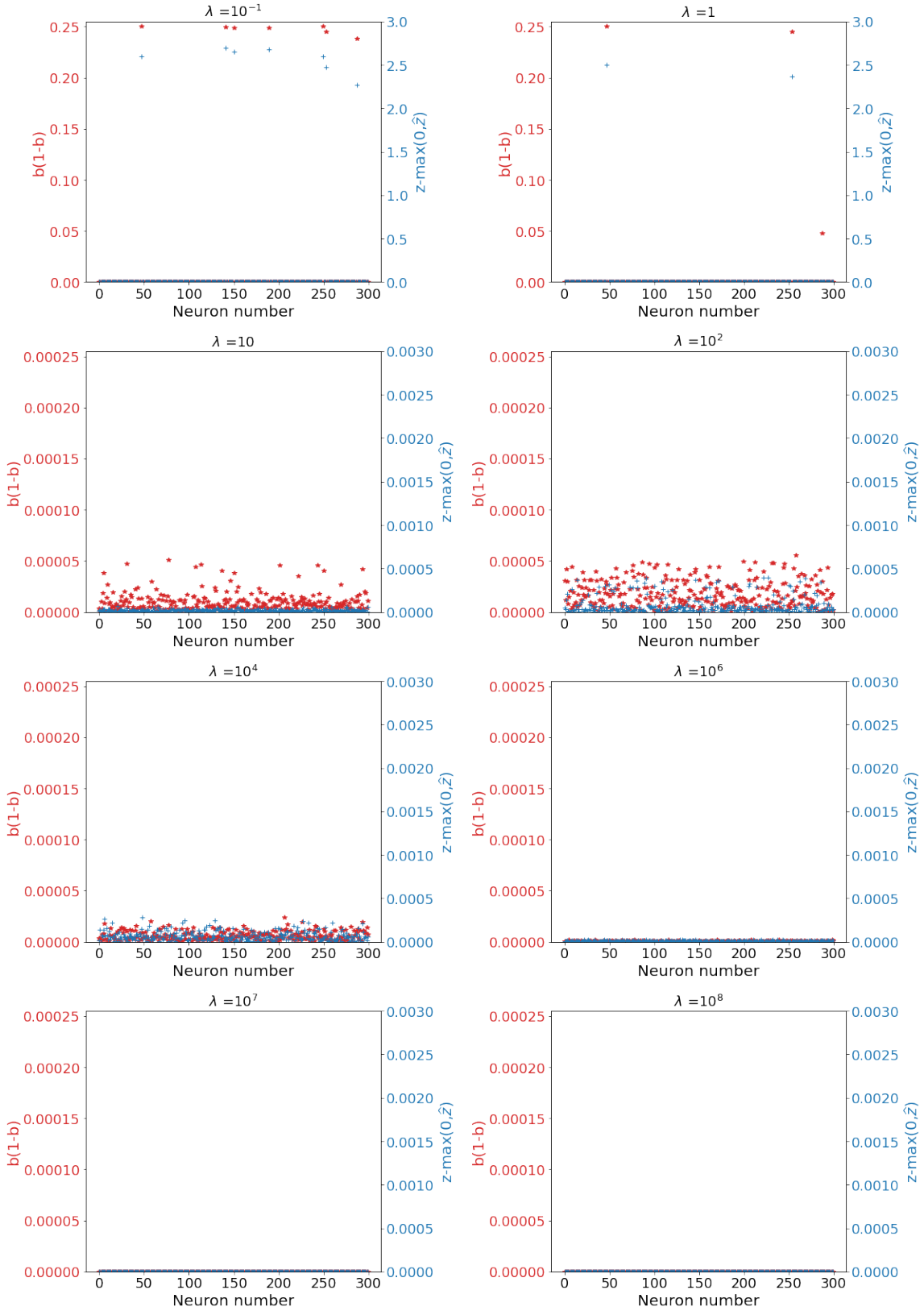


Figure 4.1 – Each subplot represents the product $\mathbf{b}_i(1-\mathbf{b}_i)$ for each neuron in red, and the difference between $\mathbf{z}_i - \max(0,\hat{\mathbf{z}}_i)$ in blue. Getting 0 for all those values means when the value $\mathbf{x}'$ found using DCA is passed through our neural network, we should get the same output final, i.e. $s = f(\mathbf{x}')$, and the intermediate are the same. Otherwise, if there are some values for which there is a difference, $s \neq f(\mathbf{x}')$ and $\mathbf{z}_i \neq \max(0,\hat{\mathbf{z}}_i)$, which means the result of the optimization is invalid. Starting from top left to down right, the first two figures show that for $\lambda = 0.1$ and $\lambda = 1$ we see that when the correlation between $\mathbf{b}_i(1-\mathbf{b}_i)$ and $\mathbf{z}_i - \max(0,\hat{\mathbf{z}}_i)$, we conclude that those results of that optimization is invalid. For the following values of λ the scale is changed for visualization purpose. On this illustration, we do not get $\mathbf{b}_i(1-\mathbf{b}_i) = 0$ and $\mathbf{z}_i - \max(0,\hat{\mathbf{z}}_i) = 0$ (or very close to 0) until $\lambda > 10^6$.

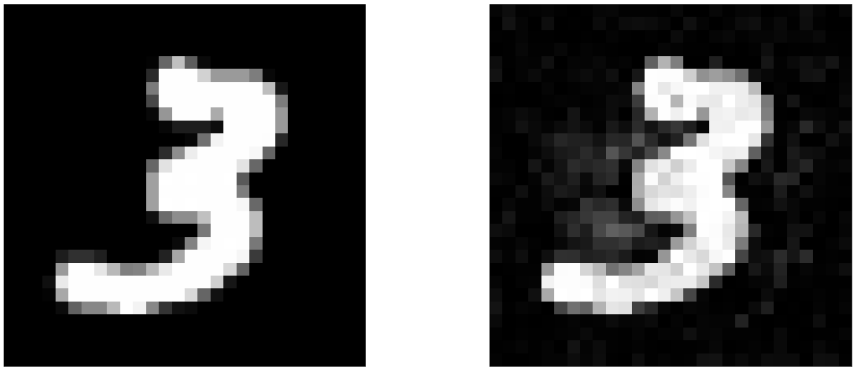


Figure 4.2 – Original and adversarial examples found using DCA with $\mathbf{b}^0$ such that $\lfloor \mathbf{b}^0 \rceil = \widetilde{\mathbf{b}}^0$. The distance l_2 between these examples is 3.56, where the MIP solution associated to the binary is 3.01.

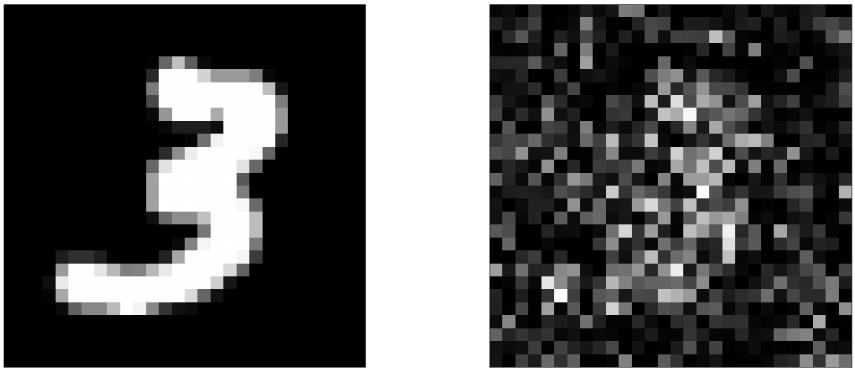


Figure 4.3 – Original and adversarial examples found using DCA with a completely random initialization. The distance l_2 between these examples is 79.99 compared to 3.56 in the previous.

solution for which the value of $\mathbf{b}$ is close to $\lfloor \mathbf{b}_{init} \rceil$, with $\mathbf{b}_{init}$ the value used to initialize $\mathbf{b}$ in DCA. We see this inability of DCA to thoroughly explore the space as a limit with respect to our problem.

4.2.4 Experiment 4: the impact of the distance constraint

In this experiment, we use a big value for λ that is fixed and we initialize with the binary as in experiment 1, but we add a constraint distance between the original sample and the adversarial example we are looking for. We used a fixed value for λ since we are now convinced that small values do not give valid results for our neural networks. We compute d_0 using the distance to the closest adversarial example without changing the activation and once we get it, we run DCA while imposing a new constraint which is, $\|\mathbf{x}-\mathbf{x}'\|_2^2 < d_0/4$ or $d_0/2$, and other values (cf Figure 4.4). The goal of this experiment was to see if DCA would be able to output a value of $\mathbf{b}$ different from $\widetilde{\mathbf{b}}^0$.

In Figure 4.4, we present the values $\mathbf{b}_i(1-\mathbf{b}_i)$ and $\mathbf{z}_i - \max(0, \widehat{\mathbf{z}}_i)$ for each neuron $i = 1, \ldots, 300$, of the hidden layer after convergence of DCA run with the additional constraint on the distance introduced previously. It presents six figures with the constraints $\|\mathbf{x}-\mathbf{x}'\|_2^2 < d$, with $d = d_0/4, d_0/2, 3d_0/4, 4d_0/5, d_0$ and $2d_0$. We observe that for $d_0/4$ and $d_0/2$ we have some neurons for which $\mathbf{b}_i(1-\mathbf{b}_i)$ is almost 0.25 even though the value of λ is big enough. It shows that DCA is unable to shift the point toward a value $\mathbf{b}$ different from $\lfloor \mathbf{b}_{init} \rceil$. For the other value of d, they are still not binary, but they are close to being binary, but still not close enough to give valid results, i.e., results matching the

output of the neural network. We observe that, even when a restriction is applied, the optimization does not jump from a binary to another.We also found that the quadratic constraint added has the effect of changing the distance found by DCA even when it should not be constraining.

4.2.5 Experiment 5: changing the objective formulation

We use the same setting as in experiment 1 (§4.2.1) with the only difference that the objective function is replaced by $\mu\|\mathbf{x}-\mathbf{x}'\| + \sum_{i=1}^{e}(\mathbf{b}_i - 2\mathbf{b}_i^k\mathbf{b}_i)$. We observe that using this formulation, we do not get binary values when μ is small, it does not get any better when we increase it (cf Figure 4.5). From that figure, we see that when μ is small, after convergence, the distance $\|\mathbf{x}-\mathbf{x}'\|_2^2$ is large and the binary values are unchanged. When μ get larger, we can observe a little decrease of $\|\mathbf{x}-\mathbf{x}'\|_2^2$ while the initial binary values remain unchanged. Then for larger values μ we see that $\|\mathbf{x}-\mathbf{x}'\|_2^2$ drops to 0 and the sum $\sum \mathbf{b}_i(1-\mathbf{b}_i)$ increases. This behaviour shows an antagonism between the minimization of $\|\mathbf{x}-\mathbf{x}'\|_2^2$ and the $\sum \mathbf{b}_i(1-\mathbf{b}_i)$ responsible for the respect of the ReLU constraints in the DCA objective.

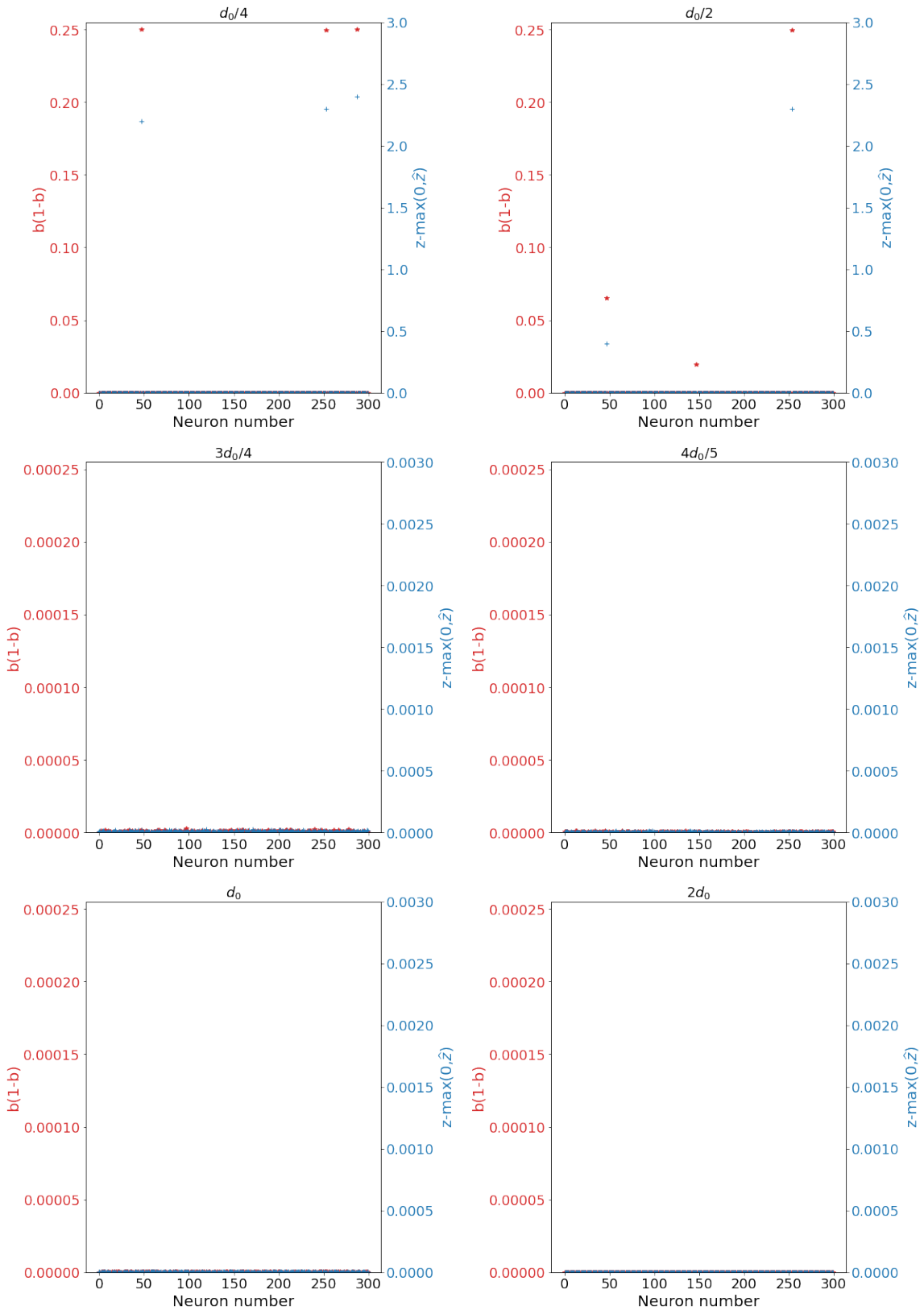


Figure 4.4 – DCA can not give a binary value for **b** when the restriction on the distance between the original sample and the example given by DCA is too hard. However, even when the restriction is less hard, for example, when we impose $\|\mathbf{x}-\mathbf{x}'\|_2^2 < 3d_0/4$ or $\|\mathbf{x}-\mathbf{x}'\|_2^2 < 4d_0/5$, the optimization look no further than $\mathbf{b}_0$ and plays on the penalization to satisfy the constraint.

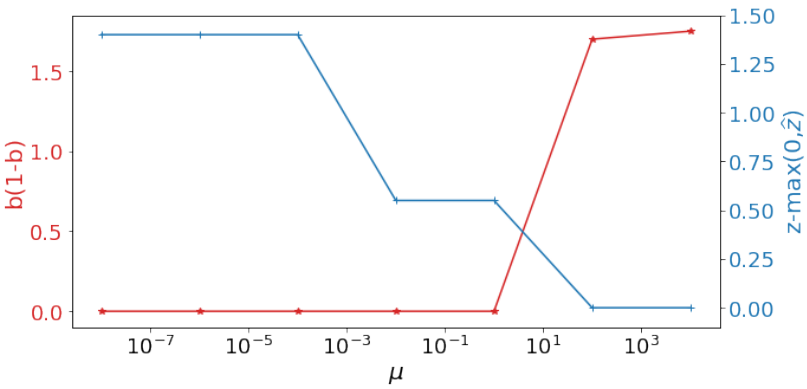


Figure 4.5 – Illustration of the antagonism between $\sum \mathbf{b}_i(1-\mathbf{b}_i)$ and $\|\mathbf{x}-\mathbf{x}'\|_2^2$. The used values for μ are the following: $1e-8, 1e-6, 1e-4, 1e-2, 1, 1e2, 1e4$. We observe that the harder we penalize $\|\mathbf{x}-\mathbf{x}'\|_2^2$ the closer it comes to 0 and the bigger $\sum \mathbf{b}_i(1-\mathbf{b}_i)$ is.

4.3 Conclusion

From these experiments, we conclude that to get a valid solution for our optimization problem with DCA, it is necessary to set a big enough value for λ. The initialization of $\mathbf{b}^0$ is very important since we noticed that DCA tends to go to the solution with $\mathbf{b}_{dca} = \lfloor \mathbf{b}^0 \rceil$ in all our experiments, where the solution found by the MIP differs from $\mathbf{b}^0$ on 11 terms. In our experiments, DCA converges towards a local minima without considering any change to the binary.

We can try to give an explanation to such a phenomenon. First, the need of a big value for λ can be explained by the sensitivity of neural networks to little changes. Due to that sensitivity, when λ is not big enough, the optimization can allow small differences between $\mathbf{z}$ and $\max(0, \hat{\mathbf{z}})$ in order to have little value of $\|\mathbf{x}-\mathbf{x}'\|_2^2$ to achieve optimal value. The $\mathbf{x}$ found then will not satisfy the constraints when passed through the neural network. That can be seen as an antagonism between $\|\mathbf{x}-\mathbf{x}'\|_2^2$ and $\sum_{i=1}^{e} \mathbf{b}_i(1-\mathbf{b}_i)$ since moving the $\mathbf{b}_i, i = 1, \ldots, e$ away from the binary allows to decrease $\|\mathbf{x}-\mathbf{x}'\|_2^2$. And when λ is big enough, it seems costly to try to deviate from $\lfloor \mathbf{b}^0 \rceil$.

A secondary reason why DCA's performance is not as great as in other problems seems to be much more complex that the problems used in the reference we had, in terms of the number of binary constraints and global number of variables and constraints, even though we used a small neural network. DCA manages to find a compromise between the two antagonistic parts of the objective which is good with respect to the optimization. Nevertheless, this compromise is undesirable since it jeopardizes the ReLU constraint making the solution given by DCA inadmissible with respect to the neural network.

Is there a way to satisfy the ReLU constraints, i.e., respecting the binary constraints, without going back to the MIP and not having a big value for λ ? Noting that once an initialization $\mathbf{b}_{init}$ is proposed, DCA tends to return $\lfloor \mathbf{b}_{init} \rceil$, one idea is to directly solve the linear program obtained by fixing the $\mathbf{b}$ to $\lfloor \mathbf{b}_{init} \rceil$ and solve the resulting linear program. Furthermore, combined with a good procedure to propose binary values $\mathbf{b}$ for which the resolution has a high chance of leading to a satisfying adversarial example, one can find the optimal adversarial example much faster than the MIP.

Chapter 5

Linear Program Powered Attack

Contents

5.1 Introduction

Neural networks are now the go-to when it comes to several tasks; image classification is one of them. However, 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 [Goodfellow et al., 2015]. The process of designing specific noise to fool the neural network is called attacking neural networks. This kind of behavior of neural networks has a deterrent effect, making the use of neural networks 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 [Papernot et al., 2016c], which proved later to be inefficient when attacks get stronger, to more recent defenses. Several defense methods have been developed, often broken some time later using more sophisticated attacks or simply giving more attempts to existing methods. Complete verification methods such as Reluplex [Katz et al., 2017] and MIP (Mixed Integer Programming) formulation of the search of adversarial examples [Tjeng et al., 2018] seem to be the ultimate approach able to give the exact robust test error [Bastani et al., 2016]. However, these complete verifiers can take up to an hour 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 the lower bound and the upper bound of the robust test error, more powerful attacks are needed to find adversarial examples in a more reasonable time when they exist in the region in which they are searched, and certification methods that are faster. Projected Gradient Descent [Madry et al., 2019], Brendel & Bethge attack [Brendel et al., 2019], Carlini & Wagner attack [Carlini and Wagner, 2017b] and the randomized gradient-free attack [Croce and Hein, 2018, Croce et al., 2020] are among the strongest attacks to date but even these attacks sometimes miss adversarial examples. The fastest implementation of MIP for adversarial robustness verification is [Tjeng et al., 2018] but it remains time consuming.We propose a new attack aiming at reducing the gap between the lower bound of the robust test error and its upper bound. We also propose a way to partition the input space and to link that partition to the MIP formulation to make the MIP faster.

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 original examples is smaller than the exponential number of combinations allowed by the MIP formulation.
- We propose a new white-box attack that exploits, given a model, the relationship between linear regions and binary values used in the MIP formulation. In a few seconds, our method is able to find an adversarial example that the MIP failed to detect given a much longer time.

5.2 Background and related work

Having a complete access to a trained model (white-box setting) makes it easier to attack that model than if the access was denied. Indeed, the weights of the networks contain information that can be exploited to craft subtle misleading examples. This can be done using either gradient-based methods or gradient-free methods.

We present the related works through the lens of information exploitation. We will thus see the gradient-based attacks, gradient-free verification methods, and some gradient-free attacks. After that, we show in 5.3 that linear regions are equivalence classes and the relationship that exists between those linear regions and the binary values used in the MIP formulation.

Gradient-based attacks On the one hand, gradient-based attacks such Fast Gradient Sign Method [Goodfellow et al., 2015] or Projected Gradient Descent [Madry et al., 2019] (see Definition 2.1) use the weight information to compute gradients that are exploited to construct adversarial examples. This has proven to be effective on neural networks which have no defense mechanism and other with some defense mechanisms as well [Athalye et al., 2018a]. 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. Nevertheless, 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* [Athalye et al., 2018a].

It has also been shown that when the minimal adversarial distortion is high, PGD fails to find adversarial example, this behavior is even more patent when the allowed distortion is close the minimal adversarial distortion [Tjeng et al., 2018].

Gradient-free verification methods On the other hand, the search for adversarial examples can be done using formal verification methods [Bunel et al., 2018] that do not use the gradient. In particular, when the neural networks can be represented as a piecewise linear function, Satisfiability Modulo Theory (SMT) solvers can be used as in [Katz et al., 2017] (discussed in paragraph Reluplex in Chapter 2) as well as Mixed Integer Programming [Tjeng et al., 2018] (discussed in paragraph MIPVerify in Chapter 2). This is the case, for instance, when the ReLU activation is used. Let us focus on the MIP formulation. Assuming that f represents a piece-wise linear function (that is the case for neural networks with ReLU activation) and $\mathbf{x}$ is an example of the input space $\mathscr{X} = [0,1]^d$, belonging to the class y. A generic formulation of the search for adversarial examples is the following [Tjeng et al., 2018]:

$$\begin{cases} \min\limits_{\mathbf{x}'} & \|\mathbf{x}-\mathbf{x}'\|_p \\ \text{s.t.} & \operatorname*{argmax}\limits_{i=1,\dots,c} f_i(\mathbf{x}') \neq y, \\ & \mathbf{x}' \in [0,1]^d\,, \end{cases} \tag{5.1}$$

where p is usually 2 or ∞.

To solve this optimization problem, the constraint $\operatorname*{argmax}\limits_{i=1,\dots,c} f_i(\mathbf{x}') \neq y$ is linearized and for each neuron a binary value is introduced, indicating if the neuron is active or not, as we will see in 5.3. We generally want to check the existence of adversarial examples in a neighborhood of clean examples, the original example for which we look adversarial examples. Computing bounds on the pre-ReLU activation reveals useful information. Computing the lower and upper bounds on the pre-ReLU activation can reveal that some neurons are always active and some others are always inactive (for whatever valid distortions are applied to the example). Those are called stable ReLUs [Tjeng et al., 2018] 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, 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 drawback is that it is time consuming, even when we simplify the problem to computing the robust test error [Bastani et al., 2016]. 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 gives also a lower bound, that if given enough time will converge to the same value, the minimal adversarial distortion.

It is possible to derive attacks from the MIP formulation, by dropping the part dedicated to computing lower bounds since any adversarial example can be used to compute an upper bound. It is the case for [Croce and Hein, 2018, Croce et al., 2020].

Gradient-free attacks The gradient-free attacks of [Croce and Hein, 2018, Croce et al., 2020] can be seen as derived from the MIP formulation. The input space is explored to find linear regions

and optimization over the found linear regions is done to find the optimal adversarial example in those linear regions. Even though the authors did not mention the link between their method and MIP verification, there exists a link that we explicit in Section 5.3.

There are also some gradient-free attacks that are not linked to the verification method such as [Alzantot et al., 2019] where a genetic algorithm is used to find adversarial examples. They create a population, find the best candidates to fool the network according to a fitness score. Then they create "children" by doing a crossover while making sure to have a higher probability to select parents among the fittest. Then the children undergo mutation. The mutated children form a new generation. Crossovers and mutations are repeated for a predefined number of generations.

5.3 Partition of the input space

In this section, we show that the linear regions used in [Croce and Hein, 2018, Croce et al., 2020] form equivalence classes in the input space. We also show how they are related to the binary values of the MIP formulation.

5.3.1 Linear Regions and equivalence classes

For simplicity purpose, let us consider a multilayer perceptron with one hidden layer with e neurons. The classification function of this MLP can be written:

$$\begin{aligned} \hat{\mathbf{z}} &= \mathrm{W}\mathbf{x} + \beta, \\ \mathbf{z} &= \max(\hat{\mathbf{z}}, 0), \\ f(\mathbf{x}) = \quad \mathbf{s} &= \mathrm{V}\mathbf{z} + \alpha. \end{aligned} \tag{5.2}$$

Let us remind the linearization of the ReLU constraint, $\mathbf{z} = \max(\hat{\mathbf{z}}, 0)$ using a big-M [Wolsey and Nemhauser, 1999] as it was already presented in Chapter 2 with equations (2.25-2.29):

$$\mathbf{z}_i \geq 0, i = 1, \dots, e \tag{5.3}$$

$$\mathbf{z}_i \leq \mathrm{M}_r\, \mathbf{b}_i, i = 1, \dots, e \tag{5.4}$$

$$\mathbf{z}_i \leq \hat{\mathbf{z}}_i + \mathrm{M}_r(1 - \mathbf{b}_i), i = 1, \dots, e \tag{5.5}$$

$$\mathbf{z}_i \geq \hat{\mathbf{z}}_i, i = 1, \dots, e \tag{5.6}$$

with M_r such that $\mathrm{M}_r \geq \max_{\mathbf{x}} |\hat{\mathbf{z}}_i|, \forall i = 1, \dots, e, \forall \mathbf{x} \in \mathscr{X}$.

Those constraints are such that:

- $\hat{\mathbf{z}}_i < 0 \Rightarrow \mathbf{b}_i = 0$,
- $\hat{\mathbf{z}}_i > 0 \Rightarrow \mathbf{b}_i = 1$.

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. However, this is not of crucial importance, since in practice, having pre-ReLU with a value of exactly 0 is very unlikely.

We define function $\mathbf{B} : [0,1]^d = \mathscr{X} \to \mathbf{B}(\mathscr{X}) \subseteq \{0,1\}^e$ which associates to each input $\mathbf{x} \in \mathscr{X}$, $\mathbf{B} : \mathbf{B}(\mathbf{x} \in \mathscr{X}) = \mathbf{b} \in \{0,1\}^e$ for $i \in \{1, \dots, e\}$:

$$\mathbf{b}_i = \begin{cases} 1, & \hat{\mathbf{z}}_i \geq 0, \\ 0, & \text{otherwise.} \end{cases} \tag{5.7}$$

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

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

Using this function, we can define the following equivalence relation:

$$\mathscr{R} : \mathscr{R}(\mathbf{x}_\phi, \mathbf{x}_\omega) \iff \mathbf{B}(\mathbf{x}_\phi) = \mathbf{B}(\mathbf{x}_\omega). \tag{5.8}$$

In other words $\mathscr{R}(\mathbf{x}_\phi, \mathbf{x}_\omega)$ means $\mathbf{x}_\phi$ and $\mathbf{x}_\omega$ have the same activation when forwarded through the network f.

This equivalence relation is then used to create equivalence classes:

$$[\mathbf{x}] = \{\mathbf{x}' \in \mathscr{X} : \mathbf{B}(\mathbf{x}') = \mathbf{B}(\mathbf{x})\}. \tag{5.9}$$

We know that the set of all equivalence classes forms a partition of the input space $\mathscr{X}$.

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, there exists 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*.

The linear regions are regions of the input space where the points have the same *activation pattern*. For a given ReLU network, activation patterns are defined in [Raghu et al., 2017] as a string of form $\{0,1\}^e$ encoding the linear region of the activation function of every neuron. In the following, we will refer to the activation patterns as binary combinations for two reasons :

1. we use binary variables in the MIP formulations of the search of adversarial examples,

2. all the activation patterns are binary combinations and but not all binary combinations are not activation patterns as shown in Figure 5.1.

5.3.2 Partitioning information in the MIP formulation

Let us now study a MIP formulation and see how to incorporate the information about the linear regions in it. To simplify the formulation,the class t is targeted using the constraint $\mathbf{s}_t > \mathbf{s}_y$.

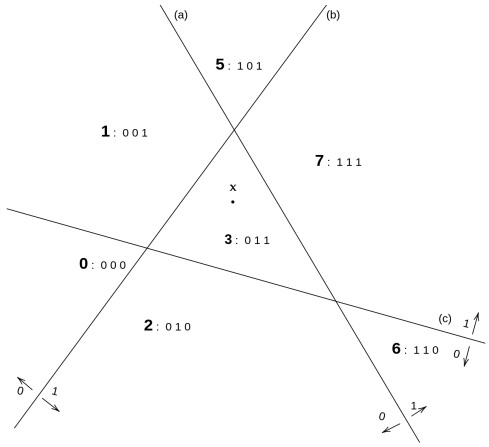
$$\begin{cases} \min & \|\mathbf{x} - \mathbf{x}'\| \\ \text{s.t.} & \hat{\mathbf{z}} = \mathrm{W}\mathbf{x}' + \beta, \\ & \mathbf{z}_i \geq 0, & i = 1,\dots,e \\ & \mathbf{z}_i \leq \mathrm{M}_r \mathbf{b}_i, & i = 1,\dots,e, \\ & \mathbf{z}_i \leq \hat{\mathbf{z}}_i + \mathrm{M}_r(1 - \mathbf{b}_i), & i = 1,\dots,e \\ & \mathbf{z}_i \geq \hat{\mathbf{z}}_i, & i = 1,\dots,e \\ & \mathbf{s} = \mathrm{V}\mathbf{z} + \alpha, \\ & \mathbf{s}_t > \mathbf{s}_y. \\ & \mathbf{x}' \in [0,1]^d, \\ & \mathbf{b} \in \{0,1\}^e, \\ & \mathbf{z} \in \mathbb{R}^e, \\ & \hat{\mathbf{z}} \in \mathbb{R}^e, \\ & \mathbf{s} \in \mathbb{R}^{\mathrm{C}}, \end{cases} \tag{5.10}$$

with big enough M_r.

Starting from this MIP formulation, we want to derive a powerful attack that can potentially find the same solution as the MIP, but faster than the MIP would do since it does not go through branching and bounding. The idea that emerged 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. To do so, we can restrict the search area to the linear region $[\mathbf{x}_\phi]$

$$\begin{cases} \min & \|\mathbf{x} - \mathbf{x}'\| \\ \text{s.t.} & s = \Gamma\mathbf{x}' + \gamma, \\ & \mathbf{x}' \in [\mathbf{x}_\phi], \\ & \mathbf{s}_t > \mathbf{s}_y. \\ & \mathbf{s} \in \mathbb{R}^{\mathrm{C}}. \end{cases} \tag{5.11}$$

Note that the optimization problem 5.11 is a linear program. Also, we do not have to determine the matrix Γ and the vector γ. Solving 5.11 is done in practice by fixing the binary values in the MIP (5.10) to $\mathbf{B}([\mathbf{x}_\phi])$ obtaining a linear program we call $\mathrm{LP}(\mathbf{B}(\mathbf{x}_\phi), t)$:



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 5.1 – Figure showing the correspondence between binary values and equivalence classes. Let us assume $\mathbf{x}$ represents a point in a two-dimensional input space and $(a), (b)$ and (c) represent neurons, the 3-digit code is the encoding corresponding to the binary values associated with each linear region of the input space. Each line represents a neuron and how it divides the plane into a part where the neuron is active and a part where it outputs 0. For example, $[\mathbf{x}]$, the equivalence class of $\mathbf{x}$, the linear region 3 corresponds to the linear region where the neuron a has negative response, and the neuron b and c have positive responses. Using the encoding abc, $[\mathbf{x}]$ corresponds to the linear region 011: 3. We draw your attention to the fact that the linear region corresponding to the binary code of number 4, 100, meaning a is active while b and c are inactive, does not have any antecedent since there can be no point in this setting verifying that. In other words, the binary combination 100 is not an activation pattern. To model the fact that if b and c are inactive the neuron a cannot be active, we can use the constraint $a \leq b + c$, which is verified by all feasible regions as shown in the rightmost column of the table.

$$
\begin{cases}
\min & \|\mathbf{x} - \mathbf{x}'\| \\
\text{s.t.} & \hat{\mathbf{z}} = \mathrm{W}\mathbf{x}' + \beta, \\
& \mathbf{z}_i \geq 0, & i = 1, \dots, e \\
& \mathbf{z}_i \leq \mathrm{M}_r \mathbf{b}_i, & i = 1, \dots, e, \\
& \mathbf{z}_i \leq \hat{\mathbf{z}}_i + \mathrm{M}_r(1 - \mathbf{b}_i), & i = 1, \dots, e \\
& \mathbf{z}_i \geq \hat{\mathbf{z}}_i, & i = 1, \dots, e \\
& \mathbf{s} = \mathrm{V}\mathbf{z} + \alpha, \\
& \mathbf{s}_t > \mathbf{s}_y, \\
& \mathbf{x}' \in [0, 1]^d, \\
& \underline{\mathbf{b} = \mathbf{B}(\mathbf{x}_\phi),} \\
& \mathbf{z} \in \mathbb{R}^e, \\
& \hat{\mathbf{z}} \in \mathbb{R}^e, \\
& \mathbf{s} \in \mathbb{R}^{\mathrm{C}},
\end{cases}
\tag{5.12}
$$

with big enough M_r.

The optimization problems 5.11 and 5.12 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 activation as $\mathbf{x}_\phi$ that is by construction, $[\mathbf{x}_\phi] = \{\mathbf{x} \in \mathscr{X} : \mathbf{B}(\mathbf{x}) = \mathbf{B}(\mathbf{x}_\phi)\}$, as stated in 5.9.

There is also a correspondence between the linear regions and binary values in $\mathbf{B}(\mathscr{X})$ such that a number can be attributed to each linear region as illustrated in Figure 5.1. Each linear region also corresponds to a leaf in the search tree associated with a MIP as shown in Figure 5.2.

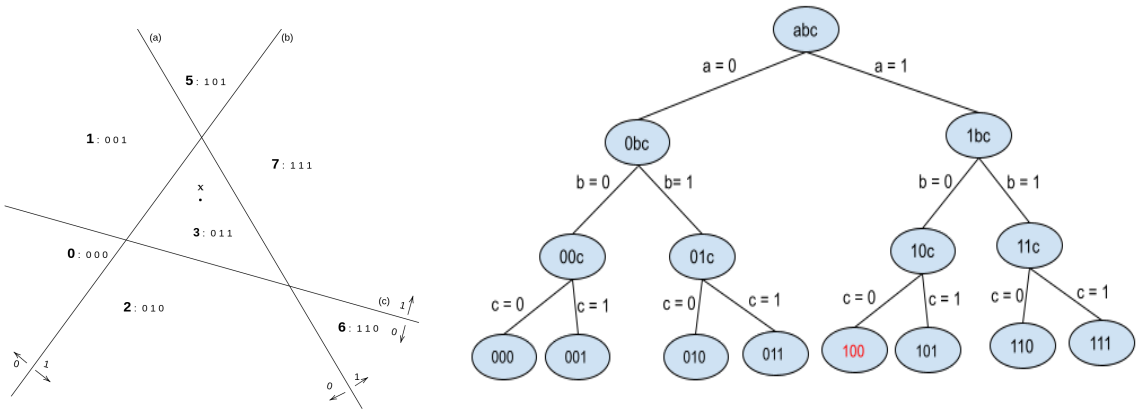


Figure 5.2 – Figure showing the relationship between the linear regions and the leaves of a search tree we associate with the MIP. The MIP needs to go through several steps before reaching a leaf. It means that several intermediate problems are solved before reaching a feasible own.

5.4 The SOTA MIP robustness verifier and Linear regions

In this section, we study how, given an input and the region in which we look for adversarial examples, the information contained in the weights can be exploited to accelerate the resolution of the MIP. We show how a part of this information has been exploited by MIPVerify [Tjeng et al., 2018], the SOTA robustness verifier, to speed up the resolution and also some proof that there are still unexploited information that could lead to an even bigger acceleration. And given the difficulty to uncover the unexploited information, we propose a method, Linear Program Powered Attack, that naturally exploits that information without having to explicitly discover it.

5.4.1 MIPVerify and the partitioning information

Let $\mathbf{x}$ be an original example and $\mathcal{N}_{\mathbf{x}}$ be the neighbourhood of $\mathbf{x}$ in which we are looking for adversarial examples. And let $\mathbf{B}(\mathcal{N}_{\mathbf{x}})$ be the set of all binary combinations corresponding to the linear regions covering $\mathcal{N}_{\mathbf{x}}$. The MIP resolution would be much faster if we could decrease the branch and bound space by having $\mathbf{b} \in \mathbf{B}(\mathcal{N}_{\mathbf{x}})$ instead of $\mathbf{b} \in \{0,1\}^e$ in the optimization problem 5.10. [Tjeng et al., 2018] already showed a correlation between the number of stable ReLUs and the solve time. The more stable ReLUs there are, the more the branch and bound trees are reduced, and the less time is spent branching and bounding to find the optimal solution and prove optimality. Several orders of magnitude in solve time were noticed when using the information carried by the bound. However, finding the stable ReLUs is a first step in reducing the branch and bound search tree. The stable ReLUs are related to the set of binary combinations that can be obtained from the input space. Indeed, the ReLUs corresponding to a neuron e_0 is stably equal to 0 (respectively 1) in $\mathcal{N}_{\mathbf{x}}$ if and only for all $\mathbf{b} \in \mathbf{B}(\mathcal{N}_{\mathbf{x}}), \mathbf{b}_{e_0} = 0$ (respectively $\mathbf{b}_{e_0} = 1$). Note that it was enough to gain several orders of magnitude in solve time compared to concurrent work.

5.4.2 Still unexploited information

Using $\mathbf{b} \in \{0,1\}^e$ in the optimization problem 5.10 is as if there is an assumption of the binary values in $\mathbf{b}$ being independent, but this assumption does not hold, there are interactions between the different coordinates of the binary vector $\mathbf{b}$, as we show in Figure 5.1 . In this example, we see that, when the values of b and c are 0, then the only value that a can take is 0, a relationship we modeled with the constraint $a \leq b + c$. This shows that [Tjeng et al., 2018] uses only a part of the information in $\mathbf{B}(\mathcal{N}_{\mathbf{x}})$.However, finding the unexploited information might be very difficult. In that example, the set of binary combinations corresponding to linear regions is $\{0,1\}^3 \backslash (1,0,0)$. In this simple example with only three neurons, we observe that the number of linear regions, 7, is less than the cardinal of $\{0,1\}^3$,8. It illustrates that, with more realistic neural networks, there may

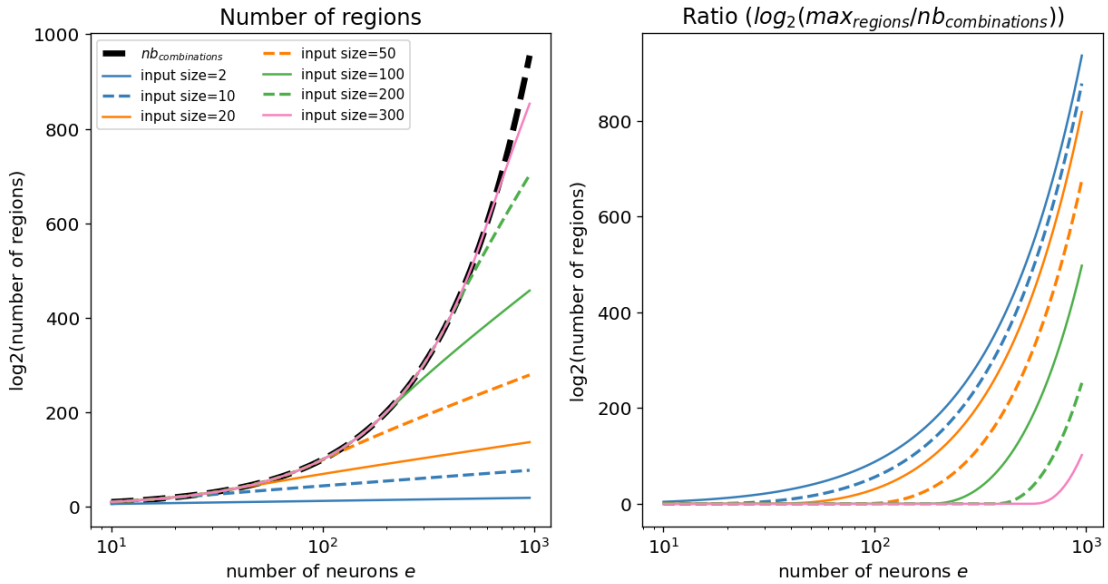


Figure 5.3 – Left: Figure showing the evolution of binary combinations $nb_{combinations}$ that can be modeled by the binary activation associated with a hidden layer, as well as the maximum number of linear regions that this hidden layer can represent using the same number of neurons, with respect to e, the number of neurons, and the input size. Right: Figure showing the evolution of the ratio between the number of binary combinations that can be modeled by the binary activation associated with a hidden layer, and the maximum number of linear regions that can be represented by that hidden with respect to the input size and e, the number of neurons.

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 toward binary combinations without correspondence to any linear region, hence those binary combinations will not produce any solution. In fact, there is nothing to prevent the MIP from going towards such useless binary combinations. And so, if we manage to find and include constraints such that the optimization is done only on binary combinations that effectively correspond to the linear regions, the combinatorial dimension of the MIP will drastically be reduced. And we would have reduce the solve time of the MIP. We can also refer to Lemma 4 of [Serra et al., 2018] to have an upper bound of the number of linear regions that be represented by e neurons (see Figure 5.3).

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}. More comparisons are available on Figure 5.3. 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 binary combinations paired with linear regions, could lead to a speed up of several orders of magnitude and make the problem of complete verification of Neural Network and the search of mean minimal adversarial distortion less prohibitively time-consuming. However, it means finding and modeling the constraints that discriminates between binary combinations that are activation paired with linear regions (i.e., activation patterns) and those that do not correspond to any activation pattern (i.e., useless binary combinations), which is out of the scope of this chapter.

5.5 Linear Program Powered Attacks

We have seen in Section 5.2 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. These attacks are based on two parts, the exploration of the linear regions and their exploitation.

5.5.1 LiPPA$_0$

LiPPA$_0$ is the first step of the full attack LiPPA. It is a "one-step" attack since it exploits a single linear region, the linear region of the clean example. Each clean example belongs to a linear region, and exploiting its linear region can be enough to find a good adversarial example, it is sometimes enough to find the optimal adversarial example as shown in [Croce et al., 2019]. They use two measures :

- $\Delta_B(\mathbf{x})$, the distance between the original example $\mathbf{x}$ and the border of its linear region, defined as $\min \|\mathbf{x}-\mathbf{z}\|$ for all $\mathbf{z}$ belonging to the border of the linear region of the clean example $\mathbf{x}$;
- $\Delta_D(\mathbf{x})$, the distance between $\mathbf{x}$ and the decision boundary in the linear region of the clean example $\mathbf{x}$, which can be defined as $\min \|\mathbf{x}-\mathbf{z}\|$ for all $\mathbf{z}$ belonging to the linear region of $\mathbf{x}$ and are misclassified.

When $\Delta_D(\mathbf{x}) \leq \Delta_B(\mathbf{x})$, then $\Delta_D(\mathbf{x})$ is the minimal adversarial distortion. Indeed, when $\Delta_D(\mathbf{x}) \leq \Delta_B(\mathbf{x})$, the point producing the minimal adversarial distortion is in the linear region of the original input, thus there is no point in looking for closer adversarial examples outside of the linear region of the original input. Using a MIP in those cases is a waste of time since LiPPA$_0$ is sufficient to find the closest adversarial examples with the resolution of a single Linear Program. LiPPA$_0$ directly exploits the linear region of the original input while the MIP would have to go through branching and bounding, solving several linear programs compared to a single linear program for LiPPA$_0$. However, LiPPA$_0$ is not always sufficient since it can find adversarial examples that are too far from the original to be interesting. In case there are no adversarial examples in the linear region of the original example, LiPPA$_0$ will definitely not find one. In that case, we have to explore the input space looking for other linear regions to find adversarial examples.

5.5.2 LiPPA

5.5.2.1 Observations and motivations

Exploitation of linear regions and its motivation Let us assume we have an oracle predicting the class t of the optimal adversarial example and the linear region $\mathbf{b}$ of the input space that contains the optimal adversarial example. Then the optimal adversarial example is found at the cost of solving a single linear program, LP($\mathbf{b}, t$). Unfortunately, such an oracle is not available, therefore we are going to use surrogates. We are going to find several ways of proposing linear regions that should lead to good adversarial examples. In practice, we solve a linear program targeting the incorrect class with the highest activations.

Exploration steps in order to find other regions to exploit Regarding the exploration of the input space, several methods can be used. Any method that allows traveling in an n-dimensional input space can be used. The goal of the exploration is to visit linear regions. The more linear regions are explored, the more is increased the chances of visiting the linear region containing the optimal adversarial example, or at least of finding a satisfying adversarial example. We will use Basic Iterative Methods [Kurakin et al., 2017] as a gradient-based surrogate to explore the input space. The choice of BIM for exploration can be motivated by the fact that following the adversarial gradient, each step increases the loss function and therefore points generally to the direction of a linear region potentially containing adversarial examples.

Our algorithm LiPPA is somewhat similar to the attack of [Croce and Hein, 2018, Croce et al., 2020], 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 5.1),
- they compute, for each linear region $[\mathbf{x}]$ the matrix Γ and the offset γ such that $f(\mathbf{x}' \in [\mathbf{x}]) = \Gamma\mathbf{x}' + \gamma$, this computation is not required in our approach,
- last but not least, they opted for a completely gradient-free approach selecting linear regions using a randomness, while our approach is hybrid since it uses the gradient to select the linear regions.

In [Croce and Hein, 2018, Croce et al., 2020] the authors generate randomly a number of points around the original example. For each random point $\mathbf{x}_\phi$, they compute the matrix Γ and the offset γ such that $f(\mathbf{x}' \in [\mathbf{x}_\phi]) = \Gamma_\phi\mathbf{x}' + \gamma_\phi$, and also compute the set of inequalities that define the polytope $[\mathbf{x}_\phi]$ over which the f is affine. In our approach, we neither compute Γ_ϕ and γ_ϕ nor do we compute the polytope, all the information we need is contained in $\mathbf{B}(\mathbf{x}_\phi)$ since as shown above it defines the linear region $[\mathbf{x}_\phi]$. They also store the activation patterns of those random points in order to check if this point lies in a linear region that were already checked and avoid repeating unnecessarily the same computation. This check is basically comparing the binary combinations of the linear region of a new point to the combinations of linear regions that were visited earlier.

Visiting binary values without antecedent with respect to B During the optimization, the MIP could visit binary values with no antecedent, i.e. binary values $\mathbf{b}$ for which there is no $\mathbf{x}$ such that $\mathbf{B}(\mathbf{x}) = \mathbf{b}$, which is not the case for our proposed methods since all the binary values are in $\mathbf{B}([\mathscr{X}])$. To have a sense of the difference that can exist between the exponential number of combination $\{0,1\}^e$ and the maximum number of linear regions that can be represented by the neural network, we can take a look at the Figure 5.3.

5.5.2.2 Algorithm

Definition 5.1 (LPStep)**.** *The exploitation of a linear region corresponds to the search of adversarial specifically on that linear region. It corresponds to the resolution of the linear program* $\mathrm{LP}(\mathbf{b}, t)$ *where t is the target class and $\mathbf{b}$ is the binary code corresponding the activation pattern of that linear region (cf Figure 5.1 and Figure 5.2) to find the closest adversarial example to the original one belonging to the class t and to the linear region corresponding to $\mathbf{b}$.*

Definition 5.2 (GSStep)**.** *Initialized with the solution of the LPStep, the exploration step is achieved using various iterative gradient steps. Iterations are stopped as soon as the current point belongs to a new linear region or after a budget limit.*

Definition 5.3 (Initialization)**.** *The initialization is called LiPPA$_0$: it is an LPStep inside the clean example's linear region.*

Definition 5.4 (Stopping criteria)**.** *Several criteria can be used :*

- *an adversarial example has been found with a distance to the clean example below the targeted ϵ,*
- *the time budget has been reached, we use a time limit of a few seconds since it is usualy enough as shown in Table 5.3,*
- *the step budget has been reached,*
- *GSStep is stuck in a linear region and can not find another,that situation hardly happens.*
- *the algorithm has "converged", it has fallen into a local minimal where it oscillates between two linear regions.*

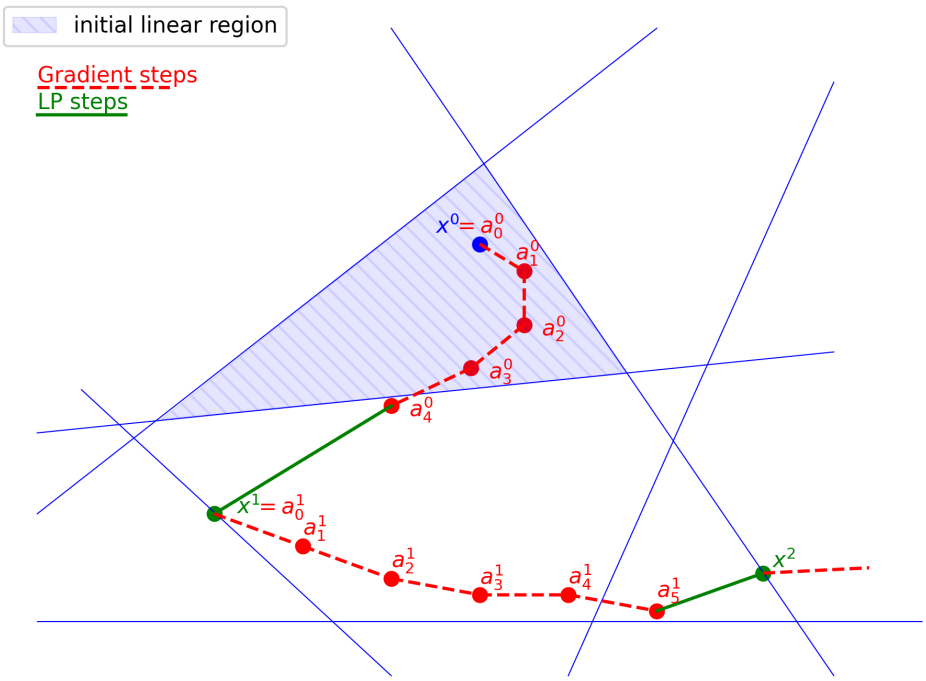


Figure 5.4 – Illustration of the behavior of LiPPA. Blue lines delimit linear regions, dashed red steps correspond to gradient steps and plain green ones to LP steps.

Algorithm 4 : LiPPA

Data : x, f, params_{stop}, params_{gs}
Result : $\hat{x}$
$[\hat{x}, b] \leftarrow$ initialisation(x,f) **[see Def.5.3]** ;
while *checkStopCriteria*$(x,\hat{x},\text{params}_{stop})$ **[see Def. 5.4] do**
  $[\hat{x}, b] \leftarrow$ GSStep$(\hat{x},f,b,\text{params}_{gs})$ **[see Def.5.2]** ;
  $[\hat{x}] \leftarrow$ LPStep(b,f) **[see Def. 5.1]**
end

Inner Loop The inner loop is responsible of going from a linear region to another linear region. In practice, we fix a limit on the number of of FGSM iterations that can be performed in order to prevent the algorithm from having an infinite loop. Even though, in practice a few steps are enough to go from a region to another. And if the number of steps needed appears to be too large, one can fix that by increasing appropriately the magnitude of each gradient step. One can also think of using variables steps, let it be randomly chosen steps or steps chosen in a specific schedule.

Initialization/Restart We initialize our algorithm using the linear region of the original example. It sometimes suffices to find a satisfying example. It is also proven that the optimal adversarial example can be found in that region [Croce et al., 2019]. For an input example $\mathbf{x}$ with its distance to the decision boundary being $\Delta_D(\mathbf{x})$ and its distance to the boundary of the linear region $\mathbf{B}(\mathbf{x})$ being $\Delta_B(\mathbf{x})$, if $\Delta_D(\mathbf{x}) < \Delta_B(\mathbf{x})$, then the closest adversarial example lies in the linear region $\mathbf{B}(\mathbf{x})$, the linear region of the original input. However, that is not the most frequent case when the model is not trained in a particular way. And sometimes the optimization over the original input space leads to adversarial examples that are too far from the original input. In that case, we use PGD to find a closer adversarial example that will serve as a new starting point. We also use PGD to restart the search when LiPPA is stuck in a local minimum when it struggles to get out a linear region while respecting the pixel range.

In summary, our method exploits both gradient-based and gradient-free information. We take

advantage of the information given by the gradient by using BIM to go from a linear region to another. We also take advantage of the particular partition in linear regions which is such that, for each target class and for each linear region, the optimal adversarial example on that linear region is found by solving a linear program, which is the gradient-free part.

5.6 Experimental results

Dataset and Models In our experiments, we used MNIST and CIFAR datasets.The architecture MLP-B represents a multilayer perceptron that has two hidden layers, each having 100 neurons. The architecture MLP-C represents a multilayer perceptron with one hidden layer with 1024 neurons. and the models of [Salman et al., 2019]. The architecture CNN is the architecture used in [Wong and Kolter, 2018] which is made of a first convolutional layer with 16 filters, a second with 32 filters each having a stride of two, followed by a hidden layer of 100 neurons and finally followed by the output layer. Models are prefixed to show how they were trained. Models prefixed with NOR were trained normally with the cross-entropy loss function. Models prefixed with ADV were trained with adversarial examples generated by Projected Gradient Descent (PGD). Models prefixed with LP_D are trained using [Wong and Kolter, 2018]. Models prefixed with Maximum Margin Regularizer (MMR) are trained using [Croce et al., 2019]. And Models prefixed with MMR-ADV used both adversarial training with PGD and MMR. All used models are available online at Salman's github[1] for architecture MLP-B, at Andriushchenko's github [2] for MLP-C and CNN except for LP_D-CNN which were taken from `https://github.com/locuslab/convex_adversarial`.

Threat model We consider a white box setting, i.e., the attacker has complete access to the model and can exploit as much information as possible. This allows us to extract information from the outputs of the hidden layers. We test robustness for ℓ_∞. However, LiPPA can also be used to look for adversarial examples for ℓ_1 and also ℓ_2 without major changes to the algorithm or the implementation. It is worth noting that for the case of ℓ_2 the obtained optimization problem is a quadratic program instead of a linear one.

Implementation details In the implementation, we avoided the use of big-M constraints to model the ReLU constraints. Since when using LiPPA, for each neuron of the network we know its state, i.e., if it is active or not, we were able to directly encode that in the formulation. Let us denote i_- the indices i such that $\mathbf{b}_i = 0$ (meaning pre-ReLU activations) are negative and i^+ the indices such that $\mathbf{b}_i = 1$ (meaning pre-ReLU activations) are nonnegative. The ReLU constraints are then:

- $\hat{\mathbf{z}}_i < 0$, $\mathbf{z}_i = 0, i \in i_-$,
- $\hat{\mathbf{z}}_i > 0$, $\mathbf{z}_i = \hat{\mathbf{z}}_i, i \in i^+$.

In practice, we start by exploiting the linear region of the original examples. That means, given a model and an example $\mathbf{x}$ belonging to the class y, with t the target class (the wrong class with the highest activation), we solve the corresponding $\text{LP}(\mathbf{B}(\mathbf{x}), t)$. It is sometimes enough to find a satisfying adversarial example. If that is not the case, then we look for adversarial examples using Projected Gradient Descent increasing the allowed distance when adversarial examples are not found, allowing PGD to find adversarial with higher distance than the desired ϵ in order to have a starting point for LiPPA. In fact, experiments showed that LiPPA tends to have trouble finding good adversarial examples when the adversarial example found in the linear region of the original is too far from the original example. Using PGD to give a starting point is therefore our workaround for this problem. If the adversarial example found using PGD is satisfying, i.e., its distance to the original example is less than the magnitude ϵ for which we want to compute the robust test error lower bounds and upper bound, then we do not need to fully launch LiPPA. If the adversarial example is not satisfying, we use it as a starting point for LiPPA by optimizing its linear region to find an adversarial example closer to the original example, and then use enough iterations of FGSM to

[1] `https://github.com/Hadisalman/robust-verify-benchmark`
[2] `https://github.com/max-andr/provable-robustness-max-linear-regions`

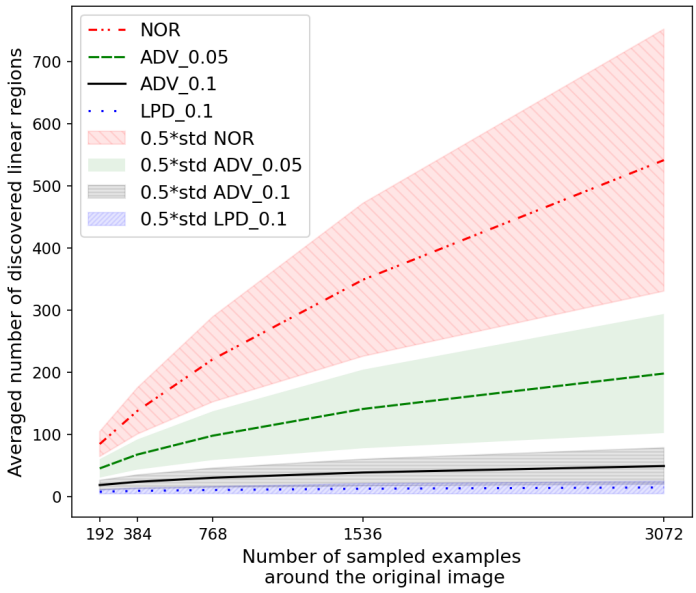


Figure 5.5 – Evolution of the number of found linear regions averaged over the first 100 images of the MNIST test set with respect to the number of random samples around the original example, using MLP-B.

get to another linear region, thus alternating between gradient-free information exploitation and gradient based one.

5.6.1 Approximation of the number of linear regions

Experimental setup and results 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. This holds even when we take 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 [3]. We count the number of uniquely discovered linear regions on 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 we found. Table 5.1 shows the average number of unstable ReLUs, n_{U}, the number of all possible binary combinations to n_{U} unstable ReLUs, and the number of linear regions we found. Our experiments show that the number of linear regions we found is smaller than $2^{n_{\mathrm{U}}}$, the number of binary combinations corresponding the n_{U} unstable ReLUs. Figure 5.5 shows that the average number of found linear regions does not grow as fast as the number of random samples.

Robustness, unstable ReLUs and number of linear regions Figure 5.5 shows that, for the same architecture, the number of found linear regions is lower for robustified models. Linking that with robustness measures, we can say that the more robust the network, the less linear regions are found. Moreover, a lower number of linear regions around the input examples plays a significant role in fastening the verification. This finding goes in the same direction as previous works. Previous results in [Tjeng et al., 2018] has already shown that verifying the robustness for models trained with the provable method [Wong and Kolter, 2018] takes less time than on models with the same architecture trained with adversarial training for example. It is coherent with the fact that, in [Tjeng et al., 2018], they found more stable ReLUs for the model trained with the provable method [Wong and Kolter, 2018] than the one trained with adversarial training for the same architecture and the same robustness target. Our experiment goes further by approximating the number of

[3] 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

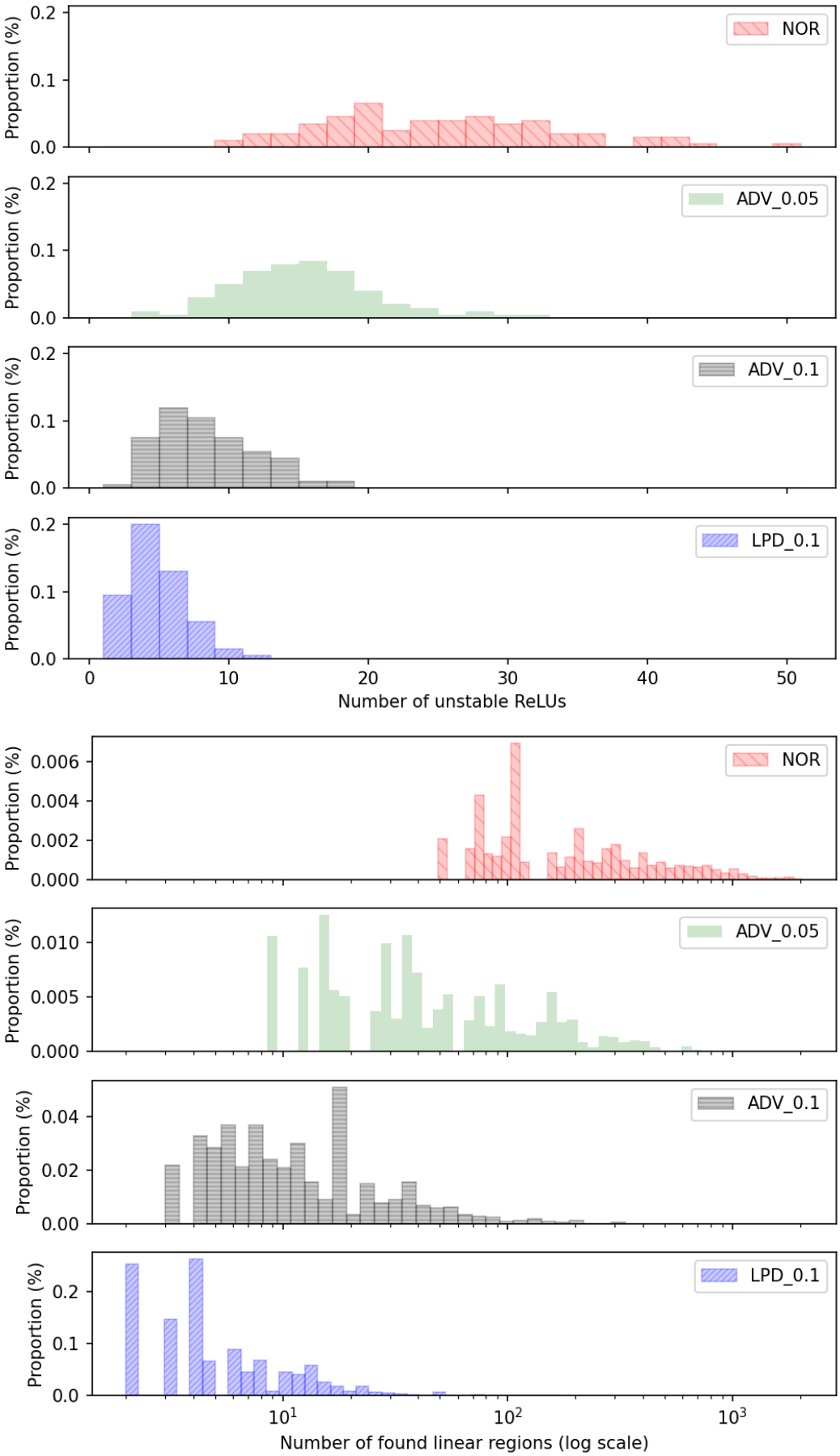


Figure 5.6 – Distribution of the number of unstable ReLUs and the number of linear regions for the MLP-B architecture and different training methods. We observe a shift of the distribution when going from not robust models to more robust models.

Table 5.1 – Comparison between the number of linear regions and the number of found linear regions across different training methods for the MLP-B architecture. The value next to the training method represents the value ℓ_∞ the model was trained to be robust to.

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

linear regions. And while in [Tjeng et al., 2018] they compute the stable ReLUs and deduce the number of possibly unstable ReLUs, we approximate the number of found unstable ReLUs, which can allow the deduction of the number of possibly stable ReLUs. And comparing the number of found linear regions and the number 2^{n_U} with n_U the number of unstable ReLUs we can see some huge differences for example for the model adversarially trained to be robust to attacks with infinite norm 0.05, we have on average nearly 198 linear regions while the average number of unstable ReLUs found is 14.89, it would correspond to $2^{14.89} > 3 \times 10^4$ possible binary combinations. We also observe that the number of found linear regions is roughly 4 times less for the model trained with adversarial training to be robust to a greater ℓ_∞ perturbation, 0.1, than for the small one, 0.05. Likewise, the number of found linear regions for LP_D is 3.5 times smaller than when trained with adversarial training for the same allowed perturbation. We can observe a shift in the distribution of the number of linear regions in Figure 5.6 further highlighting the effect of robust training. It is intuitive to say that this low number of linear regions plays a role in the fact that the MIP usually converges for the model trained with LP_D method.

This shift is a direct consequence of the reduction of the number of provably unstable found ReLUs. For MLP-B architecture, in Figure 5.6, we can observe a shift in the distribution of the number of unstable ReLUs. We can also see that the distribution of the number of unstable ReLUs seems to shrink, meaning the standard deviation of the number of unstable ReLU decreases. It means not only that the mean number of unstable ReLUs decreases, but also that the numbers are less scattered and are gathered more densely around the mean. And the model LP_D-MLP-B, which is the most robust and the most easily verified, has a very low number of unstable ReLUs when compared to the naive model, NOR-MLP-B. The reduction of the standard deviation of the number of unstable ReLUs also affects the standard deviation of the number of linear regions even though it is not that obvious on Figure 5.5 due to the logarithmic scale. Nevertheless, looking at Figure 5.3, we can clearly see that the standard deviation of the number of linear regions is several orders of magnitude greater for NOR-MLP-B than for LP_D-MLP-B. We can also remark that when using adversarial training, the model trained to be robust to adversarial perturbations with ℓ_∞-norm of 0.05 has more linear regions on average and also has a higher standard deviation than the model trained to be to robust to perturbation with ℓ_∞-norm 0.1.

We can conclude that more vulnerable networks have more unstable ReLUs. Since the models were not explicitly trained to have more stable ReLUs but to be more robust, we have an empirical hint that robustness enhances the stability of ReLUs activation around training input points. In other words, the more robust a neural network is, the more stable the ReLU activations will be around the data points.

Our experiment shows evidence that the number of linear regions around the samples is less than the number of all possible binary combinations susceptible to be explored using an MIP formulation. And, more robust models tend to have fewer linear regions around the original examples.

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. Similar results has been found by [Hanin and Rolnick, 2019] stating that "Deep ReLU networks have surprisingly few activation patterns". This step combined with a good mechanism to select the most promising linear regions/combinations are the heart of our approach.

5.6.2 LiPPA$_0$

Set-up details In this experiment, we study the first step of LiPPA, LiPPA$_0$, consisting in looking for adversarial examples in the linear region of the original examples. We use the first 1000 examples of the test set. The performances of LiPPA$_0$ are reported in Table 5.2. The Figure 5.7 also shows the distribution of the adversarial distortion necessary to fool the network while remaining in the linear region of the original examples.

LiPPA$_0$'s runtime The runtime of LiPPA$_0$ is very dependent on the architecture (see Table 5.2). For example, for the MLP-C architecture, one hidden layer of 1024 units, it takes around 2s, while it takes around 0.27s for MLP-B architecture which is made of 2 hidden layers of 100 layers each, and for CNN architecture it takes around 0.6s for MNIST. This is explained by the fact that architecture affects the number of variables in the optimization problem. We can also observe nonsignificant differences with respect to the training method for the same architecture, it is usually due to the number of samples where LiPPA$_0$ does not find a feasible solution. LiPPA$_0$'s optimization is shorter when there is no feasible solution.

Percentage of infeasible The percentage of infeasible is the percentage of samples for which LiPPA$_0$ was not able to find an adversarial example in the linear region of the original input sample such that all pixels lie between 0 and 1. This percentage seems to be related to the architecture and to the training method. Convolutional neural networks have a higher percentage of infeasible than fully connected networks. The percentage of infeasible for NOR-CNN is 18.20% while the percentage for NOR-MLP-B is 15.90% and the percentage of NOR-MLP-C is 0%. We also remark that going from naive training to robust training, the percentage of infeasible increases. It is the case for the MLP-C architecture which initially had 0% of infeasible attack which grows to 11.20% when the adversarial training is performed to be robust to perturbations of ℓ_∞ 0.1. This increase is not always observed for the architecture MLP-B where NOR-MLP-B has 15.90% of infeasible optimization problems, while for the other training methods a lower percentage of infeasible is obtained for different robust training methods. LP$_D$-MLP-B with ℓ_∞ target robustness 0.2 is the exception where 18.70% is obtained. For the architecture MLP-B and with adversarial training, we observe that the percentage of infeasible optimization problems grows when the robustness target is increased, going from 1.80% for $\varepsilon = 0.03$ to 10.00% for $\varepsilon = 0.1$. The same phenomenon is observed for the LP$_D$ training method of the architecture MLP-B.

Minimal Adversarial Distortion (MAD) In the linear region of the input example, LiPPA$_0$ outputs the minimal adversarial distortion to find an adversarial example when the corresponding optimization problem admits a solution. That MAD distribution is different between the naive training and the robust training procedures. For the average MAD, models trained naively have a smaller value than models trained to be robust to adversarial perturbations. We can clearly see that the MAD found using LiPPA$_0$ drifts toward higher values when going from naive training to robust training in Figure 5.7 . In the computation of the average MAD, examples that are misclassified (even without adversarial perturbations) are not taken into account. While using adversarial training for the architecture MLP-B, the average MAD found using LiPPA$_0$ goes from 0.19 to 0.26 when the target ℓ_∞ robustness goes from 0.03 to 0.1. LiPPA$_0$ can be used to test the presence or effectiveness of a defense mechanism.

5.6.3 LiPPA performance

Set-up details In our experiments, we used $\epsilon_{\text{FGSM}} = 0.5/256$, and as the stop criterion for the exploration step, the fact that we changed the linear region or that we did not change linear after 50 FGSM iterations, in that case we look for a new starting point. Results are in Table 5.4, where ϵ represents the magnitude for which we compute the lower bounds and upper bounds on the robust test error. It is also used to train for robustness when the model is trained to be robust to

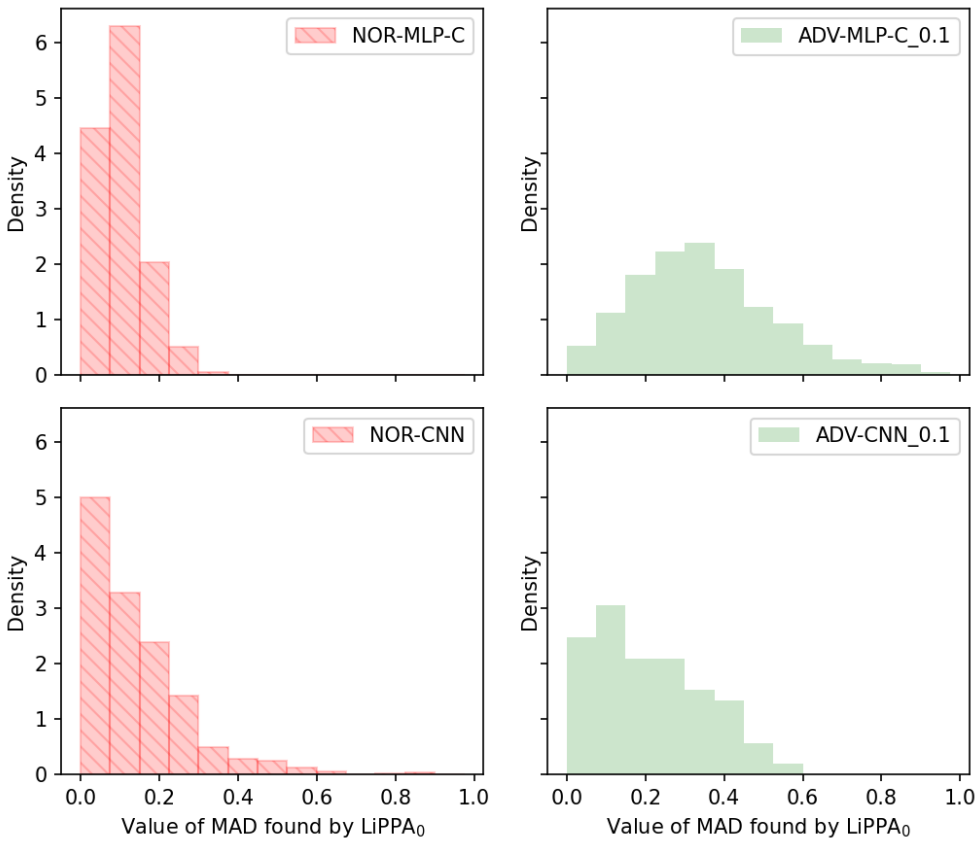


Figure 5.7 – Shift in the MAD distribution found by LiPPA$_0$. The top row represents the MLP-C architecture and the bottom row represents the CNN architecture. The left column represents the the normal training and the second column adversarial training with robustness target of 0.1.

adversarial attacks, which is the case with all networks except the ones starting with NOR. The test error is the misclassification rate on the clean examples. 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 results are from [Salman et al., 2019], and using PGD, the gradient-free attacks of [Croce and Hein, 2018, Croce et al., 2020] and a MIP with a time limit of 120s when from [Croce and Hein, 2018]. The computation of the lower and upper bounds is done using the code associated with [Tjeng et al., 2018].

Comparison to the MIP and other attacks Let us remind that when the lower bound is equal to the upper bound, it means that for every example either an adversarial example was found or it was proven that there is no adversarial example, i.e. there was no timeout for the MIP. In the same way, if we have a gap between the lower and the upper bound, 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 when the time limit was reached by the MIP.

Performance On the MNIST dataset, our attack does better than the previously reported lower bounds on NOR-MLP-C, ADV-MLP-C, and ADV-CNN where, among other attacks, the MIP had a time limit of 120s but failed to find adversarial examples found using LiPPA in at most a few seconds and usually less than a second. 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. On the CIFAR dataset, LiPPA matched the lower bound on one network and had comparable results on the other.

Table 5.2 – Table showing the performance of LiPPA_0, the percentage of examples for which LiPPA_0 found no feasible solution, the average minimal adversarial distortion (MAD) and the average runtime for the first 1000 samples of the test set.

Network	ϵ	TEST ERROR	LOWER BOUND	LiPPA_0	% infeasible	MAD	runtime
NOR-MLP-B	0.02	2.05	10.16*[Salmanet al.,2019]	6.80	15.90	0.13	0.27
NOR-MLP-B	0.03	2.05	20.43*[Salmanet al.,2019]	10.20	15.90	0.13	0.27
NOR-MLP-B	0.05	2.05	53.37*[Salmanet al.,2019]	19.10	15.90	0.13	0.27
NOR-MLP-C	0.1	1.44	93[Croceet al.,2019]	51.30	0	0.11	1.97
ADV-MLP-B	0.03	1.53	4.18*[Salmanet al.,2019]	4.40	1.80	0.19	0.28
ADV-MLP-B	0.05	1.62	6.11*[Salmanet al.,2019]	6.60	4.10	0.24	0.27
ADV-MLP-B	0.1	3.33	16.25*[Salmanet al.,2019]	16.40	10.00	0.26	0.21
ADV-MLP-C	0.1	0.92	10[Croceet al.,2019]	6.50	11.20	0.35	1.49
ADV-CNN	0.1	0.82	3[Croceet al.,2019]	2.30	92.20	0.22	0.67
LP_D-MLP-B	0.1	4.09	14.45*[Salmanet al.,2019]	16.3	0	0.25	0.21
LP_D-MLP-B	0.2	15.72	36.33*[Salmanet al.,2019]	33.9	18.70	0.36	0.18
LP_D-CNN	0.1	1.8	4.40[Croceet al.,2019]	4.1	91.5	0.21	0.55
MMR-MLP-C	0.1	2.11	22.5[Croceet al.,2019]	18.20	0	0.17	1.52
MMR-CNN	0.1	1.65	6[Croceet al.,2019]	4.60	86.30	0.27	2.14
MMR-ADV-MLP-C	0.1	2.04	14[Croceet al.,2019]	11.70	0	0.25	1.58
MMR-ADV-CNN	0.1	1.19	3.6[Croceet al.,2019]	2.70	94.50	0.19	0.51

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: if LiPPA is to find an adversarial example, it finds it real quick. If LiPPA runs more than a few seconds, it is a good prediction that it will fail. Table 5.3 illustrates this. Knowing this, a time limit of a few seconds per image could be set to save time.

Table 5.3 – Average runtime for failed and successful LiPPA search.

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 5.4 – MLP-B models are from [Salman et al., 2019] and the other from [Croce et al., 2019] completed with our new attack for comparison on the first 1000 examples. Starred results * are computed over the whole test set. **Bold results** are those beating or matching previous methods. For the results taken from [Salman et al., 2019], the lower bound reported is the best lower bound between PGD and the state-of-the-art MIP with a timeout of 3600s. For the results taken from [Croce et al., 2019] the lower bound is the best lower bound found by PGD, the gradient-free attack [Croce and Hein, 2018, Croce et al., 2020] and a MIP with a timeout of 120s. The upper bounds are reported from the same paper than the lower bound of the same line. $^+$:The upper bound is computed on the whole test set while only the first 1000 samples were used in our experiment.

DATASET	Network	ϵ	TEST ERROR	LOWER BOUND	LiPPA	UPPER BOUND
MNIST	NOR-MLP-B	0.02	2.05	10.16*[Salmanet al.,2019]	**11.40**	13.48*
	NOR-MLP-B	0.03	2.05	20.43*[Salmanet al.,2019]	**25.20**	48.67*
	NOR-MLP-B	0.05	2.05	53.37*[Salmanet al.,2019]	**63.30**	94.04*
	NOR-MLP-C	0.1	1.44	93[Croceet al.,2019]	**94.30**	100
	ADV-MLP-B	0.03	1.53	4.18*[Salmanet al.,2019]	**5.20**	5.78*
	ADV-MLP-B	0.05	1.62	6.11*[Salmanet al.,2019]	**7.50**	11.38*
	ADV-MLP-B	0.1	3.33	16.25*[Salmanet al.,2019]	**20.40**	34.3*
	ADV-MLP-C	0.1	0.92	10[Croceet al.,2019]	**11.90**	99
	ADV-CNN	0.1	0.82	3[Croceet al.,2019]	**4.50**	100
	LP_D-MLP-B	0.1	4.09	14.45*[Salmanet al.,2019]	17.40$^+$	14.45*
	LP_D-MLP-B	0.2	15.72	36.33*[Salmanet al.,2019]	38.3$^+$	36.33*
	MMR-MLP-C	0.1	2.11	22.5[Croceet al.,2019]	21.30	24.9
	MMR-CNN	0.1	1.65	6[Croceet al.,2019]	5.80	6
	MMR-ADV-MLP-C	0.1	2.04	14[Croceet al.,2019]	13.40	14.1
	MMR-ADV-CNN	0.1	1.19	3.6[Croceet al.,2019]	3.20	3.6
CIFAR-10	NOR-CNN	2/255	24.63	91[Croceet al.,2019]	87.7	100
	ADV-CNN	2/255	27.04	52.5[Croceet al.,2019]	48.1	88.5
	LP_D-CNN	2/255	38.91	46.6[Croceet al.,2019]	46.3	48
	MMR-CNN	2/255	34.61	57.5[Croceet al.,2019]	56.9	61
	MMR-ADV-CNN	2/255	35.38	47.9[Croceet al.,2019]	**47.9**	54.2

5.7 Conclusion

We proposed 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 adversarial 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. The result of the exploitation step will serve as a starting point for the following exploration step, and so on, and so forth.

Doing that, LiPPA was 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 one of the most powerful attacks. We showed that the input space of neural networks with piecewise linear activations, such as ReLU activations, 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. We showed that there is still unexploited information in the verification with respect to the linear regions of the input space. In an attempt to harness that information, the attack we propose manages to find quickly adversarial examples that the MIP could not find within the allocated time limit.

Future work includes finding better ways to propose linear regions 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 binary values corresponding the linear regions of the input space to accelerate the robustness verification.

Conclusion

Among the different challenges that deep learning models have to tackle this thesis focuses on trustworthiness and reliability through the lens of adversarial examples. We presented adversarial examples and why it is important to study them, particularly for models that are to be deployed in safety-critical environments. Even though there is not yet a consensus on the reasons for the existence of adversarial examples, several interesting hypotheses have been formulated. The *evolutionary stalking* hypothesis states that as training progresses, correctly classified samples contribute less and less to the weight updates and have little effect on the decision boundaries, leaving them very close to the decision boundaries. This hypothesis seems to be the one that explains most simply the existence of adversarial examples and it fits several domains and data types. In the image classification domain, the existence of adversarial images challenges the idea that the first layers learn to recognize concepts such as lines or color blobs while the layers closer to the output learn to identify more complex concepts such as faces and trees for example. We worked exclusively on image classification and with additive perturbation using ℓ_p norms as done by most of the work on adversarial image classification. The reason is not that robustness to those perturbations means overall robustness, but rather the fact that the lack of robustness to ℓ_p hints at vulnerability to more complex, more realistic attacks, since simple translation or rotation operations are enough to fool classifiers. There is however much more to the perturbations than just additive perturbations.

1st contribution: Our first contribution is a new defense mechanism based on adversarial training and double back-propagation. Adversarial training that we can see as an advance form of data augmentation is known to have a regularization on the loss landscape, smoothing it. Double-backpropagation has shown to be an effective regularization in the past and supports the hypothesis that flattering loss landscapes leads to better generalization. Using the idea of double-backpropagation, we further enforce that regularization by explicitly penalizing the components of the gradient of the loss not only at the training points but at potential adversarial points around them. It helped obtaining models that were at least as robust when only adversarial training was used.

2nd contribution: Once a model is trained, to ensure reliability, we test the robustness of that model against adversarial attacks. We can evaluate the robustness of a model by testing its accuracy against adversarial attacks such as gradient-based attacks. However, those attacks can give a false sense of robustness. It becomes necessary to use more powerful methods, for example the formulation of the search of adversarial examples as a MIP. However, the MIP formulation has as main drawback the fact that it is time-consuming. We then tried to find an attack based on the MIP formulation but that is much faster than the MIP using DCA. Starting from the MIP formulation, DCA relaxes the binary variables and adds a penalization that allows to obtain binary values for the relaxed variables without having to explicitly declare them as binary. That rids us of the branch-and-bound mechanism that makes the MIP slow. However, DCA quickly reached its limits. Our hypothesis on the reasons why DCA did not manage to obtain the good performances it obtained on other problems is the link between the input space and the binary values that is explicit in the last and main contribution of this thesis.

3rd contribution: This contribution is the follow-up of the previous one. Using DCA with carefully chosen parameters, we notice that we could predict the final values of pseudo-binary values knowing their initial value. In particular, when the initial values are deduced from the activation pattern of the original input image, we retrieve the same values after convergence. That is because there is an equivalence between the binary combination given by the activation pattern and the linear region of the input space. Each linear region corresponds to a binary combination. However, the flip side does not hold. There are many binary combinations that do not correspond to any linear region. This shows that the number of linear regions is smaller than the number of binary combinations as encoded in the MIP formulation. Even though there is a large number of linear regions, exploiting the gradient as done in gradient-based attacks helps to select linear regions with a higher chance of finding adversarial examples. LiPPA is, to the best of our knowledge, the first attack that combines gradient-based and gradient-free information. Combining them, LiPPA is able to achieve better performance than previous attacks including the MIP with a time limit. LiPPA is more accurate than gradient-based attacks while being much faster than the MIP. It is more accurate than the gradient-based attacks because it inherits the accuracy of the MIP. And it is faster than the MIP because LiPPA does not have to go through branch-and-bound to find adversarial examples and LiPPA exploits the gradient to find quickly linear regions while the MIP does not use the gradient. That makes LiPPA a very powerful attack that is able of finding very quickly adversarial examples that are the closest to the original inputs, the same found by the MIP given enough time. Furthermore, exploiting the linear regions allows to find precious information about the correlations between robustness, the number of linear regions, and the number of unstable ReLUs. Our experiments showed that the more robust models have less unstable ReLUs that leads to a fewer number of linear regions.

Perspective 1: on the biases of the neural networks The existence of adversarial examples questions old beliefs about how neural networks work. It also showed that neural networks do not rely on the same features as humans to perform image classification. Neural networks are believed to give much less importance to the form that humans do. Neural networks are believed to rely heavily on "texture" to make predictions. Directing the neural networks to rely more on the form of objects could help get more robust models, models that are less prone to adversarial examples. One way to do that is to extend the training datasets to pair each of their images with its results when passed through an edge detector and train on those pairs. The training phase should be crafted for the models to rely on less on "textures" and more on forms.

Perspective 2: on enhancing verifiability It is shown that having a high number of stable ReLUs helps the verification process. Finding a training method that can give robust models while at the same time guaranteeing a high number of stable ReLUs can lead to further verifiability of the obtained models.

Perspective 3: Exploiting the linear regions in the verification process Each linear region corresponds to a binary combination, that is, a leaf on the search tree associated with branch-and-bound, thus with complete verification methods such as MIP, and the flip side is not true. Finding a way to limit the search of adversarial examples only to the combinations that correspond to linear regions should lead to a speed-up of several orders of magnitude. It could be, for example, finding constraints that discriminate between the binary combinations corresponding to linear regions and binary combinations that do not correspond to any linear region.

Bibliography

[Akhtar and Mian, 2018] Akhtar, N. and Mian, A. (2018). Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey. *IEEE Access*, 6:14410–14430. 48, 49, 50

[Alzantot et al., 2019] Alzantot, M., Sharma, Y., Chakraborty, S., Zhang, H., Hsieh, C.-J., and Srivastava, M. (2019). GenAttack: Practical Black-box Attacks with Gradient-Free Optimization. *arXiv:1805.11090 [cs]*. arXiv: 1805.11090. 94

[Arthur, 2013] Arthur, C. (2013). iPhone 5S fingerprint sensor hacked by Germany's Chaos Computer Club. *The Guardian*. 37

[Athalye et al., 2018a] Athalye, A., Carlini, N., and Wagner, D. (2018a). Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. In *Proceedings of the 35th International Conference on Machine Learning*, 2018. arXiv: 1802.00420. 53, 54, 93

[Athalye et al., 2018b] Athalye, A., Engstrom, L., Ilyas, A., and Kwok, K. (2018b). Synthesizing Robust Adversarial Examples. page 10. 53, 54

[Bakator and Radosav, 2018] Bakator, M. and Radosav, D. (2018). Deep Learning and Medical Diagnosis: A Review of Literature. *Multimodal Technologies and Interaction*, 2(3):47. Number: 3 Publisher: Multidisciplinary Digital Publishing Institute. 32

[Baluja and Fischer, 2017] Baluja, S. and Fischer, I. (2017). Adversarial Transformation Networks: Learning to Generate Adversarial Examples. *arXiv:1703.09387 [cs]*. arXiv: 1703.09387. 46

[Bansal et al., 2018] Bansal, M., Krizhevsky, A., and Ogale, A. (2018). ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst. *arXiv:1812.03079 [cs]*. arXiv: 1812.03079. 32

[Bastani et al., 2016] Bastani, O., Ioannou, Y., Lampropoulos, L., Vytiniotis, D., Nori, A., and Criminisi, A. (2016). Measuring Neural Net Robustness with Constraints. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *NeurIPS 29*, pages 2613–2621. Curran Associates, Inc. 34, 92, 93

[Bhagoji et al., 2019] Bhagoji, A. N., Cullina, D., and Mittal, P. (2019). Lower Bounds on Adversarial Robustness from Optimal Transport. *Neural Information Processing Systems (NeurIPS 2019*, page 13. 62, 63

[Biere et al., 2009] Biere, A., Biere, A., Heule, M., van Maaren, H., and Walsh, T. (2009). *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press, NLD. 20

[Brendel et al., 2018] Brendel, W., Rauber, J., and Bethge, M. (2018). Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models. *arXiv:1712.04248 [cs, stat]*. arXiv: 1712.04248. 43

[Brendel et al., 2019] Brendel, W., Rauber, J., Kümmerer, M., Ustyuzhaninov, I., and Bethge, M. (2019). Accurate, reliable and fast robustness evaluation. *arXiv:1907.01003 [cs, stat]*. arXiv: 1907.01003. 92

[Brown et al., 2018] Brown, T. B., Mané, D., Roy, A., Abadi, M., and Gilmer, J. (2018). Adversarial Patch. *arXiv:1712.09665 [cs]*. arXiv: 1712.09665. 47

[Buckman et al., 2018] Buckman, J., Roy, A., Raffel, C., and Goodfellow, I. (2018). Thermometer Encoding: One Hot Way To Resist Adversarial Examples. 53

[Bunel et al., 2018] Bunel, R. R., Turkaslan, I., Torr, P., Kohli, P., and Mudigonda, P. K. (2018). A Unified View of Piecewise Linear Neural Network Verification. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31*, pages 4790–4799. Curran Associates, Inc. 61, 93

[Carlini et al., 2019] Carlini, N., Athalye, A., Papernot, N., Brendel, W., Rauber, J., Tsipras, D., Goodfellow, I., Madry, A., and Kurakin, A. (2019). On Evaluating Adversarial Robustness. *arXiv:1902.06705 [cs, stat]*. arXiv: 1902.06705. 50

[Carlini and Wagner, 2017a] Carlini, N. and Wagner, D. (2017a). Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods. *arXiv:1705.07263 [cs]*. arXiv: 1705.07263. 54

[Carlini and Wagner, 2017b] Carlini, N. and Wagner, D. (2017b). Towards Evaluating the Robustness of Neural Networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. ISSN: 2375-1207. 41, 46, 52, 92

[Carlini and Wagner, 2018] Carlini, N. and Wagner, D. (2018). Audio Adversarial Examples: Targeted Attacks on Speech-to-Text. *arXiv:1801.01944 [cs]*. arXiv: 1801.01944. 48

[Chaubey et al., 2020] Chaubey, A., Agrawal, N., Barnwal, K., Guliani, K. K., and Mehta, P. (2020). Universal Adversarial Perturbations: A Survey. *arXiv:2005.08087 [cs]*. arXiv: 2005.08087. 46

[Chen et al., 2018] Chen, P.-Y., Sharma, Y., Zhang, H., Yi, J., and Hsieh, C.-J. (2018). EAD: Elastic-Net Attacks to Deep Neural Networks via Adversarial Examples. *arXiv:1709.04114 [cs, stat]*. arXiv: 1709.04114. 41

[Chen, 2019] Chen, T. (2019). Adversarial Attack and Defense in Reinforcement Learning-From AI Security View. page 22. 48

[Cisse et al., 2017] Cisse, M., Bojanowski, P., Grave, E., Dauphin, Y., and Usunier, N. (2017). Parseval Networks: Improving Robustness to Adversarial Examples. In *International Conference on Machine Learning*, pages 854–863. 54, 72, 74

[Croce et al., 2019] Croce, F., Andriushchenko, M., and Hein, M. (2019). Provable Robustness of ReLU networks via Maximization of Linear Regions. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2057–2066. PMLR. ISSN: 2640-3498. 55, 56, 99, 101, 102, 108, 109

[Croce and Hein, 2018] Croce, F. and Hein, M. (2018). A randomized gradient-free attack on ReLU networks. *arXiv:1811.11493 [cs, stat]*. arXiv: 1811.11493. 41, 92, 93, 94, 99, 100, 107, 109

[Croce and Hein, 2019] Croce, F. and Hein, M. (2019). Sparse and Imperceivable Adversarial Attacks. pages 4724–4732. 41

[Croce et al., 2020] Croce, F., Rauber, J., and Hein, M. (2020). Scaling up the Randomized Gradient-Free Adversarial Attack Reveals Overestimation of Robustness Using Established Attacks. *International Journal of Computer Vision*, 128(4):1028–1046. 41, 92, 93, 94, 99, 100, 107, 109

[Cubuk et al., 2017] Cubuk, E. D., Zoph, B., Schoenholz, S. S., and Le, Q. V. (2017). Intriguing Properties of Adversarial Examples. *arXiv:1711.02846 [cs, stat]*. arXiv: 1711.02846. 40

[Das and Suganthan, 2011] Das, S. and Suganthan, P. N. (2011). Differential Evolution: A Survey of the State-of-the-Art. *IEEE Transactions on Evolutionary Computation*, 15(1):4–31. Conference Name: IEEE Transactions on Evolutionary Computation. 45

[Dechter and Pearl, 1985] Dechter, R. and Pearl, J. (1985). Generalized best-first search strategies and the optimality of A*. *Journal of the ACM*, 32(3):505–536. 18

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L., Kai Li, and Li Fei-Fei (2009). ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. ISSN: 1063-6919. 35

[Dhillon et al., 2018] Dhillon, G. S., Azizzadenesheli, K., Lipton, Z. C., Bernstein, J., Kossaifi, J., Khanna, A., and Anandkumar, A. (2018). Stochastic activation pruning for robust adversarial defense. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, pages undefined–undefined. 51, 53

[Dohmatob, 2019] Dohmatob, E. (2019). Generalized no free lunch theorem for adversarial robustness. In *International Conference on Machine Learning*, pages 1646–1654. PMLR. 62, 63

[Dong et al., 2018] Dong, Y., Liao, F., Pang, T., Su, H., Zhu, J., Hu, X., and Li, J. (2018). Boosting Adversarial Attacks with Momentum. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 9185–9193. 45

[Drucker and Lecun, 1992] Drucker, H. and Lecun, Y. (1992). Improving generalization performance using double backpropagation. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 3:991–7. 72, 74

[Dutta et al., 2017] Dutta, S., Jha, S., Sanakaranarayanan, S., and Tiwari, A. (2017). Output Range Analysis for Deep Neural Networks. *arXiv:1709.09130 [cs, stat]*. arXiv: 1709.09130. 59

[Ehlers, 2017] Ehlers, R. (2017). Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In D'Souza, D. and Narayan Kumar, K., editors, *Automated Technology for Verification and Analysis*, Lecture Notes in Computer Science, pages 269–286, Cham. Springer International Publishing. 59

[Engstrom et al., 2019a] Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Tran, B., and Madry, A. (2019a). Adversarial Robustness as a Prior for Learned Representations. *arXiv:1906.00945 [cs, stat]*. arXiv: 1906.00945. 39

[Engstrom et al., 2019b] Engstrom, L., Tran, B., Tsipras, D., Schmidt, L., and Madry, A. (2019b). Exploring the Landscape of Spatial Robustness. *arXiv:1712.02779 [cs, stat]*. arXiv: 1712.02779. 42

[Esteva et al., 2021] Esteva, A., Chou, K., Yeung, S., Naik, N., Madani, A., Mottaghi, A., Liu, Y., Topol, E., Dean, J., and Socher, R. (2021). Deep learning-enabled medical computer vision. *npj Digital Medicine*, 4(1):1–9. Number: 1 Publisher: Nature Publishing Group. 32, 33

[Eykholt et al., 2018] Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., and Song, D. (2018). Robust Physical-World Attacks on Deep Learning Models. *arXiv:1707.08945 [cs]*. arXiv: 1707.08945. 37

[Fang et al., 2019] Fang, X., Cao, G., Song, H., and Ouyang, Z. (2019). XGAN: adversarial attacks with GAN. In *2019 International Conference on Image and Video Processing, and Artificial Intelligence*, volume 11321, page 113211G. International Society for Optics and Photonics. 36

[Fawzi et al., 2016a] Fawzi, A., Fawzi, O., and Frossard, P. (2016a). Analysis of classifiers' robustness to adversarial perturbations. *arXiv:1502.02590 [cs, stat]*. arXiv: 1502.02590. 40

[Fawzi et al., 2016b] Fawzi, A., Moosavi-Dezfooli, S.-M., and Frossard, P. (2016b). Robustness of classifiers: from adversarial to random noise. *arXiv:1608.08967 [cs, stat]*. arXiv: 1608.08967. 40

[Finlayson et al., 2019] Finlayson, S. G., Bowers, J. D., Ito, J., Zittrain, J. L., Beam, A. L., and Kohane, I. S. (2019). Adversarial attacks on medical machine learning. *Science*, 363(6433):1287–1289. 33

[Geirhos et al., 2019] Geirhos, R., Rubisch, P., Michaelis, C., Bethge, M., Wichmann, F. A., and Brendel, W. (2019). ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. *arXiv:1811.12231 [cs, q-bio, stat]*. arXiv: 1811.12231. 39

[Goodfellow et al., 2017] Goodfellow, I., Papernot, N., Huang, S., Abbeel, P., Duan, R., and Clark, J. (2017). Attacking Machine Learning with Adversarial Examples. 36

[Goodfellow et al., 2014] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Networks. *arXiv:1406.2661 [cs, stat]*. arXiv: 1406.2661. 36

[Goodfellow et al., 2015] Goodfellow, I. J., Shlens, J., and Szegedy, C. (2015). Explaining and Harnessing Adversarial Examples. *International Conference on Learning Representation*, 1050:20. 32, 33, 38, 45, 53, 72, 73, 92, 93

[Goodfellow et al., 2013] Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). Maxout Networks. *arXiv:1302.4389 [cs, stat]*. arXiv: 1302.4389. 58

[Grigorescu et al., 2020] Grigorescu, S., Trasnea, B., Cocias, T., and Macesanu, G. (2020). A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21918. 34

[Grosse et al., 2017] Grosse, K., Papernot, N., Manoharan, P., Backes, M., and McDaniel, P. (2017). Adversarial Examples for Malware Detection. In Foley, S. N., Gollmann, D., and Snekkenes, E., editors, *Computer Security – ESORICS 2017*, Lecture Notes in Computer Science, pages 62–79, Cham. Springer International Publishing. 37

[Hanin and Rolnick, 2019] Hanin, B. and Rolnick, D. (2019). Deep ReLU Networks Have Surprisingly Few Activation Patterns. *arXiv:1906.00904 [cs, math, stat]*. arXiv: 1906.00904. 105

[Hashimoto et al., 2018] Hashimoto, D. A., Rosman, G., Rus, D., and Meireles, O. R. (2018). Artificial Intelligence in Surgery: Promises and Perils. *Annals of surgery*, 268(1):70–76. 32

[Hein and Andriushchenko, 2017] Hein, M. and Andriushchenko, M. (2017). Formal guarantees on the robustness of a classifier against adversarial manipulation. In *Advances in Neural Information Processing Systems*, pages 2266–2276. 72

[Hinton et al., 2015] Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the Knowledge in a Neural Network. *arXiv:1503.02531 [cs, stat]*. arXiv: 1503.02531. 52

[Hochreiter and Schmidhuber, 1995] Hochreiter, S. and Schmidhuber, J. (1995). SIMPLIFYING NEURAL NETS BY DISCOVERING FLAT MINIMA. pages 529–536. 72

[Holcomb et al., 2018] Holcomb, S. D., Porter, W. K., Ault, S. V., Mao, G., and Wang, J. (2018). Overview on DeepMind and Its AlphaGo Zero AI. In *Proceedings of the 2018 International Conference on Big Data and Education*, ICBDE '18, pages 67–71, New York, NY, USA. Association for Computing Machinery. 48

[Horst and Thoai, 1999] Horst, R. and Thoai, N. V. (1999). DC Programming: Overview. *Journal of Optimization Theory and Applications*, 103(1):1–43. 28

[Huang et al., 2017] Huang, S., Papernot, N., Goodfellow, I., Duan, Y., and Abbeel, P. (2017). Adversarial Attacks on Neural Network Policies. *arXiv:1702.02284 [cs, stat]*. arXiv: 1702.02284. 48

[Huang et al., 2020] Huang, X., Kroening, D., Ruan, W., Sharp, J., Sun, Y., Thamo, E., Wu, M., and Yi, X. (2020). A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review*, 37:100270. 61

[Ilyas et al., 2019] Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., and Madry, A. (2019). Adversarial Examples Are Not Bugs, They Are Features. In Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F. d., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 125–136. Curran Associates, Inc. 37, 39, 51

[Javanmard et al., 2020] Javanmard, A., Soltanolkotabi, M., and Hassani, H. (2020). Precise Tradeoffs in Adversarial Training for Linear Regression. In *Conference on Learning Theory*, pages 2034–2078. PMLR. ISSN: 2640-3498. 62

[Julian et al., 2016] Julian, K. D., Lopez, J., Brush, J. S., Owen, M. P., and Kochenderfer, M. J. (2016). Policy compression for aircraft collision avoidance systems. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pages 1–10. ISSN: 2155-7209. 59

[Katz et al., 2017] Katz, G., Barrett, C., Dill, D., Julian, K., and Kochenderfer, M. (2017). Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. *arXiv:1702.01135 [cs]*. arXiv: 1702.01135. 22, 34, 57, 59, 60, 92, 93

[Katz et al., 2019] Katz, G., Huang, D. A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., Dill, D. L., Kochenderfer, M. J., and Barrett, C. (2019). The Marabou Framework for Verification and Analysis of Deep Neural Networks. In Dillig, I. and Tasiran, S., editors, *Computer Aided Verification*, Lecture Notes in Computer Science, pages 443–452, Cham. Springer International Publishing. 64

[Kazemi et al., 2020] Kazemi, E., Kerdreux, T., and Wang, L. (2020). Trace-Norm Adversarial Examples. *arXiv:2007.01855 [cs, stat]*. arXiv: 2007.01855. 42

[Koh et al., 2018] Koh, P. W., Steinhardt, J., and Liang, P. (2018). Stronger Data Poisoning Attacks Break Data Sanitization Defenses. *arXiv:1811.00741 [cs, stat]*. arXiv: 1811.00741. 44

[Kos et al., 2017] Kos, J., Fischer, I., and Song, D. (2017). Adversarial examples for generative models. *arXiv:1702.06832 [cs, stat]*. arXiv: 1702.06832. 48

[Krizhevsky et al., 2017] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90. 32, 35

[Krotov and Hopfield, 2016] Krotov, D. and Hopfield, J. J. (2016). Dense Associative Memory for Pattern Recognition. *arXiv:1606.01164 [cond-mat, q-bio, stat]*. arXiv: 1606.01164. 38

[Krotov and Hopfield, 2018] Krotov, D. and Hopfield, J. J. (2018). Dense Associative Memory is Robust to Adversarial Inputs. *Neural Computation*, 30(12):3151–3167. arXiv: 1701.00939. 38, 39, 51

[Kurakin et al., 2017] Kurakin, A., Goodfellow, I., and Bengio, S. (2017). Adversarial examples in the physical world. *arXiv:1607.02533 [cs, stat]*. arXiv: 1607.02533. 45, 99

[Laidlaw and Feizi, 2019] Laidlaw, C. and Feizi, S. (2019). Functional Adversarial Attacks. page 11. 37

[Li et al., 2019] Li, J., Ji, S., Du, T., Li, B., and Wang, T. (2019). TextBugger: Generating Adversarial Text Against Real-world Applications. *Proceedings 2019 Network and Distributed System Security Symposium*. arXiv: 1812.05271. 65

[Liu et al., 2017] Liu, Y., Ma, S., Aafer, Y., Lee, W.-C., Zhai, J., Wang, W., and Zhang, X. (2017). Trojaning Attack on Neural Networks. page 17. 47

[Long et al., 2015] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully Convolutional Networks for Semantic Segmentation. *arXiv:1411.4038 [cs]*. arXiv: 1411.4038. 50

[Madry et al., 2019] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2019). Towards Deep Learning Models Resistant to Adversarial Attacks. *arXiv:1706.06083 [cs, stat]*. arXiv: 1706.06083. 41, 45, 92, 93

[Moosavi-Dezfooli et al., 2017a] Moosavi-Dezfooli, S.-M., Fawzi, A., Fawzi, O., and Frossard, P. (2017a). Universal adversarial perturbations. *arXiv:1610.08401 [cs, stat]*. arXiv: 1610.08401. 43, 46, 54

[Moosavi-Dezfooli et al., 2017b] Moosavi-Dezfooli, S.-M., Fawzi, A., Fawzi, O., Frossard, P., and Soatto, S. (2017b). Analysis of universal adversarial perturbations. *arXiv:1705.09554 [cs, stat]*. arXiv: 1705.09554. 40

[Moosavi-Dezfooli et al., 2016] Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. (2016). DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. pages 2574–2582. 46

[Morrison et al., 2016] Morrison, D. R., Jacobson, S. H., Sauppe, J. J., and Sewell, E. C. (2016). Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102. 16, 18, 20

[Nassi et al., 2020] Nassi, B., Mirsky, Y., Nassi, D., Ben-Netanel, R., Drokin, O., and Elovici, Y. (2020). Phantom of the ADAS: Securing Advanced Driver-Assistance Systems from Split-Second Phantom Attacks. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, pages 293–308, New York, NY, USA. Association for Computing Machinery. 34

[Nguyen et al., 2019] Nguyen, G., Dlugolinsky, S., Bobak, M., Tran, V., Lopez Garcia, A., Heredia, I., Malik, P., and Hluchy, L. (2019). Machine Learning and Deep Learning frameworks and libraries for large-scale data mining: a survey. *Artificial Intelligence Review*, 52(1):77–124. 63

[Niu, 2010] Niu, Y. S. (2010). *Programmation DC et DCA en optimisation combinatoire et optimisation polynomiale via les techniques de SDP : codes et simulations numériques*. These de doctorat, Rouen, INSA. 28, 82

[Niu and Pham Dinh, 2008] Niu, Y.-S. and Pham Dinh, T. (2008). A DC Programming Approach for Mixed-Integer Linear Programs. In Le Thi, H. A., Bouvry, P., and Pham Dinh, T., editors, *Modelling, Computation and Optimization in Information Systems and Management Sciences*, Communications in Computer and Information Science, pages 244–253. Springer Berlin Heidelberg. 28

[Papernot et al., 2016a] Papernot, N., McDaniel, P., and Goodfellow, I. (2016a). Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples. *arXiv:1605.07277 [cs]*. arXiv: 1605.07277. 37

[Papernot et al., 2016b] Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., and Swami, A. (2016b). The Limitations of Deep Learning in Adversarial Settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 372–387. 41, 45

[Papernot et al., 2016c] Papernot, N., McDaniel, P., Wu, X., Jha, S., and Swami, A. (2016c). Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks. *arXiv:1511.04508 [cs, stat]*. arXiv: 1511.04508. 34, 46, 51, 52, 72, 76, 92

[Pinot et al., 2019] Pinot, R., Meunier, L., Araujo, A., Kashima, H., Yger, F., Gouy-Pailler, C., and Atif, J. (2019). Theoretical evidence for adversarial robustness through randomization. In *Advances in Neural Information Processing Systems*, pages 11838–11848. 62

[Qayyum et al., 2020] Qayyum, A., Usama, M., Qadir, J., and Al-Fuqaha, A. (2020). Securing Connected Autonomous Vehicles: Challenges Posed by Adversarial Machine Learning and the Way Forward. *IEEE Communications Surveys Tutorials*, 22(2):998–1026. Conference Name: IEEE Communications Surveys Tutorials. 34

[Raghu et al., 2017] Raghu, M., Poole, B., Kleinberg, J., Ganguli, S., and Sohl-Dickstein, J. (2017). On the Expressive Power of Deep Neural Networks. In *International Conference on Machine Learning*, pages 2847–2854. PMLR. ISSN: 2640-3498. 95

[Raghunathan et al., 2020] Raghunathan, A., Steinhardt, J., and Liang, P. (2020). Certified Defenses against Adversarial Examples. *arXiv:1801.09344 [cs]*. arXiv: 1801.09344. 51, 55

[Rauber et al., 2018] Rauber, J., Brendel, W., and Bethge, M. (2018). Foolbox: A Python toolbox to benchmark the robustness of machine learning models. *arXiv:1707.04131 [cs, stat]*. arXiv: 1707.04131. 64

[Rauber et al., 2020] Rauber, J., Zimmermann, R., Bethge, M., and Brendel, W. (2020). Foolbox Native: Fast adversarial attacks to benchmark the robustness of machine learning models in PyTorch, TensorFlow, and JAX. *Journal of Open Source Software*, 5(53):2607. 64

[Ren et al., 2016] Ren, S., He, K., Girshick, R., and Sun, J. (2016). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *arXiv:1506.01497 [cs]*. arXiv: 1506.01497. 50

[Rozsa et al., 2018] Rozsa, A., Gunther, M., and Boult, T. E. (2018). Towards Robust Deep Neural Networks with BANG. *arXiv:1612.00138 [cs]*. arXiv: 1612.00138. 39

[Salman et al., 2019] Salman, H., Yang, G., Zhang, H., Hsieh, C.-J., and Zhang, P. (2019). A Convex Relaxation Barrier to Tight Robustness Verification of Neural Networks. In Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F. d., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 9832–9842. Curran Associates, Inc. 60, 102, 107, 108, 109

[Seck et al., 2019] Seck, I., Loosli, G., and Canu, S. (2019). L 1-norm double backpropagation adversarial defense. In *ESANN-European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. 69

[Sen et al., 2019] Sen, A., Zhu, X., Marshall, L., and Nowak, R. (2019). Should Adversarial Attacks Use Pixel p-Norm? *arXiv:1906.02439 [cs, stat]*. arXiv: 1906.02439. 40

[Sengupta et al., 2020] Sengupta, S., Basak, S., Saikia, P., Paul, S., Tsalavoutis, V., Atiah, F., Ravi, V., and Peters, A. (2020). A review of deep learning with special emphasis on architectures, applications and recent trends. *Knowledge-Based Systems*, 194:105596. 32

[Serra et al., 2018] Serra, T., Tjandraatmadja, C., and Ramalingam, S. (2018). Bounding and Counting Linear Regions of Deep Neural Networks. In *International Conference on Machine Learning*, pages 4558–4566. PMLR. ISSN: 2640-3498. 98

[Shafahi et al., 2018] Shafahi, A., Huang, W. R., Studer, C., Feizi, S., and Goldstein, T. (2018). Are adversarial examples inevitable? 62

[Shafiee et al., 2020] Shafiee, M. J., Jeddi, A., Nazemi, A., Fieguth, P., and Wong, A. (2020). Deep Neural Network Perception Models and Robust Autonomous Driving Systems. *arXiv:2003.08756 [cs, stat]*. arXiv: 2003.08756. 32

[Shaham et al., 2018] Shaham, U., Yamada, Y., and Negahban, S. (2018). Understanding Adversarial Training: Increasing Local Stability of Neural Nets through Robust Optimization. *Neurocomputing*, 307:195–204. arXiv: 1511.05432. 53

[Sharif et al., 2016] Sharif, M., Bhagavatula, S., Bauer, L., and Reiter, M. K. (2016). Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1528–1540, New York, NY, USA. Association for Computing Machinery. 37, 47

[Silva and Najafirad, 2020] Silva, S. H. and Najafirad, P. (2020). Opportunities and Challenges in Deep Learning Adversarial Robustness: A Survey. *arXiv:2007.00753 [cs, stat]*. arXiv: 2007.00753. 50

[Simon-Gabriel et al., 2019] Simon-Gabriel, C.-J., Ollivier, Y., Bottou, L., Schölkopf, B., and Lopez-Paz, D. (2019). First-order Adversarial Vulnerability of Neural Networks and Input Dimension. *arXiv:1802.01421 [cs, stat]*. arXiv: 1802.01421. 38, 72

[Singh et al., 2019] Singh, G., Gehr, T., Püschel, M., and Vechev, M. (2019). An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–30. vii, 59, 60, 61

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958. 53

[Su et al., 2019] Su, J., Vargas, D. V., and Sakurai, K. (2019). One Pixel Attack for Fooling Deep Neural Networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841. Conference Name: IEEE Transactions on Evolutionary Computation. 40, 45

[Suzuki, 2017] Suzuki, K. (2017). Overview of deep learning in medical imaging. *Radiological Physics and Technology*, 10(3):257–273. 32

[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. *arXiv:1312.6199 [cs]*. arXiv: 1312.6199. 35, 44, 46, 72

[Tabacof et al., 2016] Tabacof, P., Tavares, J., and Valle, E. (2016). Adversarial Images for Variational Autoencoders. *arXiv:1612.00155 [cs]*. arXiv: 1612.00155. 48

[Tabacof and Valle, 2016] Tabacof, P. and Valle, E. (2016). Exploring the Space of Adversarial Images. *arXiv:1510.05328 [cs]*. arXiv: 1510.05328. 40

[Tanay and Griffin, 2016] Tanay, T. and Griffin, L. (2016). A Boundary Tilting Persepective on the Phenomenon of Adversarial Examples. *arXiv:1608.07690 [cs, stat]*. arXiv: 1608.07690. 38

[Tjeng et al., 2018] Tjeng, V., Xiao, K. Y., and Tedrake, R. (2018). Evaluating Robustness of Neural Networks with Mixed Integer Programming. 34, 57, 59, 64, 92, 93, 97, 103, 105, 107

[Tramer et al., 2020] Tramer, F., Carlini, N., Brendel, W., and Madry, A. (2020). On Adaptive Attacks to Adversarial Example Defenses. *arXiv:2002.08347 [cs, stat]*. arXiv: 2002.08347. 43

[Tramèr et al., 2020] Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I., Boneh, D., and McDaniel, P. (2020). Ensemble Adversarial Training: Attacks and Defenses. *arXiv:1705.07204 [cs, stat]*. arXiv: 1705.07204. 45

[Tramèr et al., 2017] Tramèr, F., Papernot, N., Goodfellow, I., Boneh, D., and McDaniel, P. (2017). The Space of Transferable Adversarial Examples. *arXiv:1704.03453 [cs, stat]*. arXiv: 1704.03453. 40

[Tsiligkaridis and Roberts, 2021] Tsiligkaridis, T. and Roberts, J. (2021). Understanding Frank-Wolfe Adversarial Training. *arXiv:2012.12368 [cs]*. arXiv: 2012.12368. 53

[Tsipras et al., 2019] Tsipras, D., Santurkar, S., Engstrom, L., Turner, A., and Madry, A. (2019). Robustness May Be at Odds with Accuracy. *International Conference on Learning Representations*. arXiv: 1805.12152. 62

[van Hasselt et al., 2015] van Hasselt, H., Guez, A., and Silver, D. (2015). Deep Reinforcement Learning with Double Q-learning. *arXiv:1509.06461 [cs]*. arXiv: 1509.06461. 53

[Vechev, 2020] Vechev, M. (2020). Reliable and Interpretable Artificial Intelligence – Lecture 4b (Certification of Neural Networks). 56

[Wang et al., 2021] Wang, W., Wang, R., Wang, L., Wang, Z., and Ye, A. (2021). Towards a Robust Deep Neural Network in Texts: A Survey. *arXiv:1902.07285 [cs]*. arXiv: 1902.07285. 48

[Wang and Bovik, 2009] Wang, Z. and Bovik, A. C. (2009). Mean squared error: Love it or leave it? A new look at Signal Fidelity Measures. *IEEE Signal Processing Magazine*, 26(1):98–117. Conference Name: IEEE Signal Processing Magazine. 41

[Wolsey and Nemhauser, 1999] Wolsey, L. A. and Nemhauser, G. L. (1999). *Integer and Combinatorial Optimization*. John Wiley & Sons. Google-Books-ID: vvm4DwAAQBAJ. 14, 18, 26, 58, 94

[Wong and Kolter, 2018] Wong, E. and Kolter, Z. (2018). Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope. In *International Conference on Machine Learning*, pages 5283–5292. 51, 55, 57, 60, 102, 103

[Wong et al., 2019] Wong, E., Schmidt, F., and Kolter, Z. (2019). Wasserstein Adversarial Examples via Projected Sinkhorn Iterations. In *International Conference on Machine Learning*, pages 6808–6817. PMLR. ISSN: 2640-3498. 40, 42

[Wong et al., 2018] Wong, E., Schmidt, F., Metzen, J. H., and Kolter, J. Z. (2018). Scaling provable adversarial defenses. In *Advances in Neural Information Processing Systems*, pages 8400–8409. 55

[Xiao et al., 2019a] Xiao, C., Li, B., Zhu, J.-Y., He, W., Liu, M., and Song, D. (2019a). Generating Adversarial Examples with Adversarial Networks. *arXiv:1801.02610 [cs, stat]*. arXiv: 1801.02610. 47

[Xiao et al., 2019b] Xiao, C., Zhong, P., and Zheng, C. (2019b). Enhancing Adversarial Defense by k-Winners-Take-All. 51, 52

[Xie et al., 2017] Xie, C., Wang, J., Zhang, Z., Zhou, Y., Xie, L., and Yuille, A. (2017). Adversarial Examples for Semantic Segmentation and Object Detection. *arXiv:1703.08603 [cs]*. arXiv: 1703.08603. 48, 50

[Yang et al., 2021] Yang, B., Xu, K., Wang, H., and Zhang, H. (2021). Random Transformation of Image Brightness for Adversarial Attack. *arXiv:2101.04321 [cs]*. arXiv: 2101.04321. 42

[Yin et al., 2019] Yin, D., Kannan, R., and Bartlett, P. (2019). Rademacher Complexity for Adversarially Robust Generalization. In *International Conference on Machine Learning*, pages 7085–7094. PMLR. ISSN: 2640-3498. 62

[Zhang et al., 2018] Zhang, H., Weng, T.-W., Chen, P.-Y., Hsieh, C.-J., and Daniel, L. (2018). Efficient Neural Network Robustness Certification with General Activation Functions. page 10. 60