Università degli Studi di Cagliari

PhD DEGREE

Electronic and Computer Engineering

Cycle XXXIV

TITLE OF THE PhD THESIS

Towards Debugging and Improving Adversarial Robustness Evaluations

Scientific and Disciplinary Sector(s)

ING-INF/05

PhD Student: Maura Pintor

Supervisors Dr. Fabio Roli
Dr. Battista Biggio

Final exam. Academic Year 2020/2021
Thesis defence: February 2022

Abstract

Despite exhibiting unprecedented success in many application domains, machine-learning models have been shown to be vulnerable to adversarial examples, i.e., maliciously perturbed inputs that are able to subvert their predictions at test time. Rigorous testing against such perturbations requires enumerating all possible outputs for all possible inputs, and despite impressive results in this field, these methods remain still difficult to scale to modern deep learning systems. For these reasons, empirical methods are often used. These adversarial perturbations are optimized via gradient descent, minimizing a loss function that aims to increase the probability of misleading the model's predictions. To understand the sensitivity of the model to such attacks, and to counter the effects, machine-learning model designers craft worst-case adversarial perturbations and test them against the model they are evaluating. However, many of the proposed defenses have been shown to provide a false sense of security due to failures of the attacks, rather than actual improvements in the machine-learning models' robustness. They have been broken indeed under more rigorous evaluations. Although guidelines and best practices have been suggested to improve current adversarial robustness evaluations, the lack of automatic testing and debugging tools makes it difficult to apply these recommendations in a systematic and automated manner. To this end, we tackle three different challenges: (1) we investigate how adversarial robustness evaluations can be performed efficiently, by proposing a novel attack that can be used to find minimum-norm adversarial perturbations; (2) we propose a framework for debugging adversarial robustness evaluations, by defining metrics that reveal faulty evaluations as well as mitigations to patch the detected problems; and (3) we show how to employ a surrogate model for improving the success of transfer-based attacks, that are useful when gradient-based attacks are failing due to problems in the gradient information.

To improve the quality of robustness evaluations, we propose a novel attack, referred to as Fast Minimum-Norm (FMN) attack, which competes with state-of-the-art attacks in terms of quality of the solution while outperforming them in terms of computational complexity and robustness to sub-optimal configurations of the attack hyperparameters. These are all desirable characteristics of attacks used in robustness evaluations, as the aforementioned problems often arise from the use of sub-optimal attack hyperparameters, including, e.g., the number of attack iterations, the step size, and the use of an inappropriate loss function. The correct

refinement of these variables is often neglected, hence we designed a novel framework that helps debug the optimization process of adversarial examples, by means of quantitative indicators that unveil common problems and failures during the attack optimization process, e.g., in the configuration of the hyperparameters. Using such indicators, specific mitigation strategies can help avoid the common pitfalls encountered in state-of-the-art adversarial robustness evaluations of machine-learning models, providing a first concrete step towards improving the reliability of such evaluation processes and designing more trustworthy machine-learning technologies. Unfortunately, even a careful analysis with gradient-based attacks is sometimes not sufficient to prove that a defense is robust. Commonly-accepted best practices suggest further validating the target model with alternative strategies, among which is the usage of a surrogate model to craft the adversarial examples to transfer to the model being evaluated is useful to check for gradient obfuscation. However, how to effectively create transferable adversarial examples is not an easy process, as many factors influence the success of this strategy. In the context of this research, we utilize a first-order model to show what are the main underlying phenomena that affect transferability and suggest best practices to create adversarial examples that transfer well to the target models.

Contents

List of Figures

List of Tables

Symbols

$\boldsymbol{x}$	input sample
y	sample label
$\boldsymbol{x}_{\text{lb}}$	lower bound of the feature space
$\boldsymbol{x}_{\text{ub}}$	upper bound of the feature space
$\hat{y}$	predicted label
f	decision function
f_c	output score of class c
z_c	probability score of class c
$\boldsymbol{\theta}$	model parameters
$\mathcal{D}_{train}$	training data
$\mathcal{D}_{test}$	testing data
$\boldsymbol{\delta}$	perturbation
$\|\| \cdot \|\|_p$	ℓ_p norm of a vector
$\mathcal{L}$	loss function
α	step size
$\mathcal{A}$	accuracy
$\nabla_{\boldsymbol{x}} f$	gradient of function f w.r.t. $\boldsymbol{x}$

Chapter 1

Introduction

The broad use of Machine Learning in our everyday life has reached the point in which we do not notice anymore that it is there. Machine Learning is indeed not particularly new, in fact being around since the 1960s. What is new is the applications. We see machine learning operate successfully in the most complex tasks, sometimes even impossible for humans. For example, in the cybersecurity domain, it is often machine learning that does the job, due to the massive amount of data to process. It is exactly this use of machine learning in increasingly safety-critical applications that motivates the research in Adversarial Machine Learning. In a nutshell, this study analyzes how machine learning can be broken and how to prevent that from happening. The first step is understanding what is the impact of the attacks on the machine learning model. Of course, such impact depends on the application and there is not much control on that, other than limiting the effects of a possible attack. In order to understand the risk deriving from adversarial attacks, machine-learning system designers should assess the sensitivity of the model against such attacks, by crafting them and testing the performances of the models against them. This process requires creating adversarial perturbations that target the worst-case scenarios for the specific model, hence it is often necessary to have deep knowledge of the possible breaking points of the model (see Figure 1.1), and some expertise in how to target them appropriately.

Once it has been established what is the potential attacker's entry point, it is important to assess what is the sensitivity of the model to the adversarial attacks that it might be exposed to. In this thesis, we focus on a particular case of models, namely classification models, and of attacks, namely the evasion attacks. The first evasion attacks date back to 2004, but they became popular around 2013 when they were crafted against deep networks. These attacks specifically target the ability of the model to correctly classify an input sample at test time by applying an imperceptible perturbation to it. This perturbation, despite being very small, causes a strong reaction to the model that is misguided and misclassifies the sample. One example of this is the popular paper where a face recognition algorithm was fooled with a physically-realizable attack through the use of a pair of adversarial glasses. With the

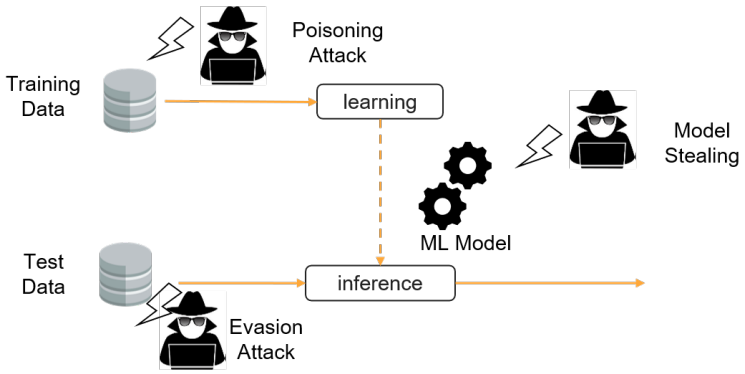


Figure 1.1: Possible entry points of the attacker that targets a classification model.

accessory on, the person could be recognized as a different individual, evading the recognition system. Now, we move to safety-critical applications. What if, instead, your antivirus checks a software that you just downloaded as safe, instead of recognizing it as a virus? This is the reason why we need to understand how sensitive is our model to adversarial evasion attacks. The approach that is often used in these cases is to create a lot of adversarial examples for different levels of perturbation, and then check what is the performance of the model on those perturbed inputs. Techniques such as formal verification, i.e. ensuring consistency of the model on all possible inputs, can provide some robustness guarantees, but unfortunately, they are not suitable to scale well on modern machine learning systems. For this reason, we have to rely on empirical approaches, creating adversarial examples with more practical search techniques. This can be a very time-consuming process, as these adversarial attacks require defining an optimization problem to solve for finding the perturbations and the algorithms for solving such optimization. The selection of both these aspects cannot be taken lightly. There is a wide range of algorithms that attempt to find an adversarial perturbation. Technically speaking, the space where we are looking for adversarial examples is big, in fact, imagine trying to search among all the possible variations of an image and finding the worst case. Here comes to the rescue the gradient-based optimization. For finding adversarial attacks, we first define a loss function that depends on the parameters of the model and on the input that we pass to it. Then, if we have a starting point, we can follow the direction of the gradient by changing the input, i.e., we exploit the direction for which the loss changes the most, and we can travel along the loss landscape and reach optimal points, likely adversarial examples. This is often accompanied by a requirement on the maximum-norm of the perturbation, which enforces a trade-off between the adversarial confidence of the resulting classification and how much perturbation we are adding. All this process adds up to the security evaluation curve, i.e., a complete evaluation of the model's performances as the perturbation level increases. Each point of the security evaluation curve constitutes the robust accuracy of the model under the given perturbation bound.

Once everything is set up, we can craft our adversarial attacks. What is usually done in this context is to take a set of algorithms and run them on the testing set. Then, the designer should select among all the results the worst cases for that model and for the threat model they selected. Then we will have the evaluation of the security of that model. What to do when the model is not robust at all? For standard training models, i.e., models with no particular defense in place, the robust accuracy drops pretty quickly. In these cases, we can try to limit the effect of worst-case perturbation by considering a defense mechanism. But what should we do when the model shows instead to be robust to the adversarial perturbation we create? In these cases, there are two possibilities. The first is that there are no adversarial examples in the region considered, and we did a good job evaluating them. The second possibility is, unfortunately, more common: we just did not find adversarial examples, but they might still be there. When we create a defense, we might actually be breaking the mechanism that we use for creating attacks, rather than actually making the model robust. There is a long history of the never-ending battle between who publishes a defense and who shows that it can be broken with a more careful evaluation. The key problem is that although the presence of guidelines, there is no systematic way to apply them.
These defenses have all been published, and soon broken, because of repeating patterns that have been neglected in the past, but are also constantly overlooked in the state-of-the-art. Some of them were just evaluated lightly, with incomplete optimization or wrong attack configurations. Many of these evaluations, however, suffer from a more specific problem, *gradient obfuscation*, i.e., the defense mechanism makes the information contained in the input gradients useless. It is important to understand here that hiding the adversarial examples by making them difficult to find is not the same as removing them by learning more robust decision functions or detecting/rejecting the malicious inputs. In these cases, it is just a matter of time before someone finds out a way to create adversarial perturbations that work. Other problems might arise also for wrong assumptions or wrong implementation of the attacks. Understanding what is the correct attack strategy is definitely a more difficult problem than the one of creating adversarial attacks *per se*, and the absence of easy-to-use debugging tools makes it even more complicated.
Attacking a model with only one strategy is also not the best way of understanding its robustness to a wide range of potentially unknown attacks. Current guidelines suggest complementing the gradient-based evaluation with sanity checks, which often reveal the presence of problems in the security evaluation process itself, rather than in the model. For example, in Tramer et al. (2020), 13 defenses were evaluated and broken by reversing the mechanism and adapting the attack strategy to target the specific model. Optimization attacks are by far among the most powerful algorithms, however, they tend to fail when applied blindly to models that do not consider aspects of the implemented defense. Sanity checks are almost always helpful to understand if the model is being attacked in the wrong way. This is when we should move to gradient-free attacks and transfer attacks. Gradient-free attacks

try to optimize the loss function just by exploration of the landscape and inspection of the answers returned by the model. By utilizing transferability instead, one can optimize a loss landscape that behaves a lot better than the one of the original model, and then attempt to transfer the attack on the target model. If this attack succeeds, that can be a clear indication of obfuscated gradients, as the adversarial examples are proved to be still there. Unfortunately, crafting transferable adversarial examples is still a complicated process that requires a lot of effort to succeed, especially without the help of guidelines on how to choose the target model or the right attack algorithm.
In general, evaluating adversarial robustness is a key aspect of designing machine-learning algorithms that have to operate in security-critical applications or environments. The process of creating adversarial attacks and testing the model against them involves many steps that have to be taken carefully and require regard to many aspects that are not easy to understand and apply by practitioners.

1.1 Thesis Statement

In this thesis, we aim to overcome some of the aforementioned problems that affect the process of evaluating adversarial robustness by improving the main time-consuming and difficult-to-apply steps.

1. we want to improve the process of searching for adversarial examples, by making it more efficient and less difficult to configure;

2. we want to introduce a framework that allows debugging the creation of adversarial examples, by creating specific indicators to reveal failures and proposing mitigations to the identified problems, including a novel metric for detecting when a defense is obfuscating the gradients; and

3. we want to provide additional tools to test the robustness of a model, by proposing novel metrics to assess the intrinsic vulnerability of a model, and its sensitivity to transferable adversarial examples created with a surrogate model.

Improving the efficiency of the evaluations. One of the problems is that evaluating a model's robustness takes time. It takes time to run the attacks, as computing the gradient direction requires computing the gradient, and this is even worsened by the difficult search for the perfect hyperparameters (i.e. variables that have to be chosen by the designer) that have to be tuned in order to achieve good enough solutions. On top of that, if we want to compute the drop in the model's performance when attacked with different perturbation strengths, we need to compute all the adversarial examples, with their respective constraints, for every value tested. We aim to improve this aspect of the robustness evaluation by proposing a novel attack

algorithm for finding the minimum perturbation required to reach the adversarial class. The main advantage of this approach is that it does not have to be restarted for every value of the perturbation size threshold, as a single run allows to set the evaluation threshold and find out immediately how many adversarial examples are found within a given constraint in the norm of perturbation. This allows to greatly reduce the computational cost of the complete evaluation. Second, with a single algorithm, it is possible to evaluate under 4 different perturbation models, i.e., different ℓ_p norms, and with two different scenarios, namely *targeted* and *untargeted*. To reduce the time spent in crafting the attacks, it reaches fast convergence with a few lightweight steps, and to reduce the time spent in finding the best configuration it is robust to the hyperparameter choice.

Providing a framework for the optimization, debugging, and visualization of adversarial attacks. It is easy to understand that the approach used in security evaluations is very empirical and it can go incredibly wrong as we don't have guarantees on how good is the optimum that we found with our algorithm. Adversarial attacks are difficult to configure and debug: if the search is not initialized well, or conducted well, or reported well, the algorithms fail. The even bigger problem is that they fail in the same way even when the actual adversarial example is not there, and there is no way to understand the difference, unless by delving deeply into the concepts of optimization, loss functions, and adaptive evaluations. This research work provides a systematic framework for debugging optimization-based adversarial attacks and making sure that the misconfiguration problems, as well as the misconceptions about the model, are detected and fixed.

Providing guidelines on how to evaluate the model beyond gradient-based attacks. The problem is only half-solved by conducting a thorough security evaluation. Then, if gradient-based optimization fails, we cannot be certain that the attacks are not suffering from other problems, e.g., gradient obfuscation. One very simple check is to create alternative adversarial examples by optimizing the perturbation on a similar, but different model, and then submitting the result to the model under evaluation. If this *transfer test* fails, hence the adversarial attacks do not succeed upon transfer, we are still in the same situation. We cannot tell if the adversarial attacks are failing because they are created in the wrong way, or because the model is actually robust. For this reason, in order to improve the quality of the attacks being created, we felt the need of finding out what are the underlying factors that affect transferability. We provide an extensive evaluation of these phenomena and suggest guidelines on how to create transferable adversarial attacks.

1.2 Outline of the Thesis

In this thesis, first, we give some background and context information (Chapter 2), then we provide an overview of the state of the art and its limitations (Chapter 3),

present our contributions (Chapter 4) along with the experimental analysis (Chapter 5), and finally discuss the main results, conclusions and future directions (Chapter 6).

Significance and advantage. We believe this work will facilitate reliable robustness evaluations, and we hope that the field of adversarial robustness will benefit from the insight presented in this work towards designing more robust machine learning algorithms.

1.3 List of Publications

This thesis is based on the following publications:

- **M. Pintor**, F. Roli, W. Brendel, and B. Biggio. Fast minimum-norm adversarial attacks through adaptive norm constraints. In *Thirty-fifth Conference on Neural Information Processing Systems (NeurIPS 2021)*, 2021;

- **M. Pintor**, L. Demetrio, G. Manca, B. Biggio, and F. Roli. Slope: A first-order approach for measuring gradient obfuscation. In *Proceedings of the ESANN, 29th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2021)*, 2021;

- **M. Pintor**, L. Demetrio, A. Sotgiu, G. Manca, A. Demontis, N. Carlini, B. Biggio, and F. Roli. Indicators of attack failure: Debugging and improving optimization of adversarial examples, *arXiv preprint*, 2021;

- A. Demontis, M. Melis, **M. Pintor**, M. Jagielski, B. Biggio, A. Oprea, C. Nita-Rotaru, and F. Roli. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In *28th USENIX Security Symposium (USENIX Security 19)*, 2019.

Other works carried out during the Ph.D. course, that are not included in this thesis:

- U. Ozbulak, **M. Pintor**, A. Van Messem, W. De Neve. Evaluating Adversarial Attacks on ImageNet: A Reality Check on Misclassification Classes. In *NeurIPS 2021 Workshop on ImageNet: Past, Present, and Future*, 2021.

- Y. Zheng, X. Feng, Z. Xia, X. Jiang, A. Demontis, **M. Pintor**, B. Biggio, F. Roli. Why adversarial reprogramming works, when it fails, and how to tell the difference.arXiv preprint arXiv:2108.11673, 2021.

- G. Orrù, D. Ghiani, **M. Pintor**, G. L. Marcialis, F. Roli. Detecting anomalies from video-sequences: A novel descriptor. In *25th international conference on pattern recognition (ICPR 2020)*, 2020.

- P. Meloni, D. Loi, G. Deriu, A. D. Pimentel, D. Sapra, **M. Pintor**, B. Biggio, O. Ripolles, D. Solans, F. Conti, L. Benini, T. Stefanov, S. Minakova, B. Moser, N. Shepeleva, M. Masin, F. Palumbo, N. Fragoulis, I. Theodorakopoulos. Architecture-aware design and implementation of CNN algorithms for embedded inference: The aloha project. In *2018 30th international conference on microelectronics (ICM)*.

- P. Meloni, D. Loi, G. Deriu, A. D. Pimentel, D. Sapra, B. Moser, N. Shepeleva, F. Conti, L. Benini, O. Ripolles, D. Solans, **M. Pintor**, B. Biggio, T. Stefanov, S. Minakova, N. Fragoulis, I. Theodorakopoulos M. Masin, F. Palumbo. Aloha: An architectural-aware framework for deep learning at the edge. In *Proceedings of the workshop on intelligent embedded systems architectures and applications*.

- P. Meloni, D. Loi, P. Busia, G. Deriu, A. D. Pimentel, D. Sapra, T. Stefanov, S. Minakova, F. Conti, L. Benini, **M. Pintor**, B. Biggio, B. Moser, N. Shepeleva, N. Fragoulis, I. Theodorakopoulos, M. Masin, F. Palumbo. Optimization and deployment of CNNs at the edge: The aloha experience. In Proceedings of the 16th ACM international conference on Computing Frontiers.

Chapter 2

Background Concepts

Nowadays machine learning is used for the most complicated tasks, included in industrial processes, used as a workforce, and considered a strong ally for solving difficult data-heavy problems. However, while many people consider machine learning such a powerful tool, there is still a chance that these algorithms may fail in unexpected ways. For this reason, there is much interest in explaining the decisions taken by machine learning, and in testing its robustness against crafted attacks.

In this chapter we will provide background concepts on machine learning and adversarial machine learning, and we will introduce the notation that will be used for the rest of the thesis.

2.1 Machine Learning and Supervised Learning Algorithms

Machine Learning (ML) is the science of automatically improving computer algorithms, through the use of data, to perform tasks for which they are not explicitly programmed. In other words, machine learning is nothing more than parameter estimation for modeling a decision process by looking at the data.

A machine learning model is an algorithm that learns from data to perform a specific task. Building such a system requires creating a machine able to recognize patterns in the data, that are represented through measurable characteristics called *features*. Thereafter, we will represent an input sample as $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)$, where each x_i is one of its features.

In general, since domains have different bounds, it is usually a good practice to rescale everything in a known interval, e.g., between 0 and 1, or reshape as a Standard distribution, i.e. a Gaussian with zero mean and unit variance. After such preprocessing step, each sample will be encoded on a space of features with values included in a range between a minimum value x_{lb} (lower bound) and a maximum value x_{ub} (upper bound). We can then define a vector that contains all the lower bounds, $\boldsymbol{x}_{\text{lb}}$, and another vector that will be containing all the upper bounds $\boldsymbol{x}_{\text{ub}}$.

Hence, we can say that $\boldsymbol{x}_{\text{lb}} \preceq \boldsymbol{x} \preceq \boldsymbol{x}_{\text{ub}}$, where the $\preceq$ operator indicates the $\leq$ operator applied feature-wise.
On top of this feature extraction phase, depending on the use case and the data that are used, machine learning can be divided into three main categories:

- **Supervised Learning.** The algorithm is trained on labeled data and has to find the relationship between the data and the provided labels.
- **Unsupervised Learning.** Algorithms that learn from unlabeled data, hence learning hidden structures within it.
- **Reinforcement Learning.** Algorithms that work by putting a model in an environment and letting it learn by itself through observation of the environment and a reward system that provides feedback on the actions taken. The algorithm is trained to learn an optimal policy for achieving the maximum reward.

In this thesis, we will focus on *classification* problems, a family of algorithms that falls in the supervised learning category. A *classifier* is a machine learning algorithm that learns a mapping $f : \boldsymbol{x} \mapsto y$, where $\boldsymbol{x} \in \mathbb{R}^d$ is a feature vector, d is the input space dimension, $y \in \{0, \dots, C-1\}$ is the label, C is the number of classes, and f is the learned function that maps the input vector $\boldsymbol{x}$ to the label y. The model is defined by a set of parameters $\boldsymbol{\theta}$, that are optimized to obtain the mapping and control the behavior of the system.
The model will produce output scores, one for each of the classes C, and the classification is achieved by taking

$$\hat{y} = \underset{C \in \{0,\dots,C-1\}}{\operatorname{argmax}} f(\boldsymbol{x}; \boldsymbol{\theta})$$

as the predicted label. Note that the classes are numbered from 0 to $C-1$, and we will keep this convention for the rest of this thesis.
The model outputs C different scores, one for each of the classes in the dataset, that are called *logits*. The logits for a class c will be denoted as $f_c(\boldsymbol{x}; \boldsymbol{\theta})$. Since it is usually desirable to have the outputs modeled as probabilities, sometimes a Softmax function is applied to the logits. The Softmax is a function that takes a vector of values and produces another vector of the same dimension, where the values satisfy the conditions to be in the $[0, 1]$ range, and add up to 1.0, i.e., they can be considered a probability of the sample being of class c. The Softmax defined as

$$\sigma(\boldsymbol{t})_j = \frac{e^{t_j}}{\sum_{k=1}^{K} e^{t_k}},$$

takes elements of the vector, computes the element-wise exponential, and divides each element by the sum of the exponentials of the whole vector. Hence, we can

define the probability scores $z(\boldsymbol{x};\boldsymbol{\theta}) = \sigma(f(\boldsymbol{x};\boldsymbol{\theta}))$ as the Softmax output of the model, where each value z_c is the probability of the input sample of belonging to class c.

The parameters of the model are estimated through modeling of the empirical distributions contained in the data. The original dataset $\mathcal{D}$ is divided into the *training set*, $\mathcal{D}_{train}$, and the *testing set*, $\mathcal{D}_{test}$. The training set is used to train the model by optimizing its parameters, and the testing set is used to test its performance since they mimic future unknown data. The parameters $\boldsymbol{\theta}$ are learned by defining a loss function $\mathcal{L} : (\boldsymbol{x}, y; \boldsymbol{\theta}) \mapsto \mathbb{R}$, that returns a real value for evaluating a candidate set of parameters in the given task. In simpler words, the loss can be seen as the ability of the model on classifying the sample $\boldsymbol{x}$ as the true label y. A loss function (or cost function) is a function that returns a numerical value typically proportional to the difference between the desired outputs and the output produced by the model on a set of training samples. The loss function can be defined in different ways, depending on which errors we want to prioritize and fix from the output. One typical loss function that is used for training machine-learning models is the *cross-entropy loss*, as it penalizes the misclassifications, i.e. when the model predicts the wrong label, in the training data.

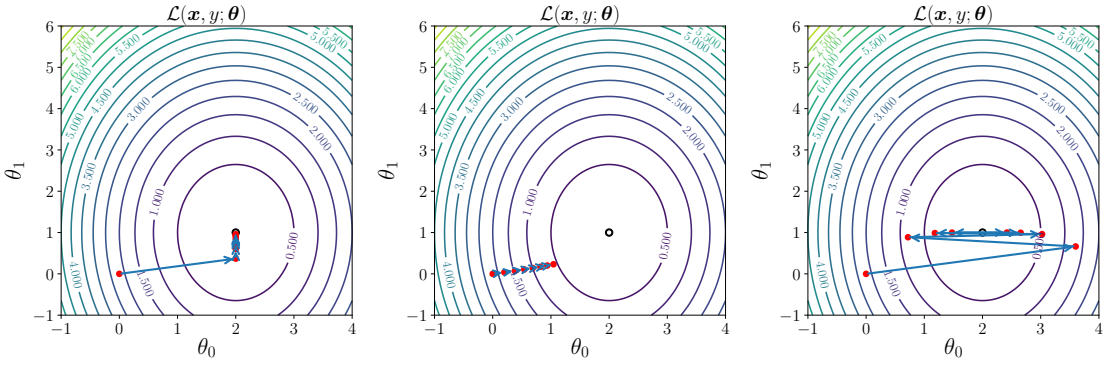
The cross-entropy loss is defined as

$$\mathcal{L}(\boldsymbol{x}, y; \boldsymbol{\theta}) = -\sum_{c=0}^{C-1}(y_c \log((z_c(\boldsymbol{x};\boldsymbol{\theta})))). \tag{2.1}$$

where the value of y_c is 1 if the correct label for the sample is c, otherwise 0, i.e. the one-hot-encoded vector corresponding to the label y, and $z_c(\boldsymbol{x};\boldsymbol{\theta})$ is the probability score for the class c. For example, if the problem includes 10 classes, the label $y = 2$ should be encoded as $[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$. Note, as already stated before, that the numbering starts from class 0. Thus, the goal is to make the score of the correct class higher for each sample, i.e. minimizing the loss.

The most straightforward and popular way to optimize loss functions is through the use of gradient descent. Gradient descent requires a differentiable function, e.g. the one in Eq. 2.1, and iteratively optimizes it to find the values that achieve the minimization. The gradient measures how much the loss changes when changing the input parameters of the loss, and gives the steeper direction to the minima of the loss. In the case of training, we use the partial derivative of the loss $\mathcal{L}$ with respect to the weights $\boldsymbol{\theta}$, i.e. $\nabla_{\boldsymbol{\theta}}\mathcal{L} = \frac{\partial\mathcal{L}(\boldsymbol{x},y;\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}$, and we iteratively update the weights to achieve a decrease in the loss. Since we want to modulate the amount of movement inside the space of the parameters, we set a hyperparameter that controls the size of each step in the descent. We must set a multiplier, called the *learning rate* (α), to an appropriate value. Each update is thus obtained through the product:

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha\nabla_{\boldsymbol{\theta}}\mathcal{L}$$



(a) $\alpha = 0.1$ (b) $\alpha = 1.0$ (c) $\alpha = 0.01$

Figure 2.1: Importance of tuning the step size (α) properly in the optimization process. The step size regulates the optimization of the loss $\mathcal{L}(\boldsymbol{x}, y; \boldsymbol{\theta})$ in the space spanned by θ_0 and θ_1. When the step size is correct (a), the updates bring the optimized point to the minimum of the loss. When the step size is too small (b), the updates are not sufficient for the optimized point to reach the optimum. When the step size is too big (c), the optimized point overshoots the minimum and it is unable to descend properly.

In this context, it is important to highlight the distinction between parameters and hyperparameters. *Parameters* are the variables in the models that must be determined using the training data. These are fitted by the optimization algorithm. *Hyperparameters* are adjustable values that must be tuned in order to obtain training with good performance. Under this notation, the model's weights $\boldsymbol{\theta}$ are clearly parameters, whereas the step size α falls into the category of hyperparameters. In general, this separation stands true also in other optimization problems, such as the optimization-based adversarial attacks, which will be presented in the following sections.
Tuning the hyperparameters, e.g. α, is a consistent part of training the models, as the choice of these might affect significantly the result of the optimization. Figure 2.1 shows an example of gradient descent using different step sizes for optimization. This is the size of the change from one step and the other, and making it too big might make the optimization skip the optimal value while making it too small might require many steps to reach convergence.
The final performance of a model can be computed as its accuracy, defined as:

$$\mathcal{A} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}_{\hat{y}_i = y_i},$$

where the $\mathbb{1}$ is the indicator function, equal to 1 only when the condition $\hat{y}_i = y_i$ is satisfied, hence when the predicted label for the sample $\boldsymbol{x} \in \mathcal{D}_{test}$ $\hat{y}_i$ is equal to the true label y_i.

2.2 Adversarial Machine Learning

As clear from the news, AI is being used in many applications. To demonstrate how this field is advancing quickly, we want to mention that in 2021, Israeli operatives reportedly carried out an assassination mission with an AI-guided Sniper Rifle[1]. The progressive use of AI in increasingly risky environments places even more emphasis on the importance of protecting these systems from intentional attacks and blatant mistakes. As more and more systems are now based mainly on machine learning models, researchers are showing that these powerful methods are not without weaknesses. The inputs and the systems themselves are vulnerable to noteworthy adversarial attacks, i.e. manipulated samples that tamper the model's decision or prevent its normal functioning. These attacks, often imperceptible to humans, can be inconvenient or even threatening to deployed AI systems. The highly critical tasks for which these systems are being used have raised questions about their trustworthiness and the interpretation of their results. The extensive use of machine learning in safety-critical systems can expose new weaknesses that are specific to these kinds of models and for which not only a few defenses exist, but they are also not trivial to evaluate effectively.

The first popular examples of adversarial perturbation appeared in 2012 (Biggio et al., 2013; Szegedy et al., 2014) and are known under the name of *adversarial examples*. In 2016 Carlini et al. (2016) showed that it is possible to design perturbed audio signals that fool the voice-recognition systems to act on commands that are hidden to our hearing but interpreted by our trusted assistants. In April 2019, an attack against self-driving cars has shown that it takes only some small sticker in the road for causing the autopilot to follow an unexpected path or an image that looks like noise to activate the windshield wipers[2].

These findings are part of the field of research of Adversarial Machine Learning. This topic has gained soaring interest in recent years precisely because of this alarming evidence and of the disastrous consequences of a successful attack, depending on the targeted application scenario. This phenomenon has been growing especially in the context of security domains, including cybersecurity, typically involved in an adversarial context where the impostor wants to access undetected restricted resources. Current machine learning methods have been designed carelessly of the security aspect required for trusting them in the real world. These systems can be fooled with well-crafted adversarial manipulation to the input data, also known as evasion attacks (Biggio et al., 2013; Szegedy et al., 2014), and the manipulation of the training sets used for updating the model based on online training, performing attacks known as poisoning attacks (Biggio et al., 2012; Koh and Liang, 2017).

[1] https://www.nytimes.com/2021/09/18/world/middleeast/iran-nuclear-fakhrizadeh-assassination-israel.html

[2] `https://keenlab.tencent.com/en/2019/03/29/Tencent-Keen-Security-Lab-Experimental-Security-Research-of-Tesla-Autopilot/`

Other attacks discover hidden information – potentially personal or sensitive data - from the deployed model (Papernot et al., 2016a).

In the following section, we will provide a taxonomy that organizes the adversarial attacks against classification systems and, in general, supervised learning algorithms. The description of adversarial attacks against unsupervised learning algorithms and reinforcement learning algorithms is beyond the scope of this thesis.

2.2.1 Taxonomy of Adversarial Attacks Against Classifiers

Adversarial attacks against classification learning algorithms can be categorized along three primary axes: the attacker's goal, the attacker's knowledge, and the attacker's capability. A widely-established taxonomy was presented by Biggio and Roli (2018) and will be summarized in the following while localizing the adversarial attacks that will be included in this thesis, i.e. evasion attacks, within the different aspects presented.

Attacker's Goal. The attacker's goal is what the attackers aim to achieve with their actions. Cybersecurity concepts define the CIA Triad[3], which models the information security threats in three axes, namely *Confidentiality*, *Integrity*, and *Availability*. In evasion attacks, the attacker aims to achieve an *integrity* security violation, i.e., to compromise the normal system operation. The attack can be error-generic, i.e. aiming to have the input sample classified as any class excluding the true label, or error-specific, i.e. aiming to have the input sample misclassified as a specific class. This distinction is also known in the literature as *untargeted* and *targeted* attacks respectively, and this notation will be used for the remaining part of the thesis.

Attacker's Knowledge. The attacker's knowledge indicates the information that is available to the attacker. Intuitively, the more information available, the more impact the attacks should have on the target model. The knowledge of the attacker in adversarial attacks against classification can be divided on the availability to the parameters of the model, i.e. the weights, and the output of the classifier. Moreover, different shades of availability can be identified. The attacker can have direct access to the model, or know at least the type of model and how it was trained, hence having the possibility of creating a similar model, or can have access to the outputs in two different modes, knowing either the output scores or the classification labels.

- *White-box attacks.* This is the ideal case for attackers, as they have complete access to the model. They can query the classifier, obtain its outputs, but most importantly they can compute the gradients with respect to the input features, that are required by many attacks. More information will be given in Sect. 3.1.1.

[3] `https://en.wikipedia.org/wiki/Information_security`

- *Black-box attacks.* Attackers can only query the model and obtain either the predicted labels or, if accessible, the output scores of the decision function. This is often the case of Machine Learning as a Service (MLaaS), where the user can send queries to a model that is deployed on the web and not downloadable, and the platform returns only the model's output. More information about black-box attacks will be provided in Sect. 3.2.2.

- *Gray-box attacks.* In this case, the attacker knows additional information on how the model was created. One example can be an MLaaS platform that gives access to training data or open-sources the model architecture that is being used. This case is more unusual, as often MLaaS are owned by companies and avoid disclosing information about the deployed models.

Attacker's Capability. This axis indicates if the attacker can act on the training phase of the model, and perturb the training data, i.e. altering the weights that will be learned, or only on the testing phase, i.e. only changing the output for the samples that are being passed. This divides the attacks between *poisoning attacks*, that aim to achieve a specific behavior of the model after training, and *evasion attacks*, that aim to obtain a specific output in the inference phase of the model. Another limitation for the capability of the attackers is if the specific application includes further constraints on the things that they can change. For example, when perturbing a sample in the image domain, the features have to remain in the bounds of the representation, typically within a value of 0 and 255 for each *pixel* (see Sect. 2.1 for recalling about feature space bounds).

Table 2.1 shows the taxonomy of the adversarial attacks against machine learning, highlighting where evasion attacks are located depending on the attacker's goal and capability. In this thesis, we will focus on evasion attacks with both *white-box* and *black-box* attacker's knowledge scenarios.

	Attacker's Goal		
Attacker's Capability	**Integrity**	**Availability**	**Privacy/Confidentiality**
Test data	Evasion (a.k.a. *adversarial examples*)	Energy-latency attacks (a.k.a. *sponge examples*)	Model extraction / stealing and model inversion (a.k.a. *hill-climbing attacks*)
Training data	Poisoning (to allow subsequent intrusions, e.g. backdoors or neural network trojans)	Poisoning (to maximize classification error)	-

Table 2.1: Categorization of attacks against machine learning depending on the capability and the goal of the attacker, as defined in Biggio and Roli (2018).

2.2.2 Evasion Attacks

As previously discussed, evasion attacks allow to find, given an input $\boldsymbol{x}$ correctly classified as y, a small perturbation for which $\boldsymbol{x} + \boldsymbol{\delta}$ is classified as $y' \neq y$. The attack can be *targeted* or *untargeted*. A *targeted* attack aims to have the sample

misclassified as a specific label $y'_t \neq y$, whereas an *untargeted* attack will create adversarial examples that are simply misclassified, hence having simply $y' \neq y$. Creating adversarial examples amounts to optimizing a trade-off between the size of the perturbation and the confidence of the classifier on the misclassification objective within a constrained region Δ:

$$\min_{\boldsymbol{\delta} \in \Delta} (\mathcal{L}(\boldsymbol{x} + \boldsymbol{\delta}, y; \theta), ||\boldsymbol{\delta}||_p). \tag{2.2}$$

A targeted attack will maximize the score of the target class y_t and an untargeted attack will simply minimize the score of the original label y. The region Δ is defined by several bounds, that limit the search space. One bound constrains the sample to remain within the input space of the model, i.e., $\boldsymbol{x}_{\text{lb}} \preceq (\boldsymbol{x} + \boldsymbol{\delta}) \preceq \boldsymbol{x}_{\text{ub}}$ as defined in Sect. 2.1. The norm constraint $||\boldsymbol{\delta}||_p$ bounds instead the maximum perturbation size in the given perturbation model. Commonly-used perturbation models are ℓ_0, ℓ_1, ℓ_2, ℓ_∞, and will produce respectively sparse to increasingly dense perturbations. In summary, we can write such constrained regions as

$$\Delta = \{\boldsymbol{\delta} \in \mathbb{R}^d \mid \|\boldsymbol{\delta}\|_p \leq \epsilon \cap \boldsymbol{x}_{\text{lb}} \preceq \boldsymbol{x} + \boldsymbol{\delta} \preceq \boldsymbol{x}_{\text{ub}}\}.$$

The loss function $\mathcal{L}(\boldsymbol{x} + \boldsymbol{\delta}, y; \boldsymbol{\theta})$ determines the landscape where we are optimizing our attacks and whether the attack is targeted or untargeted by including the specification in the objective. In the following of this thesis, we will adopt the convention for which the desired adversarial examples are found in the minima of the loss function (changing that will only require swapping the signs in all the equations). Thus, attackers aim to minimize the loss for achieving their goals.

Depending on the case, the attack can have different loss functions as objective.

- **Cross-entropy Loss (CE-Loss).** The loss that Szegedy et al. (2014) used for the FGSM attack, defined as in Eq. 2.1, but slightly modified for achieving misclassification, rather than the correct functioning of the model. The attacker aims to minimize:

$$\mathcal{L}(\boldsymbol{x}, y; \boldsymbol{\theta}) = \sum_{c=1}^{C} (y_c \log(z_c(\boldsymbol{x}; \boldsymbol{\theta}))), \tag{2.3}$$

  where, in contrast to the case of training, the loss has swapped sign, i.e. for each sample the attacker aims to minimize the probability score of the correct class. The score z_c is the probability output of the model for classifying the sample $\boldsymbol{x}$ as class y. For a targeted attack, it is sufficient instead to maximize the probability of the target class z_t, i.e. using Eq. 2.3 with the opposite sign, and substituting the original label y with the target label y_t.

- **Logit Difference (CW-Loss).** This loss was introduced by Carlini and Wagner (2017b), in the popular attack that brings their name. In this case, the loss is directly applied to the logits, i.e. the outputs of the model.

$$\mathcal{L}(\boldsymbol{x}, y; \boldsymbol{\theta}) = f_c(\boldsymbol{x}; \boldsymbol{\theta}) - \max_{j \neq y} f_j(\boldsymbol{x}; \boldsymbol{\theta}) \,, \tag{2.4}$$

  where $f_j(\boldsymbol{x}, \boldsymbol{\theta})$ is the score output of the model for the sample $\boldsymbol{x}$ as class j, and $\boldsymbol{\theta}$ is the set of its learned parameters. Assuming the classifier assigns higher scores to the correct class, the loss function takes on negative values when $\boldsymbol{x}$ becomes adversarial. As for the previous case, this loss can become targeted if the score $f_t(\boldsymbol{x}; \boldsymbol{\theta})$ is used instead of the score for the original class, and the sign of the loss is swapped, i.e. the objective becomes maximizing the logit of the target class. It is worth noting that setting this loss equal to zero means finding the decision boundary while imposing an offset means enforcing a margin of misclassification confidence on the sample.

- **Difference of Logit Ratio (DLR).** This loss function was introduced by Croce and Hein (2020b), and it is both shift and rescaling invariant to the outputs of the model. This is obtained by using the Logit Difference and dividing it by a scaling factor computed on the logits themselves.

$$\mathrm{DLR}(\boldsymbol{x}, y; \boldsymbol{\theta}) = \frac{f_c(\boldsymbol{x}; \boldsymbol{\theta}) - \max_{j \neq y} f_j(\boldsymbol{x}; \boldsymbol{\theta})}{f_{\pi 1}(\boldsymbol{x}; \boldsymbol{\theta}) - f_{\pi 3}(\boldsymbol{x}; \boldsymbol{\theta})}, \tag{2.5}$$

  where π is the ordering of the components of the logits in decreasing order. The shift invariance is achieved by having a difference of logits in the denominator. The targeted version of this loss is further complicated and will not be used in the context of this thesis, suggesting the interested readers to refer to the original paper for more details (Croce and Hein, 2020b).

Keeping on with the parallel to training machine learning models, also adversarial attacks define losses that they need to optimize, and the most straightforward way to do so is using gradient descent. In contrast to what happens in training, in this case, we are keeping the parameters of the model $\boldsymbol{\theta}$ fixed, while varying the perturbation added to the input data $\boldsymbol{\delta}$. For this reason, we need to know how much the loss changes w.r.t. the features of the sample, i.e. we need to compute the gradient of the loss w.r.t. to the input data. The gradient is expressed as $\nabla \mathcal{L}_x = \frac{\partial \mathcal{L}(\boldsymbol{x}, y; \boldsymbol{\theta})}{\partial \boldsymbol{x}}$. The algorithms then update, in many cases iteratively, the attack sample along the negative gradient of the objective function, in order for it to decrease.

An adversarial attack is thus defined by its objective function, i.e. the *loss*, and an optimizer, i.e. the *algorithm*, that aims to minimize the objective function within the constraints. A list and description of the state-of-the-art attacks, along with the associated objectives and algorithms, will be given in Chapter 3.

2.3 Assessing Robustness of Machine Learning Models

For model designers, it is thus important to estimate robustness to the adversarial attacks. One simple evaluation can be to evaluate the model's sensitivity as the expected value of its correct classification after the inputs have been perturbed with a given perturbation model. The *robust accuracy* is defined by setting a region delimited by $||\boldsymbol{\delta}||_p \leq \epsilon$ and centered in $\boldsymbol{x}$ and computing the classification accuracy of the samples perturbed with an ℓ_p norm smaller than ϵ for all the points in a testing set.

$$\mathop{\mathbb{E}}_{(\boldsymbol{x},y)\sim\mathcal{D}_{test}} [\mathcal{A}(\boldsymbol{x}+\boldsymbol{\delta}, y; \boldsymbol{\theta}) | \, \boldsymbol{\delta} \in \Delta]$$

Thus, it is clear that stronger attacks will produce, for the same perturbation bounds, lower robust accuracies. The threshold $\epsilon > 0$ is set for different datasets and models, ranging in values that produce robust accuracies between the clean accuracies and zero. For example, common standards for the MNISTdataset are to set the ℓ_∞ norm at 0.3 and the ℓ_2 norm at 0.5. For having a complete snapshot of the robustness of a model, and for better comparing different models, one can also produce the security evaluation curves, that compute the robust accuracy as a function of the distance, by running attacks for different values of the threshold ϵ and inspecting their success. Figure 2.2 shows an example of security evaluation on two different models.

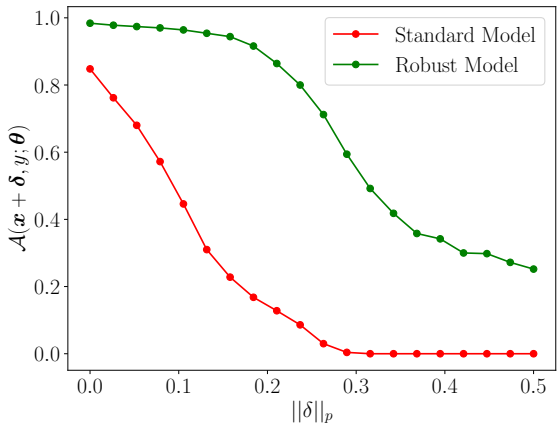


Figure 2.2: Example of security evaluation curve. The robust model (*green*) retains higher accuracy than the standard model (*red*) on increasing amounts of perturbation.

Alternatively, one can fix the other objective of the trade-off, i.e. the accuracy (under attack), and inspect the minimum distance. For this, an evaluation procedure was established in Schott et al. (2018) and followed in other minimum-distance

attack papers. The idea is to inspect the distance for which 50% of the samples are adversarial, i.e. for which the attack succeeds. For achieving that, it is sufficient to compute the median of the minimum distances found for the points in the testing set, taking care to set the samples that are not adversarial after the attack to a distance of ∞ in order for them to not be taken into account (unless the majority of the samples are not adversarial, then the median distance will result equal to ∞).

2.3.1 Defenses Against Evasion Attacks

To improve robustness, various defense techniques against adversarial examples have been proposed. Such defenses can be roughly categorized into 7 categories:

- **Adversarial training.** This is the most straightforward defense, introduced by Goodfellow et al. (2015) and Kurakin et al. (2017), and then extended by Madry et al. (2018). The robustness of this model is achieved by augmenting the training data with adversarial examples so that the model can learn how to map them to their true labels. This improves adversarial robustness but it is resource-demanding and time-consuming, as it requires creating adversarial examples during training. Training time is slowed down by the inner attack iterations that create the augmented samples, however, applying faster attacks still produces good results (Wong et al., 2019; Rony et al., 2019). This method is often combined with other defense mechanisms for further increasing robustness.

- **Robust regularization.** The method consists in penalizing the input gradients, hence the sensitivity to input perturbations, during training. The cost of computing adversarial examples is removed, but the method still requires the computation of the input gradients to regularize them along with the objective function. This method is shown to be equivalent to adversarial training in Ross and Doshi-Velez (2018).

- **Manifold projections.** In this defense, the input data is projected into a learned data manifold, that models the distribution of the training samples. In this way, if an adversarial example is submitted to the model, it is reprojected in the data distribution, with the intent of removing the adversarial perturbation (Jalal et al., 2017; Shen et al., 2017; Samangouei et al., 2018; Song et al., 2018; Schott et al., 2018).

- **Stochasticity.** The defense adds randomness to the input (Xie et al., 2018; Prakash et al., 2018), the activations (Xiao et al., 2020; Dhillon et al., 2018), or to the outputs (Park et al., 2021). These defenses are often broken by the use of smoothing techniques in the loss (Tramer et al., 2020).

- **Preprocessing/Input Quantization.** This defense mechanism preprocesses the inputs (Guo et al., 2018; Munusamy Kabilan et al., 2018) or the activations (Buckman et al., 2018), often by quantizing them (Papernot et al., 2016b; Lu et al., 2017). Unfortunately, these are often found ineffective (Carlini and Wagner, 2016).

- **Detectors.** A separate model is trained to detect the adversarial examples (Bendale and Boult, 2016; Meng and Chen, 2017; Li and Li, 2017; Melis et al., 2017; Yu et al., 2019). The output of the model is the label and the probability that the sample is adversarial, which can be thresholded to reject the decision. The peculiarity of this model is that it does not provide the prediction if the sample is rejected unless a separate algorithm recovers the input sample.

- **Certified Defenses.** These defenses provide guarantees of robustness to norm-bounded attacks by ensuring no adversarial example is present in the certified radius (Cohen et al., 2019; Zhang et al., 2020). The limitation of these defenses is that they do not scale well, or are supported for very specific types of models or attacks.

Despite the considerable number of defense papers proposed in the last years, only a few of them stood the test of time. Why, you may ask. The main problem that plagues these defenses lies in the method by which they are evaluated. Searching for adversarial examples produces a binary output, i.e., it indicates whether they were found in the region in which they were searched. Unfortunately, the state-of-the-art algorithms and frameworks are not capable of providing support to understand if the optimization process went correctly. In the next chapter, we will dive deeper into the reasons why adversarial robustness evaluations are not an easy task, and what should be done to make them more efficient and effective.

Chapter 3

Adversarial Robustness Evaluations

Research on adversarial machine learning has been carried out for many years and has shown impressive improvements in the attack side, while the mitigation techniques proceed slower mainly due to the different nature of the proof: it takes only a sample to show that a defense does not work, while the attack can work in some case and still be effective. In other words, for proving a defense effective it should be evaluated with all possible inputs to the model, a task impossible even for the most advanced systems.

The design of the system is a constantly-updating arms race, where the attacker and the defender act based on the opponent's moves evolving over time to reach their opposite goals (Biggio et al., 2013; Biggio and Roli, 2018). An update to the model happening only after a successful attack – a reactive approach - may be too latecomer to avoid damage. A proactive approach, i.e. the anticipation of the possible attacks, is preferable especially in case of critical applications. This approach advocates the secure-by-design paradigm, that is the identification of potential threats against the system and the embedding on the system itself of specific countermeasures that address the problem before it can be exploited by the attacker.

It is of paramount importance to define an experimental protocol and state clearly the settings of the scenario, the model, the processes, and the attacker, in order to have a complete view of the problem. The threat model defines the attacker's *goal*, *knowledge*, and *capability* for manipulation, which have been already discussed in Sect. 2.2.1. For this, a substantial help, at least to start the analysis, is provided by the white-box attacks and by the libraries that implement them, which allow in few lines of code to set up a preliminary robustness evaluation. We have deliberately used the adjective *preliminary* because the search for adversarial examples cannot be limited to a hasty analysis with things already built, but this is just the beginning. Once this step is completed, and no adversarial examples are found, one should move on to more complicated and dedicated search techniques.

Unfortunately, the designed defenses often lack a fulfilling evaluation protocol and have been proved ineffective (Athalye et al., 2018). The main limitations of these evaluations are the following:

- many defenses are able to identify *only* existing attacks. The solutions provided by these defenses are the attack identification, e.g. with a specifically trained classifier, and the reject option, i.e. the rejection of low-confidence decisions. These solutions are unfortunately not able to detect the unknown. Adaptive attacks, i.e. attacks aware of the defense in place, have successfully fooled this kind of defense;
- some of the known defenses only *hide* the gradients of the decision function, so that they cannot be exploited for creating the attacks. This is not a solution, as they can be bypassed by using an approximation of the target function (Athalye et al., 2018);
- in many other cases, the attacks used for testing the defenses are run with the default hyperparameters, resulting in a complete underestimation of the impact of the attack.

In Chapter 2 we introduced the background on machine learning, adversarial machine learning, and what it means to assess adversarial robustness. In this chapter we will first detail the robustness evaluation process, then we will describe its main limitations, along with stating what will be addressed in this thesis.

3.1 Existing Tools and Frameworks

Learning algorithms are vulnerable to adversarial examples, i.e., intentionally-perturbed inputs aimed to mislead classification at test time (Szegedy et al., 2014; Biggio et al., 2013). Since their discovery and formalization, there has been a proliferation of attack algorithms that attempt to find these small perturbations. In this thesis, we will focus on the *gradient-based* category of attacks, i.e. all the attacks that aim to find the steepest descent direction on a loss function.

3.1.1 Gradient-based Adversarial Attack Algorithms for Evading Classifiers

The first test to assess the robustness of a model is always to try the defense with the strategy that already works well enough, in the majority of the cases, to reveal that the defense is weak. This restricts the first part of the analysis to gradient-based attacks. Depending on the strategy adopted, these attacks can be divided into three main sub-categories:

- **Projected-gradient attacks.** These algorithms perform a maximum-confidence attack in each step under a given perturbation budget ϵ. Some of them also iteratively adjust ϵ to reduce the perturbation size.
- **Soft-constraint attacks.** They optimize a trade-off between the confidence of the misclassified samples and perturbation size.
- **Boundary attacks.** These attacks move along the decision boundary towards the closest point to the input sample.

An important distinction has to be highlighted here. All these attacks define an optimization problem where *maximum-confidence* and *minimum-distance* attacks are either the objective or the constraint. Specifically, maximum-confidence attacks optimize the following problem:

$$\begin{aligned} \boldsymbol{\delta}^\star \in \arg\min_{\boldsymbol{\delta}} \quad & \mathcal{L}(\boldsymbol{x}+\boldsymbol{\delta}, y, \boldsymbol{\theta}) && (3.1)\\ \text{s.t.} \quad & \|\boldsymbol{\delta}\|_p \leq \epsilon && (3.2)\\ & \boldsymbol{x}_{\text{lb}} \preceq \boldsymbol{x}+\boldsymbol{\delta} \preceq \boldsymbol{x}_{\text{ub}}\,, && (3.3) \end{aligned}$$

where $||\cdot||_p$ indicates the ℓ_p-norm operator. The loss L in the constraint in Eq. (3.2) can be any of the losses defined in Sect. 2.2.2, e.g., the cross-entropy loss, defined accordingly to the goal of the attack, i.e. untargeted or targeted misclassification, and keeping in mind that we defined them as having adversarial examples in the minima. Minimum-distance attacks instead have the misclassification confidence as the main objective, while constraining the maximum distance, i.e.

$$\begin{aligned} \boldsymbol{\delta}^\star \in \arg\min_{\boldsymbol{\delta}} \quad & \|\boldsymbol{\delta}\|_p && (3.4)\\ \text{s.t.} \quad & f_y(\boldsymbol{x}+\boldsymbol{\delta}, \boldsymbol{\theta}) \neq f_y(\boldsymbol{x}, \boldsymbol{\theta}) && (3.5)\\ & \boldsymbol{x}_{\text{lb}} \preceq \boldsymbol{x}+\boldsymbol{\delta} \preceq \boldsymbol{x}_{\text{ub}}\,, && (3.6) \end{aligned}$$

where $||\cdot||_p$ indicates the ℓ_p-norm operator. The constraint in Eq. (3.5) can be, for example, the difference between the output score of the original class y and the scores of the other classes, hence enforcing a misclassification whenever the loss is negative. The problem may be formulated also as a targeted attack by setting accordingly the sign of the loss, and targeting the class $\hat{y}$. Hence, in this problem, the misclassification confidence is the constraint, and the distance is the objective to minimize.

In contrast to *maximum-confidence* attacks, which maximize confidence in a wrong class within a given perturbation budget, the latter are better suited to evaluate adversarial robustness as one can compute the accuracy of a classifier under attack

for any perturbation budget without re-running the attack. Having a set of diverse attacks available is however important, as for evaluating a model's robustness it does not matter how the adversarial example is found, only whether it was found or not. This emphasizes the importance of understanding what are the main attacks and their principles, as it might reveal where they can work better than others and where they can fail.

Projected Gradient Descent (PGD). PGD is a powerful projected-gradient iterative attack that was proposed by Kurakin et al. (2017) and by Madry et al. (2018). This attack implements the multiple-step version of a previously-proposed attack named Fast Gradient Sign Method (FGSM) (Goodfellow et al., 2015), that attacks the ℓ_∞-bounded region with a single step of normalized gradient descent on the negative cross-entropy loss (as in the one defined in Eq. 2.3), to maximize the misclassification rate. Other methods that incrementally improve FGSM or PGD have also been proposed later on (Dong et al., 2018; Lin et al., 2020), but the underlying principle remains the same. FGSM computes a single iteration as

$$\boldsymbol{x} - \epsilon \operatorname{sign}(\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, y; \boldsymbol{\theta})),$$

on the cross-entropy loss $L(\boldsymbol{x}, y; \boldsymbol{\theta})$. Note that the attack can aim to increase the cross-entropy loss of the model on the original labels to achieve error-generic misclassifications, i.e. untargeted attacks, or can reduce the loss on the classification of the input as a specific target class to achieve an error-specific misclassification, i.e. a targeted attack. This attack assumes local linearity of the loss function within the step size and moves the perturbed points in the normalized vertices of an ℓ_∞ box around the current iterate with a radius of ϵ.
The iterative versions, i.e. PGD, computes the multi-step variant

$$\boldsymbol{x}_{t+1} = \Pi_\Delta(\boldsymbol{x}_i - \alpha \operatorname{sign}(\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, y; \boldsymbol{\theta})),$$

that produces a more refined optimization on the loss landscape by performing small steps of maximum size α while keeping the updates inside the ϵ-bounded region. Other variants introduce improvements such as momentum (Dong et al., 2018), or Nesterov acceleration and scale invariance (Lin et al., 2020).

Carlini & Wagner Attack (CW). This is a soft-constraint attack, introduced by Carlini and Wagner (2017a), that minimizes the soft-constraint version of the attack problem, i.e.,

$$\min_{\boldsymbol{\delta}} \|\boldsymbol{\delta}\|_p + c \cdot \min(\mathcal{L}(\boldsymbol{x} + \boldsymbol{\delta}, y, \boldsymbol{\theta}), -\kappa).$$

The loss used here is the logit-difference loss, as in Eq. 2.4. The hyperparameters κ and c are used to tune the trade-off between perturbation size and misclassification confidence. Respectively, κ regulates the misclassification confidence by setting a minimum required value for the loss, while c is a multiplier for the loss term that is

searched through line search in an outer loop of the attack algorithm (the inner loop computes the gradient-descent steps). To find minimum-norm perturbations, CW requires setting $\kappa = 0$, while the constant c is tuned via binary search (re-running the attack at each iteration).

Decoupling Direction and Norm (DDN). This boundary attack, proposed by Rony et al. (2019), can be seen as a minimum-distance version of the PGD attack, as it optimizes the cross-entropy loss within an ϵ-sized constraint, but with the difference that it adjusts ϵ to minimize the perturbation size.

Brendel & Bethge Attack (BB). This boundary attack, proposed by Brendel et al. (2020), starts from a randomly-drawn adversarial point, performs a multi-step binary search to find a point that is closer to the decision boundary, and then updates the point to minimize its perturbation size by following the decision boundary. The direction of the update moves alongside the boundary, by using the gradient of the difference of logits. In each iteration, BB computes the optimal update within a given trust-region of radius ρ. In contrast to standard gradient-based attacks, which start from the clean sample, this attack starts from a point that is adversarial and far away from the initial one, and walks along the boundary towards the smallest perturbations that produce an adversarial example.

Fast Adaptive Boundary (FAB). This boundary attack, proposed by Croce and Hein (2020a), iteratively optimizes the attack point by linearly approximating its distance to the decision boundary. FAB uses a scale-invariant version of the logit difference loss, which selects the steepest descent on the class that is the closest, according to the linear approximation, to the current iterate, i.e. it selects the closest hyperplane that leads to untargeted misclassification. This attack does not implement a *targeted* version in the sense that is commonly intended in this field. The "*targeted*" version of the FAB attack is actually a method to accelerate finding an untargeted adversarial example. It uses an adaptive step size bounded by $\alpha_{\max}$ and an extrapolation step η to facilitate finding adversarial points. For each step, it computes the direction and performs a trade-off update including a forward step towards the adversarial region, and a backward step to reduce the perturbation size.

Other attacks and variations of the presented attacks. Many of these attacks have been later incrementally improved by small variations that improve their optimization process in some specific cases. For example, the popular PGD attack has been later on enhanced with momentum (Dong et al., 2018). We are restricting our analysis to the category of PGD attacks, without diving into much detail on all the variations. The same goes for the CW attack, which was also proposed to support a mixed trade-off of the ℓ_1 and ℓ_2 norms, as ElasticNet attack (Chen et al., 2018).
Moreover, minimum-norm attacks like DDN, FAB, and BB do not produce high-confidence adversarial examples (otherwise they could always find smaller perturbations). Despite that, they are not weak. In fact, minimum-norm attacks such as CW

and BB have served as standard tools in the breaking of many defenses (Athalye et al., 2018; Brendel et al., 2020), and are recommended in widely-accepted community guidelines (Carlini, 2019; Tramer et al., 2020), that will be presented in Sect 3.2. Aside from that, there exist some weak minimum-norm attacks like DeepFool (Moosavi-Dezfooli et al., 2016), which make strong assumptions on the models to increase query efficiency but are considered sub-optimal algorithms compared to the ones listed here as they aim to achieve primarily different objectives, e.g. a limited computational cost. For these reasons, we are not giving many details on a number of adversarial attacks that fall outside the categories presented here and outside the scope of this thesis.

AutoAttack. Special mention should be made of the work by Croce and Hein (2020b) that partially automates the choice of attacks and their hyperparameters. AutoAttack is a hyperparameter-free attack that includes 4 sub-strategies, namely:

- AutoPGD with Cross-entropy Loss (APGD-CE), which is an extension of PGD that automatically tunes the step size with an internal search;
- AutoPGD with the Difference of Logits Ratio (APGD-DLR), which in addition to doing the step-size tuning, changes the original cross-entropy loss to the DLR loss defined in Eq. 2.5.
- FAB, the boundary attack presented in the previous overview of gradient-based attacks (Croce and Hein, 2020a);
- SquareAttack, a query-efficient black-box attack by Andriushchenko et al. (2020), that searches for adversarial examples with ℓ_2 and ℓ_∞ perturbation models by exploring at each iteration in localized square-shaped regions.

The APGD attack partially solves the problem of the careful selection of the step size, trading it off with fewer queries available for actually running the attack. Furthermore, the AutoAttack can be considered a framework of adversarial attacks itself, as it runs for each input point all the available internal strategies and selects the best one.

Table 3.1: Summary of the main state-of-the-art evasion attacks. CE and DL refer to the Cross-Entropy Loss and to the Difference of Logits respectively.

Attack	**Category**	**Loss**	**Pert. Model**	**Targeted**	**Untargeted**
PGD	projected gradient	CE	ℓ_1, ℓ_2, ℓ_∞	✓	✓
CW	soft constraint	DL	ℓ_2	✓	✓
DDN	projected gradient	CE	ℓ_2	✓	✓
BB	boundary	DL	ℓ_0, ℓ_1, ℓ_2, ℓ_∞	✓	✓
FAB	boundary	DL[a]	ℓ_1, ℓ_2, ℓ_∞		✓

[a] FAB uses a scale-invariant version of the difference of logits.

As described here, there is a great variety of attacks and implementations, that are also summarized in Table 3.1. However, the main difficulty remains: it is not easy to choose the right one, tune the hyperparameters correctly, without incurring mistakes and failures that can lead to a faulty evaluation. First, every attack has its strengths and weaknesses. Soft-constraint attacks like CW optimize a trade-off between the confidence of the misclassified samples and perturbation size. This class of attacks needs a sample-wise tuning of the trade-off hyperparameter to find the smallest possible perturbation, thus requiring many steps to converge. Boundary attacks like BB or FAB move along the decision boundary towards the closest point to the input sample. These attacks converge within relatively few steps but may require an adversarial starting point and need to solve a relatively expensive optimization problem in each step. Finally, recent minimum-norm projected-gradient attacks like DDN perform a maximum-confidence attack in each step under a given perturbation budget ϵ, while iteratively adjusting ϵ to reduce the perturbation size. DDN combines the effectiveness of boundary attacks with the simplicity and per-step speed of soft-constraint attacks; however, it is specific to the ℓ_2 norm and cannot be readily extended to other norms.

All these attacks are gradient-based, hence they optimize a loss function towards finding adversarial examples. An important aspect to keep in mind, that will be described profusely in Sect. 3.2, is the problem of *gradient obfuscation*. In short, gradient obfuscation happens when the defense breaks the gradient-descent mechanism by either making the loss function highly non-linear or by making the gradients zero in the local region where the attack is being optimized, i.e. there is no direction to follow to find the optimum.

3.1.2 Toolboxes and Benchmarks

The implementation of the aforementioned attacks can be found in many different *ad hoc* open-source code libraries. The adversarial toolboxes provide powerful off-the-shelf, ready-to-use attacks, to evaluate machine learning models. Such libraries are useful to evaluate individual schemes, i.e., a single attack against a single model. These libraries provide unified interfaces for the attacks so that it is easy to test attacks in a plug-and-play manner.

Toolboxes for Adversarial Attacks. Compared to the years when adversarial machine learning was born, there is now a wide variety of open-source code libraries that focus on implementing attack algorithms. Some of them also provide implementations of several basic defenses, but they do not generally include up-to-date state-of-the-art pre-trained models. Here we give a quick overview of the most used libraries for adversarial attacks.

CleverHans[1] (Papernot et al., 2018) is an adversarial-machine-learning library that provides standardized reference implementations of attack algorithms and

[1] `https://github.com/cleverhans-lab/cleverhans`

adversarial training techniques. This library originally supported only Tensorflow models, but it was recently updated to support more back-end frameworks for deep learning.
AdverTorch[2] (Ding et al., 2019), similarly to CleverHans implements state-of-the-art attacks through the use of standard optimizers commonly used for deep learning, but it is restricted to PyTorch users.
Foolbox[3] (Rauber et al., 2017, 2020) is a library that lets you easily run adversarial attacks against machine learning models like deep neural networks. It is built on top of EagerPy, a wrapper that makes it works natively with models in different deep learning frameworks. It is nowadays the most used library for adversarial robustness evaluations.
The Robustness Library[4] (Engstrom et al., 2019) implements primarily methods for training, evaluating, and exploring robust neural networks. It contains some of the most used attacks, but it focuses more on the defenses.
Adversarial Robustness Toolbox (ART)[5] (Nicolae et al., 2019) contains by far one of the widest collections of attacks and applications against machine learning. It includes the adversarial threats of Evasion, Poisoning, Extraction, and Inference, supports many popular machine learning frameworks, data types (images, tables, audio, video, etc.), and machine learning tasks (classification, object detection, speech recognition, generation, certification, etc.).
SecML[6] (Melis et al., 2019) is a security evaluation library for machine-learning algorithms, that supports the PyTorch deep learning framework as well as the scikit-learn models, and includes many useful tools for inspecting the internal process of creating adversarial attacks.

Despite seeming very easy to use, the majority of these libraries often come with default hyperparameters and lack of support to correctly configure the attacks. This scenario, often combined with tight time constraints for testing the defense properly, might expose the model to the risk of being evaluated too lightly, as will be explained in the following sections. It is worth pointing out that there is also a small number of tools to compare attacks other than in terms of their success rate, as these libraries rarely collect logs and statistics during the optimization process.

Benchmarks. Related to the aforementioned libraries, there are a number of attack benchmarks that have been constructed. Instead of measuring the robustness of individual schemes as the just-described libraries do, these benchmarks aim to provide a complete evaluation framework that can be extended to any future model and attack as well.

[2] `https://github.com/BorealisAI/advertorch`
[3] `https://github.com/bethgelab/foolbox`
[4] `https://github.com/MadryLab/robustness`
[5] `https://github.com/Trusted-AI/adversarial-robustness-toolbox`
[6] `https://github.com/pralab/secml`

The MNIST[7] Challenge and CIFAR10[8] Challenge (Madry et al., 2018) were among the first benchmarks that appeared in the field, hosted at NeurIPS 2017 and 2018. The aim of these challenges was to find, in a black-box manner, the adversarial examples for the models trained with adversarial training (Madry et al., 2018) on the respective datasets, for which the weights were initially kept secret. The challenge received many submissions, and later on, the models were released to the public. Ling et al. (2019) proposed DEEPSEC[9], a benchmark that tests several attacks against a wide range of defenses. However, this framework was shown to be flawed by several implementation issues and problems in the configuration of the attacks (Carlini, 2019). One of the first public benchmarks for evaluating and comparing different models was RobustML[10], that despite being a valuable contribution for the community, relied on evaluations performed by users, and the last submitted model dates back to 2019. Croce et al. (2020) propose RobustBench[11], that accepts state-of-the-art models as submissions, and it tests their robust accuracy by applying AutoAttack (Croce and Hein, 2020b). However, this benchmark suite is not able to determine which are the possible causes of such scored performance, e.g. it does not inspect why the attacks failed. This is a limitation of all available benchmarks, as it will be clear after reading the next section.

3.2 Current Guidelines for Evaluating Adversarial Robustness

There have been a number of prior papers evaluating the robustness of particular defense schemes (Carlini and Wagner, 2017a; Athalye et al., 2018; Tramer et al., 2020). These papers focus on understanding whether the robustness claims of particular defenses are true, often by performing one-off attacks or by proposing new general attack approaches that can be used to break future defenses. The main aspect of these papers is that the authors focus their efforts on breaking defenses by manually reversing their mechanisms, showing what the authors of the original evaluation had overlooked. In many cases, it involved problems in the optimization of the attacks, rather than errors in the implementation or in the code itself. Carlini et al. (2019) systematized various suggestions from the literature for how to ensure that adversarial robustness evaluations are performed thoroughly.

[7] `https://github.com/MadryLab/mnist_challenge`
[8] `https://github.com/MadryLab/cifar10_challenge`
[9] `https://github.com/kleincup/DEEPSEC`
[10] `https://www.robust-ml.org/`
[11] `https://github.com/RobustBench/robustbench`

3.2.1 Obfuscated Gradients and the Rise of Adaptive Evaluations

At this point, it is clear that performing security evaluations is not just taking a set of attacks and running them against the defense, but it requires a careful analysis of the problem. In the field, many published defenses have been shown to conduct an incorrect or incomplete evaluation. Figure 3.1 shows a timeline demonstrating a few representative defense papers, published also in top venues for the field, that were broken by subsequent evaluations. The fact that many defenses were created and evaluated even after the publications of the most relevant paper containing the guidelines for robustness evaluations (Carlini et al., 2019) proves that we are still far from having a well-defined protocol for properly checking a defense.

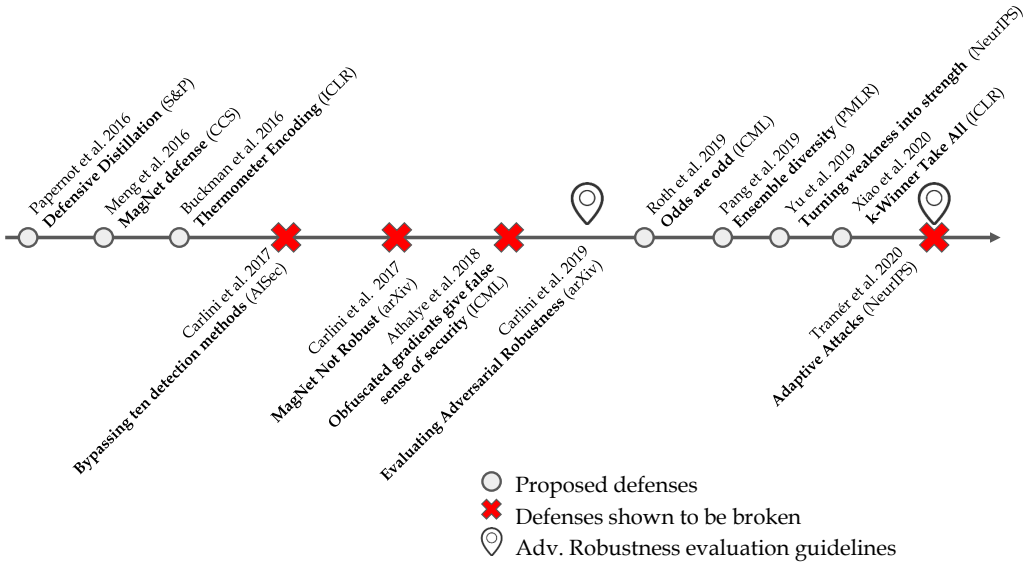
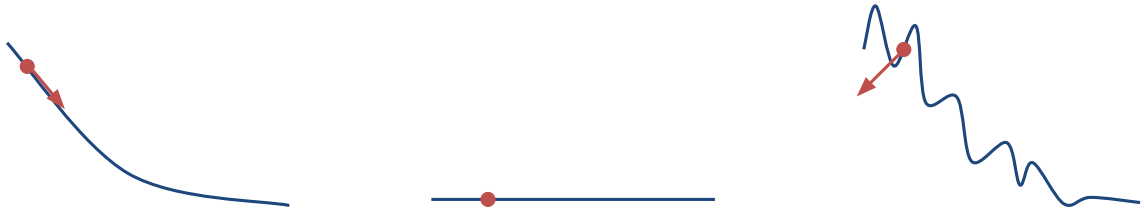


Figure 3.1: Timeline of the arms race for adversarial robustness evaluations.

The main problem that often leads to the defense being broken after publication is that the evaluation does not cover properly the possibility of unknown attacks. Moreover, it is not possible to rule out the presence of adversarial examples if they were not found with an empirical strategy, as a different method may find them and reveal a broken defense.
Even the state-of-the-art attacks might be not strong enough, as they are tested only against existing defenses, hence leaving a good chance these attacks will not work against a new defense. One typical example of this is the paper by Athalye et al. (2018), where the authors found that 7 published defenses were indeed only apparently robust to adversarial examples. These models were not creating a robust decision landscape with no adversarial points, but in fact, they were only making



(a) (b) (c)

Figure 3.2: When the landscape is sufficiently smooth (a), gradient descend is able to decrease the loss. When the gradient is zero (b), it is not possible to find the steepest direction. When the landscape is highly irregular, the descent is noisy as the local steepest descent might not point to a good solution.

them difficult to find by making the gradients of the models useless for generating adversarial examples. Figure 3.2 shows examples of how gradient-descent works well when the loss is smooth (Figure 3.2a), while it becomes difficult when the loss is locally flat (Figure 3.2b), or when the landscape is noisy (Figure 3.2c). The *gradient obfuscation* problem was thus defined as a defense that either causes the loss to be non-differentiable or with incorrect gradients (*gradient shattering*), to have a highly variable landscape by introducing randomness (*stochastic gradients*), or to be causing numerical instabilities that cause the gradients to be infinite or zero (*exploding/vanishing gradients*). This paper notably set the bar high for authors that wanted to publish their own robust models, as it demonstrated that a lot of care should be put in the evaluation part, rather than focusing only on the design of the defense.
Later on, this difficulty of properly assessing the robustness of defenses led to the creation of guidelines that are generically applicable to all evaluations, in order to at least avoid common problems and check the model as hard as possible. In Chapter 4 we will formalize these problems as 4 failures that affect the robustness evaluations, and provide a concrete advancement in the methods that can be used to detect them and to fix them automatically. However, at the current state, only a checklist for the evaluation is available. The list divides the items into three main categories:

- *Common severe flaws.* This category includes items whose invalidity results in the invalidity of all robustness claims derived from the evaluation. It states to define a precise threat model, to validate that with strong attacks, to release the model and evaluation as open-source code, and to perform sanity tests to ensure that the approach is valid. One particular point is to evaluate the model with *adaptive attacks*, i.e. attacks that have full knowledge of and access to the defense.

- *Common pitfalls.* This category includes mistakes that might prevent the detection of ineffective defenses, but do not apply to all defense mechanisms. Failing items in this category might not result in a completely invalid claim,

however, they are still important to carefully check the correctness of the evaluation. These include testing with a set of diverse attacks, trying attacks that do not use the gradients of the target model, verify that attacks have converged and have been configured well.

- *Special-case pitfalls.* The items in this category apply only to a restricted fraction of defenses, but they can provide additional evidence that the evaluation was performed correctly. These include the use of provable approaches when possible, the use of random-search attacks, and the investigation of domains different from the image domain.

Recently, Tramer et al. (2020) proved once again that using a well-defined checklist is not yet sufficient for having the complete picture of the robustness of a model. In this work, the authors tested thirteen defenses that were evaluated with reportedly adaptive attacks, but then a more thorough evaluation showed that the robust accuracy of the models was nowhere near the declared one. The authors also try to systematize again the main strategies to use for creating attacks, by defining 6 recurring "*attack themes*", i.e. repeating patterns of wrong assumptions, that they fixed to break the included defenses. This is a very interesting work, where each defense is mapped to which attack themes broke it, however, it still requires a lot of manual work and a deep knowledge of the topic.

The guidelines by Carlini et al. (2019) and the attack paper by Tramer et al. (2020) themselves stress the importance of evaluating the defenses with all the available tools, and of trying also new strategies that target the particular defense, without using the presented strategies as templates. Unfortunately, there is little help from the attack algorithms, that do not provide much feedback on the failures. This requires inspecting the failures one-by-one, and manually applying strategies designed for each specific case. One little guidance can be found in performing sanity checks that reveal if the assumptions on the attack strategy are being overlooked in some way. In the next section, we will dive deeper into the problem of gradient obfuscation, and describe additional strategies that can be used as tests for inspecting carefully if the evaluation is being affected by such a problem.

3.2.2 Additional Attack Strategies and sanity checks

The guidelines in Carlini et al. (2019) present, along with the standard pitfalls, a comprehensive list of sanity checks that should be performed to complement the robustness evaluation, i.e. to make sure that there are no inconsistencies in what to expect from the results.

Among the sanity checks, the simplest but also extremely useful ones include (i) making sure the performance of the model goes to zero when attacked with unbounded perturbations; (ii) ensuring doubling attack iterations does not increase

attack success rate; (iii) trying brute-force and random-search attacks; (iv) trying both targeted and untargeted attacks[12]; and (iv) verify iterative attacks perform better than single-step attacks. All these checks are very helpful to make sure that the results make sense logically, as there should be a strict improvement of the attack's success rate with the increase in either computational budget or perturbation budget.

There is then a set of sanity checks that require more attention. First, adaptive attacks should be stronger than baseline attacks. Second, all white-box attacks should work better than black-box attacks. The black-box threat model is a subset of the white-box threat model, as it works with less information and it usually finds solutions that are less close to the optimal ones. Surprisingly, there are some cases where using the gradients of another model, or using no gradients at all, works better. If this happens, it means that the loss function is affected by gradient obfuscation as the white-box attacks are not able to use the information contained in the gradients to optimize the objective.

Creating adversarial examples without using the gradients of the model means that we should either use another model's gradients, i.e. using transfer attacks, or optimize the loss function with different methods, e.g. with query-based attacks. Query-based attacks can be divided into score-based attacks (Chen et al., 2017; Narodytska and Prasad Kasiviswanathan, 2016), which use the output scores of the model for approximating the gradient, and decision-based attacks (Brendel et al., 2018), which rely only on the final decision of the model. Other attacks do not necessarily rely on model information nor model outputs[13], and are called transfer-based attacks (Goodfellow et al., 2015; Szegedy et al., 2014). This last category of attacks needs only some information on the training data, that is either known or at least reproduced as best as possible. Then, the data is used to train a model from which adversarial perturbations can be created. This is based on empirical evidence that adversarial examples transfer between models, but unfortunately it is indeed not easy to maximize their success. Notwithstanding that all the premises made for the correct configuration and optimization of attacks are valid, in this case also selecting the model and selecting attacks that are effective for this threat model is not yet a process of which all aspects are perfectly understood. The variables in play are many, and understanding which ones influence the success of transferability the most, and in which way to change them, is not yet a well-known process.

[12]In many situations, untargeted attacks succeed more easily, however, there are cases in which the loss behaves better when targeting, for example, only a subset of the classes (Gowal et al., 2019; Tramer et al., 2020).

[13]Transfer attacks can however use information collected from the model for creating a model as similar as possible to the target, or to verify the success of the attacks between iterations.

3.3 Limitations of the State of the Art

Although the process of evaluating adversarial robustness seems to be well-defined, it is indeed much more difficult and time-consuming and requires expert knowledge of different subjects, including machine learning, optimization techniques, and last but not least good coding skills. This is further demonstrated by the state-of-the-art defenses that are broken in the never-ending arms race of robustness evaluations. Recent papers like the one by Tramer et al. (2020) provide further evidence that the field is still much divided between *defense* and *attack* papers, leaving practitioners of both sides very far apart from each other. In this work, we are attempting to bridge the gap between the guidelines for robustness evaluations, and the papers that break the defenses one-by-one. For now, proving a system is robust has been treated as a failure of all the known attacks. This is clearly a limitation of the approach, and raises the following open questions:

- How can we create strong attacks in an efficient way?
- Are these attacks effectively testing the defense? How can we understand if the attacks used for this evaluation are suffering from sub-optimal configuration or problems like gradient obfuscation?
- How can we conduct a thorough evaluation of a defense, making sure that every weak spot has been effectively tested in the worst-case scenario?

These are the main open problems that will be investigated in this thesis and tackled in the next chapter. We firmly believe that the proposed scheme could significantly advance the state of the art and favor a much larger adoption of secure machine learning models.

3.3.1 Main Thesis Goals and Outputs

This thesis aims to create tools and methods for evaluating a defense strategy for a machine learning system in a reliable and rigorous way. Existing guidelines (Carlini et al., 2019) will be used as starting point, and new directions will be explored in this work.

First, evaluations are often performed with adversarial attacks that have to be properly configured, might be taking a long time to run, and might also be limited to only one perturbation model. To overcome these limitations, in this work we propose a novel, fast minimum-norm (FMN) attack (Sect. 4.2), which retains the main advantages of many of the competing attacks, while generalizing it to different ℓ_p norms ($p = 0, 1, 2, \infty$). We perform large-scale experiments on different datasets and models (Sect. 5.1), showing that FMN is able to significantly outperform current minimum-norm attacks in terms of convergence speed and computation time while finding equal or better optima across almost all tested scenarios and ℓ_p norms.

FMN thus combines all desirable traits a good adversarial attack should have, providing an important step towards improving adversarial robustness evaluations. This contribution is included in a recently-accepted NeurIPS paper:

- M. Pintor, F. Roli, W. Brendel, and B. Biggio. Fast minimum-norm adversarial attacks through adaptive norm constraints. In *Thirty-fifth Conference on Neural Information Processing Systems (NeurIPS 2021)*, 2021.

Second, it is not easy to debug if the optimization within the attack algorithms is encountering problems that are preventing the attacks to succeed. To tackle this limitation, we introduce a unified attack framework that captures the predominant styles of existing gradient-based attack methods, and allows us to categorize the five main causes of failure that may arise during their optimization, we propose five *indicators of attack failures* (IoAF), i.e., metrics and principles that help understand why and when gradient-based attack algorithms fail (Sect. 4.3), and we empirically evaluate the utility of our metrics on four recently-published defenses, showing how their robustness evaluations could have been improved by monitoring the IoAF values and following our evaluation protocol (Sect. 5.2). We also add to the presented metrics a non-trivial technique, called *Slope*, to measure in general the ease of decreasing the attacker loss, including the presence of gradient obfuscation. The triggering of this metric may suggest changing strategy as gradient-based white-box attacks might be failing due to problems in the gradient. This contribution encloses the work of two papers, one accepted at ESANN 2021, and the other in preprint, available on ArXiv:

- M. Pintor, L. Demetrio, G. Manca, B. Biggio, and F. Roli. Slope: A first-order approach for measuring gradient obfuscation. In *Proceedings of the ESANN, 29th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2021)*, 2021;

- M. Pintor, L. Demetrio, A. Sotgiu, G. Manca, A. Demontis, N. Carlini, B. Biggio, and F. Roli. Indicators of attack failure: Debugging and improving optimization of adversarial examples, *arXiv preprint*, 2021.

Third, within the sanity checks included in the guidelines, we want to improve the transferability analysis by understanding which phenomena affect the success rate of transferable evasion attacks. We highlight two main factors contributing to attack transferability: the intrinsic adversarial vulnerability of the target model, and the complexity of the surrogate model used to optimize the attack. Based on these insights, we define three metrics that impact an attack's transferability. The publication that accompanies this contribution was presented at USENIX 2019:

- A. Demontis, M. Melis, M. Pintor, M. Jagielski, B. Biggio, A. Oprea, C. Nita-Rotaru, and F. Roli. Why do adversarial attacks transfer? explaining

transferability of evasion and poisoning attacks. In *28th USENIX Security Symposium (USENIX Security 19)*, 2019.

In the next chapter, we will describe in detail these contributions, and how they concretely improve the overall process of evaluating the adversarial robustness of machine learning models.

Chapter 4

Efficient and Bug-free Adversarial Robustness Evaluations

This chapter will introduce a mathematical framework that can be used for creating gradient-based adversarial attacks (Sect. 4.1), use it for designing a powerful optimization algorithm (Sect. 4.2), and for analyzing the point where generic gradient-based attacks can fail (Sect. 4.3). Finally, the framework can also be used for creating transferable adversarial examples, following the guidelines that we found in our study (Sect. 4.4).

4.1 Attack Framework

Resuming from Chapter 2, adversarial attack algorithms solve an optimization problem defined as objective and constraints. We argue here that optimizing adversarial examples amounts to solving a multi-objective optimization:

$$\min_{\boldsymbol{\delta} \in \Delta} \left(\mathcal{L}(\boldsymbol{x} + \boldsymbol{\delta}, y; \boldsymbol{\theta}), \|\boldsymbol{\delta}\|_p \right) , \tag{4.1}$$

where $\boldsymbol{x} \in [0,1]^d$ is the input sample, $y \in \{1, \dots, c\}$ is either its label (for untargeted attacks) or the label of the target class (for targeted attacks), and $\boldsymbol{\delta} \in \Delta$ is the perturbation optimized to have the perturbed sample $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{\delta}$ misclassified as desired, within the given input domain. The target model is parameterized by $\boldsymbol{\theta}$. The given problem presents an inherent tradeoff: minimizing $\mathcal{L}$ amounts to finding an adversarial example with large misclassification confidence and perturbation size, while minimizing $\|\boldsymbol{\delta}\|_p$ penalizes larger perturbations (in the given ℓ_p norm) at the expense of decreasing misclassification confidence.[1] Typically the attacker loss $\mathcal{L}$ is defined as the Cross-Entropy (CE) loss (Eq. 2.3), the logit difference by Carlini and Wagner (2017b) (Eq. 2.4), or the Difference of Logit Ratio (DLR) by Croce and Hein (2020b) (Eq. 2.5).

[1] Note that the sign of $\mathcal{L}$ may be adjusted internally in our formulation to properly account for both untargeted and targeted attacks.

In addition, when the input space is bounded, e.g., $\boldsymbol{x} \in [0,1]^d$, a box constraint is also enforced on $\boldsymbol{\delta}$ to ensure that the perturbed sample $\boldsymbol{x}'$ lies in the same space.

Multiobjective problems define a Pareto frontier in a plane with two axes, namely the misclassification confidence and perturbation size. Depending on the strongest requirements, one can select the best trade-off between the given objectives along the Pareto frontier, by either using soft- or hard-constraint reformulations. For example, Carlini and Wagner (CW) (Carlini and Wagner, 2017b) is a soft-constraint attack, which reformulates the aforementioned multiobjective problem as an unconstrained optimization:

$$\min_{\boldsymbol{\delta}} \|\boldsymbol{\delta}\|_p + c \cdot \min(\mathcal{L}(\boldsymbol{x}+\boldsymbol{\delta}, y, \boldsymbol{\theta}), -\kappa),$$

where the hyperparameters κ and c tune the trade-off between misclassification confidence and perturbation size. Hard-constraint reformulations instead aim to minimize one objective while constraining the other. They include maximum-confidence attacks like Projected Gradient Descent (PGD) (Madry et al., 2018), which is formulated as

$$\min_{\boldsymbol{\delta}} \mathcal{L}(\boldsymbol{x}+\boldsymbol{\delta}, y; \boldsymbol{\theta}) \quad \text{s.t.} \quad \|\boldsymbol{\delta}\|_p \leq \epsilon,$$

and minimum-norm attacks like Brendel and Bethge (BB) (Brendel et al., 2020) and Decoupling-Direction-Norm (DDN) (Rony et al., 2019), which can be formulated as

$$\min_{\boldsymbol{\delta}} \|\boldsymbol{\delta}\|_p \quad \text{s.t.} \quad \mathcal{L}(\boldsymbol{x}+\boldsymbol{\delta}, y; \boldsymbol{\theta}) \leq k.$$

In these cases, ϵ and k upper bound the perturbation size and the misclassification confidence, respectively, thereby optimizing a different trade-off between these two quantities.

The aforementioned attacks often need to use an approximation $\hat{\boldsymbol{\theta}}$ of the target model, since the latter may be either non-differentiable or not sufficiently smooth (Athalye et al., 2018), hindering the gradient-based attack optimization process. In this case, once the attacker loss has been optimized on the surrogate model $\hat{\boldsymbol{\theta}}$, the attack is considered successful if it evades the target model $\boldsymbol{\theta}$.

Attack Algorithm. According to the previous discussion, even if different attacks minimize different objectives or require different constraints, they can all be seen as solutions to a common multi-objective problem, based on gradient descent. We summarize their main steps in Algorithm 1. First, an *initialization point* (line 1) needs to be defined, which can be achieved by directly using the input point $\boldsymbol{x}$, a randomly-perturbed version of it, or even a sample from the target class (Brendel et al., 2020). Then, if the target model $\boldsymbol{\theta}$ is difficult to handle (e.g., because it is non-differentiable) the attacker must choose a surrogate model $\hat{\boldsymbol{\theta}}$ that approximates the real target $\boldsymbol{\theta}$ (line 2). The attack then iteratively updates the initial point searching for a better and better adversarial example (line 4), computing in each iteration one (or more) gradient descent steps (line 5) using the initial point and the perturbation $\boldsymbol{\delta}_i$ computed so far. Hence, the new perturbation $\boldsymbol{\delta}_{i+1}$ is obtained by enforcing the

Algorithm 1 Our framework for computing adversarial attacks

Input: $\boldsymbol{x}$, the input sample; y, the target (true) class label if the attack is targeted (untargeted); n, the number of iterations; α, the learning rate; f, the target model; Δ, the considered region.

Output: The adversarial example $\boldsymbol{x}^\star$ that satisfies the constraints.

1: $\boldsymbol{x}_0 \leftarrow \texttt{initialize}(\boldsymbol{x})$ *// initialize starting point*
2: $\hat{\boldsymbol{\theta}} \leftarrow \texttt{approximation}(\boldsymbol{\theta})$ *// Approximate model parameters*
3: $\boldsymbol{\delta}_0 \leftarrow \mathbf{0}$ *//Initial δ*
4: **for** $i \in [1, n]$ **do**
5: $\quad \boldsymbol{\delta}' \leftarrow \boldsymbol{\delta}_i - \alpha \nabla_{\boldsymbol{x}_i} \mathcal{L}(\boldsymbol{x}_0 + \boldsymbol{\delta}_i, y; \hat{\boldsymbol{\theta}})$ *// Compute optimizer step*
6: $\quad \boldsymbol{\delta}_{i+1} \leftarrow \texttt{apply-constraints}(\boldsymbol{x}_0, \boldsymbol{\delta}', \Delta)$ *// Apply constraints (if needed)*
7: **end for**
8: $\boldsymbol{\delta}^\star \leftarrow \texttt{best}(\boldsymbol{\delta}_0, ..., \boldsymbol{\delta}_n)$ *// Choose best perturbation*
9: **return** $\boldsymbol{\delta}^\star$

constraints defined in the problem (line 6), which can be updated accordingly to the chosen strategy (Rony et al., 2019). For *maximum confidence* approaches, the attack can not exit the Δ region, and samples are projected accordingly on this ball when reaching the constraints. Alternatively, we consider *minimum distance* attacks successful only if they found adversarial examples inside the Δ region. At the end of the gradient descent iterations, the attacker collects all the perturbations along with the iterations into an *attack path*. The final result of the algorithm is the best perturbation contained anywhere in the attack path, w.r.t. the loss being minimized (line 8).

4.2 Fast Minimum-norm Adversarial Attacks through Adaptive Norm Constraints

As the first contribution, we present an efficient attack algorithm that finds minimum-norm perturbations in different ℓ_p norms ($p = 0, 1, 2, \infty$), that combines the main advantages of many other attack algorithms becoming an efficient and reliable tool for testing adversarial defenses. In other words, we aim to achieve an algorithm that is faster, more efficient, and that produces adversarial examples with a smaller perturbation budget, i.e. with a smaller norm of perturbation. We expand from the formulation of gradient-based attacks in Algorithm 1, and implement the steps to provide substantial improvement w.r.t. the current strategies.

Problem formulation. Given an input sample $\boldsymbol{x}_{\text{lb}} \preceq \boldsymbol{x} + \boldsymbol{\delta} \preceq \boldsymbol{x}_{\text{ub}}$, belonging to class $y \in \{1, \dots, c\}$, the goal of an untargeted attack is to find the minimum-norm perturbation $\boldsymbol{\delta}^\star$ such that the corresponding adversarial example $\boldsymbol{x}^\star = \boldsymbol{x} + \boldsymbol{\delta}^\star$ is

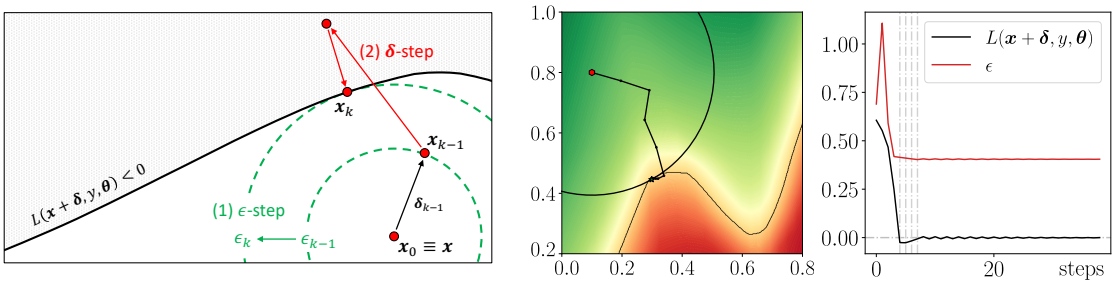


Figure 4.1: (a) Conceptual representation of the FMN attack algorithm (*leftmost plot*). The ϵ-step updates the constraint size ϵ to minimize its distance to the boundary. The $\boldsymbol{\delta}$-step updates the perturbation $\boldsymbol{\delta}$ with a projected-gradient step to maximize misclassification confidence within the current ϵ-sized constraint. (b) Example of execution of our attack on a bi-dimensional problem (*middle plot*), along with the corresponding values of the loss function $\mathcal{L}$ and the constraint size ϵ across iterations (*rightmost plot*). Our algorithm works by first pushing the initial point (red dot) towards the adversarial region (in red), and then perturbing it around the decision boundary to improve the current solution towards a local optimum. The vertical lines in the rightmost plot highlight the steps in which a better solution (smaller $\|\boldsymbol{\delta}^\star\|$ and $\mathcal{L} < 0$) is found.

misclassified. This problem can be formulated as:

$$\boldsymbol{\delta}^\star \in \arg\min_{\boldsymbol{\delta}} \quad \|\boldsymbol{\delta}\|_p\,, \tag{4.2}$$

$$\text{s.t.} \quad \mathcal{L}(\boldsymbol{x}+\boldsymbol{\delta}, y, \boldsymbol{\theta}) < 0\,, \tag{4.3}$$

$$\boldsymbol{x}_{\text{lb}} \preceq \boldsymbol{x}+\boldsymbol{\delta} \preceq \boldsymbol{x}_{\text{ub}}, \tag{4.4}$$

where $||\cdot||_p$ indicates the ℓ_p-norm operator. The loss $\mathcal{L}$ in the constraint in Eq. (4.3) is defined as:

$$\mathcal{L}(\boldsymbol{x}, y, \boldsymbol{\theta}) = f_y(\boldsymbol{x}, \boldsymbol{\theta}) - \max_{j \neq y} f_j(\boldsymbol{x}, \boldsymbol{\theta})\,, \tag{4.5}$$

where $f_j(\boldsymbol{x}, \boldsymbol{\theta})$ is the confidence given by the model f for classifying $\boldsymbol{x}$ as class j, and $\boldsymbol{\theta}$ is the set of its learned parameters. Note that this is the difference of logits that is used also in Carlini and Wagner (2017b), and similarly to the CW attack, misclassification confidence can be enforced by adding an offset κ to the difference of logits, i.e. by setting $\mathcal{L}(\boldsymbol{x}, y, \boldsymbol{\theta}) < -\kappa$.
Assuming that the classifier assigns $\boldsymbol{x}$ to the class exhibiting the highest confidence, i.e.,

$$y^\star = \arg\max_{j \in 1,\ldots,c} f_j(\boldsymbol{x}, \boldsymbol{\theta}),$$

the loss function $\mathcal{L}(\boldsymbol{x}, y, \boldsymbol{\theta})$ takes on negative values only when $\boldsymbol{x}$ is misclassified. Finally, the box constraint in Eq. (4.4) ensures that the perturbed sample $\boldsymbol{x}+\boldsymbol{\delta}$ lies in the feasible input space. The aforementioned problem typically involves a

non-convex loss function $\mathcal{L}$ (w.r.t. its first argument), due to the non-convexity of the underlying decision function f. For this reason, it may admit different locally-optimal solutions. Note also that the solution is trivial (i.e., $\boldsymbol{\delta}^\star = \mathbf{0}$) when the input sample $\boldsymbol{x}$ is already adversarial (i.e., $\mathcal{L}(\boldsymbol{x}, y, \boldsymbol{\theta}) < 0$).

Extension to the targeted case. The goal of a targeted attack is to have the input sample misclassified in a given target class y'. This can be accounted for by modifying the loss function in Eq. (4.5) as

$$\mathcal{L}^t(\boldsymbol{x}, y', \boldsymbol{\theta}) = \max_{j \neq y'} f_j(\boldsymbol{x}, \boldsymbol{\theta}) - f_{y'}(\boldsymbol{x}, \boldsymbol{\theta}) = -\mathcal{L}(\boldsymbol{x}, y', \boldsymbol{\theta}),$$

i.e., changing its sign and using the target class label y' instead of the true class label y.

Solution algorithm. To solve Problem (4.2)-(4.4), we reformulate it using an upper bound ϵ on $\|\boldsymbol{\delta}\|_p$:

$$\min_{\epsilon, \boldsymbol{\delta}} \epsilon\,, \quad \text{s.t.}\, \|\boldsymbol{\delta}\|_p \leq \epsilon, \tag{4.6}$$

and to the constraints in Eqs. (4.3)-(4.4). This allows us to derive an algorithm that works in two main steps, similarly to DDN (Rony et al., 2019), by updating the maximum perturbation size ϵ separately from the actual perturbation $\boldsymbol{\delta}$, as represented in Fig. 4.1(a). In particular, the constraint size ϵ is adapted to reduce the distance of the constraint to the boundary (ϵ-step), while the perturbation $\boldsymbol{\delta}$ is updated using a projected-gradient step to minimize the loss function $\mathcal{L}$ within the given ϵ-sized constraint ($\boldsymbol{\delta}$-step). This essentially amounts to a projected gradient descent algorithm that iteratively adapts the constraint size ϵ to find the minimum-norm adversarial example. The complete algorithm is given as Algorithm 2, while a more detailed explanation of the two aforementioned steps is given below.

ϵ-step. This step updates the upper bound ϵ on the perturbation norm (lines 4-12 in Algorithm 2). The underlying idea is to increase ϵ if the current sample is not adversarial (i.e., $\mathcal{L}(\boldsymbol{x}_{k-1}, y, \boldsymbol{\theta}) \geq 0$), and to decrease it otherwise, while reducing the step size to dampen oscillations around the boundary and reach convergence. In the former case (ϵ-*increase*), the increment of ϵ depends on whether an adversarial example has been previously found or not. If not, we estimate the distance to the boundary with a first-order linear approximation and set

$$\epsilon_k = \|\boldsymbol{\delta}_{k-1}\|_p + \mathcal{L}(\boldsymbol{x}_{k-1}, y, \boldsymbol{\theta}) / ||\nabla \mathcal{L}(\boldsymbol{x}_{k-1}, y, \boldsymbol{\theta})||_q,$$

being q the dual norm of p. This approximation allows the attack point to make faster progress towards the decision boundary. Conversely, if an adversarial sample has been previously found, but the current sample is not adversarial, it is likely that the current estimate of ϵ is only slightly smaller than the minimum-norm solution. We thus increase ϵ by a small fraction as $\epsilon_k = \epsilon_{k-1}\,(1 + \gamma_k)$, being γ_k a decaying

Algorithm 2 Fast Minimum-norm (FMN) Attack

Input: $\boldsymbol{x}$, the input sample; t, a variable denoting whether the attack is targeted ($t = +1$) or untargeted ($t = -1$); y, the target (true) class label if the attack is targeted (untargeted); γ_0 and γ_K, the initial and final ϵ-step sizes; α_0 and α_K, the initial and final $\boldsymbol{\delta}$-step sizes; K, the total number of iterations.

Output: The minimum-norm adversarial example $\boldsymbol{x}^\star$.

1: $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}$, $\epsilon_0 = 0$, $\boldsymbol{\delta}_0 \leftarrow \mathbf{0}$, $\boldsymbol{\delta}^\star \leftarrow \boldsymbol{\infty}$
2: **for** $k = 1, \dots, K$ **do**
3: $\quad \boldsymbol{g} \leftarrow t \cdot \nabla_{\boldsymbol{\delta}} \mathcal{L}(\boldsymbol{x}_{k-1} + \boldsymbol{\delta}, y, \boldsymbol{\theta})$ *// loss gradient*
4: $\quad \gamma_k \leftarrow h(\gamma_0, \gamma_K, k, K)$ *// ϵ-step size decay* (Eq. 4.7)
5: $\quad$ **if** $\mathcal{L}(\boldsymbol{x}_{k-1}, y, \boldsymbol{\theta}) \geq 0$ **then**
6: $\quad\quad \epsilon_k = \|\boldsymbol{\delta}_{k-1}\|_p + \mathcal{L}(\boldsymbol{x}_{k-1}, y, \boldsymbol{\theta}) / \|\boldsymbol{g}\|_q$ **if** *adversarial not found yet* **else** $\epsilon_k = \epsilon_{k-1}(1 + \gamma_k)$
7: $\quad$ **else**
8: $\quad\quad$ **if** $\|\boldsymbol{\delta}_{k-1}\|_p \leq \|\boldsymbol{\delta}^\star\|_p$ **then**
9: $\quad\quad\quad \boldsymbol{\delta}^\star \leftarrow \boldsymbol{\delta}_{k-1}$ *// update best min-norm solution*
10: $\quad\quad$ **end if**
11: $\quad\quad \epsilon_k = \min(\epsilon_{k-1}(1 - \gamma_k), \|\boldsymbol{\delta}^\star\|_p)$
12: $\quad$ **end if**
13: $\quad \alpha_k \leftarrow h(\alpha_0, \alpha_K, k, K)$ *// $\boldsymbol{\delta}$-step size decay* (Eq. 4.7)
14: $\quad \boldsymbol{\delta}_k \leftarrow \boldsymbol{\delta}_{k-1} + \alpha_k \cdot \boldsymbol{g} / \|\boldsymbol{g}\|_2$
15: $\quad \boldsymbol{\delta}_k \leftarrow \Pi_\epsilon(\boldsymbol{x}_0 + \boldsymbol{\delta}_k) - \boldsymbol{x}_0$
16: $\quad \boldsymbol{\delta}_k \leftarrow \mathrm{clip}(\boldsymbol{x}_0 + \boldsymbol{\delta}_k) - \boldsymbol{x}_0$
17: $\quad \boldsymbol{x}_k \leftarrow \boldsymbol{x}_0 + \boldsymbol{\delta}_k$
18: **end for**
19: **return** $\boldsymbol{x}^\star \leftarrow \boldsymbol{x}_0 + \boldsymbol{\delta}^\star$

step size. In the latter case (ϵ-*decrease*), if the current sample is adversarial, i.e., $\mathcal{L}(\boldsymbol{x}_{k-1}, y, \boldsymbol{\theta}) < 0$, we decrease ϵ as $\epsilon_k = \epsilon_{k-1}(1 - \gamma_k)$, to check whether the current solution can be improved. If the corresponding ϵ_k value is larger than the optimal $\|\boldsymbol{\delta}^\star\|_p$ found so far, we retain the best value and set $\epsilon_k = \|\boldsymbol{\delta}^\star\|_p$. These multiplicative updates of ϵ exhibit an oscillating behavior around the decision boundary, due to the conflicting requirements of minimizing the perturbation size and finding an adversarial point. To ensure convergence, as anticipated before, the step size γ_k is decayed with cosine annealing:

$$\gamma_k = h(\gamma_0, \gamma_K, k, K) = \gamma_K + \tfrac{1}{2}(\gamma_0 - \gamma_K)\left(1 + \cos\left(\tfrac{k\pi}{K}\right)\right) , \tag{4.7}$$

being k the current step, K the total number of steps, and γ_0 and γ_K the initial and final step sizes.

$\boldsymbol{\delta}$-step. This step updates $\boldsymbol{\delta}$ (lines 13-17 in Algorithm 2). The goal is to find the adversarial example that is misclassified with maximum confidence (i.e., for which $\mathcal{L}$

is minimized) within the current ϵ-sized constraint (Eq. 4.6) and bounds (Eq. 4.4). This amounts to performing a projected-gradient step along the negative gradient of $\mathcal{L}$. We consider a normalized steepest descent with decaying step size α to overcome potential issues related to noisy gradients while ensuring convergence (line 14). The step size α is decayed using cosine annealing (Eq. 4.7). Once $\boldsymbol{\delta}$ is updated, we project it onto the given ϵ-sized ℓ_p-norm constraint via a projection operator Π_ϵ (line 15), to fulfill the constraint in Eq. (4.6). The projection is trivial for $p = \infty$ and $p = 2$. For $p = 1$, we use the efficient algorithm by Duchi et al. (2008). For $p = 0$, we retain only the first ϵ components of $\boldsymbol{\delta}$ exhibiting the largest absolute value. We finally clip the components of $\boldsymbol{\delta}$ that violate the bounds in Eq. (4.4) (line 16).

Execution example. In Fig. 4.1(b), we report an example of the execution of our algorithm on a bi-dimensional problem. The initial sample is updated to follow the negative gradient of $\mathcal{L}$ towards the decision boundary. When an adversarial point is found, the algorithm reduces ϵ to find a better solution. The point is thus projected back onto the non-adversarial region, and ϵ increased (by a smaller, decaying amount). These oscillations allow the point to walk on the boundary towards a local optimum, i.e., an adversarial point lying on the boundary, where the gradient of the loss function and that of the norm constraint have opposite directions. FMN tends to quickly converge to a good local optimum, provided that the step size is reduced to a sufficiently-small value and that a sufficiently large number of iterations are performed. This is also confirmed empirically in Sect. 5.1.

Adversarial initialization. Our attack can be initialized from the input sample $\boldsymbol{x}$, or from a point $\boldsymbol{x}_{\text{init}}$ belonging either to a different class (if the attack is untargeted) or to the target class (if the attack is targeted). When initializing the attack from $\boldsymbol{x}_{\text{init}}$, we perform a 10-step binary search between $\boldsymbol{x}$ and $\boldsymbol{x}_{\text{init}}$, to find an adversarial point which is closer to the decision boundary. In particular, we aim to find the minimum ϵ such that $\mathcal{L}(\boldsymbol{x} + \Pi_\epsilon(\boldsymbol{x}_{\text{init}} - \boldsymbol{x}), y, \boldsymbol{\theta}) < 0$ (or $\mathcal{L}^t < 0$ for targeted attacks). Then we run our attack starting from the corresponding values of $\boldsymbol{x}_k$, ϵ_k, $\boldsymbol{\delta}_k$ and $\boldsymbol{\delta}^\star$.

Differences with DDN. FMN applies substantial changes to both the algorithm and the formulation of DDN. The main difference is that (i) DDN always rescales the perturbation to have size ϵ. This operation is problematic when using other norms, especially sparse ones, as it hinders the ability of the attack to explore the neighboring space and find a suitable descent direction. Another difference is that (ii) FMN does not use the cross-entropy loss but rather the logit difference as the loss function $\mathcal{L}$. Moreover, (iii) FMN does not need an initial value for ϵ, as ϵ is dynamically estimated; and (iv) γ is decayed to improve convergence around better minimum-norm solutions, by more effectively dampening oscillations around the boundary. Finally, we include the possibility of (v) initializing the attack from an adversarial point, which can greatly increase the convergence speed of the algorithm.

In this section, we presented a powerful attack that efficiently finds adversarial perturbations with gradient-based techniques combined with query-optimization strate-

gies. Unfortunately, even such powerful tools might still be affected by failures that prevent the attack to work with the standard settings. In the next section, we present useful indicators that detect problems in gradient-based optimization of adversarial attacks, along with mitigation strategies to correct the problems found, in order to open the way to bug-free and more reliable security evaluations.

4.3 Understanding Failures of Gradient-based Attacks

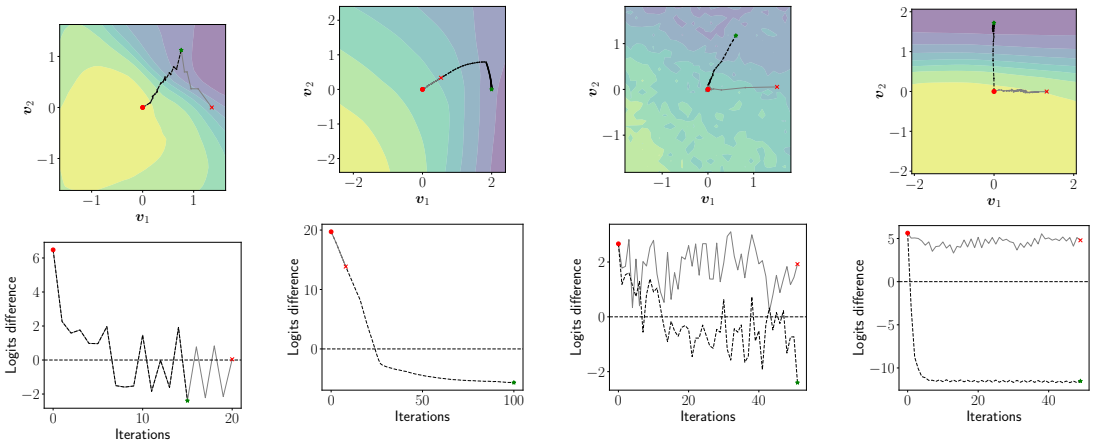
We want to point out that the currently available algorithms, which can all be seen as different expansions of Algorithm 1, lack any form of debugging system. In this context, we would like to start by drawing a parallel with software debugging, where specific tools help to pinpoint exactly the line that introduces the problem and make it much easier to fix it. With adversarial attacks instead, what happens is the equivalent of a program saying "*failure*", but without understanding where the problem is located, or without even having a vague indication through error messages. In this section we will first introduce a systematic analysis of the known possible failures that might affect the attacks, then we design measurable indicators that are able to detect when a known failure is happening, allowing us to fix the problem right away and patch the security evaluation.

4.3.1 Attack Failures

We now systematize the ways that gradient descent attacks fail into four failure categories, each of which corresponds to a particular step of Algorithm 1.

***F₁*: Implementation Problems.** If no adversarial examples are found by the attack, it is possible that the attack implementation has errors or bugs. For example, we isolated a problem inside the PGD procedure proposed by Madry et al. (2018). As described, the attack returns the adversarial example only by looking at the *last* point of the attack path (line 8 of Algorithm 1), as shown in Fig. 4.2a, but would not return an adversarial example if one was found during search and then passed over.

***F₂*: Non-converging attack.** When performing gradient-based attacks, a common problem is that they may not converge to any local minimum, as shown in Fig. 4.2b. This problem can be caused by sub-optimal choices of the attack hyperparameters, including the step size α and the number of iterations n (as shown in Algorithm 1). If α is too small, the gradient update step is not exploring the space (line 5 of Algorithm 1), while using too few iterations n might cause an early stopping of the attack (line 4 of Algorithm 1). An example of this failure can be found in the evaluation of the defense proposed by Buckman et al. (2018), where the authors only used 7 steps of PGD for testing the robustness of their defense, or by the one



(a) Impl. problems. (b) Non-converging attack (c) Bad local optimum. (d) Non-adaptive attack.

Figure 4.2: The four attack failures that can be encountered during the optimization of an attack. The failed attack path is shown in *gray*, while the successful attack is displayed in *black*. The point $\boldsymbol{x}_0$ is marked with the *red* dot, the returned point of the failed attack with a *red* cross, and the successful adversarial point with the *green* star. The top row shows the loss landscape, as $\mathcal{L}(\boldsymbol{x} + a\boldsymbol{v}_1 + b\boldsymbol{v}_2, y_i; \boldsymbol{\theta})$. $\boldsymbol{v}_1$ is the normalized direction $(\boldsymbol{x}_n - \boldsymbol{x}_0)$, while $\boldsymbol{v}_2$ is a representative direction for the displayed case. In the second row we show the value of $\mathcal{L}(\boldsymbol{x} + \boldsymbol{\delta}_i, y_i; \boldsymbol{\theta})$ for the evaluated model.

proposed by Pang et al. (2019), where the defense has been evaluated with only 10 steps of PGD. Alternatively, this failure might be triggered either by setting the step size too large, which leads the optimizer to keep overshooting the local minimum, or the presence of *gradient obfuscation techniques* that alter the gradients of the model to point to random directions, leading gradient descent to fail. An example of such failure is found in the defense proposed by Xiao et al. (2020), where the model keeps only the top-k neurons for each layer, resulting in a noisy loss landscape, where gradients point in many different directions.

***F₃*: Bad local optimum.** Once the attack reached convergence, the computed point might not be adversarial, if the optimizer has reached a region where it can no longer update the adversarial perturbation, as shown in Fig. 4.2c. There are a few reasons that might lead to such a failure. One of them is again caused by the presence of gradient obfuscation, where the optimizer is unable to continue the descent since it arrived in a region where the norms of gradients are (nearly) zero (i.e. flat regions), or again because the gradients are noisy, and the optimization lands on a bad local optimum (line 5 of Algorithm 1). An example of such failure is detected inside the defense proposed by Papernot et al. (2016b), where the model is trained to have signal in correspondence of samples, and producing regions with

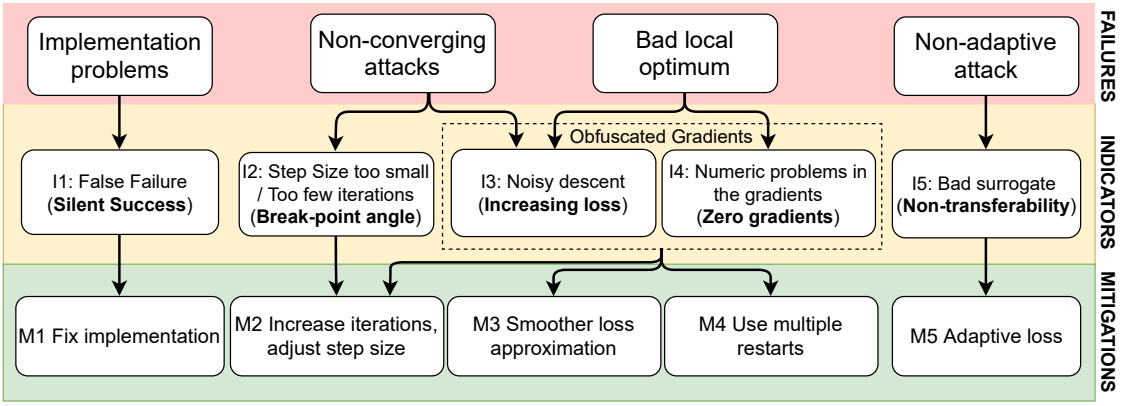


Figure 4.3: Indicators of Attack Failures. The top row lists the four general failures encountered in gradient-based attacks. The second row lists the Indicators of Attack failures we propose, and the last row depicts possible mitigations that can be applied.

no gradient all around them. Another reason might be triggered by the choice of the initialization point itself (line 1 of Algorithm 1), which leads the optimizer into a region where no adversarial examples can be found. In other words, a poor initialization might prevent the algorithm from properly exploring some zone of the landscape that might produce better solutions. This problem has been detected by the analysis conducted by Tramer et al. (2020) against the defense proposed by Pang et al. (2019), where a different initialization point leads the attack to find a better solution.

***F_4*: Non-adaptive attack.** In this failure, the loss function that the attacker optimizes does not take into account the actual defense that has been evaluated, i.e. the objective does not match the actual loss of the target system, and this is caused by a bad choice of the surrogate model (line 2 of Algorithm 1), as also shown in Fig. 4.2d. This issue manifests when either the attack is computed on an undefended model, and later tested against the defense, or the target model is not differentiable and the surrogate is not a correct approximation of the differentiable function. Since we consider both cases, we differ from the literature, where the term *non-adaptive* has been used only for attacks that were not specifically designed to target a given defense (Tramer et al., 2020). We thus restrict our definition of adaptive *specifically* to the case in which the *attacker's* loss goes down (i.e., the attack is being correctly optimized), but the *defender's* loss does not (i.e., despite the attack working on the surrogate, it does not work on the actual defended model). An example of this failure is found in the defense proposed by Yu et al. (2019), where the attack has been computed against the undefended model, and then evaluated against the defense later.

To maximize the likelihood of creating successful attacks and hence avoiding such failures, current recommendations (Carlini et al., 2019; Tramer et al., 2020) suggest to (i) select the strongest attacks against the model that is being tested; (ii) state the

precise threat model being considered; (iii) select the correct hyperparameters for the attack being used; and (iv) compute charts to understand how the attacks behave by varying the size of the perturbation. Indeed useful, these are only qualitative recommendations that require *ad-hoc* inspection of each failed attack.

4.3.2 Indicators of Attack Failure

Above we list several problems that can occur during adversarial example defense evaluations, however, it is not clear how one might go about checking for their presence. The existing literature documents crude techniques to diagnose some failures (e.g., that applying more iterations of an attack should increase the attack success rate) but most recent defense evaluations pass all of these basic sanity checks.

To further complement evaluations we now describe our Indicators of Attack Failures (IoAF): actionable tests designed to help an analyst debug a failing attack. Each of these tests outputs a value bounded between 0 and 1, where values towards 1 imply the presence of the failure described by the test. Along with the indicators to check the attacks, we also introduce a metric that can be used to detect when a model is obfuscating the gradients, hence making the optimization implicitly difficult for gradient-based attacks.

Informed by the results of the indicators, we propose potential mitigations that can resolve the presence of the detected failure. An overview of such an approach can be appreciated in Fig. 4.3, where we connect failures with the indicators that quantify them, along with possible mitigations.

***I_1*: Silent Success.** This indicator is designed as a binary flag that triggers when the attack is failing, but a legitimate adversarial example is found inside the attack path, as described by the implementation problem failure (F_1). While it is true that this is a zeroth-order and trivial strategy that developers can apply, we find that it is missing in most of the state-of-the-art evaluations and implementations.

***I_2*: Break-point angle.** This indicator is designed to quantify the non-convergence of the attack (F_2) caused by the choice of too small hyperparameters. Note that this indicator checks only if the loss stabilizes to a certain value by inspecting if the optimization trend has a specific shape of the curve, without looking at the specific value reached, which can be different for every model. We normalize the loss along the attack path and the iteration, to fit the loss in the domain $[0, 1] \times [0, 1]$, and, ideally, a well-converged loss should approximate a triangle in that domain, as shown in Fig. 4.4. To create that triangle, we connect the first and the last point in the loss curve, and we conclude the shape by considering the point

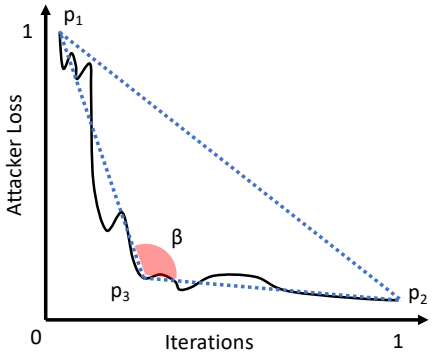


Figure 4.4: I_2 indicator.

of the loss curve that is further to such conjunction. Thus, the three points that define the triangle are the initial point used for gradient descent $p_1 = (0, \mathcal{L}(\boldsymbol{x}, y; \theta))$, the final adversarial example returned $p_2 = (1, \mathcal{L}(\boldsymbol{x} + \delta_n, y; \theta))$ and a third point $p_3(\frac{b}{n}, \mathcal{L}(\boldsymbol{x} + \delta_b, y; \theta))$. The index b is the index of the point in the attack path whose loss is further away from the line $p_2 - p_1$, i.e. the break-point of the loss curve, and n is the total number of iterations of the attack. We are interested in the amplitude of the basis β angle since it is the one that characterizes the shape of the triangle: when $\beta \approx \pi$, the triangle is flat, implying that the loss is still decreasing. For this reason, the indicator computes $1 - |cos\beta|$, matching such intended behavior. On the other hand, this indicator is close to 0 when the triangle is close to being right, hence $\beta \approx \frac{\pi}{2}$.

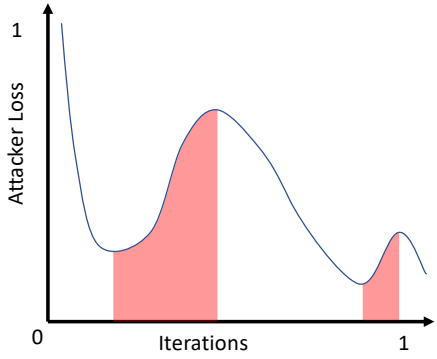


Figure 4.5: I_3 indicator.

***I_3*: Increasing loss.** This indicator is designed to quantify either the non-convergence of the attack (F_2), or the inability of converging to a good local optimum (F_3), both caused by the presence of noisy gradients. In this case, the loss of the attack does not decrease smoothly as the number of iterations grows, typically exhibiting an oscillating behavior, as reported in Fig. 4.5. To characterize such behavior, we first normalize the loss of the attack and the iterations as done in I_2. We then consider only the iterations for which the loss increases, and compute the corresponding area under the curve, as shown in Fig. 4.5. Accordingly, this indicator is 0 only when the attack loss monotonically decreases, while it tends to 1 when the loss increases across iterations (rather than decreasing). Note that in this case, as no formal methods can state if a point is the global minimum or maximum of a non-convex/non-linear function, we rely on the fact that, if the loss is increasing over the iterations, the final solution is very likely not to be a good local optimum. Moreover, since this indicator characterizes the path followed while optimizing, which is different from attack to attack, the value of this indicator is not comparable across different attacks.

***I_4*: Zero gradients.** This indicator is designed to quantify the bad-local optimum failure (F_3), caused by the absence of gradient information. For this reason, we compute how many times, along the attack path, the gradients of the loss function are zero: $\frac{1}{n+1}\sum_{i=0}^{n} \mathbb{1}_{\|\nabla_{\boldsymbol{x}+\delta_i} L\|=0}$. This indicator is close to 1 when most of the norms of the gradient are 0, causing the attack step to fail. Here we check if the product of the gradient and step size (line 5 of Algorithm 1) is zero by checking the norm of the computed gradient (ignoring the trivial case where the step size is zero).

***I_5*: Non-transferability.** This indicator is designed to quantify the non-adaptive failure (F_4), by measuring if the optimized attack fails against the real target model while succeeding against the surrogate one. If the attack transfers successfully, the

indicator is set to 0, otherwise it is set to 1. Note this is different from standard transfer attacks, that transfer adversarial examples from an unrelated source model and so the success rate is often low. In this case, we are considering a surrogate that is the *same* model as the target but is neglecting the defense mechanism. One example of this is the case where the defense is composed of a classifier model that performs the original classification task and a detector model that is able to identify an adversarial example, even if correctly optimized, crafted with the model that performs classification. In this case, the attacker should change the objective of the optimization by also enforcing the detection to not be triggered.

Slope: a specific metric to measure gradient obfuscation. As a follow-up work of the indicators, we designed an additional metric to detect the presence of gradient obfuscation. This metric is different from I_3, and I_4, as it characterizes the loss being used and the model under attack, rather than the single optimization path. The underlying idea of the *Slope* metric is to compare the expected loss after one gradient update of size η against the actual observed loss after the update. To this end, we consider one normalized PGD step of size η, for which the expected loss increment is obtained by solving this linearized problem:

$$\max_{\|\boldsymbol{\delta}\|_p \leq \eta} \mathcal{L}(\boldsymbol{x}+\boldsymbol{\delta}; y; \boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{x}, y; \boldsymbol{\theta}) \approx \max_{\|\boldsymbol{\delta}\|_p \leq \eta} \boldsymbol{\delta}^\top \underbrace{\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, y; \boldsymbol{\theta})}_{\boldsymbol{g}} = \eta \|\boldsymbol{g}\|_q \,, \tag{4.8}$$

where $\|\cdot\|_q$ is the dual norm of $\|\cdot\|_p$. This equation tells us that the loss increment depends on the dual norm of the loss gradient and the step size η, as the optimal perturbation $\boldsymbol{\delta}^\star$ is found by aligning $\boldsymbol{\delta}$ with the loss gradient $\boldsymbol{g}$ depending on the given norm; e.g., $\boldsymbol{\delta}^\star = \eta \operatorname{sign}(\boldsymbol{g})$ when $p = \infty$, and $\boldsymbol{\delta}^\star = \eta \boldsymbol{g} / ||\boldsymbol{g}||_2$ when $p = 2$.
Based on the aforementioned first-order approximation, we define the *Slope* metric P evaluated at $\boldsymbol{x}$ as the ratio between the estimated loss increment and the actual one:

$$P(\boldsymbol{x}) = \frac{\eta \|\boldsymbol{g}\|_q}{\mathcal{L}(\boldsymbol{x}+\boldsymbol{\delta}, y; \boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{x}, y; \boldsymbol{\theta})}. \tag{4.9}$$

The ideal scenario would be $P(\boldsymbol{x}) \geq 1$, where the approximation is consistent with the actual loss increment. If $0 < P(\boldsymbol{x}) < 1$, the approximation is following a good direction, but it could be improved since the actual loss increment is higher than the computed approximation. However, if $P(\boldsymbol{x}) \leq 0$ means that either the loss can not be increased, or the step is performed in the opposite direction, thus resulting in a decrease in the attacker loss. Such is the case when the attacker should rethink or debug their strategy since it is not being correctly optimized.
The Slope metric is calculated sample-wise, and we can also compute the mean value over many observations $\boldsymbol{x}_0, ..., \boldsymbol{x}_n$ as

$$\bar{P} = \frac{1}{n} \sum_{i=1}^{n} P(\boldsymbol{x}_i). \tag{4.10}$$

This average acts as a global measure for the ease of increasing the attacker loss against a particular model.

4.3.3 Mitigate the Failures of Security Evaluations

To set up the evaluation, the attacker should choose a target model to attack, and decide on a surrogate for this model (or use the model unchanged if it is already smoothly differentiable). Then, they must choose an attack and its hyperparameters, and run the attack against the surrogate model they chose. If the attack succeeds then nothing needs to be done. If the attack fails, however, the result can be inspected using our framework to determine the causes of the failures, and hence apply the proposed mitigations. For each point, the attacker should check the feedback of the indicators and mitigate accordingly the detected failures.

***M*$_1$: Fix the implementation.** If I_1 is active, the attack is considered failed, but there exists an adversarial point inside the computed path that satisfies the attack objective. Hence, the resulting robust accuracy must be lowered to reflect this patch accordingly. Also, the attacker would want to run again their evaluations using another library, or a patched version of the same attack.

***M*$_2$: Tune the hyperparameters.** If I_2 activates, it means that the optimization can be improved, and hence both the step size and iteration hyperparameters can be increased. Otherwise, if I_3 activates, the attack should consider a smaller step size, since the loss might be overshooting local minima.

***M*$_3$: Use a different loss function.** If I_3 activates, and the decrement of the step size did not work, the attack should change the loss to be optimized (Tramer et al., 2020), preferring one that has a smoother behavior. If I_4 activates, the attack should consider loss functions that do not saturate (e.g. avoid the softmax) (Carlini and Wagner, 2016), or also increase the step size of the attack to avoid regions with zero gradients.

***M*$_4$: Consider different restarts for the attack.** If I_3 or I_4 activates, the attack might also consider repeating the experiments with more initialization points and restarts, as the failure could be the result of added randomness or an unlucky initialization.

***M*$_5$: Perform adaptive attacks.** Lastly, if none of the above is applied, the attack might be optimizing against a bad surrogate model. If I_5 is active, the attack should be repeated by changing the surrogate to better approximate the target, or include the defense inside the attack itself (Tramer et al., 2020). This step implies repeating the evaluation, as the change of the surrogate might trigger other previously-fixed failures.

When attacks fail even after the application of recommended mitigations, it would be easy to assume that the evaluated defense is strong against adversarial attacks. However, the only thing known is that baseline attacks, properly tested, are not working against the defense. Hence, the designer of the defense should try as hard as possible to break the proposed defense with further investigations (Carlini et al., 2019), and by performing sanity checks, e.g., ensuring that the robust accuracy

drops to 0% when the perturbation size is unbounded, or by trying different attack strategies, e.g., using gradient-free attacks, transfer attacks, or attacks designed by specifically reversing the defense mechanism. In the next section, we will provide some helpful guidelines to successfully create transferable adversarial examples, that can be used as alternative optimization strategies to find further weak spots on the decision landscape of a model.

4.4 Creating Transferable Adversarial Attacks

In transfer attacks, the steps in Algorithm 1 are performed with a model $\hat{\boldsymbol{\theta}}$ that is used as an approximation of the different model parametrized by $\boldsymbol{\theta}$.
We discuss here a connection among transferability of evasion attacks, input gradients, and model complexity. First, we will detail how the size of input gradients of a model can be used to understand its intrinsic vulnerability, then we will specify how the complexity of the surrogate used for crafting the adversarial examples affects the quality of the solutions found for evasion attacks. Here, for model complexity, we intend the capacity of a learning algorithm to fit the training data. It is typically controlled by tuning either the number of classifier parameters to be learned or their size (e.g., via regularization).

Transferability of Evasion Attacks. Given an evasion attack point $\boldsymbol{x}'$, crafted against a surrogate learner (parameterized by $\hat{\boldsymbol{\theta}}$), we measure its *transferability* as the loss attained by the target classifier f (parameterized by $\boldsymbol{\theta}$) on that point, i.e., $T = \mathcal{L}(y, \boldsymbol{x} + \hat{\boldsymbol{\delta}}; \boldsymbol{\theta})$. This can be simplified through a linear approximation of the loss function as:

$$T = \mathcal{L}(\boldsymbol{x} + \hat{\boldsymbol{\delta}}, y; \boldsymbol{\theta}) \approxeq \mathcal{L}(\boldsymbol{x}, y; \boldsymbol{\theta}) + \hat{\boldsymbol{\delta}}^\top \nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, y; \boldsymbol{\theta}) \,. \tag{4.11}$$

This approximation may not only hold for sufficiently-small input perturbations. It may also hold for larger perturbations if the classification function is linear or has a small curvature (e.g., if it is strongly regularized).
Under the same linear approximation, for any given point $\boldsymbol{x}, y$, the evasion problem can be rewritten as:

$$\hat{\boldsymbol{\delta}} \in \arg\max_{\|\boldsymbol{\delta}\|_p \leq \epsilon} \mathcal{L}(\boldsymbol{x} + \boldsymbol{\delta}, y; \hat{\boldsymbol{\theta}}) \,, \tag{4.12}$$

i.e. the attacker should look for the local change in the input sample that *linearly* produces the biggest change in the loss. Under the same linear approximation, this corresponds to the maximization of an inner product over an ϵ-sized ball:

$$\max_{\|\boldsymbol{\delta}\|_p \leq \epsilon} \boldsymbol{\delta}^\top \nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, y; \hat{\boldsymbol{\theta}}) = \epsilon \|\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, y; \hat{\boldsymbol{\theta}})\|_q \,, \tag{4.13}$$

where ℓ_q is the dual norm of ℓ_p.
The above problem is maximized as follows:

1. For $p = 2$, the maximum is $\hat{\boldsymbol{\delta}} = \epsilon \frac{\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, y; \hat{\boldsymbol{\theta}})}{\|\nabla_{\boldsymbol{x}} \ell(\boldsymbol{x}, y; \hat{\boldsymbol{\theta}})\|_2}$;
2. For $p = \infty$, the maximum is $\hat{\boldsymbol{\delta}} \in \epsilon \cdot \text{sign}\{\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, y; \hat{\boldsymbol{\theta}})\}$;
3. For $p = 1$, the maximum is achieved by setting the values of $\hat{\boldsymbol{\delta}}$ that correspond to the maximum absolute values of $\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, y; \hat{\boldsymbol{\theta}})$ to their sign, i.e., ± 1, and 0 otherwise.

Substituting the optimal value of $\hat{\boldsymbol{\delta}}$ into Eq. (4.11), we can compute the loss increment $\Delta\mathcal{L} = \hat{\boldsymbol{\delta}}^\top \nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, y; \boldsymbol{\theta})$ under a transfer attack in closed form; e.g., for $p = 2$, it is given as:

$$\Delta\ell = \epsilon \frac{\nabla_{\boldsymbol{x}} \hat{\mathcal{L}}^\top}{\|\nabla_{\boldsymbol{x}} \hat{\mathcal{L}}\|_2} \nabla_{\boldsymbol{x}} \mathcal{L} \leq \epsilon \|\nabla_{\boldsymbol{x}} \mathcal{L}\|_2 \,, \tag{4.14}$$

where, for compactness, we use $\hat{\mathcal{L}} = \mathcal{L}(\boldsymbol{x}, y; \hat{\boldsymbol{\theta}})$ and $\mathcal{L} = \mathcal{L}(\boldsymbol{x}, y; \boldsymbol{\theta})$.
In this equation, the left-hand side is the increase in the loss function in the black-box case, while the right-hand side corresponds to the white-box case. The upper bound is obtained when the surrogate classifier $\hat{\boldsymbol{\theta}}$ is equal to the target $\boldsymbol{\theta}$ (white-box attacks). Similar results hold for $p = 1$ and $p = \infty$ (using the dual norm in the right-hand side).

Intriguing Connections and Transferability Metrics. The above findings reveal some interesting connections among transferability of attacks, model complexity (controlled by the classifier hyperparameters), and input gradients, as detailed below, and allow us to define simple and computationally-efficient transferability metrics.
(1) Size of Input Gradients. The first interesting observation is that transferability depends on the size of the gradient of the loss ℓ computed using the *target* classifier, regardless of the surrogate: the larger this gradient is, the larger the attack impact may be. This is inferred from the upper bound in Eq. (4.14). We define the corresponding metric $S(\boldsymbol{x}, y)$ as:

$$S(\boldsymbol{x}, y) = \|\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, y; \boldsymbol{\theta})\|_q \,, \tag{4.15}$$

where q is the dual of the perturbation norm.
The size of the input gradient also depends on the complexity of the given model, controlled, e.g., by its regularization hyperparameter. Less complex, strongly-regularized classifiers tend to have smaller input gradients, i.e., they learn smoother functions that are more robust to attacks, and vice-versa. In Fig. 4.6 we report an example showing how increasing regularization (i.e., decreasing complexity) for a neural network trained on MNIST89, by controlling its *weight decay*, reduces the average size of its input gradients, improving adversarial robustness to evasion. It is however worth remarking that, since complexity is a model-dependent characteristic,

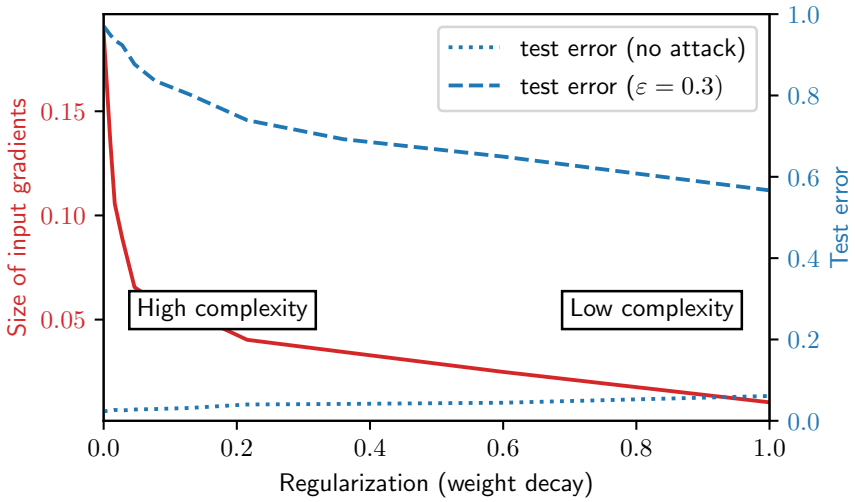


Figure 4.6: Size of input gradients (averaged on the test set) and test error (in the absence and presence of evasion attacks) against regularization (controlled via weight decay) for a neural network trained on MNIST89. Note how the size of input gradients and the test error under attack decrease as regularization (complexity) increases (decreases).

the size of input gradients cannot be directly compared across different learning algorithms; e.g., if a linear SVM exhibits larger input gradients than a neural network, we cannot conclude that the former will be more vulnerable.

Another interesting observation is that, if a classifier has large input gradients (e.g., due to high-dimensionality of the input space and low level of regularization), for an attack to succeed it may suffice to apply only tiny, *imperceptible* perturbations. This explains why adversarial examples against deep neural networks can often only be slightly perturbed to mislead detection, while when attacking less complex classifiers in low dimensions, modifications become more evident.

(2) Gradient Alignment. The second relevant impact factor on transferability is based on the alignment of the input gradients of the loss function computed using the target and the surrogate learners. If we compare the increase in the loss function in the black-box case (the left-hand side of Eq. 4.14) against that corresponding to white-box attacks (the right-hand side), we find that the relative increase in loss, at least for ℓ_2 perturbations, is given by the following value:

$$R(\boldsymbol{x}, y) = \frac{\nabla_{\boldsymbol{x}} \hat{\mathcal{L}}^{\top} \nabla_{\boldsymbol{x}} \mathcal{L}}{\|\nabla_{\boldsymbol{x}} \hat{\ell}\|_2 \|\nabla_{\boldsymbol{x}} \ell\|_2} \,. \tag{4.16}$$

Interestingly, this is exactly the cosine of the angle between the gradient of the loss of the surrogate and that of the target classifier. This is a novel finding which explains why the cosine angle metric between the target and surrogate gradients can well characterize the transferability of attacks, confirming empirical results from

previous work (Liu et al., 2017). For other kinds of perturbation, this definition slightly changes, but gradient alignment can be similarly evaluated. Differently from the gradient size S, gradient alignment is a pairwise metric, allowing comparisons across different surrogate models. These two metrics can be used to understand the intrinsic vulnerability of a model, and if an adversarial attack can transfer well to other scenarios.

In the next chapter, we will report our experimental analysis that supports the main contributions of this thesis.

Chapter 5

Experiments

In this Chapter, we provide experimental evidence to support the contributions presented in Chapter 4. For each of the sections, we introduce the scenario, evaluation criteria, and the experimental protocol, then report the results and discuss them. In the next sections, we present the experiments for the Fast Minimum-norm attack (Sect. 5.1), for the Indicators of Failure and Slope (Sect. 5.2), and for the transferability analysis (Sect. 5.3).

5.1 Fast Minimum-norm Adversarial Attacks

We report here an extensive experimental analysis involving several state-of-the-art defenses and minimum-norm attacks, covering ℓ_0, ℓ_1, ℓ_2 and ℓ_∞ norms. The goal is to empirically benchmark the proposed FMN attack and assess its effectiveness and efficiency as a tool for adversarial robustness evaluation.

5.1.1 Experimental Setup

Datasets. We consider two commonly-used datasets for benchmarking adversarial robustness of deep neural networks, i.e., the MNIST handwritten digits and CIFAR10. We normalize the datasets in the range $[0, 1]$, i.e. we set $\boldsymbol{x}_{\text{lb}} = 0$ and $\boldsymbol{x}_{\text{ub}} = 1$, and then we apply the preprocessing specified in the original implementations of the models. Following the experimental setup in Brendel et al. (2020), we use a subset of 1000 test samples to evaluate the considered attacks and defenses.

Models. We use a diverse selection of models to thoroughly evaluate attacks under different conditions. For MNIST, we consider the following four models:

- *M1*, the 9-layer network used as the undefended baseline model by Papernot et al. (2016b); Carlini and Wagner (2017b);
- *M2*, the robust model by Madry et al. (2018), trained on ℓ_∞ attacks (robustness claim: 89.6% accuracy with $\|\boldsymbol{\delta}\|_\infty \leq 0.3$, current best evaluation: 88.0%);

- *M3*, the robust model by Rony et al. (2019), trained on ℓ_2 attacks (robustness claim: 87.6% accuracy with $\|\boldsymbol{\delta}\|_2 \leq 1.5$); and
- *M4*, the IBP Large Model by Zhang et al. (2020) (robustness claim: 94.3% accuracy with $\|\boldsymbol{\delta}\|_\infty \leq 0.3$).

For CIFAR10, we consider three state-of-the-art robust models from RobustBench (Croce et al., 2020):

- *C1*, the robust model by Madry et al. (2018), trained on ℓ_∞ attacks (robustness claim: 44.7% accuracy with $\|\boldsymbol{\delta}\|_\infty \leq 8/255$, current best evaluation: 44.0%);
- *C2*, the defended model by Carmon et al. (2019) (top-5 in RobustBench), trained on ℓ_∞ attacks and additional unsupervised data (robustness claim: 62.5% accuracy with $\|\boldsymbol{\delta}\|_\infty \leq 8/255$, current best evaluation: 59.5%); and
- *C3*, the robust model by Rony et al. (2019), trained on ℓ_2 attacks (robustness claim: 67.9% accuracy with $\|\boldsymbol{\delta}\|_2 \leq 0.5$, current best evaluation: 66.4%).

Attacks. We compare our algorithm against different state-of-the-art attacks for finding minimum-norm adversarial perturbations across different norms: the Carlini & Wagner (CW) attack (Carlini and Wagner, 2017a), the Decoupling Direction and Norm (DDN) attack (Rony et al., 2019), the Brendel & Bethge (BB) attack (Brendel et al., 2020), and the Fast Adaptive Boundary (FAB) attack (Croce and Hein, 2020a). We use the implementation of FAB from Ding et al. (2019), while for all the remaining attacks we use the implementation available in Foolbox (Rauber et al., 2017, 2020), in which we also implemented ours. All these attacks are defined on the ℓ_2 norm. BB and FAB are also defined on the ℓ_1 and ℓ_∞ norms, and only BB is defined on the ℓ_0 norm. We consider both untargeted and targeted attack scenarios, as defined in Sect. 4.2, except for FAB, which is only evaluated in the untargeted case.[1].

Hyperparameters. To ensure a fair comparison, we perform an extensive hyperparameter search for each of the considered attacks. We consider two main scenarios: tuning the hyperparameters at the *sample-level* and at the *dataset-level*. In the sample-level scenario, we select the optimal hyperparameters separately for each input sample by running each attack 10 to 16 times per sample, with a different hyperparameter configuration or random initialization point each time. In the dataset-level scenario, we choose the same hyperparameters for all samples, selecting the configuration that yields the best attack performance. While sample-level

[1]The reason is that FAB implements a substantially different attack in the targeted case. The targeted version of FAB aims to find a closer untargeted misclassification by running the attack a number of times, each time targeting a different candidate class, and then selecting the best solution (Croce and Hein, 2020a,b)

tuning provides a fairer comparison across attacks, it is more computationally demanding and less practical than dataset-level tuning. In addition, the latter allows us to understand how robust attacks are to suboptimal hyperparameter choices. We select the hyperparameters to be optimized for each attack as recommended by the corresponding authors (Brendel et al., 2020; Carlini and Wagner, 2017b; Croce and Hein, 2020a; Rony et al., 2019). The hyperparameter configurations considered for each attack are detailed below. For attacks that are claimed to be robust to hyperparameter changes, like BB and FAB, we follow the recommendation of using a larger number of random restarts rather than increasing the number of hyperparameter configurations to be tested. In addition, as BB requires being initialized from an adversarial starting point, we initialize it by randomly selecting a sample either from a different class (in the untargeted case) or from the target class (in the targeted case). Finally, as each attack performs operations with different levels of complexity within each iteration, possibly querying the model multiple times, we set the number of steps for each attack such that at least $1,000$ forward passes (i.e., *queries*) are performed. This ensures a fairer comparison also in terms of the computational time and resources required to execute each attack.
In the experiments, we will use the following attacks, introduced in Sect. 3.1.1:

- *CW.* We set the number of binary-search steps to 9, and the maximum number of iterations to 250, to ensure that at least $1,000$ queries are performed. We also set different values for $c, \eta \in \{10^{-3}, 10^{-2}, 10^{-1}, 1\}$.

- *DDN.* We consider initial values of $\epsilon_0 \in \{0.03, 0.1, 0.3, 1, 3\}$, and run the attack with a different number of iterations $K \in \{200, 1000\}$, as this affects the size of each update on $\boldsymbol{\delta}$.

- *BB.* We consider different values for $\rho \in \{10^{-3}, 10^{-2}, 10^{-1}, 1\}$, while we fix the number of steps to 1000. We run the attack 3 times by considering different initialization points and eventually retain the best solution.

- *FAB.* As suggested by Croce and Hein (2020a), we tune $\alpha_{\max} \in \{0.1, 0.05\}$ and $\eta \in \{1.05, 1, 3\}$. We consider 3 different random initialization points, and run the attack for 500 steps each time, eventually selecting the best solution.

- *FMN.* We run FMN for $K = 1000$ steps, using $\gamma_0 \in \{0.05, 0.3\}$, $\gamma_K = 10^{-4}$, and $\alpha_K = 10^{-5}$. For ℓ_0, ℓ_1, and ℓ_2, we set $\alpha_0 \in \{1, 5, 10\}$. For ℓ_∞, we set $\alpha_0 \in \{10^1, 10^2, 10^3\}$, as the normalized ℓ_2 step yields much smaller updates in the ℓ_∞ norm. For each hyperparameter setting we run the attack twice, starting from (i) the input sample and (ii) an adversarial point.

Evaluation criteria. We evaluate the attacks with four different criteria: (i) *perturbation size* and (ii) *robustness to hyperparameter selection*, measured as the median $\|\boldsymbol{\delta}^\star\|_p$ on the test set (for a fixed budget of Q queries and for sample- and

dataset-level hyperparameter tuning); (iii) *execution time*, measured as the average time spent per query (in milliseconds); and (iv) *convergence speed*, measured as the average number of queries required to converge to a good-enough solution (within 10% of the best value found at $Q = 1000$). When computing the median, we follow the evaluation in Brendel et al. (2020): the perturbation size is set to 0 if a clean sample is misclassified, while it is set to ∞ when the attack fails (no adversarial is found). The median perturbation size thus represents the value for which 50% of the samples evade a particular model.

5.1.2 Experimental Results

Query-distortion (QD) curves. To evaluate each attack in terms of perturbation size under the same query budget Q, we use the so-called QD curves introduced by Brendel et al. (2020). These curves report, for each attack, the median value of $\boldsymbol{\delta}^\star$ as a function of the number of queries Q. For each given Q value, the optimal $\boldsymbol{\delta}^\star$ for each point is selected among the different attack executions (i.e., using different hyperparameters and/or initialization points, as described before).
In Fig. 5.1-5.4, we report the QD curves for the MNIST and CIFAR10 models, in both targeted and untargeted scenarios.
On the MNIST dataset, our attacks generally reach smaller norms with fewer queries, with the exception of M2 (Figs. 5.1-5.2), where it seems to reach convergence more slowly than BB in ℓ_0 and ℓ_∞. In ℓ_2, the CW attack is the slowest to converge, due to the need to carefully tune the weighting term.
On the CIFAR10 dataset (Figs. 5.3-5.4), our attack always rivals or outperforms the others, with the notable exception of DDN for the ℓ_2 norm, which sometimes finds smaller perturbations more quickly, as also shown in Table 5.3.
It is worth noting that our attack attains comparable results in terms of perturbation size across all norms, while significantly outperforming FAB and BB in the ℓ_1 case. It typically requires also fewer iterations than the other attacks to converge. While the QD curves show the complete behavior of each attack as Q increases, a more compact and thorough summary of our evaluation is reported below, according to the four evaluation criteria described in Sect. 5.1.1.

Perturbation size. Table 5.1 reports the median value of $\|\boldsymbol{\delta}^\star\|$ at $Q = 1000$ queries (i.e., the last value from the query-distortion curve), for all models, attacks and norms. The values obtained with sample-level hyperparameter tuning confirm that our attack can find smaller or comparable perturbations with those found by the competing attacks, in most of the untargeted and targeted cases, and that the biggest margin is achieved in the ℓ_1 case. FMN is only slightly worse than DDN and BB in a few cases, including ℓ_2-DDN on M4 and ℓ_∞-BB on M2 and M4. The reason may be that these robust models exhibit noisy gradients and flat regions around the clean input samples, hindering the initial optimization steps of the FMN attack.

Robustness to hyperparameter selection. The values reported in the lower

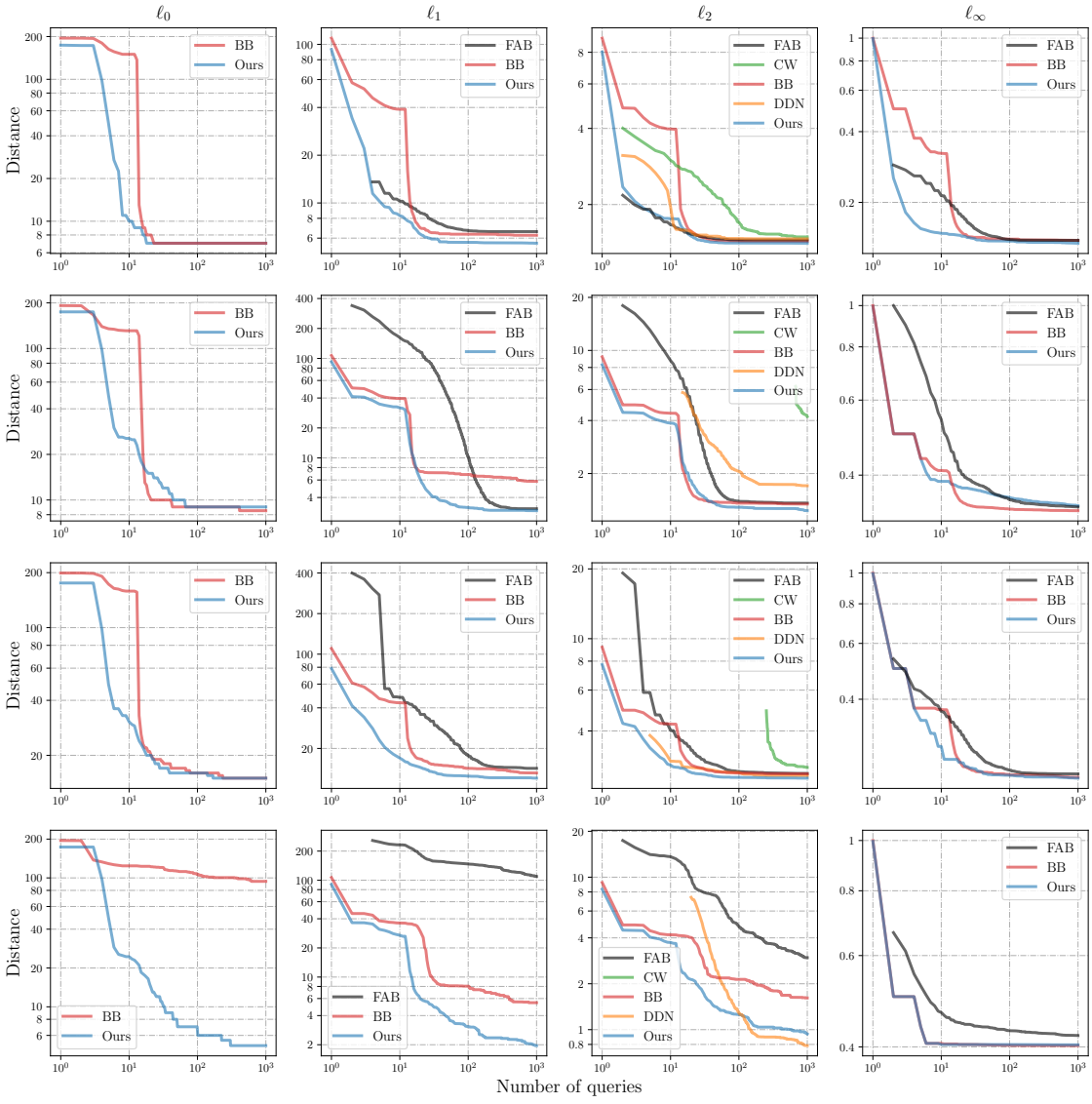


Figure 5.1: Query-distortion curves for untargeted (U) attacks on the M1, M2, M3, and M4 MNIST models.

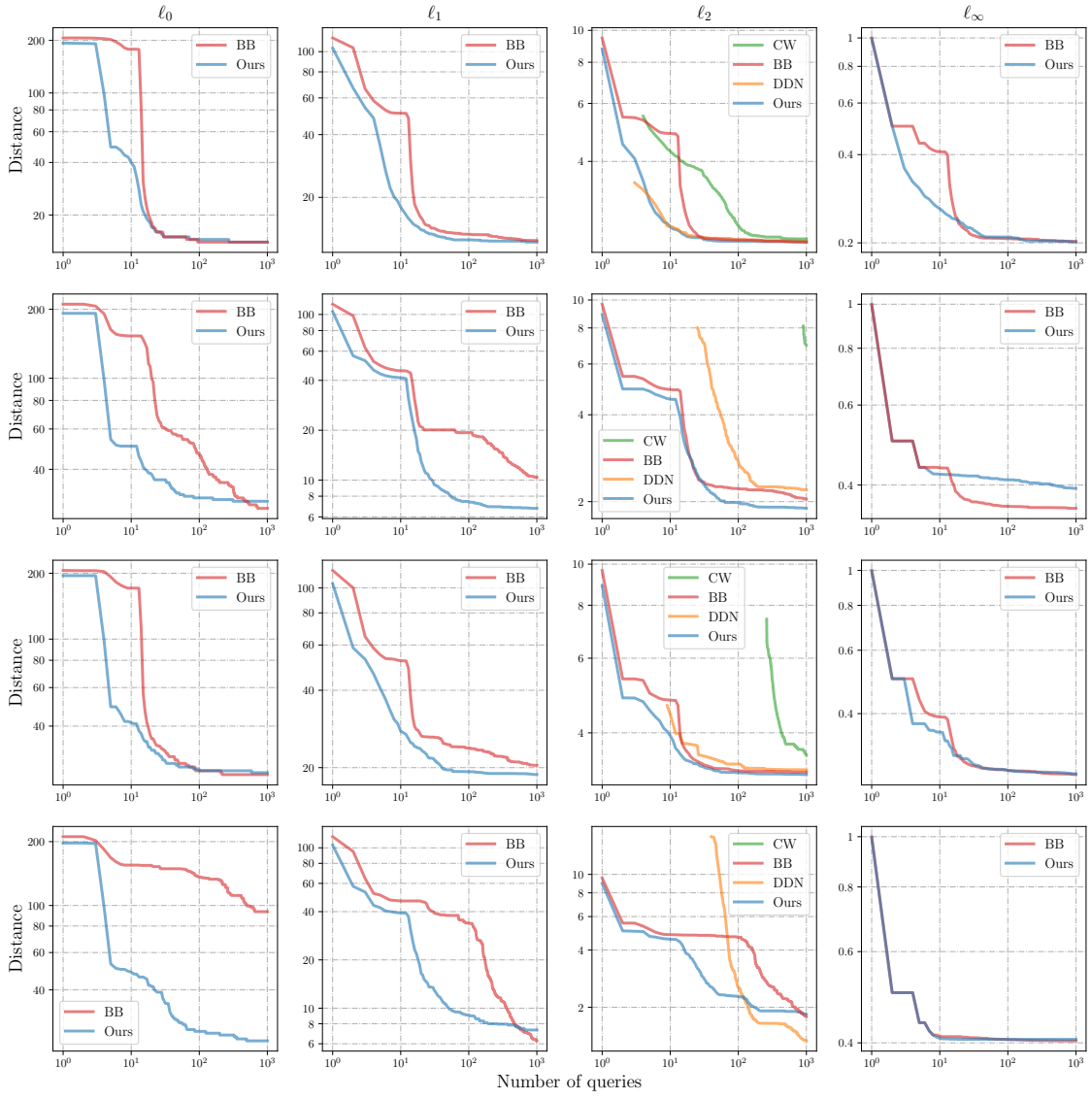


Figure 5.2: Query-distortion curves for targeted (T) attacks on the M1, M2, M3, and M4 MNIST models.

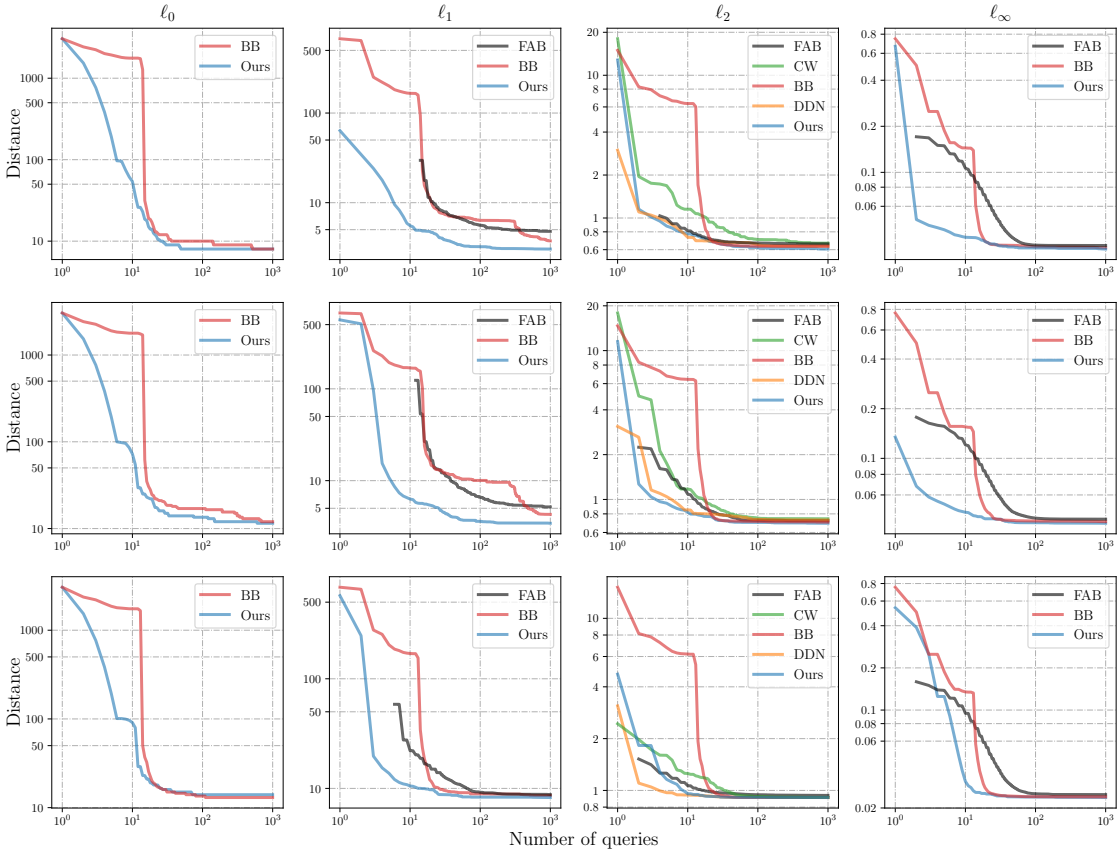


Figure 5.3: Query-distortion curves for untargeted (U) attacks on the C1 (*top*), C2 (*middle*), and C3 (*bottom*) CIFAR10 models.

part of Table 5.1 show that, when using dataset-level hyperparameter tuning, FMN outperforms the other attacks in a much larger number of cases. This shows that FMN is more robust to hyperparameter changes, while other attacks like ℓ_0- and ℓ_1-BB suffer when using the same hyperparameters for all samples.

Execution time. The average runtime per query for each attack-model pair, measured on a workstation with an NVIDIA GeForce RTX 2080 Ti GPU with 11GB of RAM, can be found in Table 5.2. The results show that our attack is up to 2-3 times faster, with the exception of DDN in the ℓ_2 case. This is however compensated by the fact that FMN finds better solutions. The advantage is that our attack avoids costly inner projections as in BB and FAB. FMN is slightly less time-efficient than DDN and CW, as it simultaneously updates the adversarial point and the norm constraint. In particular, the update on the constraint may initially require computing the norm of the gradient $\boldsymbol{g}$ (line 6 in Algorithm 2), which increases the runtime of our attack. FAB computes a similar step, but for all the output classes, which hinders its scalability to problems with many classes.

Convergence speed. To get an estimate of the convergence speed, we measure the

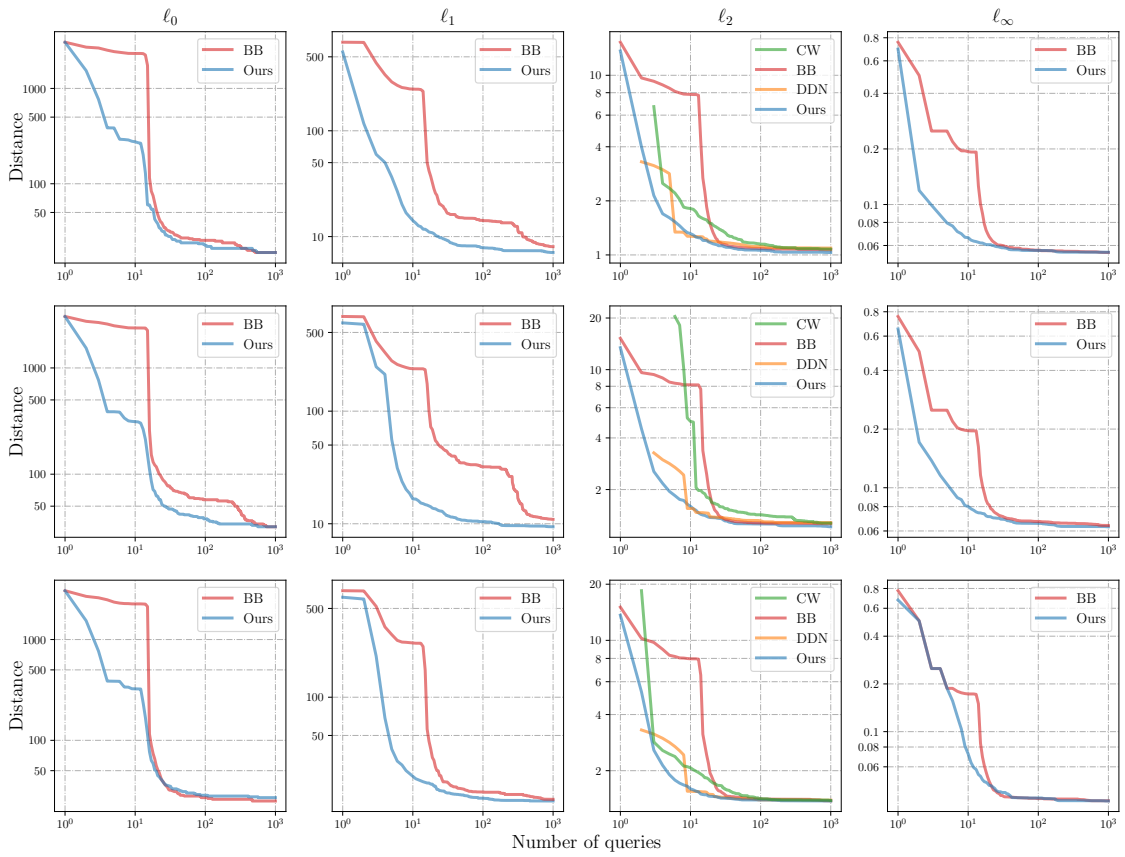


Figure 5.4: Query-distortion curves for targeted (T) attacks on the C1 (*top*), C2 (*middle*), and C3 (*bottom*) CIFAR10 models.

Table 5.1: Median $\|\boldsymbol{\delta}^\star\|_p$ value at $Q = 1000$ queries for targeted and untargeted attacks, with sample-level and dataset-level hyperparameter tuning.

		MNIST								**CIFAR10**					
		Untargeted				*Targeted*				*Untargeted*			*Targeted*		
	Model	M1	M2	M3	M4	M1	M2	M3	M4	C1	C2	C3	C1	C2	C3
		Sample-level Hyperparameter Tuning													
ℓ_0	BB	**7**	**8**	**15**	94	**14**	27	**24**	93	**8**	12	**13**	**19**	**32**	**25**
	Ours	**7**	9	**15**	**5**	**14**	**20**	**24**	**23**	**8**	**11**	14	**19**	**32**	27
ℓ_1	FAB	6.60	3.08	14.23	109.4	-	-	-	-	4.79	5.17	8.79	-	-	-
	BB	6.26	5.81	13.16	5.44	12.42	10.38	20.41	**6.25**	3.75	4.29	8.62	8.04	10.93	15.71
	Ours	**5.57**	**2.95**	**12.04**	**1.96**	**12.20**	**6.75**	**18.79**	7.31	**3.04**	**3.43**	**8.26**	**7.07**	**9.40**	**15.24**
ℓ_2	FAB	1.45	1.36	2.62	2.97	-	-	-	-	0.66	0.72	0.94	-	-	-
	CW	1.49	4.22	2.78	-	2.33	6.97	3.54	-	0.67	0.74	**0.91**	1.08	1.27	1.38
	BB	1.43	1.34	2.61	1.61	**2.27**	2.04	3.23	1.79	0.63	0.70	**0.91**	1.07	1.26	1.38
	DDN	1.46	1.71	2.56	**0.79**	2.29	2.20	3.27	**1.33**	0.64	0.73	**0.91**	1.09	1.29	1.39
	Ours	**1.41**	**1.23**	**2.50**	0.94	2.28	**1.89**	**3.19**	1.85	**0.61**	**0.69**	**0.91**	**1.03**	**1.21**	**1.38**
ℓ_∞	FAB	.138	.337	.233	.421	-	-	-	-	.033	.043	.025	-	-	-
	BB	.138	**.330**	.227	**.402**	.202	**.355**	**.271**	**.403**	.032	.041	**.024**	**.055**	.064	**.037**
	Ours	**.134**	.339	**.226**	.404	**.201**	.389	.272	.406	**.032**	**.040**	**.024**	**.055**	**.063**	**.037**
		Dataset-level Hyperparameter Tuning													
ℓ_0	BB	12	152	52	145	20	179	39	183	28	44	32	29	65	33
	Ours	**9**	**33**	**18**	**15**	**16**	**48**	**28**	**55**	**11**	**17**	**16**	**25**	**38**	**32**
ℓ_1	FAB	8.66	225.7	163.9	312.3	-	-	-	-	-	-	20.48	-	-	-
	BB	10.60	49.83	17.57	46.99	16.60	53.11	29.89	54.31	7.02	10.20	17.13	11.41	15.26	23.37
	Ours	**7.13**	**4.18**	**13.66**	**4.99**	**13.18**	**8.33**	**21.37**	**12.16**	**4.28**	**4.82**	**9.52**	**8.51**	**10.40**	**17.32**
ℓ_2	FAB	1.54	1.59	2.81	16.30	-	-	-	-	0.77	1.11	1.06	-	-	-
	CW	1.63	5.15	3.71	-	2.50	-	4.72	-	0.86	1.00	0.99	1.36	2.90	1.55
	BB	1.75	1.82	3.02	4.57	2.64	2.59	3.52	5.31	0.86	0.95	1.10	1.25	1.45	1.73
	DDN	**1.47**	2.01	2.62	**1.15**	2.31	2.72	3.36	**1.96**	**0.66**	0.77	0.91	1.11	1.31	1.40
	Ours	1.61	**1.42**	**2.61**	1.56	**2.30**	**2.13**	**3.24**	2.41	0.67	**0.74**	**0.91**	**1.09**	**1.28**	**1.38**
ℓ_∞	FAB	.148	.365	.248	.900	-	-	-	-	.038	.052	.029	-	-	-
	BB	.159	**.336**	.243	.409	.223	**.361**	.280	.477	.044	.054	.029	.059	.074	.042
	Ours	**.140**	.357	**.233**	**.408**	**.206**	.426	**.277**	**.434**	**.034**	**.042**	**.024**	**.057**	**.066**	**.037**

number of queries required by each attack to reach a perturbation size that is within 10% of the value found at $Q = 1000$ queries (the lower the better). Results are shown in Table 5.3. Our attack converges on par with or faster than all other attacks for almost all models, often requiring only half or a fifth as many queries as the state of the art. Exceptions are MNIST and CIFAR10 challenge models (M2 and C1) for ℓ_2 and ℓ_∞, where BB and DDN occasionally converge faster. FMN rarely needs more than 100 steps, reaching the minimal perturbation after only 10-30 queries on many datasets, models, and norms.

Table 5.2: Average execution time (milliseconds / query) for each attack-model pair.

		MNIST								**CIFAR10**					
		Untargeted				*Targeted*				*Untargeted*			*Targeted*		
	Model	M1	M2	M3	M4	M1	M2	M3	M4	C1	C2	C3	C1	C2	C3
ℓ_0	BB	10.76	11.85	10.19	12.02	60.88	62.17	62.31	57.74	46.51	50.31	50.43	99.71	105.28	103.53
	Ours	**5.15**	**4.87**	**5.87**	**9.70**	**5.14**	**4.75**	**5.85**	**9.71**	**26.26**	**30.54**	**30.89**	**26.13**	**30.26**	**30.81**
ℓ_1	FAB	9.38	8.88	12.61	36.00	-	-	-	-	84.04	108.91	108.64	-	-	-
	BB	6.73	7.03	7.31	12.50	43.25	43.54	43.69	43.86	32.56	37.40	37.59	68.99	73.33	74.03
	Ours	**5.43**	**5.14**	**6.10**	**9.35**	**5.44**	**5.10**	**6.09**	**9.35**	**27.34**	**31.17**	**31.18**	**26.00**	**30.98**	**31.03**
ℓ_2	FAB	10.22	10.13	13.45	36.72	-	-	-	-	84.27	109.43	108.87	-	-	-
	CW	4.22	4.09	5.17	10.07	4.23	4.14	5.15	10.06	25.90	31.32	31.31	25.78	31.32	31.30
	BB	4.44	4.15	5.03	12.38	26.20	26.76	27.24	31.00	26.64	31.82	31.90	48.74	54.35	54.07
	DDN	**3.42**	**3.33**	**4.30**	**8.59**	**3.42**	**3.35**	**4.32**	**8.60**	**24.14**	**29.62**	**29.48**	**23.61**	**29.61**	**29.52**
	Ours	4.46	4.42	5.48	9.15	4.50	4.44	5.47	9.09	24.88	30.22	30.08	25.39	30.21	30.04
ℓ_∞	FAB	10.85	10.61	14.05	36.23	-	-	-	-	84.62	109.83	109.57	-	-	-
	BB	14.26	16.36	13.51	15.44	38.61	38.87	36.39	34.85	61.34	62.36	62.63	83.70	87.64	88.90
	Ours	**4.25**	**4.33**	**5.30**	**9.17**	**4.33**	**4.23**	**5.31**	**9.10**	**24.84**	**30.15**	**30.01**	**24.78**	**30.19**	**30.03**

Table 5.4: Success rate (%) of FMN against PGD on ImageNet models.

		ResNet18	VGG
ℓ_1 ($\epsilon = 1.0$)	PGD	31.4	30.4
	FMN	**38.4**	**39.8**
ℓ_2 ($\epsilon = 0.15$)	PGD	61.7	61.4
	FMN	**65.8**	**66.2**
ℓ_∞ ($\epsilon = 4 \cdot 10^{-4}$)	PGD	51.0	**49.0**
	FMN	**55.2**	**49.0**

Robust accuracy. Despite our attack being not tailored to target specific defenses, and our evaluation restricted to a subset of the testing samples, it is worth remarking that the robust accuracies of the models against our attack are aligned with that reported in current evaluations, with the notable exception of C3, where our attack can decrease robust accuracy from 67.9% to 65.5%.

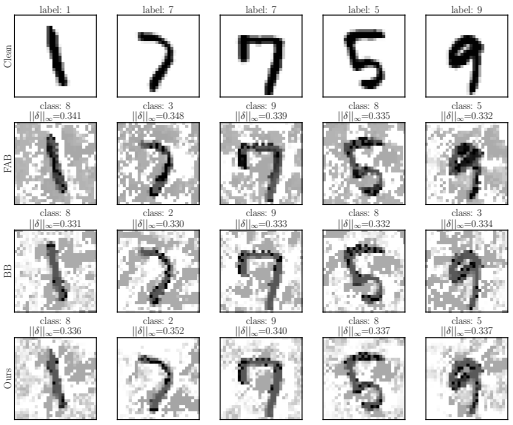
Experiments on ImageNet. We conclude our experiments by running an additional comparison between FMN and a widely-used maximum-confidence attack, i.e., the Projected Gradient Descent (PGD) attack (Madry et al., 2018), on two pre-trained ImageNet models (i.e., ResNet18 and VGG16), considering ℓ_1, ℓ_2 and ℓ_∞ norms. The hyperparameters are tuned at the *dataset-level* using 20 validation samples. For FMN, we fix the hyperparameters as discussed before, and only tune $\alpha_0 \in \{0.1, 1, 2, 8\}$, without using adversarial initialization. For PGD, we tune the step size $\alpha \in \{0.001, 0.01, 0.1, 1, 2, 8\}$. We run both attacks for $Q = 1,000$ queries on a separate set of $1,000$ samples. The success rates of both attacks at fixed ϵ values are reported in Table 5.4. The results show that FMN outperforms or equals PGD in all norms.

Table 5.3: Number of queries required by each attack to reach a perturbation size that is within 10% of the value obtained at $Q = 1000$.

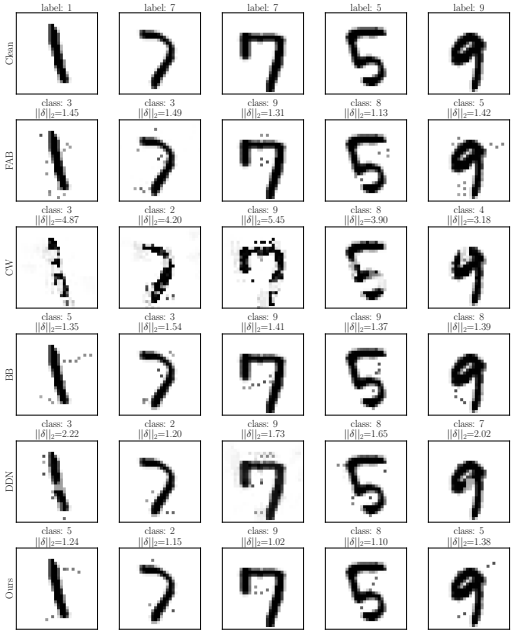
		MNIST								**CIFAR10**					
		Untargeted				*Targeted*				*Untargeted*			*Targeted*		
	Model	M1	M2	M3	M4	M1	M2	M3	M4	C1	C2	C3	C1	C2	C3
ℓ_0	BB	**22**	**43**	68	**114**	30	443	71	376	497	372	58	384	500	85
	Ours	**22**	82	**38**	182	**27**	**165**	**46**	**145**	**48**	**71**	**37**	**271**	**146**	**70**
ℓ_1	FAB	44	**242**	152	569	-	-	-	-	124	220	72	-	-	-
	BB	24	314	83	**391**	45	614	233	722	674	570	34	526	464	206
	Ours	**21**	363	**34**	631	**25**	**243**	**37**	**336**	**48**	**85**	**31**	**89**	**130**	**38**
ℓ_2	FAB	14	60	40	532	-	-	-	-	18	28	14	-	-	-
	CW	110	799	335	-	100	913	469	-	67	39	33	56	144	42
	BB	20	**24**	20	337	21	**61**	20	692	22	23	22	26	27	29
	DDN	**12**	136	**15**	474	12	149	26	670	**13**	**20**	**4**	**18**	**19**	18
	Ours	16	94	16	**190**	**11**	136	**16**	**188**	28	23	7	25	29	**13**
ℓ_∞	FAB	36	50	44	11	-	-	-	-	50	50	54	-	-	-
	BB	19	17	**20**	**5**	**24**	17	**22**	**5**	**20**	24	21	27	33	**29**
	Ours	**9**	**10**	22	**5**	27	**8**	26	**5**	22	**15**	**14**	**20**	**29**	34

In this section, we first show adversarial examples obtained by different ℓ_p attacks on MNIST and CIFAR10 data for visual comparison. These examples highlight the different behavior exhibited by each attack. We then report the query-distortion curves for all datasets, models, and attacks used in this paper, showing that our attack outperforms current attacks on the ℓ_1 norm and rivals their performance on other norms, while typically converging with much fewer queries.

Adversarial examples. In Figs. 5.5-5.6, we report adversarial examples generated by all attacks against model M2 and C2, respectively, on MNIST and CIFAR10 datasets, in the untargeted scenario. As expected, for each metric we find different levels of sparsity in the applied perturbation. The clean samples and the original label are displayed in the first row of each figure. In the remaining rows, we show the perturbed sample along with the predicted class and the corresponding norm of perturbation $\|\boldsymbol{\delta}^\star\|_p$. It is worth noting that the output class for different untargeted attacks is not always the same, which might sometimes explain differences in the perturbation sizes. An example is given in Fig. 5.6b, where the sample in the fourth column, labeled as "ship", is perturbed by most of the attacks towards the class "airplane", while in our case it outputs the class "dog" with a much smaller distance. The presence of so many different local optima provides an additional indication that many blind spots exist especially in high-dimensional spaces, and that improving adversarial robustness in this setting remains thus a very challenging problem. On the other hand, the observation that different attacks find different local optima also points out that using a diverse set of attacks can help obtain more reliable adversarial robustness evaluations. For this reason, we firmly believe that FMN can be a valuable, complementary addition to the current arsenal of attacks towards improving the evaluation of adversarial robustness.

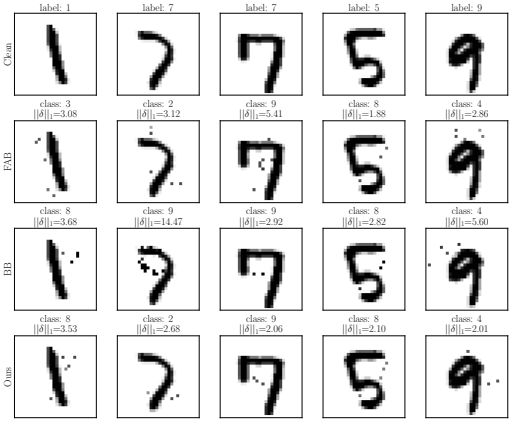


(a) Untargeted ℓ_∞ attacks against M2 (Madry et al., 2018)

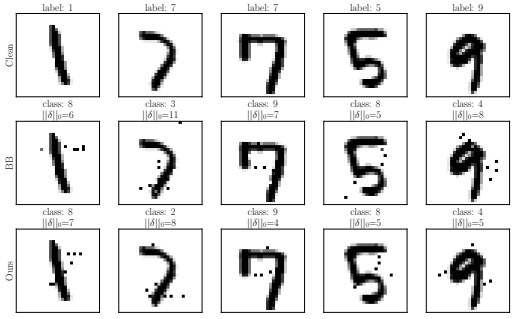


(b) Untargeted ℓ_2 attacks against M2 (Madry et al., 2018).

.

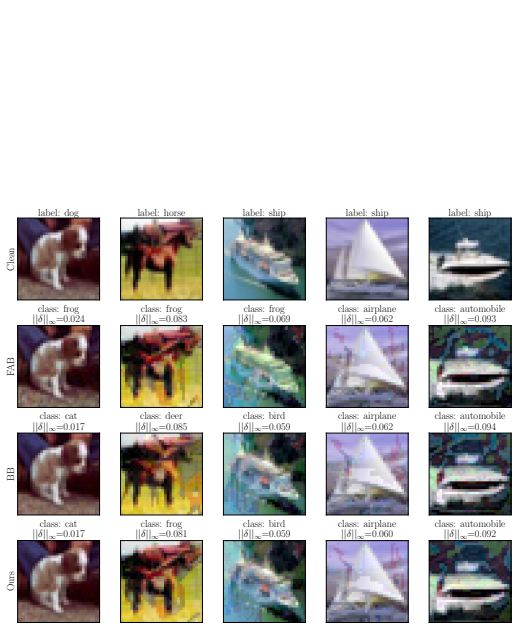


(c) Untargeted ℓ_1 attacks against M2 (Madry et al., 2018).

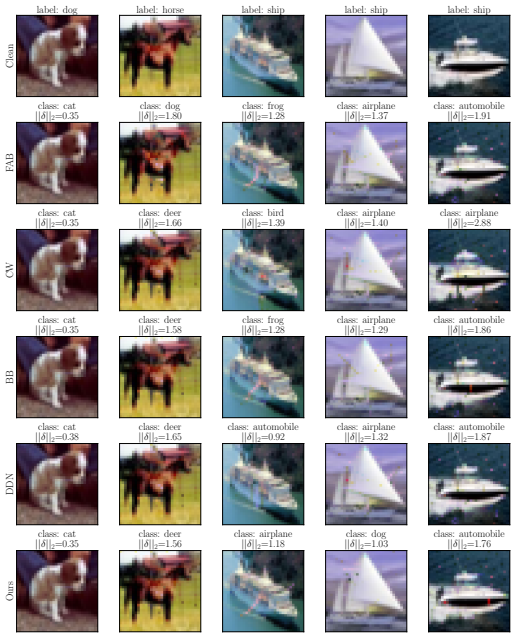


(d) Untargeted ℓ_0 attacks against M2 (Madry et al., 2018).

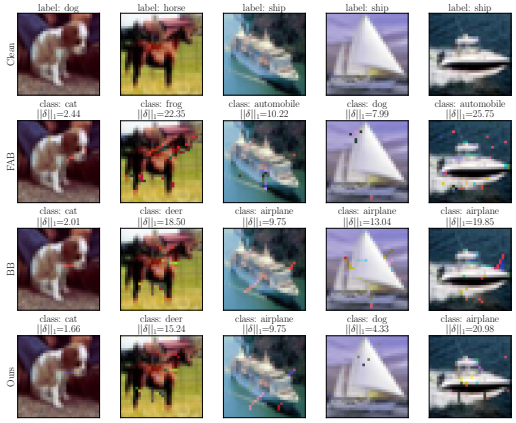
Figure 5.5: Adversarial examples on MNIST dataset.



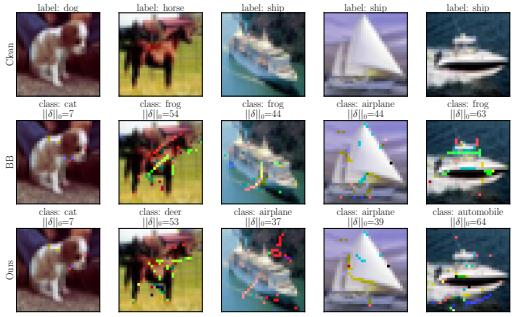
(a) Untargeted ℓ_∞ attacks against C2 (Carmon et al., 2019).



(b) Untargeted ℓ_2 attacks against C2 (Carmon et al., 2019).



(c) Untargeted ℓ_1 attacks against C2 (Carmon et al., 2019).



(d) Untargeted ℓ_0 attacks against C2 (Carmon et al., 2019).

Figure 5.6: Adversarial examples on the CIFAR10 dataset.

5.2 Indicators of Attack Failures

In Sect. 4.3 we presented the Indicators of Attack Failure (IoAF), useful tools for debugging adversarial robustness evaluations. We now exhibit the results of our experiments, by showing the correlation between the feedback of our indicators, and the false sense of security given by badly-evaluated defenses.

Experimental setup. We run our attacks on an Intel® Xeon® CPU E5-2670 v3, with 48 cores, 126 GB of RAM, and equipped with an Nvidia Quadro M6000 with 24 GB of memory. All the attacks and models have been wrapped and run by using the SecML library (Melis et al., 2019). We select four defenses that have been reported as failing, and we show that our indicators would have detected such evaluation errors. For each of them, we set the hyperparameters for the attack as done in the original evaluation, in order to collect similar results, and we compute the robust accuracy (as defined in Sect 2.3) for the given threat model.

- *k-Winners-Take-All (kWTA)*, the defense proposed by Xiao et al. (2020) uses only the top-k outputs from each layer, generating many discontinuities in the loss landscape, and hence resulting in the non-converging failure due to noisy gradients (F_2). We use the implementation provided by Tramer et al. (2020), trained on CIFAR10, and we test its robustness by attacking it with ℓ_∞-PGD (Madry et al., 2018) with a step size of $\alpha = 0.003$, maximum perturbation $\epsilon = 8/255$ and 50 iterations, with 5 restarts for each attack, scoring a robust accuracy of 58% on 100 samples.

- *Distillation*, the defense proposed by Papernot et al. (2016b), works by training a model to have zero gradients around the training points, leading gradient-based attacks towards bad local optimum (F_3). We re-implemented such defense, by training a distilled classifier on the MNIST dataset to mimic the original evaluation. Then, we apply ℓ_∞-PGD (Madry et al., 2018), with step size $\alpha = 0.01$, maximum perturbation $\epsilon = 0.3$ for 50 iterations on 100 samples, resulting in a robust accuracy of 94,2%.

- *Ensemble diversity*, the defense proposed by Pang et al. (2019) is composed of different neural networks, trained with a regularizer that encourages diversity. We adopt the implementation provided by Tramer et al. (2020). Then, following its original evaluation, we apply ℓ_∞-PGD (Madry et al., 2018), with step size $\alpha = 0.001$, maximum perturbation $\epsilon = 0.01$ for 10 iterations on 100 samples, resulting in a robust accuracy of 38%.

- *Turning a Weakness into a Strenght (TWS)*, the defense proposed by Yu et al. (2019), applies a mechanism for detecting the presence of adversarial examples on top of an undefended model, measuring how much the decision changes locally around a sample. Even if the authors also apply other rejection mechanisms, we take into account only the described one, as we wish to show that attacks optimized neglecting such term will trigger the non-adaptive attack failure (F_4).

We apply this defended on a WideResNet model trained on CIFAR10, provided by RobustBench (Croce et al., 2020). We attack this model with ℓ_∞-PGD (Madry et al., 2018), with step size $\alpha = 0.1$, maximum perturbation $\epsilon = 0.3$ for 50 iterations on 100 samples, and then we query the defended model with all the computed adversarial examples. While the attacks work against the standard model, some of them are rejected by the defense, resulting in a robust accuracy of 35%, highlighted by the trigger of the I_5 indicator. In this case, we consider an attack unsuccessful if the original sample is not misclassified and the adversarial point is either belonging to the same class or is labeled as rejected.
Each of these attacks has been executed with 5 random restarts. We also attack all these models with the version of AutoPGD (APGD) (Croce and Hein, 2020b) that uses the difference of logit (DLR) as a loss to optimize. This strategy will take care to automatically tune its hyperparameters while optimizing, reducing possible errors that occur while deciding the values of step size, and iterations. Lastly, we compute attacks that take into account all the mitigations we prescribed, and they will be analyzed further in the paper.

Identifying failures. We want now to understand if our indicators are correlated with faults of the security evaluations of defenses. We collect the results of all the attacks against the selected targets, and we compute our indicators, by listing their values in Table 5.5, along with their mean score. With a glance, it is possible to grasp that our hypothesis is right: the detection of a failure is linked with higher values for the robust accuracy, and also the opposite. Each original evaluation is characterized by high values of one or more indicators, while the opposite happens for stronger attacks. For instance, APGD automatically tunes its hyperparameter while optimizing, hence it is able to apply some mitigations directly during the attack. To gain a quantitative evaluation of our hypothesis, we compute both the p-value and the correlation between the average score of the indicators and the robust accuracy, depicting this result in Fig. 5.7. Both p-value and correlation suggest a strong connection between these analyzed quantities, confirming our initial belief.

Mitigating failures. We can now use our indicators to improve the quality of the security evaluations, and we apply the following pipeline: (i) we test the defense with a set of points with the original attack strategy proposed by the author of the defense; (ii) we select the failure cases and inspect the feedback of our indicators *per-sample*; (iii) for each cause of failure, we apply the specific remediation suggested

Model	Attack	I_1	I_2	I_3	I_4	I_5	$\bar{I}$	RA
	PGD	0.33	0.43	0.77	-	-	0.306	58%
k-WTA (Xiao et al., 2020)	APGD	-	0.31	0.33	-	-	0.128	36%
	PGD*	0.07	0.48	0.55	-	-	0.220	6%
	PGD	-	0.98	-	0.97	-	0.39	94%
Distillation (Papernot et al., 2016b)	APGD	-	0.40	0.21	-	-	0.122	0%
	PGD*	-	0.04	-	-	-	0.008	0%
	PGD	-	0.76	-	-	-	0.152	38%
Ensemble Div. (Pang et al., 2019)	APGD	-	0.37	0.14	-	-	0.102	0%
	PGD*	0.08	0.17	0.15	-	-	0.080	9%
	PGD	-	0.49	0.07	-	0.37	0.186	35%
TWS (Yu et al., 2019)	APGD	-	0.41	0.09	-	-	0.100	0%
	PGD*	-	0.37	0.10	-	-	0.094	0%

Table 5.5: Values of the Indicators of Attack Failures, computed for all the attacks against all the evaluated models. We denote the attacks that apply also the mitigations as PGD*.

by the metric; and (iv) we show that the attack now succeeds, thus reducing the robust accuracy of the target model, and also the values of the indicators.
We report all the results of this process in Table 5.6, where each row shows the original robust accuracy, and how it is decreased, mitigation after mitigation. Also, all the individual values of each indicator computed on these patched attacks can be found in Table 5.5, marked as PGD*.

- *Mitigating k-WTA failures.* For many failing attacks, the *I1* indicator triggers, implying that the attack found an adversarial example inside the path. We then apply mitigation M_1, and we lower accordingly the robust accuracy of the model to 36,4%. We then analyze the feedback of the I_3 indicator, the one that detects the presence of noisy gradients. We apply mitigation M_3, and we change the loss of the attack as described by Tramer et al. (2020). This loss is computed by averaging the gradient of every single point of the attack path with the information of the surrounding ones. The resulting direction is then able to correctly descent toward a minimum. We run ℓ_∞-PGD with the same parameters, but smoothing the gradients by averaging 100 neighboring points from a normal distribution $\mathcal{N}(\mu = \boldsymbol{x}_i, \sigma = 0.031)$, where x_i is a point in the attack path. After such mitigation, the robust accuracy drops to $6,4\%$, and so follows the indicator (Fig. 5.8a).

- *Mitigating Distillation failures.* All the attacks fail because of the absence of gradient information, leading the attack to a bad local optimum (F_3), and such is highlighted by the feedback of the I_3 indicator. We apply mitigation M_3, and we change the loss optimized during the attack, following the strategy applied by Carlini and Wagner (2016), that computes the loss of the attack on the logit of the model rather than the final softmax layer. We repeat the

PGD attack with such fix, and the robust accuracy drops to 0%, along with the indicator I_3 (Fig. 5.8b).

- *Mitigating Ensemble diversity failures.* Firstly, the I_1 indicator highlighted the presence of F_1, implying that some failing attacks are due to the implementation itself. We apply mitigation M_1, and the robust accuracy decreases to 36%. Also, I_2 indicator is active, implying that the loss of failing attacks could be optimized more. For this reason, we apply mitigation M_2, and we increase the step size to 0.05 and the iterations to 50. This patch is enough for lowering the robust accuracy to 9%. (Fig. 5.8c).

- *Mitigating TWS failures.* The detector is rejecting adversarial attacks successfully computed on the undefended model, triggering the I_5 indicator. Hence we apply mitigation M_5, and we adapt the attack to consider also the rejection class. This version of PGD minimizes the usual loss function of the attacker, but it also minimizes the score of the rejection class when encountered, allowing it to evade the rejection. We run such an attack, and we obtain a new robust accuracy of 0% (Fig. 5.8d).

Model	**Initial**	M_1	M_2	M_3	M_4	M_5	**Final**
k-WTA (Xiao et al., 2020)	58.2%	36.4%	36.4%	6.4%	6.4%	6.4%	**6.4%**
Distillation (Papernot et al., 2016b)	94.2%	94.2%	94.2%	0.4%	0.4%	0.4%	**0.4%**
Ensemble Diversity (Pang et al., 2019)	38.0%	38.0%	9.0%	9.0%	9.0%	9.0%	**9.0%**
TWS (Yu et al., 2019)	35.0%	35.0%	35.0%	35.0%	35.0%	0.0%	**0.0%**

Table 5.6: Robust accuracies (%) after patching the security evaluations with the prescribed mitigations.

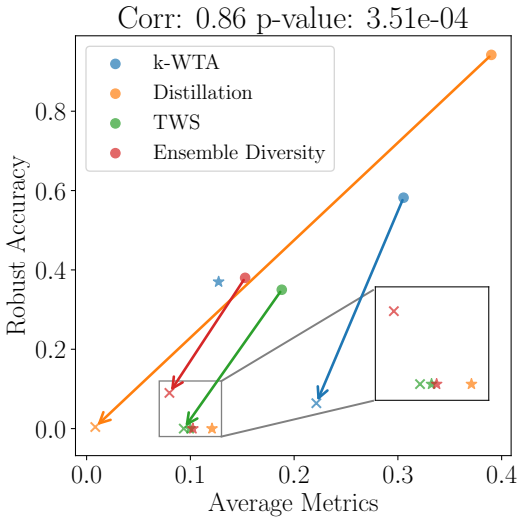


Figure 5.7: Robust accuracy vs. average value of the indicators. Incorrect evaluations (denoted with '∘') report high robust accuracy but also trigger most of the indicators. Better evaluations, performed by either mitigating the attack failures (denoted with '×'), or using APGD (denoted with '⋆'), correctly report a lower robust accuracy along with a lower average value of our indicators.

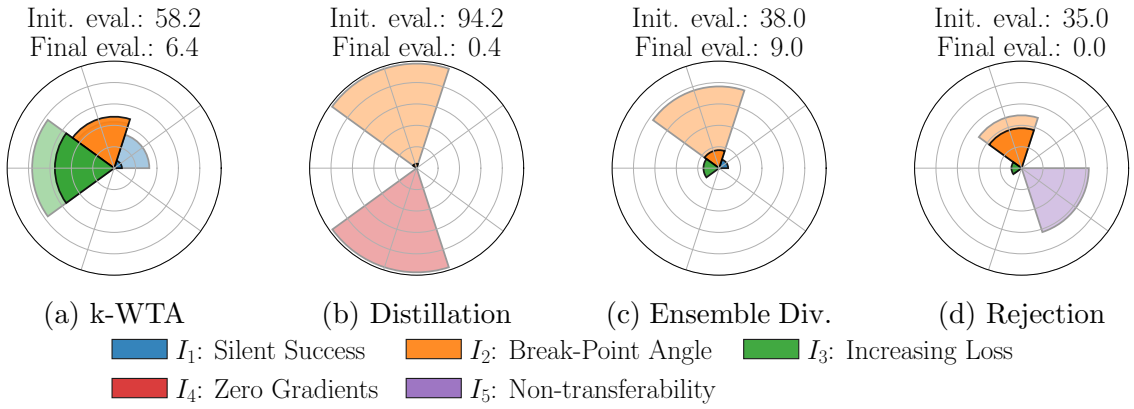


Figure 5.8: The values of our indicators and the success rate (SR) of the attack, before (semi-transparent colored area) and after (solid colored area) fixing the failures, computed for the analyzed models.

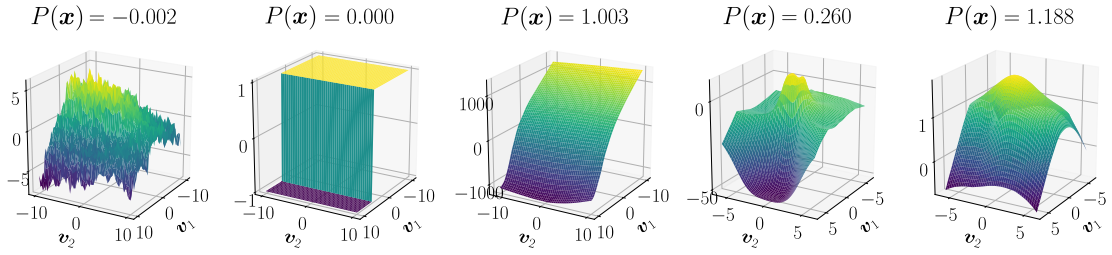
5.2.1 Measuring Gradient Obfuscation With Slope

From the results presented in the previous section, it is clear that some of the used models produce gradient obfuscation in the loss function used by the attacks. We now show how our metric Slope, which measures gradient obfuscation, is correlated with the ease or difficulty of optimizing the loss of an ℓ_p PGD attack, against a target model.

Target models We select four models, two of them are already presented in the previous evaluation and trained using particular techniques that are known to cause gradient obfuscation (Tramer et al., 2020), one is a standard-trained model, and the last one is an adversarially-trained one.

- *k-Winners Take All (k-WTA)* (Xiao et al., 2020): the model already described in Sect. 5.2, that causes many discontinuities in the loss surface, as shown in Figure 5.9a, and known to be affected by noisy descent (Tramer et al., 2020);
- *Distillation* (Papernot et al., 2016b): the model already described in Sect. 5.2, that causes zero gradients and instabilities in most of the loss landscape, as shown in Fig. 5.9b. The attack proposed by Athalye et al. (2018) discards the last softmax layer, using only the logits as the output of the classifier. This trick smooths the loss function, as shown in Fig. 5.9c, successfully removing the effect of the defense. We test such a model by computing the gradients of the Cross-Entropy loss (obfuscated) and on the pre-softmax scores (not obfuscated).
- *Standard training*: we use the implementation provided by RobustBench (Croce et al., 2020), which is a WideResNet (Zagoruyko and Komodakis, 2016) model trained on CIFAR10. Since it is not trained with any defenses, the loss landscape is smooth, but it contains many local minima and maxima, as shown in Fig. 5.9d. Hence, all classes are very close one to another, and adversarial examples are very easy to be found.
- *Adversarial training* (Madry et al., 2018): the model is trained with both normal and adversarial examples, computed with an attack of choice, and such process is repeated until a desired robust accuracy is reached. As a result, the loss landscape becomes smoother, reducing the blind-spot areas where adversarial examples lay, as shown in Fig. 5.9e. For our work, we use the ResNet (He et al., 2016) model trained by Madry et al. (2018) on CIFAR10.

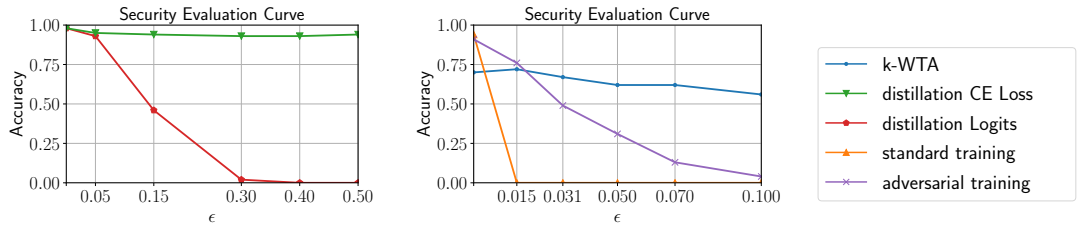
Results We report the results of our experimental analysis, by taking into account the correlation between our metric $\bar{P}$ and the robust accuracy of each target. For computing $\bar{P}$, we use $\eta \in [0.01, 0.1]$ when using the ℓ_2 norm (Fig. 5.11a), and $\eta \in [0.001, 0.01]$ when using the ℓ_∞ norm (Fig. 5.11b). The security evaluation is computed by attacking all the targets with PGD ℓ_∞. For the model trained on



(a) (b) (c) (d) (e)

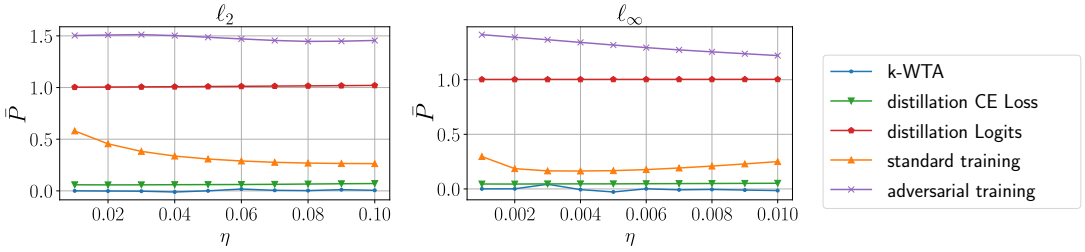
Figure 5.9: Loss landscape visualizations on the bi-dimensional space spanned by the ℓ_∞ adversarial direction ($\boldsymbol{v}_1$) and a random direction ($\boldsymbol{v}_2$) for the models used in our experiments: (a) K-WTA, (b) Distillation with Cross Entropy loss, (c) Distillation with logits loss, (d) Standard model, and (e) Adversarial training. Note that $P(\boldsymbol{x}) \leq 0$ only for models (a) and (b), which present obfuscated gradients.

CIFAR10, we set the maximum perturbation $\epsilon \in [0, 0.1]$ (Fig. 5.10b), where 0 implies the accuracy in absence of the attack, while we use $\epsilon \in [0, 0.5]$ for the attacks against the models trained on MNIST (Fig.5.10a).
By looking at the output of $\bar{P}$, we notice that models trained with obfuscated gradients are characterized by values that are less than or equal to zero, implying difficulty in detecting the right direction to follow during the attack. For instance, the Distillation model with Cross-Entropy loss has null gradients, while k-WTA is characterized by noisy gradients that lead the loss to a decrement rather than an increment.
The other models' scores are positive, meaning that their gradients are informative for the attack and aligned with the loss to increase, hence converging to adversarial points. These effects are very similar for both ℓ_2 and ℓ_∞ norms, implying that such trend is characteristic of the model itself rather than the norm used for computing one step of PGD.
These results are confirmed by the security evaluations in Fig. 5.10: all the defended obfuscated models are less stressed by the adversarial attacks performed with PGD. The fact that the robust accuracy does not fall even with a high perturbation budget, i.e. in the rightmost part of the security evaluation plots, confirms the hypothesis that the attack strategy is not suitable for those models. Also, the smoother models are affected by our attacks when the perturbation budges increases, as also predicted by $\bar{P}$. Hence, Slope is a good proxy for detecting the presence of gradient obfuscation inside the model, allowing the attacker to rethink their strategy, and land successful evasion attacks.



(a) MNIST models. (b) CIFAR models.

Figure 5.10: Security evaluations of the four models. On the x-axis, the values for the perturbation budget ϵ used for the evaluation, while on the y axis the robust accuracy.



(a) (b)

Figure 5.11: The values of the mean Slope $\bar{P}$, computed for both the ℓ_2 (a) and the ℓ_∞ (b) norms. On the x-axis, the step sizes η used for computing the approximation, while on the y axis we report the values of $\bar{P}$.

5.3 Transferability of Adversarial Examples

In this section, we evaluate the transferability of evasion attacks across a range of ML models. We highlight some interesting findings on transferability, based on the three metrics developed in Sect. 4.4. In particular, we analyze attack transferability in terms of its connection to the size of the input gradients of the loss function, and the gradient alignment between surrogate and target classifiers. We provide recommendations on how to choose the most effective surrogate models to craft transferable evasion attacks in the black-box setting.

Experimental setup. The MNIST89 data includes the MNIST handwritten digits from classes 8 and 9. Each digit image consists of 784 pixels ranging from 0 to 255, normalized in $[0, 1]$ by dividing such values by 255. We run 10 independent repetitions to average the results on different training-test splits. In each repetition, we run white-box and black-box attacks, using $5,900$ samples to train the target classifier, $5,900$ distinct samples to train the surrogate classifier (without even relabeling the surrogate data with labels predicted by the target classifier; i.e., we do not perform any query on the target), and $1,000$ test samples. We modified test digits in both classes using Algorithm 1 under the ℓ_2 distance constraint $\|\boldsymbol{x} - \boldsymbol{x}'\|_2 \leq \epsilon$, with $\epsilon \in [0, 5]$.

For each of the following learning algorithms, we train a high-complexity (H) and a low-complexity (L) model, by changing its hyperparameters: (*i*) SVMs with linear kernel (SVM_H with $C = 100$ and SVM_L with $C = 0.01$); (*ii*) SVMs with RBF kernel (SVM-RBF_H with $C = 100$ and SVM-RBF_L with $C = 1$, both with $\gamma = 0.01$); (*iii*) logistic classifiers (logistic_H with $C = 10$ and logistic_L with $C = 1$); (*iv*) ridge classifiers (ridge_H with $\alpha = 1$ and ridge_L with $\alpha = 10$);[2] (*v*) fully-connected neural networks with two hidden layers including 50 neurons each, and ReLU activations (NN_H with no regularization, i.e., weight decay set to 0, and NN_L with weight decay set to 0.01), trained via cross-entropy loss minimization; and (*vi*) random forests consisting of 30 trees (RF_H with no limit on the depth of the trees and RF_L with a maximum depth of 8). These configurations are chosen to evaluate the robustness of classifiers that exhibit similar test accuracies but different levels of complexity.

Model complexity analysis in white-box settings. The results for white-box evasion attacks are reported for all classifiers that fall under our framework and can be tested for evasion with gradient-based attacks (SVM, Logistic, Ridge, and NN). This excludes random forests, as they are not differentiable. We report the complete *security evaluation curves* Biggio and Roli (2018) in Fig. 5.12, showing the mean test error (over 10 runs) against an increasing maximum admissible distortion ϵ. In Fig. 5.13a we report the mean test error at $\epsilon = 1$ for each target model against the size of its input gradients (S, averaged on the test samples and on the 10 runs).

The results show that, for each learning algorithm, the low-complexity model has smaller input gradients, and it is less vulnerable to evasion than its high-complexity

[2]Recall that the level of regularization increases as α increases, and as C decreases.

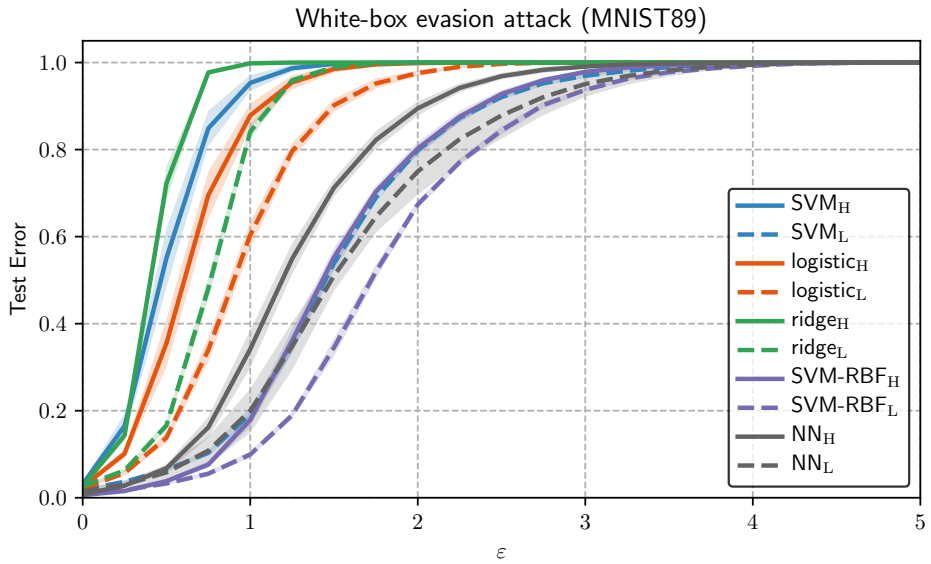
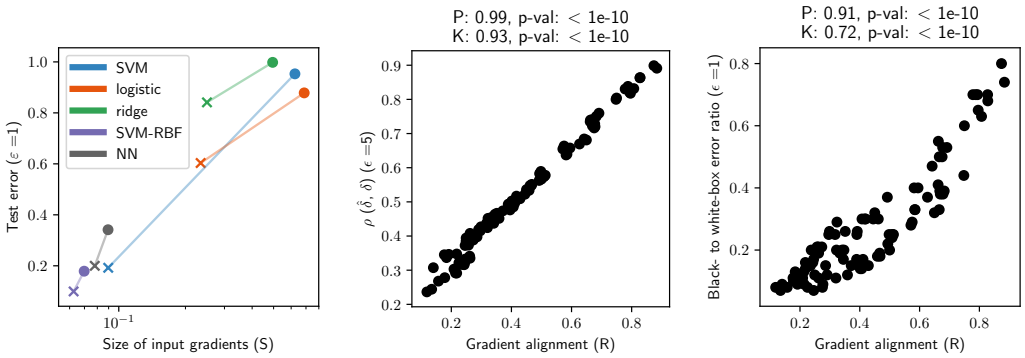


Figure 5.12: White-box evasion attacks on MNIST89. Test error against increasing maximum perturbation ϵ.

counterpart, confirming our theoretical analysis. Recall that these results hold only when comparing models trained using the same learning algorithm. This means that we can compare, e.g., the S value of SVM_H against SVM_L, but not that of SVM_H against logistic_H. In fact, even though logistic_H exhibits the largest S value, it is not the most vulnerable classifier. Another interesting finding is that nonlinear classifiers tend to be less vulnerable than linear ones.

Transferability of evasion attacks. In Fig. 5.14 we report the results for black-box evasion attacks, in which the attacks against surrogate models (in rows) are transferred to the target models (in columns). The top row shows results for surrogates trained using only 20% of the surrogate training data, while in the bottom row surrogates are trained using all surrogate data, i.e., a training set of the same size as that of the target. The three columns report results for $\epsilon \in \{1, 2, 5\}$.
It can be noted that lower-complexity models (with stronger regularization) provide better surrogate models, on average. In particular, this can be seen best in the middle column for a medium level of perturbation, in which the lower-complexity models (SVM_L, logistic_L, ridge_L, and SVM-RBF_L) provide on average higher error when transferred to other models. The reason is that they learn smoother and stabler functions, that are capable of better approximating the target function. Surprisingly, this holds also when using only 20% of training data, as the black-box attacks relying on such low-complexity models still transfer with similar test errors. This means that most classifiers can be attacked in this black-box setting with almost no knowledge of the model, no query access, but provided that one can get a small amount of data similar to that used to train the target model.



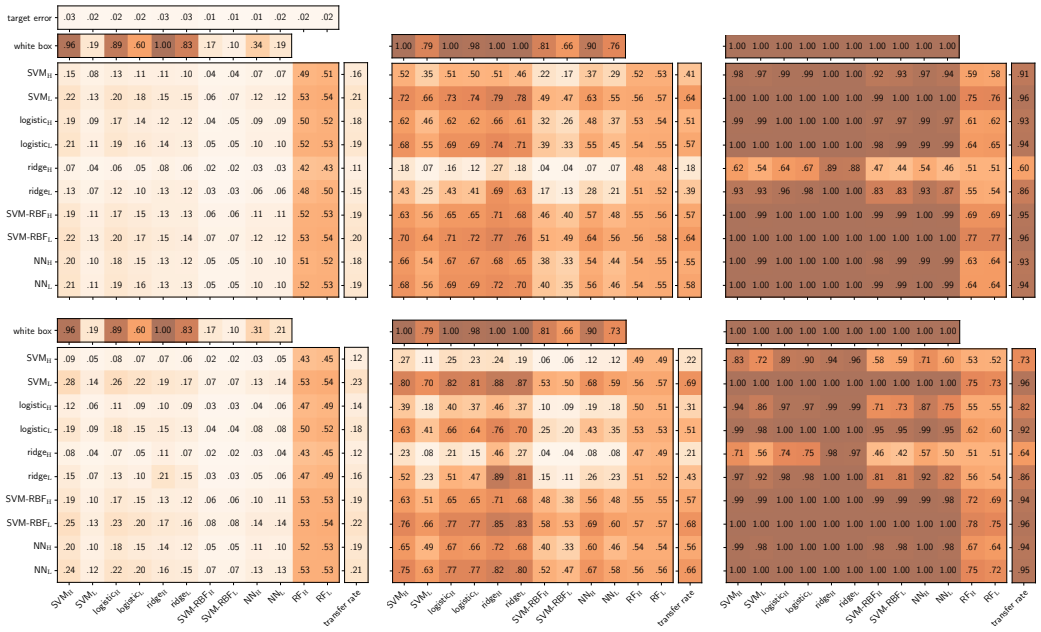
(a) (b) (c)

Figure 5.13: Evaluation of our metrics for evasion attacks on MNIST89. (a) Test error under attack vs average size of input gradients (S) for low- (denoted with '$\times$') and high-complexity (denoted with '$\circ$') classifiers. (b) Pearson correlation coefficient $\rho(\hat{\boldsymbol{\delta}}, \boldsymbol{\delta})$ between black-box ($\hat{\boldsymbol{\delta}}$) and white-box ($\boldsymbol{\delta}$) perturbations (values in Fig. 5.15, right) vs gradient alignment (R, values in Fig. 5.15, left) for each target-surrogate pair. Pearson (P) and Kendall (K) correlations between ρ and R are also reported along with the p-values obtained from a permutation test to assess statistical significance.

Gradient alignment and transferability. In Fig. 5.15, we report on the left the gradient alignment computed between surrogate and target models, and on the right the Pearson correlation coefficient $\rho(\hat{\boldsymbol{\delta}}, \boldsymbol{\delta})$ between the perturbation optimized against the surrogate (i.e., the black-box perturbation $\hat{\boldsymbol{\delta}}$) and that optimized against the target (i.e., the white-box perturbation $\boldsymbol{\delta}$). We observe immediately that gradient alignment provides an accurate measure of transferability: the higher the cosine similarity, the higher the correlation (meaning that the adversarial examples crafted against the two models are similar). We correlate these two measures in Fig. 5.13b, and show that such correlation is statistically significant for both Pearson and Kendall coefficients. In Fig. 5.13c we also correlate gradient alignment with the ratio between the test error of the target model in the black- and white-box setting (extrapolated from the matrix corresponding to $\epsilon = 1$ in the bottom row of Fig. 5.14), as suggested by our theoretical derivation. The corresponding permutation tests confirm statistical significance. We finally remark that gradient alignment is extremely fast to evaluate, as it does not require simulating any attack, but it is only a relative measure of the attack transferability, as the latter also depends on the complexity of the target model; i.e., on the size of its input gradients.

5.3.1 Summary of Transferability Evaluation

We summarize the results of transferability for evasion and poisoning attacks below.



(a) $\epsilon = 1$ (b) $\epsilon = 2$ (c) $\epsilon = 5$

Figure 5.14: Black-box (transfer) evasion attacks on MNIST89. Each cell contains the test error of the target classifier (in columns) computed on the attack samples crafted against the surrogate (in rows). Matrices in the top (bottom) row correspond to attacks crafted against surrogate models trained with 20% (100%) of the surrogate training data, for $\epsilon \in \{1, 2, 5\}$. The test error of each target classifier in the absence of an attack (target error) and under (white-box) attack are also reported for comparison, along with the mean transfer rate of each surrogate across targets. Darker colors mean higher test error, i.e., better transferability.

(1) Size of input gradients. Low-complexity target classifiers are less vulnerable to evasion and poisoning attacks than high-complexity target classifiers trained with the same learning algorithm, due to the reduced size of their input gradients. In general, nonlinear models are more robust than linear models to both types of attacks.

(2) Gradient alignment. Gradient alignment is correlated with transferability. Even though it cannot be directly measured in black-box scenarios, some useful guidelines can be derived from our analysis. For evasion attacks, low-complexity surrogate classifiers provide stabler gradients that are better aligned, on average, with those of the target models; thus, it is generally preferable to use strongly-regularized surrogates.

To summarize, for evasion attacks, decreasing the complexity of the surrogate model by properly adjusting the hyperparameters of its learning algorithm provides adversarial examples that transfer better to a range of models.

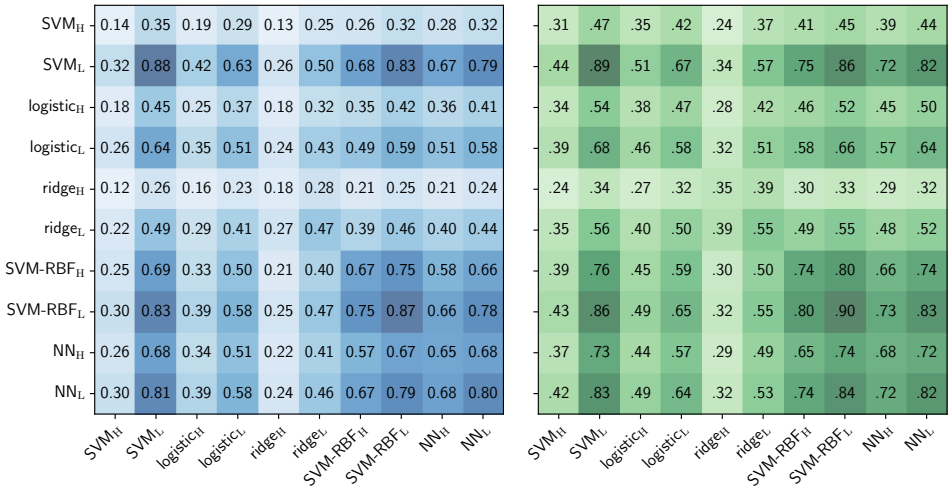


Figure 5.15: Gradient alignment and perturbation correlation for evasion attacks on MNIST89. *Left:* Gradient alignment R (Eq. 4.16) between surrogate (rows) and target (columns) classifiers, averaged on the unmodified test samples. *Right:* Pearson correlation coefficient $\rho(\boldsymbol{\delta}, \hat{\boldsymbol{\delta}})$ between white-box and black-box perturbations for $\epsilon = 5$.

Understanding attack transferability has two main implications. First, even when attackers do not know the target classifier, our findings suggest that low-complexity surrogates have a better chance of transferring to other models. Our recommendation to perform black-box evasion attacks is to choose surrogates with low complexity (e.g., by using strong regularization and reducing model variance). Second, our analysis also provides recommendations to defenders on how to design more robust models against evasion and poisoning attacks. In particular, lower-complexity models tend to have more resilience compared to more complex models. Of course, we need to take into account the bias-variance trade-off and choose models that still perform relatively well on the original prediction tasks.

Chapter 6

Conclusions

We now summarize the main contributions of this thesis, along with the main results found with the analysis, and draw future plans for extending the topics of this research to new directions.

6.1 Summary of the Main Achievements

This work introduces improvements over the security evaluation of machine learning models. Our big picture is to automate, systematize, and raise awareness on the complicated and unforgiving process of proactively evaluating machine learning defenses.

We introduced a novel minimum-norm attack that combines all desirable traits to help improve current adversarial evaluations: (i) finding smaller or comparable minimum-norm perturbations across a range of models and datasets; (ii) being less sensitive to hyperparameter choices; (iii) reducing runtime up to 3 times per query with respect to competing attacks and (iv) converging within fewer iterations. FMN also works with different ℓ_p norms ($p = 0, 1, 2, \infty$) and it does not necessarily require being initialized from an adversarial starting point. Our experiments have shown that FMN rivals or surpasses other attacks in speed, reliability, efficacy, and versatility. We firmly believe that FMN will establish itself as a useful tool in the arsenal of robustness evaluation. By facilitating more reliable robustness evaluations, we expect that FMN will foster advancements in the development of machine-learning models with improved robustness guarantees.

We proposed the Indicators of Attack Failures (IoAF), quantitative tests that help the debugging of faulty-conducted security evaluations. Using these indicators, we then designed a pipeline for mitigating their issues, leading to a fairer evaluation. We selected defenses that have been previously shown to be weak against adversarial attacks, and we evaluated them with the lens of our indicators, showing that we could have detected their misconduct in advance. We empirically proved that these tests are correlated with wrongly high robust accuracy, while they drop when

attacks are successful. We hope that future work will include our indicators during the evaluation phase of new methods, in order to identify when attacks are failing for known reasons, and thus contributing to the creation of better defense mechanisms. Also, this work poses a preliminary step towards the creation of interactive dashboards that can be inspected as web applications.

We proposed a metric for detecting the presence of gradients obfuscation, matching them with the ease of increasing the loss of the attack with PGD. Such metric is based on the intuition that obfuscated gradients can not be approximated during a step of PGD attacks, hence the computed loss is not representative of the real loss of the model. We tested the metric against four models, where two of them are defended with obfuscated gradients, showing that the output of our metric is correlated with the inability of decreasing the loss of the obfuscated ones. Hence, our metric can be used as a tool for detecting such defense, helping the attacker devise a strategy against the target.

We have conducted an analysis of the transferability of evasion attacks. Our theoretical transferability formalization sheds light on various factors impacting the transfer success rates. In particular, we have defined two metrics that impact the transferability of an attack, namely the complexity of the target model and the gradient alignment between the surrogate and target models. The lesson to system designers is to evaluate their classifiers against these criteria and select lower-complexity, stronger regularized models that tend to provide higher robustness to evasion attacks.

Despite the improvements brought by this thesis, we are still far from having a fully-automated, completely-trustworthy process. In the next section, we will list, for each of the contributions presented, limitations and possible future research directions to further extend and enhance this work.

6.2 Limitations and Future Directions

While FMN is able to find smaller perturbations consistently when compared against ℓ_0 and ℓ_1 attacks, it only rivals the performance of other attacks for ℓ_2 and ℓ_∞ norms, especially when tested against robust models which may present obfuscated gradients. To overcome this limitation, FMN may be extended using *smoothing* strategies that help find better descent directions, e.g., by averaging gradients on randomly-perturbed inputs. This can be regarded as an interesting extension of FMN towards attacking robust models. In this respect, we also believe that FMN may facilitate minimum-norm adaptive evaluations in a more general sense. Adaptive evaluations, where the attack is modified to be maximally effective against a new defense, are the key element towards properly evaluating adversarial robustness (Carlini et al., 2019; Tramer et al., 2020). PGD attacks are popular because they can be easily

adapted to new defenses. Since FMN combines PGD with a dynamic minimization of the perturbation size, we argue that our attack can also be easily adapted to new defenses, thereby facilitating adaptive evaluations. FMN may also benefit from other improvements that have been suggested for PGD, including momentum, cyclical step sizes or restarts. We leave such improvements to future work.

The indicators presented are helpful tools for finding problems in the optimization of gradient-based adversarial attacks. However, our analysis is not yet fully-automated since an operator must manually decide how to turn an attack into its adaptive version against a particular defense. Our quantitative tools for helping an informed decision among all the possible solutions that the attacker could choose. Another limitation lurks in the choice of the attack itself since some unknown-and-adaptive attack could behave very differently than a standard one, triggering some indicator in the process. However, these tests can be patched accordingly to take care of these newly-proposed patched attacks, and are still being helpful as debugging tools. Lastly, as already discussed in Sect. 4.3.2, if the evaluated defense is not triggering any indicators it does not imply it is secure, but rather it forces the application of other sanity checks Carlini et al. (2019). We believe some part of this last process can be automatized with additional indicators, however, we leave this as future work. Finally, it would be insightful to attach our pipeline of indicators and mitigations to already-available benchmarks (i.e. RobustBench Croce et al. (2020)), possibly detecting other failures in security evaluations we did not cover in our experiments.

On the transferability perspective, we believe interesting avenues for future work include extending our analysis to multi-class classification settings and considering a range of gray-box models in which attackers might have additional knowledge of the machine learning system (as in Suciu et al. (2018)). Application-dependent scenarios such as cybersecurity might provide additional constraints on threat models and attack scenarios and could impact transferability in interesting ways.

6.3 Concluding Remarks

To conclude, all these directions suggest that obtaining reliable machine learning models is still an open problem, where progress can be obtained only with small and careful steps. Knowing how to evaluate adversarial robustness is one big obstacle that slows down the advancements in proposing new defenses. This thesis provides concrete improvements in the overall security evaluation process by proposing an efficient attack, debugging strategies for the optimization of adversarial examples, and an analysis of the effects of transferability. We believe that this work provides useful tools and guidelines with the hope that this will improve, in the future, the state of research in this field and the trustworthiness of machine-learning models, especially in security-critical scenarios.

Acknowledgements

This work has been supported by BMK, BMDW, and the Province of Upper Austria in the frame of the COMET Programme managed by FFG in the COMET Module S3AI.

Bibliography

M. Andriushchenko, F. Croce, N. Flammarion, and M. Hein. Square attack: a query-efficient black-box adversarial attack via random search. In *European Conference on Computer Vision*, pages 484–501. Springer, 2020.

A. Athalye, N. Carlini, and D. A. Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *ICML*, volume 80 of *JMLR Workshop and Conference Proceedings*, pages 274–283. JMLR.org, 2018.

A. Bendale and T. E. Boult. Towards open set deep networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1563–1572, 2016.

B. Biggio and F. Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, 2018.

B. Biggio, B. Nelson, and P. Laskov. Poisoning attacks against support vector machines. In J. Langford and J. Pineau, editors, *29th Int'l Conf. on Machine Learning*, pages 1807–1814. Omnipress, 2012.

B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time. In H. Blockeel, K. Kersting, S. Nijssen, and F. Železný, editors, *Machine Learning and Knowledge Discovery in Databases (ECML PKDD), Part III*, volume 8190 of *LNCS*, pages 387–402. Springer Berlin Heidelberg, 2013.

W. Brendel, J. Rauber, and M. Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *International Conference on Learning Representations*, 2018.

W. Brendel, J. Rauber, M. Kümmerer, I. Ustyuzhaninov, and M. Bethge. Accurate, reliable and fast robustness evaluation. In *Thirty-third Conference on Neural Information Processing Systems (NeurIPS 2019)*, pages 12817–12827. Curran, 2020.

J. Buckman, A. Roy, C. Raffel, and I. Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. In *International Conference on Learning Representations*, 2018.

N. Carlini. A critique of the deepsec platform for security analysis of deep learning models, 2019.

N. Carlini and D. Wagner. Defensive distillation is not robust to adversarial examples, 2016.

N. Carlini and D. A. Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In B. M. Thuraisingham, B. Biggio, D. M. Freeman, B. Miller, and A. Sinha, editors, *10th ACM Workshop on Artificial Intelligence and Security*, AISec '17, pages 3–14, New York, NY, USA, 2017a. ACM.

N. Carlini and D. A. Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy*, pages 39–57. IEEE Computer Society, 2017b.

N. Carlini, P. Mishra, T. Vaidya, Y. Zhang, M. Sherr, C. Shields, D. Wagner, and W. Zhou. Hidden voice commands. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 513–530, Austin, TX, Aug. 2016. USENIX Association. ISBN 978-1-931971-32-4. URL `https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/carlini`.

N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. Goodfellow, A. Madry, and A. Kurakin. On evaluating adversarial robustness, 2019.

Y. Carmon, A. Raghunathan, L. Schmidt, P. Liang, and J. C. Duchi. Unlabeled data improves adversarial robustness. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 11192–11203, 2019.

P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *10th ACM Workshop on Artificial Intelligence and Security*, AISec '17, pages 15–26, New York, NY, USA, 2017. ACM.

P.-Y. Chen, Y. Sharma, H. Zhang, J. Yi, and C.-J. Hsieh. Ead: elastic-net attacks to deep neural networks via adversarial examples. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

J. Cohen, E. Rosenfeld, and Z. Kolter. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning*, pages 1310–1320. PMLR, 2019.

F. Croce and M. Hein. Minimally distorted adversarial examples with a fast adaptive boundary attack. In *International Conference on Machine Learning*, pages 2196–2205. PMLR, 2020a.

F. Croce and M. Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *ICML*, 2020b.

F. Croce, M. Andriushchenko, V. Sehwag, N. Flammarion, M. Chiang, P. Mittal, and M. Hein. Robustbench: a standardized adversarial robustness benchmark. *arXiv preprint arXiv:2010.09670*, 2020.

G. S. Dhillon, K. Azizzadenesheli, J. D. Bernstein, J. Kossaifi, A. Khanna, Z. C. Lipton, and A. Anandkumar. Stochastic activation pruning for robust adversarial defense. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=H1uR4GZRZ`.

G. W. Ding, L. Wang, and X. Jin. AdverTorch v0.1: An adversarial robustness toolbox based on pytorch. *arXiv preprint arXiv:1902.07623*, 2019.

Y. Dong, F. Liao, T. Pang, X. Hu, and J. Zhu. Boosting adversarial examples with momentum. In *CVPR*, 2018.

J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the l1-ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 272–279, New York, NY, USA, 2008. ACM.

L. Engstrom, A. Ilyas, H. Salman, S. Santurkar, and D. Tsipras. Robustness (python library), 2019. URL `https://github.com/MadryLab/robustness`.

I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

S. Gowal, J. Uesato, C. Qin, P.-S. Huang, T. Mann, and P. Kohli. An alternative surrogate loss for pgd-based adversarial testing. *arXiv e-prints*, pages arXiv–1910, 2019.

C. Guo, M. Rana, M. Cisse, and L. van der Maaten. Countering adversarial images using input transformations. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=SyJ7ClWCb`.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

A. Jalal, A. Ilyas, C. Daskalakis, and A. G. Dimakis. The robust manifold defense: Adversarial training using generative models. *arXiv e-prints*, pages arXiv–1712, 2017.

P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1885–1894. JMLR. org, 2017.

A. Kurakin, I. J. Goodfellow, and S. Bengio. Adversarial machine learning at scale. In *ICLR*, 2017. URL `https://arxiv.org/abs/1611.01236`.

X. Li and F. Li. Adversarial examples detection in deep networks with convolutional filter statistics. In *The IEEE International Conference on Computer Vision (ICCV)*, 2017.

J. Lin, C. Song, K. He, L. Wang, and J. E. Hopcroft. Nesterov accelerated gradient and scale invariance for adversarial attacks. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=SJlHwkBYDH`.

X. Ling, S. Ji, J. Zou, J. Wang, C. Wu, B. Li, and T. Wang. Deepsec: A uniform platform for security analysis of deep learning model. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 673–690, 2019. doi: 10.1109/SP.2019.00023.

Y. Liu, X. Chen, C. Liu, and D. Song. Delving into transferable adversarial examples and black-box attacks. In *ICLR*, 2017.

J. Lu, T. Issaranon, and D. Forsyth. Safetynet: Detecting and rejecting adversarial examples robustly. In *The IEEE International Conference on Computer Vision (ICCV)*, 2017.

A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018.

M. Melis, A. Demontis, B. Biggio, G. Brown, G. Fumera, and F. Roli. Is deep learning safe for robot vision? Adversarial examples against the iCub humanoid. In *ICCVW Vision in Practice on Autonomous Robots (ViPAR)*, pages 751–759. IEEE, 2017.

M. Melis, A. Demontis, M. Pintor, A. Sotgiu, and B. Biggio. secml: A python library for secure and explainable machine learning. *arXiv preprint arXiv:1912.10013*, 2019.

D. Meng and H. Chen. MagNet: a two-pronged defense against adversarial examples. In *24th ACM Conf. Computer and Comm. Sec. (CCS)*, 2017.

S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 2574–2582, 2016.

V. Munusamy Kabilan, B. Morris, and A. Nguyen. Vectordefense: Vectorization as a defense to adversarial examples. 2018. arXiv preprint arXiv:1804.08529.

N. Narodytska and S. Prasad Kasiviswanathan. Simple black-box adversarial perturbations for deep networks. *arXiv e-prints*, pages arXiv–1612, 2016.

M.-I. Nicolae, M. Sinn, M. N. Tran, B. Buesser, A. Rawat, M. Wistuba, V. Zantedeschi, N. Baracaldo, B. Chen, H. Ludwig, I. M. Molloy, and B. Edwards. Adversarial robustness toolbox v1.0.0, 2019.

T. Pang, K. Xu, C. Du, N. Chen, and J. Zhu. Improving adversarial robustness via promoting ensemble diversity. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4970–4979. PMLR, 09–15 Jun 2019. URL `http://proceedings.mlr.press/v97/pang19a.html`.

N. Papernot, P. McDaniel, A. Sinha, and M. Wellman. Towards the science of security and privacy in machine learning. *arXiv preprint arXiv:1611.03814*, 2016a.

N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597, May 2016b. doi: 10.1109/SP.2016.41.

N. Papernot, F. Faghri, N. Carlini, I. Goodfellow, R. Feinman, A. Kurakin, C. Xie, Y. Sharma, T. Brown, A. Roy, A. Matyasko, V. Behzadan, K. Hambardzumyan, Z. Zhang, Y.-L. Juang, Z. Li, R. Sheatsley, A. Garg, J. Uesato, W. Gierke, Y. Dong, D. Berthelot, P. Hendricks, J. Rauber, and R. Long. Technical report on the cleverhans v2.1.0 adversarial examples library. *arXiv preprint arXiv:1610.00768*, 2018.

D. Park, H. Khan, A. Khan, A. Gittens, and B. Yener. Output randomization: A novel defense for both white-box and black-box adversarial models, 2021.

A. Prakash, N. Moran, S. Garber, A. DiLillo, and J. A. Storer. Deflecting adversarial attacks with pixel deflection. In *CVPR*, 2018.

J. Rauber, W. Brendel, and M. Bethge. Foolbox: A python toolbox to benchmark the robustness of machine learning models. In *Reliable Machine Learning in the Wild Workshop, 34th International Conference on Machine Learning*, 2017. URL `http://arxiv.org/abs/1707.04131`.

J. Rauber, R. Zimmermann, M. Bethge, and W. Brendel. 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, 2020. doi: 10.21105/joss.02607. URL `https://doi.org/10.21105/joss.02607`.

J. Rony, L. G. Hafemann, L. S. Oliveira, I. B. Ayed, R. Sabourin, and E. Granger. Decoupling direction and norm for efficient gradient-based l2 adversarial attacks and defenses. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4322–4330, 2019.

A. S. Ross and F. Doshi-Velez. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. In *AAAI*. AAAI Press, 2018.

P. Samangouei, M. Kabkab, and R. Chellappa. Defense-gan: Protecting classifiers against adversarial attacks using generative models. In *International Conference on Learning Representations*, 2018.

L. Schott, J. Rauber, M. Bethge, and W. Brendel. Towards the first adversarially robust neural network model on mnist. In *International Conference on Learning Representations*, 2018.

S. Shen, G. Jin, K. Gao, and Y. Zhang. Ape-gan: Adversarial perturbation elimination with gan. *arXiv preprint arXiv:1707.05474*, 2017.

Y. Song, T. Kim, S. Nowozin, S. Ermon, and N. Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rJUYGxbCW`.

O. Suciu, R. Marginean, Y. Kaya, H. D. III, and T. Dumitras. When does machine learning FAIL? generalized transferability for evasion and poisoning attacks. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1299–1316, Baltimore, MD, Aug. 2018. USENIX Association. ISBN 978-1-939133-04-5. URL `https://www.usenix.org/conference/usenixsecurity18/presentation/suciu`.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014. URL `http://arxiv.org/abs/1312.6199`.

F. Tramer, N. Carlini, W. Brendel, and A. Madry. On adaptive attacks to adversarial example defenses. *Advances in Neural Information Processing Systems*, 33, 2020.

E. Wong, L. Rice, and J. Z. Kolter. Fast is better than free: Revisiting adversarial training. In *International Conference on Learning Representations*, 2019.

C. Xiao, P. Zhong, and C. Zheng. Enhancing adversarial defense by k-winners-take-all. In *8th International Conference on Learning Representations*, 2020.

C. Xie, J. Wang, Z. Zhang, Z. Ren, and A. Yuille. Mitigating adversarial effects through randomization. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=Sk9yuql0Z`.

T. Yu, S. Hu, C. Guo, W. Chao, and K. Weinberger. A new defense against adversarial images: Turning a weakness into a strength. In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, Oct. 2019.

S. Zagoruyko and N. Komodakis. Wide residual networks. In *British Machine Vision Conference 2016*. British Machine Vision Association, 2016.

H. Zhang, H. Chen, C. Xiao, S. Gowal, R. Stanforth, B. Li, D. Boning, and C.-J. Hsieh. Towards stable and efficient training of verifiably robust neural networks. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=Skxuk1rFwB`.

Ringraziamenti

Ho sempre pensato che ogni cosa nella vita sia come uno sport di squadra. L'unione non solo fa la forza, ma fa anche il divertimento e l'entusiasmo di affrontare nuove sfide con le persone su cui si può contare, perché da soli fare i gol è molto più difficile (soprattutto per un portiere). Non conta solo il risultato, ma anche e soprattutto chi c'è stato, che è la vera cosa che dà valore alla vittoria. Desidero ringraziare tutte le persone che mi hanno aiutato nella ricerca e nella scrittura di questa tesi e nel mio percorso di dottorato.

Ringrazio Battista e Fabio, che hanno seguito con me questo percorso guidandomi e incoraggiandomi a fare sempre il meglio, sapendo che poi i risultati arrivano. Grazie per tutta la conoscenza che mi avete trasmesso in questi anni. Ho ancora tanto da imparare, ma la vostra naturalezza nell'insegnamento e la vostra passione per la ricerca è sempre stata per me la più grande ispirazione.

Ringrazio Ambra, sempre disponibile e preparata, che ha la capacità incredibile di trovare prontamente soluzioni per risolvere qualunque problema.
Ringrazio Angelo, Daniele, e Giorgio, per essere sempre stati disponibili nei momenti di necessità e per aver contribuito anche loro attivamente nel raggiungimento di questo traguardo.
Ringrazio anche tutti gli altri membri del PRALab, per tutte le interessanti conversazioni e gli ottimi consigli.

Ringrazio Stefania, Guido, Enrico, Davide, Luca, Igino e tutti gli altri per avermi accolto nella famiglia di Pluribus One e avermi dato l'opportunità di partecipare ai loro grandi progetti e di imparare nuove cose ogni giorno, nonché di non aver mai fatto mancare occasioni per festeggiare tutti insieme ogni evento della vita. Ho sempre la certezza che se vengo in sede da voi troverò Stefy che prepara caffè con *crastuli* freschi, Guido e Enrico che mi parlano delle loro nuove sfide di coding, Davide e Luca che mi chiedono come va il dottorato e delle mie avventure calcistiche, e qualche volta Igino, anche se nessuno lo ha sentito arrivare, che ha sempre il sorriso e l'entusiasmo di chi è fortissimo nel suo campo ma non se ne vanta mai.

Ringrazio Rina e Gerhard, per avermi dato una casa in Germania, aver condiviso tante serate insieme, e avermi fatto trovare un po' d'Italia anche lì. Terrò sempre con me i giri in bici, le serate passate a parlare, la vostra disponibilità incondizionata, e le ore passate a giocare a giochi da tavolo con voi cercando di imparare un po' di tedesco. Spero di poter venire a trovarvi al più presto.

Ringrazio Daniele, il mio esperto di tecnologia in continuo aggiornamento che ha sempre qualcosa da raccontarmi ogni volta che ci sentiamo, che siano storie di gatti, caprette, o computer. Un giorno ti chiamerò per qualche progetto strampalato e tu accetterai di aiutarmi, me lo sento.

Thank you, Utku, for all the chats we had and for that last project that we squeezed in for both our PhDs. I still have and cherish the book you gave me at the airport in Sicily, kind of a survival guide for my career that helped me set the goals for these years.

Ringrazio Enrico, Marco, Roberto e Nicola per tutte le riunioni della "Gilda degli Ingegneri", rigorosamente tenute attorno a un tavolo di sushi.

Ringrazio il Sinnai Calcio a 5, per aver sempre offerto un posto dove allenarsi e alleviare lo stress quotidiano con allenamenti, partite, e tantissimi momenti condivisi.
Grazie a Mauro per aver sempre tenuto a spingere oltre i miei limiti per migliorare.
Grazie a Francesco per tutte le chiacchierate e quesiti matematici e ingegneristici, di cui non ne avremo mai abbastanza.
Ringrazio la Jasnagora, per avermi fatto vivere l'esperienza mistica di conciliare il dottorato e la serie A2, e per avermi sempre accolta a braccia aperte e col sorriso ogni volta che ci rivediamo.
Ringrazio Roby, Petra, Anna, e Maira, compagne di squadra da una vita con cui ho condiviso tante partite e importanti vittorie possibili solo grazie a costanza negli allenamenti e passione per lo sport.
Ringrazio Filipa per le lunghissime camminate che però sembrano sempre troppo brevi per raccontarci tutto quello che succede attorno a noi.

Ringrazio Maura, Manu e Sbè per i momenti trascorsi insieme dentro e (soprattutto) fuori dal campo. So sempre di poter contare su di voi, nonostante tutto. Ogni volta che sono con voi il tempo vola, ma nel frattempo come minimo abbiamo parlato di almeno cento argomenti diversi (perché bisogna sempre aggiornarsi su tutto), abbiamo scritto una canzone, e anche inventato un nuovo gioco da tavolo.

Ringrazio i miei colleghi e amici Luca e Kathrin, per tutte le avventure passate insieme in questo ultimo anno. Dalle prime "facili" passeggiate a un week end in campeggio con la tenda bucata e la pioggia, me le sono godute tutte e non vedo l'ora di farne tante altre.
Grazie a Luca, con cui condivido non solo tantissime serate di scrittura codice e videogiochi, ma anche un accesso non autorizzato al palazzo di Avast. Si è visto dall'inizio che non ci saremmo fermati davanti a nulla.
Grazie a Kathrin, con la quale non importa se stiamo camminando o nuotando, non finiremo mai gli argomenti di cui parlare. Ci saranno sempre nuovi posti da vedere, e nuove cose da cucinare.
Ringrazio Antonio per aver condiviso consigli culinari e tante risate, e ogni tanto qualche discussione scientifica. So solo che dopo aver parlato con te le cose sono sempre più chiare (sarà il vino?).

Ringrazio Waleska, la mia metà calcistica, per aver sempre messo se stessa tra il tiro e la mia porta, non c'è sicurezza più grande di lottare in due per affrontare le difficoltà più insidiose. Grazie per non essere mai mancata, nei momenti difficili e in

quelli dove si festeggia. Grazie per avermi sempre ricordato che quando siamo più stanche diamo anche di più, perché noi siamo fatte così.
Grazie anche a Manuela, Maria Assunta e Aldo per avermi accolta a casa loro e aver condiviso con me tanti pranzi e giornate insieme. Grazie per essere sempre i miei più grandi tifosi.

Ringrazio Luca (Rinaldo) che ha sempre assicurato cibo, conversazioni e divertimento in ogni serata passata in compagnia. Se qualcuno mi ha insegnato a godersi la vita, quello sei tu. Ho difficoltà a trovare un momento dove siamo stati seri, perché hai sempre un buonumore contagioso e qualcosa su cui farmi ridere.

Ringrazio la mia famiglia, senza di voi non avrei avuto modo di fare tutto questo. Grazie a Lucia e Carla, sorelle e complici per qualunque "monelleria".
Grazie a babbo, che ha sempre dato importanza ai momenti trascorsi insieme, e ha sempre avuto qualcosa da insegnarmi.
Un ringraziamento speciale va a mamma, che sarebbe stata fiera di me in questo momento, e che mi ha dato sempre tutto senza chiedere mai niente.

A tutte queste persone, e a molte altre, vanno i miei più sentiti ringraziamenti. Senza di loro, conseguire questo dottorato sarebbe stato impossibile. Grazie a loro, è stato una grande partita, di quelle che si vincono e fanno tornare a casa stanchi, ma col sorriso.

The research reported in this PhD thesis has been partly supported by BMK, BMDW, and the Province of Upper Austria in the frame of the COMET Programme managed by FFG in the COMET Module S3AI.