Advances in Reliably Evaluating and Improving Adversarial Robustness

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von

Jonas Rauber
aus Rottweil

Tübingen
2021

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	5. November 2021
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Matthias Bethge
2. Berichterstatterin:	Prof. Dr. Zeynep Akata
3. Berichterstatter:	Prof. Dr. Alois Knoll

Summary

Machine learning has made enormous progress in the last five to ten years. We can now make a computer, a *machine*, *learn* complex perceptual tasks *from data* rather than explicitly programming it. When we compare modern speech or image recognition systems to those from a decade ago, the advances are awe-inspiring.

The susceptibility of machine learning systems to small, maliciously crafted *adversarial perturbations* is less impressive. Almost imperceptible pixel shifts or background noises can completely derail their performance. While humans are often amused by the *stupidity* of artificial *intelligence*, engineers worry about the security and safety of their machine learning applications, and scientists wonder how to make machine learning models more robust and more human-like. This dissertation summarizes and discusses advances in three areas of adversarial robustness.

First, we introduce a new type of adversarial attack against machine learning models in real-world black-box scenarios. Unlike previous attacks, it does not require any insider knowledge or special access. Our results demonstrate the concrete threat caused by the current lack of robustness in machine learning applications.

Second, we present several contributions to deal with the diverse challenges around evaluating adversarial robustness. The most fundamental challenge is that common attacks cannot distinguish robust models from models with misleading gradients. We help uncover and solve this problem through two new types of attacks immune to gradient masking. Misaligned incentives are another reason for insufficient evaluations. We published joint guidelines and organized an interactive competition to mitigate this problem. Finally, our open-source adversarial attacks library *Foolbox* empowers countless researchers to overcome common technical obstacles. Since robustness evaluations are inherently unstandardized, straightforward access to various attacks is more than a technical convenience; it promotes thorough evaluations.

Third, we showcase a fundamentally new neural network architecture for robust classification. It uses a generative analysis-by-synthesis approach. We demonstrate its robustness using a digit recognition task and simultaneously reveal the limitations of prior work that uses *adversarial training*. Moreover, further studies have shown that our model best predicts human judgments on so-called controversial stimuli and that our approach scales to more complex datasets.

Zusammenfassung

Machine Learning hat in den letzten fünf bis zehn Jahren enorme Fortschritte gemacht. Heutzutage können wir Computer, *Maschinen*, dazu bringen, komplexe Wahrnehmungsaufgaben aus Daten zu *lernen*, anstatt sie explizit zu programmieren. Besonders moderne Sprach- und Bilderkennungssysteme erreichen im Vergleich zu denen von vor einem Jahrzehnt mittlerweile eine beeindruckende Genauigkeit.

Weniger beeindruckend ist die Anfälligkeit von Machine-Learning-Systemen für kleine, böswillig herbeigeführte Störungen. Kaum wahrnehmbare Hintergrundgeräusche oder Veränderungen ausgewählter Pixel können sie komplett in die Irre führen. Während Menschen sich oft über diese *Dummheit* künstlicher *Intelligenz* amüsieren, machen sich Entwickler Sorgen um die Sicherheit ihrer Machine-Learning-Anwendungen, und Wissenschaftler suchen nach robusteren Machine-Learning-Modellen, deren Wahrnehmung mehr der des Menschen entspricht. Diese Dissertation fasst Fortschritte in drei Bereichen rund um die Robustheit gegen gezielte Störungen zusammen und diskutiert ihre Implikationen.

Erstens stellen wir eine neue Art Attacke vor, die Machine-Learning-Anwendungen ganz unmittelbar angreifen kann. Im Gegensatz zu vorangegangenen Attacken erfordert sie weder Insiderwissen noch besonderen Zugang zum Modell. Unsere Ergebnisse zeigen die konkrete Bedrohung, die durch die derzeitig fehlende Robustheit von Machine-Learning-Anwendungen entsteht.

Zweitens präsentieren wir mehrere Arbeiten, die sich mit den verschiedenen Herausforderungen bei der Robustheits-Evaluierung befassen. Die grundlegendste Herausforderung dabei ist, dass gängige Testmethoden robuste Modelle nicht von Modellen mit irreführenden Gradienten unterscheiden können. Durch zwei neue Arten von Testmethoden, die immun gegen irreführende Gradienten sind, helfen wir, dieses Problem aufzudecken und zu lösen. Falsche Anreize sind ein weiterer Grund für fehlerhafte Evaluierungen. Um dieses Problem zu lindern, haben wir gemeinsame Richtlinien veröffentlicht und einen interaktiven Wettbewerb organisiert. Schlussendlich haben wir mit *Foolbox* eine Open-Source-Softwarebibliothek mit Testmethoden veröffentlicht, die unzähligen Forschern hilft, gängige technische Hindernisse beim Evaluieren von Modellen zu überwinden. Da die Evaluierung von Robustheit grundsätzlich nicht standardisiert werden kann, führt der einfache Zugang zu verschiedenen Testmetho-

den in der Praxis darüberhinaus zu gründlicheren Evaluierungen und verlässlicheren Ergebnissen.

Drittens haben wir eine völlig neue neuronale Netzwerk-Architektur entwickelt, die robustes Klassifizieren ermöglichen soll. Sie verwendet einen generativen Analysis-by-Synthesis-Ansatz. Am Beispiel eines Modells zur Ziffernerkennung demonstrieren wir die Robustheit dieser Architektur und zeigen gleichzeitig die Grenzen früherer Arbeiten auf, die *Adversarial Training* verwenden. Neuere Studien haben außerdem gezeigt, dass unser Modell die menschliche Wahrnehmung sogenannter kontroverser Stimuli besser als andere Modelle vorhersagt und dass unser Ansatz auch auf komplexere Datensätze skaliert.

Contents

Chapter 1

Introduction

This dissertation addresses the challenge of reliably evaluating and improving the adversarial robustness of machine learning models. Even though adversarial robustness only became a prominent field of research in the last few years, many of us have a decade-long personal connection to it. A spam filter screens every email sent to our address and decides what goes into our inbox and what is discarded right away. Nevertheless, most of us still receive spam emails. This is not simply because spam filters have not learned to recognize spam despite years of training. It is because spam emails are created with spam filters in mind (Dalvi et al. 2004). Spammers are the adversaries of our spam filters.

Spam filters are one of the first widespread applications of machine learning, the technique to make a computer, a *machine*, *learn* the operations needed to perform a task *from data* rather than explicitly programming it. In the last five to ten years, machine learning has advanced rapidly and revolutionized related fields such as computer vision, natural language processing, and speech recognition. Machine learning is now applied throughout the sciences, from archeology to astronomy. In commerce and industry, machine learning is considered a key driver for innovation and automation. For example, it is used to detect production defects, predict the need for maintenance, target advertisements, filter undesired content, forecast market demand, optimize logistics, recommend related products, or reduce pesticides. Under the buzzword *artificial intelligence*, we also see increasing use of machine learning in consumer products. For example, we now have voice assistants that understand what we say, cars that drive themselves, smartphones that recognize our faces, apps that identify plants, services that translate our texts, and wearables that call an ambulance when we become unconscious.

For some of these applications, it is hard to imagine anyone having adversarial interests. However, for other applications, an adversary who could mislead the machine learning model would pose a substantial security risk. So are such adversarial attacks against machine learning models practically possible in real-world scenarios? This is an important question, one that we would like to answer in this dissertation.

First Research Challenge: *Attacking machine learning models in real-world scenarios.*

The potential security threat of adversarial attacks is one reason for studying how to improve the adversarial robustness of machine learning models, but not the only one. There are two other reasons worth considering, the first of which is safety.

Security and safety may sound like two sides of the same coin. In fact, in German, they share the same word: *Sicherheit*. In this context, however, their goals can be nicely differentiated. Security aims at protecting the machine learning application from adversaries or, more generally, from the environment. In contrast, safety aims at protecting the environment (e.g., humans) from the machine learning system's inherent imperfection. Of course, both are related because attacks against insecure applications in safety-critical environments may cause unsafe behavior, but safety as such is an important goal even in settings without an explicit adversary. Imagine a machine learning model with sufficiently high accuracy to be, in theory, safely deployed in a particular application. In practice, however, the distribution of inputs in a real-world scenario might be ever so slightly different from the distribution seen during training. How can we guarantee safety when we do not precisely know how the inputs will change once the model is deployed? One solution is to aim for robustness against worst-case perturbations. If a model can safely handle worst-case perturbations, it can safely handle real-world perturbations.

A shift in perspective—away from applications—reveals the third, more fundamental reason for studying how to improve adversarial robustness: progress towards human abilities. Humans can, for example, usually explain their decisions. One approach to explaining a machine learning model's decision is to reveal the features crucial for that decision. A counterfactual explanation (Sokol et al. 2019) does so by asking the model for the smallest modification to its input that would change its prediction in a certain way. The same question also arises when we aim to influence human perception by targeting a machine learning model trained to predict human perception. Imagine, for example, a model that predicts where people look in images (Kümmerer et al. 2015). Influencing where people look is then—in principle—nothing else than asking the model for the smallest modification of the input image that would change the predicted gaze in the desired way. Mathematically, asking this question, in turn, is the same as adversarially attacking the model. We only get helpful counterfactual explanations (or perturbations that effectively influence the human gaze) if the model's minimal adversarial perturbations are large enough and semantically meaningful to affect humans.

Unfortunately, improving a model's adversarial robustness is not as straightforward as one might hope.

> *What you cannot measure, you cannot improve.*
> — Peter Drucker

This statement is not merely a platitude; before we can aim at improving adversarial robustness, we have to have a way to measure it.

Of course, some metrics are trivial to measure, and in such cases, the sole focus can be directly on improving them. *Test accuracy* is a case in point. Improving it can be an art in itself, but reliably and accurately estimating it is usually as trivial as it gets.

Estimating adversarial robustness is less trivial. There is no fixed algorithm that returns the one correct estimate of a model's adversarial robustness. Bounds on a model's robustness from above and below may hint at its actual robustness, but it remains doubtful whether these bounds are equally tight (or loose) for different models. To complicate matters further, a model's adversarial robustness is not even a single number but a distribution of individual robustness estimates, one for each data point. Depending on how the distributions are aggregated, one or the other model may seem more robust. Before attempting to *improve* a model's adversarial robustness, this dissertation thus puts particular emphasis on reliably *evaluating* adversarial robustness.

Second Research Challenge: *Reliably evaluating adversarial robustness.*

With progress on reliably evaluating adversarial robustness, we then have the tools, methods, and understanding to test new hypotheses for making machine learning applications safer and more secure and machine learning models more human-like. In short, we can finally aim at improving adversarial robustness.

Third Research Challenge: *Improving adversarial robustness.*

These are the three research challenges that underlie the work presented in this dissertation. Before we derive concrete research questions from these challenges, we review the relevant prior work and summarize important background information. More recent literature is being discussed as needed in the individual publications (chapter 2) as well as the overall discussion (chapter 3).

1.1 Definitions & Background

The term *adversarial examples* was first introduced in the context of modern machine learning in 2013 when it was found that by "applying an *imperceptible* non-random perturbation to a test image, it is possible to arbitrarily change the [...] prediction [of a neural network]" (Szegedy et al. 2013).[1] This *observation* should not be mistaken for a *definition* of adversarial examples (though, unfortunately, it often is).

Instead, adversarial examples are "inputs formed by applying small but intentionally worst-case perturbations to examples from the dataset, such that the perturbed input results in the model outputting an incorrect answer" (Goodfellow et al. 2015). Worst-case *adversarial perturbations* are generated by minimizing the perturbation under the constraint that the perturbed input is misclassified (or by maximizing the prediction error given a bound on the perturbation size); they are not necessarily imperceptible.

A classifier's *adversarial robustness* $r_{x,l}$ on an input x with label l can then be defined as the distance to the closest input $\tilde{x}$ that is not classified as l. For an input $x \in D$ with a label $l \in C$, this can be formalized as

$$r_{x,l} = \min_{\tilde{x} \in D} \; d(x, \tilde{x}) \quad \text{subject to} \quad f(\tilde{x}) \neq l,$$

where D is the input domain, $C = \{1, \dots, N\}$ is the set of classes, N is the number of classes, $d(x, y)$ is a distance metric, and $f : D \to C$ is the classifier. Note that the

[1] The adversarial robustness of classic machine learning algorithms like linear classifiers has been studied much earlier (e.g., Lowd et al. 2005).

above focus on classification is for simplicity; the definition can be generalized to other machine learning tasks if necessary. Unless f is a degenerated classifier that classifies everything into the same class, an $\tilde{x}$ that solves the above equation always exists.

For the common case of an *additive* perturbation δ, an input domain $D \subseteq \mathbb{R}^N$, and a simple p-norm, the equation simplifies to

$$r_{x,l} = \min_{\delta \in \mathbb{R}^N} \|\delta\|_p \quad \text{s. t.} \quad f(x+\delta) \neq l \quad \text{and} \quad x + \delta \in D.$$

After the minimization, $\tilde{x}$ and δ—the arguments of the minimization—represent the adversarial example and the adversarial perturbation, respectively.

With this definition, *adversarial attacks* are algorithms to approximate the minimum (to find adversarial examples that humans still classify like the original input and to measure the robustness r), not to find adversarial examples at all. For classifiers with high accuracy, practically any sample from the training set belonging to a different class is already an adversarial example, though, of course, an extremely sub-optimal one. Not the existence of adversarial examples is disturbing, but the observation that "[f]or all the networks we studied [...], for each sample, we have always managed to generate very close, visually hard to distinguish, adversarial examples" (Szegedy et al. 2013).

In contrast to adversarial attacks, *adversarial defenses* try to change (or replace) the classifier f such that larger perturbations are necessary to flip the classification (or, more generally, change the model's prediction). An optimal adversarial defense is one for which even the smallest possible adversarial perturbations are still so large that the human perception is influenced in ways similar to the machine learning model's perception.

Additionally, an optimal adversarial defense should classify unrecognizable inputs with low confidence. This goes back to an observation by Nguyen et al., who found that it is possible to create "images that are completely unrecognizable to humans, but that state-of-the-art DNNs believe to be recognizable objects with 99.99 % confidence" (Nguyen et al. 2015). Such (adversarially created) unrecognizable inputs are not classic adversarial examples because they are not perceptually similar to real data points and because they cannot be misclassified (for the lack of ground truth), but the fact that they are classified with high confidence reveals another important discrepancy between human and machine perception.

While the work by Szegedy et al. can be seen as starting a new field, the concept of an adversary that attempts to *evade* a machine learning system at test time was not new. In the computer security community, such attacks against machine learning systems are known as *evasion attacks* and were studied in particular in the context of spam classification and malware detection (Biggio, Corona, et al. 2013). *Evasion attacks* that perturb inputs at test time can be distinguished from *poisoning attacks* that manipulate the training data itself. Like evasion attacks, poisoning attacks are not new (Barreno et al. 2006; Rubinstein et al. 2009; Biggio, Nelson, et al. 2012) but have regained interest in the context of modern machine learning, for example, through works that *poison* the training data to introduce backdoors into neural networks (Gu et al. 2019; X. Chen et al. 2017; Yingqi Liu et al. 2018). Other adversarial problems include model stealing, where the attacker tries to obtain a copy of the model (Tramèr, F. Zhang, et al. 2016), and adversarial reprogramming, where the attacker tries to steal computational resources by repurposing a publicly accessible model (Elsayed et al. 2019). While these are all important adversarial problems, particularly from a security perspective, in this dissertation, we focus—as motivated in the introduction—on evasion attacks and the robustness against them and refer to them as *adversarial attacks* as defined above.

Adversarial Attacks

The space of adversarial attacks is huge. With dozens of new adversarial attacks or variations of other attacks in 2020 alone[1], there are far too many to list them all, let alone to go into detail. Instead, we will introduce the adversarial attacks that are most important, either for historical reasons or because they are most widely used or otherwise well known. We will focus primarily on those attacks that predate the attacks presented in this dissertation (sections 2.1 and 2.2).

For their first-ever adversarial attack against deep neural networks, Szegedy et al. (2013) reformulated the above optimization problem such that it can be (approximately) solved using L-BFGS-B (Byrd et al. 1995). While L-BFGS-B can handle the domain constraint $x + \delta \in D$ natively (assuming $D = [0, 1]^N$), the misclassification constraint $f(x + \delta) \neq l$ has to be continuously approximated (e.g., using the negative cross-entropy). It can then be integrated into the optimization through a weighted

[1] In 2020, hundreds of publications on adversarial robustness were published on arXiv (Carlini 2020). More than two hundred of them contain "adversarial attack" in their title, and a significant share of those are actual new attacks or attack variations.

combination with $\|\delta\|_p$. The relative weighting of the two terms determines the emphasis that L-BFGS-B puts on minimizing the perturbation vs. achieving misclassification when solving the resulting optimization problem

$$\min_{\delta \in \mathbb{R}^N} \; c\,\|\delta\|_p + \mathrm{loss}_{f,l}(x+\delta) \quad \text{s. t.} \quad x+\delta \in [0,1]^N.$$

The relative weight c of the minimization term is chosen using a line search that determines the largest c such that L-BFGS-B still returns a δ for which $f(x+\delta) \neq l$.

Instead of minimizing the negative cross-entropy between the predicted distribution and the label l, Szegedy et al. actually minimized the (positive) cross-entropy between the predicted distribution and some fixed target class $\hat{l} \neq l$ (and perform the line search such that $f(x+\delta) = \hat{l}$). Such *targeted attacks* can be more worrisome from a security perspective and more relevant if a dataset has many similar classes (e.g., ImageNet (Deng et al. 2009)), but for consistency and to avoid clutter, we will confine ourselves to the *untargeted* case when describing the following attacks.

The popular *Carlini-Wagner attack* is a second attack that builds on the same reformulation of the optimization problem as above. In contrast to the *L-BFGS-B attack*, Carlini and Wagner (2017b) changed the loss function ($\mathrm{loss}_{f,l}$) used to encourage misclassification from cross-entropy to the logit difference, avoid the box constraint ($x+\delta \in [0,1]^N$) using a change of variables, and—with the box constraint out of the way—then switched to the Adam optimizer (Kingma et al. 2015). Like the L-BFGS-B attack, the Carlini-Wagner attack is computationally expensive because the inner optimization has to be repeated for each step of the line search.

DeepFool (Moosavi-Dezfooli et al. 2016) sits at the other end of the spectrum; it is one of the fastest adversarial attacks because it radically approximates the problem. First, it assumes that the decision boundaries are linear. Second, it only considers the top k other classes (e.g., k = 10). Third, it analytically computes the optimal step to cross the (linear) decision boundary. To compensate for the linearity assumption, it performs a small overshoot (e.g., 5 %). If this approach is not successful (the class has not changed), it is repeated a few times until it is. Despite these coarse approximations, DeepFool can find surprisingly small adversarial perturbations.

The three adversarial attacks we have looked at so far all ultimately *minimize the perturbation* under the constraint that the perturbed input is misclassified. This approach follows naturally from the optimization problem stated above. Another group of adversarial attacks does, however, take the orthogonal approach. They attempt to *maximize misclassification* (e.g., maximize the cross-entropy between the predicted distribution and the label) of the perturbed input under the constraint that the perturbation is smaller than some bound ϵ. To achieve this, they perform gradient ascent on the loss and project or clip the result to the ϵ-ball. The size of the returned perturbations is then no longer minimal but predefined as ϵ. This approach is taken by popular adversarial attacks such as FGSM, BIM, and PGD.

The *Fast Gradient Sign Method* FGSM (Goodfellow et al. 2015) can be seen as the single-step special case of these attacks. It simply perturbs the input in the direction of the pointwise sign of the gradient scaled with ϵ (and clips the result to the valid space). It takes the sign because it assumes ϵ to be an L_∞ bound. The L_2 equivalent of the attack (FGM) omits the sign and scales the L_2-normalized gradient instead. FGSM and FGM are arguably the fastest adversarial attacks.[1] Importantly, this is the very reason why FGSM was created: to efficiently construct adversarial examples on the fly during training (in an attempt to perform adversarial training, which we will discuss in a moment). In other words, FGSM was not meant for robustness evaluation but, unfortunately, it is often misused for precisely that.

The *Basic Iterative Method* BIM (Kurakin et al. 2017) was motivated as the multi-step iterative extension of FGSM. It repeatedly takes the sign of the gradient, rescales it with an additional step size parameter, perturbs the input, and clips the result to the input domain and the ϵ-ball.

The *Projected Gradient Descent attack* PGD (Madry et al. 2018) is, in effect, nothing else than the Basic Iterative Method except that it is typically started from a random point within the ϵ-ball rather than from the original unperturbed input. It thus avoids potentially relatively flat regions around the data points, and it can be restarted several times with different random initializations (something that has been shown to make the attack much more effective (Mosbach et al. 2018)). In practice, PGD also uses smaller step sizes and more steps than BIM. The increased number of steps and the repeated random restarts make PGD more effective than BIM at the cost of increased computational needs.

Other noteworthy adversarial attacks—excluding ours[2]—are JSMA, EAD, MIM, DDN,

[1] FGSM and FGM still have to pay the cost of computing the gradient. Simply perturbing an input in the direction of a data point from another class could be seen as an even faster (yet extremely naive) adversarial attack.

[2] The above list excludes the attacks that we will introduce later, our Boundary Attack (section 2.1) and our

and—last but not least—transfer attacks. The *Jacobian-Saliency-Map-Attack* JSMA (Papernot, McDaniel, Jha, et al. 2016) was the first attack that minimized the L_0 norm of the perturbation. The *EAD attack* (P.-Y. Chen, Sharma, et al. 2018) generalized the L_0, L_2, L_∞ Carlini-Wagner attack to the L_1 norm. The *Momentum Iterative Method* MIM (Dong et al. 2018) added a momentum term to BIM and PGD. And the *DDN attack* improves attack efficiency by "decoupling the direction and the norm of the adversarial perturbation" (Rony et al. 2019).

The term *transfer attack* does not describe yet another adversarial attack but an alternative method to indirectly attack a model with any of the above adversarial attacks. Transfer attacks work as follows. First, the actual attack is run against some other model, the *substitute model*. The obtained adversarial example is then tested on the target model. Interestingly, even though the adversarial examples were created for the substitute model, they can fool the target model.

This surprising *transferability* of adversarial examples, of course, depends on different factors. The more similar the two models are, the better the adversarial examples transfer from one to the other. Targeted adversarial examples transfer best if they are sourced from an ensemble of models (Yanpei Liu et al. 2017). Papernot, McDaniel, Goodfellow, et al. (2017) showed that a good substitute model can be trained by creating labels on-the-fly from the target model's predictions (it still helps to have a similar architecture). The size of the adversarial perturbation also plays a role. Minimal adversarial perturbations (found by "better" adversarial attacks) are less likely to transfer than larger perturbations because they are sensitive to small changes of the decision boundary. Thus, it works best to transfer only the direction of the adversarial perturbation and then perform a line search until the target model is fooled or, more simply, to upscale the perturbation a bit. Transfer attacks require access to the target model to test different samples (possibly already needed to train the substitute model), but all the gradient information comes from the substitute model.

The transferability of adversarial examples between similar models even allows us to attack machine learning models operating in the physical world. The physical transformation caused by the camera and the environment is noisy and variable, and gradients cannot be backpropagated through it, but upscaled adversarial perturbations created for the machine learning model behind the camera can survive physical transforma-

Linear Region Attack (section 2.2), as well as our other attacks, the Pointwise Attack (Schott et al. 2019) and the Brendel Bethge Attack (Brendel, Rauber, Kümmerer, et al. 2019).

tions such as printing and recording with a camera and transfer to the physical model (Kurakin et al. 2017).

Adversarial Attack Libraries

The previous section contains references to fourteen different adversarial attacks, and these are just the most important ones. Even if all authors were sharing reference implementations of their attacks, evaluating one's own model with the various adversarial attacks would be difficult because of inconsistent interfaces, assumptions, and framework requirements. CleverHans (Papernot, Goodfellow, et al. 2016), an adversarial attacks library for TensorFlow (Abadi et al. 2016), was the first library seeking to solve this problem by establishing common standards. Other noteworthy adversarial attack libraries—excluding ours[1]—are AdverTorch (Ding et al. 2019) and ART (Nicolae et al. 2018). They all come with their own advantages and disadvantages (see sections 2.3 and 2.4).

Adversarial Defenses

Szegedy et al. (2013) not only attacked deep neural networks (and discovered their adversarial susceptibility) but also already *suggested* (but not implemented) a first potential adversarial defense: *adversarial training*. The number of proclaimed adversarial defenses—like the number of adversarial attacks—has since skyrocketed. Proposed defense mechanisms include distillation ([#1]Papernot, McDaniel, Wu, et al. 2016), saturating activations ([#2]Nayebi et al. 2017), input discretization ([#3]Buckman et al. 2018), regularization ([#4]Kannan et al. 2018), input transformations ([#5]Guo et al. 2018), preprocessing ([#6]Bafna et al. 2018; [#7]Y. Yang et al. 2019), new activation functions ([#8]Zantedeschi et al. 2017; [#9]Xiao et al. 2020), new loss functions ([#10]Pang, Xu, Dong, et al. 2020), learned projections ([#11]Meng et al. 2017; [#12]Shen et al. 2019; [#13]Song et al. 2018; [#14]Samangouei et al. 2018), denoising ([#15]Liao et al. 2018), stochasticity ([#16]Xie et al. 2018; [#17]Prakash et al. 2018; [#18]Pang, Xu, and Zhu 2020), generative models ([#19]Y. Li et al. 2019), pruning ([#20]Dhillon et al. 2018), outlier detection ([#21]Roth et al. 2019; [#22]Ma et al. 2018; [#23]Z. Yang et al. 2019; [#24]Hu et al. 2019), and ensembling ([#25]Verma et al. 2019; [#26]Pang, Xu, Du, et al. 2019; [#27]Sen et al. 2020).

Unfortunately, the above defense mechanisms are not robust; all of the above publica-

[1] Our own adversarial attack libraries, Foolbox (Rauber, Brendel, et al. 2017) and Foolbox Native (Rauber, Zimmermann, et al. 2020), will be introduced in sections 2.3 and 2.4, respectively.

tions overestimated their robustness (Carlini and Wagner 2016 *broke #1*; Brendel and Bethge 2017 *broke #2*; Carlini and Wagner 2017a *broke #8, 11, 12*; Athalye, Carlini, and Wagner 2018 *broke #3, 5, 13, 14, 16, 20, 22*; Athalye and Carlini 2018 *broke #15, 17*; Engstrom et al. 2018 *broke #4*; *and* Tramèr, Carlini, et al. 2020 *broke #6, 7, 9, 10, 18, 19, 21, and 23–27*). Moreover, the above defenses' problems were not spotted after years of improvements in evaluating adversarial robustness, but often shortly after publication. For example, Athalye, Carlini, and Wagner (2018) and Athalye and Carlini (2018) broke as many as nine defenses within days after publication at ICLR 2018 and CVPR 2018, respectively. The defense *suggested* by Szegedy et al.—adversarial training—is an exception to this rule.

Adversarial training augments the training data with adversarially perturbed training samples. Similar to how data augmentation can make models more robust against particular types of noise or perturbations, adversarial training is expected to improve the robustness against adversarial perturbations. The adversarial perturbations have to be created on-the-fly because—unlike random noise—they depend on the model (and the model changes in every training step). This is why Szegedy et al. (2013) only *suggested* adversarial training; their L-BFGS-B attack was too slow to try adversarial training in practice (given the computational resources at the time).

FGSM—as mentioned above—was specifically created to overcome this problem (Goodfellow et al. 2015). It was fast enough to create adversarial perturbations on-the-fly. After the adversarial training, Goodfellow et al. tested their model's robustness against the FGSM attack and—as hoped—found that the model was robust. Unfortunately (or fortunately), Tramèr, Papernot, et al. (2017) later showed that it was only robust against FGSM and not against adversarial attacks in general. Goodfellow et al. had overestimated the model's robustness because they evaluated it with the same attack they had used during training.

This raised the question of whether adversarial training would always overfit to the specific attack (or attacks) used during training or whether it could lead to real robustness against any attack. This question was empirically answered by Madry et al. (2018). Using PGD with 40 steps instead of FGSM to create tiny adversarial perturbations on-the-fly, they adversarially trained both an MNIST (LeCun et al. 1998) and a CIFAR-10 (Krizhevsky et al. 2009) model. Again, both models seemed much more robust than standard models, but to be sure, they invited the research community to scrutinize their results. Despite great efforts, the independent evaluations revealed only a small overestimation of robustness by Madry et al. and, overall, confirmed the robustness of both models against adversarial perturbations (with a bounded L_∞ norm)—in contrast to the many failed defenses above.

Robustness Guarantees

The underlying problem of all the failed robustness evaluations is that adversarial attacks give an upper bound on the adversarial robustness by constructing adversarial examples. Whether this upper bound is tight or loose is hard to know. Lower bounds on the size of the minimal adversarial perturbation, on the other hand, would guarantee a certain robustness. Deriving such guarantees, like developing better adversarial attacks, is an active area of research (Hein et al. 2017; Raghunathan et al. 2018; Anil et al. 2019; Croce, Andriushchenko, and Hein 2019; Q. Li et al. 2019; Croce and Hein 2020). Despite substantial improvements, there remains a large gap between the lower bounds (guarantees) and the upper bounds (attacks).

1.2 Research Questions

In the introduction, we have taken different perspectives on adversarial examples and identified three research challenges to focus on. In this section, we revisit each of these three research challenges (RC1, RC2, and RC3) in light of the existing prior work and derive concrete open research questions that we aim to answer in this dissertation.

Figure 1 on page 30 provides a comprehensive overview of the relationships between the research challenges and research questions identified in this chapter, the results and publications summarized in chapter 2, and the overall discussions in chapter 3.

Attacking Machine Learning Models in Real-World Scenarios (RC1)

One property all the adversarial attacks reviewed above have in common is that they use gradient descent. This does not come as a surprise. Neural networks are trained with gradient descent and differentiable by design, so it seems obvious that adversarial attacks should exploit this.

When trying to attack models in real-world scenarios, using gradient descent becomes a problem. Machine learning models deployed in applications are black-boxes. Outside access to the models' gradients is an unrealistic assumption, and so the above adversarial attacks cannot be used directly. Transferring adversarial examples from a similar model offers a way out, but only if we already know a similar model (Kurakin et al. 2017) or can access the training data to construct one (Papernot, McDaniel, Goodfellow, et al. 2017). Moreover, transfer attacks operate on a relatively coarse scale; the adversarial perturbations are typically much larger than those found by direct attacks. Without gradients or a good proxy model, none of the existing attacks are possible. This raises the following question:

Research Question 1: *Are adversarial attacks a threat to real-world black-box machine learning models, or is robustness through obscurity a viable option?*

We address this question in our work on decision-based adversarial attacks (section 2.1) and discuss its implications in section 3.1.

Reliably Evaluating Adversarial Robustness (RC2)

The following four research questions represent different aspects of what makes it difficult to evaluate adversarial robustness.

When CleverHans was first released in 2016, it solved an important problem: combining various adversarial attacks into one joint library. It was no longer necessary to study each attack's code individually and to adapt the evaluation to the respective interface. What CleverHans—a TensorFlow library—could not solve were the problems caused by the different deep learning frameworks. TensorFlow was in its infancy at the time; many researchers (and thus models) were still using Theano (Team et al. 2016) and Lasagne (Dieleman et al. 2015) or Caffe (Jia et al. 2014). The dominant frameworks changed over time, but the fundamental problem remained. In early 2017, before new adversarial attack libraries for other frameworks emerged, we, therefore, wondered whether adversarial attacks could be implemented in a future-proof way—independent of a specific deep learning framework—so that different frameworks could be supported as they would emerge or become obsolete. Besides, a framework-agnostic adversarial attacks library would ensure consistent and comparable results across frameworks.

Research Question 2: *How can we build an adversarial attacks library that supports evaluating the robustness of machine learning models implemented in different deep learning frameworks?*

A solution to this problem (see section 2.3 for our attempt) would be an important first step to make evaluations possible. In practice, new problems occur as soon as we do not just evaluate the robustness of simple standard models created without considering adversarial examples but instead analyze defended models (also known as *defenses*) that are supposed to be robust against adversarial examples.

Gradient masking (Papernot, McDaniel, Goodfellow, et al. 2017) is the most prominent issue when evaluating the adversarial robustness of a defended model. Gradients that are masked on purpose are also known as obfuscated gradients (Athalye, Carlini, and Wagner 2018). This is not the only form of gradient masking. In practice, gradients can become unreliable accidentally just as easily as they can be obfuscated intentionally.

The idea behind intentional gradient masking is *robustness through obscurity*. All the common adversarial attacks, e.g., DeepFool, the Carlini-Wagner attack, or PGD, require reliable gradients. When the gradient backpropagation through the neural network is disturbed, e.g., by introducing non-differentiable elements, these adversarial attacks will fail or at least be less effective. The size of the adversarial perturbations found by the attacks will grow, and the “defended” model will seem more robust. Obviously, this does not improve the true robustness, but it makes it more difficult to find small adversarial perturbations.

Accidental gradient masking is even more tricky because it might easily stay unnoticed. It can be caused by defense mechanisms that look innocent. Even adversarial training—without any change to the architecture—might cause some amount of gradient masking because it encourages flat regions around training samples, and in these flat regions, it can be difficult to perform gradient descent (Madry et al. 2018, updated version from September 4, 2019). Consequently, potential gradient masking is an omnipresent problem and deserves appropriate attention.

Research Question 3: *How can we find small, close to minimal adversarial perturbations despite gradient masking?*

We explore two alternative approaches to answer this question: decision-based adversarial attacks (section 2.1) and architecture-based adversarial attacks (section 2.2). Together, they reveal a common misconception about *black-box* and *white-box* attacks that we discuss in section 3.2.

Apart from the quite technical gradient masking problem, the evaluation of adversarial defenses suffers from a significant incentive problem. Researchers that want to claim robustness have little incentive to prove themselves wrong. The lack of incentives does not require willful misconduct to become a problem; even unintentionally, it can cause an insufficient evaluation. Whether at ICLR (Athalye, Carlini, and Wagner 2018), ICML (Tramèr, Carlini, et al. 2020), CVPR (Athalye and Carlini 2018), or NeurIPS (Tramèr, Carlini, et al. 2020), the current peer-review process has proven inadequate to catch even common evaluation problems—and calls for changes.

Research Question 4: *How can we incentivize strong robustness evaluations and ensure an adequate peer-review process?*

We tested three different ideas to create new incentives and to improve the peer-review process. First, we joined efforts with several other groups to publish comprehensive guidelines on evaluating adversarial robustness (Carlini, Athalye, et al. 2019). In this way, we hoped to increase the awareness of both reviewers and defense researchers for common pitfalls and best practices. Second, to make the review process itself more adversarial, we organized a competition that pitched proposed defenses against active, opposing evaluators (Brendel, Rauber, Kurakin, Papernot, Veliqi, Salathé, et al. 2018). Third, to simulate this approach in a standard review process, we assigned different roles to different co-authors when publishing our own defense (Schott et al. 2019). We discuss all three ideas and their effectiveness in section 3.3.

The above research questions illustrate the progress made in recent years. The second research question was aimed at laying the technical foundation to evaluate adversarial robustness. The third and the fourth research question concerned the two main issues when evaluating adversarial robustness: gradient masking and the incentive problem, respectively. The fifth research question will now focus on making robustness evaluations fast.

> *Make it work. Make it right. Make it fast.*
>
> — Kent Beck

Running adversarial attacks as fast as possible is not merely about saving time. That is not to say that saving time is not important: Of course, saving time is economically and ecologically beneficial and facilitates rapid iteration. However, the ability to run an adversarial attack *faster* can also be correlated with making it *stronger*. Running an attack for more steps (e.g., when performing gradient descent) or restarting an attack multiple times (e.g., when the attack entails randomness) generally makes the attack more effective at reducing the size of the adversarial perturbations. As there are no fixed evaluation procedures that dictate a certain number of iterations or restarts (for reasons being discussed in section 3.3) and adversarial attacks are expensive to run, researchers often have to trade off strength (iterations, restarts) against what they can afford (time, money). Faster attack implementations can thus promote stronger evaluations.

Unfortunately, using a highly optimized deep learning framework to run adversarial attacks natively on GPUs for maximum performance is at odds with supporting different deep learning frameworks (research question 2). This conflict has led to several adversarial attack libraries that all sacrifice one or the other: our original Foolbox library (section 2.3)—developed in response to the second research question—and ART (Nicolae et al. 2018) sacrifice performance while CleverHans (Papernot, Goodfellow, et al. 2016) and AdverTorch (Ding et al. 2019) sacrifice the advantages of supporting different frameworks. Resolving this conflict would liberate researchers from the need to use different libraries in different situations.

Research Question 5: *Can we resolve the contradiction between performance and support for different deep learning frameworks?*

We attempt to answer this question in section 2.4. With the above four research questions on evaluating adversarial robustness (and our answers in chapter 2), we have laid the groundwork for our ultimate goal: improving adversarial robustness.

Improving Adversarial Robustness (RC3)

Since Madry et al. (2018) showed how to make *adversarial training* effective, it is often seen as the solution to the problems posed by adversarial examples. Conceptually that makes sense. Adversarial training is a form of *robust optimization*, a well-known approach to achieve robustness in worst-case scenarios. Nevertheless, considering the numerous variables concerning the definition of adversarial robustness (e.g., the various datasets or the different metrics), it is worth investigating the current limitations of adversarial training.

Research Question 6: *Does adversarial training with PGD—as done by Madry et al.—solve the problems posed by adversarial examples, or what are its limitations?*

This question is not answered simply by assessing the correctness of Madry et al.'s results but also depends on the relevance of the setting they considered and on the generalizability of their results to other settings.

Either way, adversarial training is a naive “brute-force” approach that tries to teach robustness through plenty of examples. It would be great if we could instead design neural networks with built-in robustness. They might be more efficient, might guarantee robustness, and might be closer to human perception in other ways as well.

Research Question 7: *Can a neural network architecture be robust by design rather than through adversarial training?*

In section 2.5, we try to find such an inherently robust architecture by taking inspiration from the generative understanding of the physical world attributed to humans.

1.3 Publications

I worked on these research questions together with numerous collaborators. Below, I list the publications that have resulted from these collaborations together with the names of all co-authors. The publications are grouped according to their role in this dissertation. A star indicates joint first authorship.

Publications Included in This Dissertation

The six publications below—four peer-reviewed conference papers or journal articles, one workshop contribution, and one manuscript—form the foundation of this dissertation. They are included in full in the appendix, and their main motivation, results, and discussion are summarized in the next chapter. In addition, the appendix contains a detailed description of each author’s contribution to each paper.

- Wieland Brendel*, Jonas Rauber*, Matthias Bethge (2018). “Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models”. *Sixth International Conference on Learning Representations (ICLR 2018).*
- Francesco Croce*, Jonas Rauber*, Matthias Hein (2020). “Scaling up the Randomized Gradient-Free Adversarial Attack Reveals Overestimation of Robustness Using Established Attacks”. *International Journal of Computer Vision (IJCV) 128:1028-1046.*
- Jonas Rauber*, Wieland Brendel*, Matthias Bethge (2017). “Foolbox: A Python

toolbox to benchmark the robustness of machine learning models". *Reliable Machine Learning in the Wild Workshop, 34th International Conference on Machine Learning (ICML 2017).*

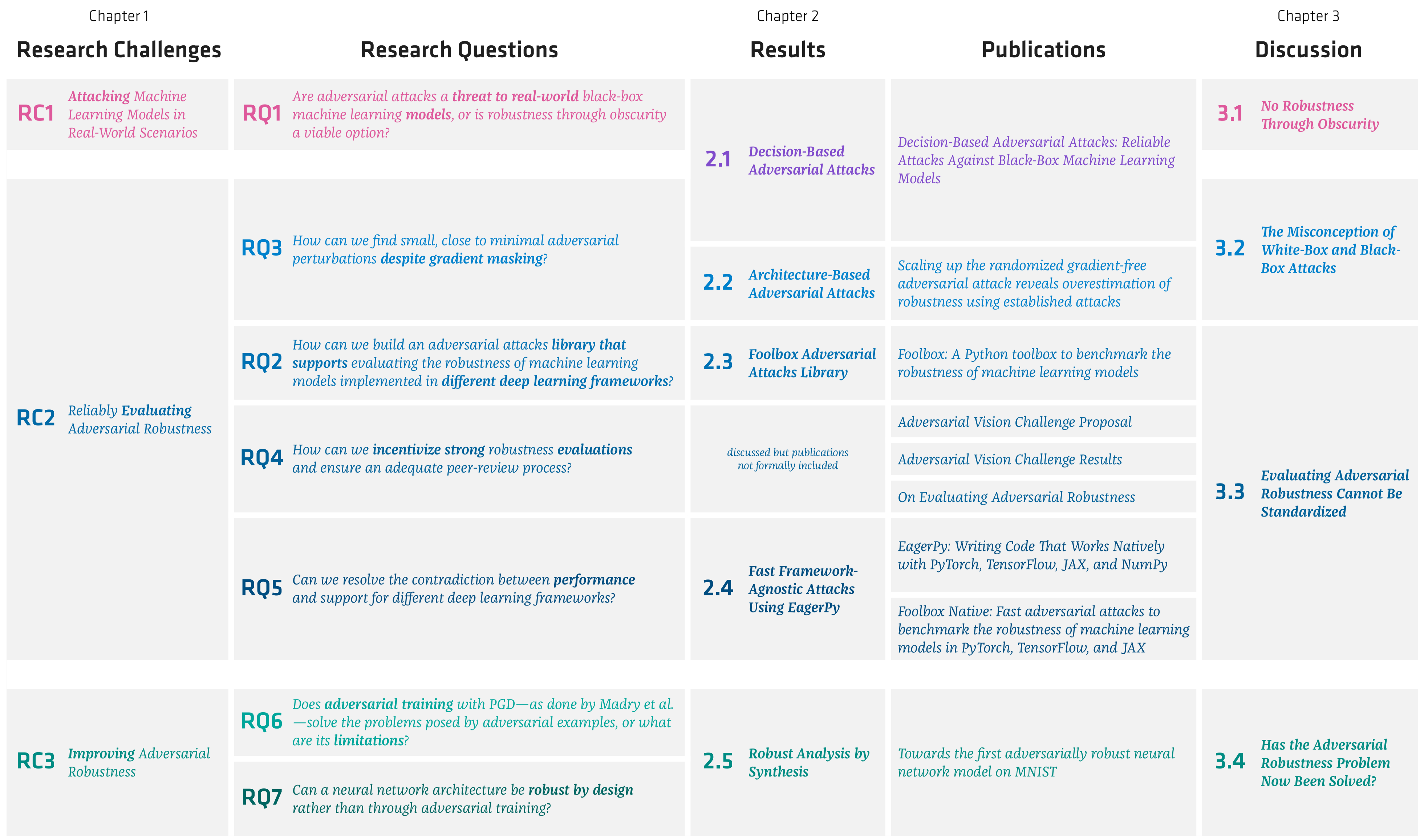
- Jonas Rauber, Matthias Bethge, Wieland Brendel (2020). "EagerPy: Writing Code That Works Natively with PyTorch, TensorFlow, JAX, and NumPy". *Preprint available on arXiv, 2008.04175.*
- Jonas Rauber, Roland Zimmermann, Matthias Bethge, Wieland Brendel (2020). "Foolbox Native: Fast adversarial attacks to benchmark the robustness of machine learning models in PyTorch, TensorFlow, and JAX". *Journal of Open Source Software, 5(53), 2607.*
- Lukas Schott*, Jonas Rauber*, Matthias Bethge, Wieland Brendel (2019). "Towards the first adversarially robust neural network model on MNIST". *Seventh International Conference on Learning Representations (ICLR 2019).*

Related Work Not Included in This Dissertation

The following five publications comprise two peer-reviewed conference papers, a competition proposal, a living document, and a book chapter. They are not formally included in this dissertation but will, in some cases, be referred to in the joint discussion in chapter 3.

- Wieland Brendel, Jonas Rauber, Alexey Kurakin, Nicolas Papernot, Behar Veliqi, Marcel Salathé, Sharada P. Mohanty, Matthias Bethge (2018). "Adversarial Vision Challenge". *Competition Track of the Thirty-second Conference on Neural Information Processing Systems (NeurIPS 2018).*
- Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, Alexey Kurakin (2019). "On Evaluating Adversarial Robustness". *Living Document (arXiv:1902.06705).*
- Robert Geirhos*, Carlos R. Medina Temme*, Jonas Rauber*, Heiko H. Schütt, Matthias Bethge, Felix A. Wichmann (2018). "Generalisation in humans and deep neural networks". *Advances in Neural Information Processing Systems 31 (NeurIPS 2018).*
- Wieland Brendel, Jonas Rauber, Matthias Kümmerer, Ivan Ustyuzhaninov, Matthias Bethge (2019). "Accurate, reliable and fast robustness evaluation". *Advances in Neural Information Processing Systems 32 (NeurIPS 2019).*

- Wieland Brendel, Jonas Rauber, Alexey Kurakin, Nicolas Papernot, Behar Veliqi, Sharada P. Mohanty, Florian Laurent, Marcel Salathé, Matthias Bethge, Yaodong Yu, Hongyang Zhang, Susu Xu, Hongbao Zhang, Pengtao Xie, Eric P. Xing, Thomas Brunner, Frederik Diehl, Jérôme Rony, Luiz Gustavo Hafemann, Shuyu Cheng, Yinpeng Dong, Xuefei Ning, Wenshuo Li, Yu Wang (2020). "The NeurIPS '18 Competition: Adversarial Vision Challenge". *The Springer Series on Challenges in Machine Learning.*



Chapter 1 Research Challenges	Research Questions	Chapter 2 Results	Publications	Chapter 3 Discussion
RC1 ***Attacking*** *Machine Learning Models in Real-World Scenarios*	**RQ1** *Are adversarial attacks a **threat to real-world** black-box machine learning **models**, or is robustness through obscurity a viable option?*	**2.1** ***Decision-Based Adversarial Attacks***	*Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models*	**3.1** ***No Robustness Through Obscurity***
RC2 *Reliably **Evaluating** Adversarial Robustness*	**RQ3** *How can we find small, close to minimal adversarial perturbations **despite gradient masking**?*	**2.2** ***Architecture-Based Adversarial Attacks***	*Scaling up the randomized gradient-free adversarial attack reveals overestimation of robustness using established attacks*	**3.2** ***The Misconception of White-Box and Black-Box Attacks***
	RQ2 *How can we build an adversarial attacks **library that supports** evaluating the robustness of machine learning models implemented in **different deep learning frameworks**?*	**2.3** ***Foolbox Adversarial Attacks Library***	*Foolbox: A Python toolbox to benchmark the robustness of machine learning models*	**3.3** ***Evaluating Adversarial Robustness Cannot Be Standardized***
	RQ4 *How can we **incentivize strong** robustness **evaluations** and ensure an adequate peer-review process?*	*discussed but publications not formally included*	*Adversarial Vision Challenge Proposal* *Adversarial Vision Challenge Results* *On Evaluating Adversarial Robustness*	
	RQ5 *Can we resolve the contradiction between **performance** and support for different deep learning frameworks?*	**2.4** ***Fast Framework-Agnostic Attacks Using EagerPy***	*EagerPy: Writing Code That Works Natively with PyTorch, TensorFlow, JAX, and NumPy* *Foolbox Native: Fast adversarial attacks to benchmark the robustness of machine learning models in PyTorch, TensorFlow, and JAX*	
RC3 ***Improving*** *Adversarial Robustness*	**RQ6** *Does **adversarial training** with PGD—as done by Madry et al.—solve the problems posed by adversarial examples, or what are its **limitations**?*	**2.5** ***Robust Analysis by Synthesis***	*Towards the first adversarially robust neural network model on MNIST*	**3.4** ***Has the Adversarial Robustness Problem Now Been Solved?***
	RQ7 *Can a neural network architecture be **robust by design** rather than through adversarial training?*			

Figure 1: Overview of the relationships between different parts of this dissertation.

Chapter 2

Results

2.1 Decision-Based Adversarial Attacks

This section summarizes:

Wieland Brendel*, Jonas Rauber*, Matthias Bethge (2018). "Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models". *Sixth International Conference on Learning Representations (ICLR 2018).*

The full publication can be found in the appendix on page 77. * joint first authorship

Motivation

Many existing adversarial attacks rely on gradients (section 1.1) or scores (e.g., P.-Y. Chen, H. Zhang, et al. 2017; Narodytska et al. 2017) to find small adversarial perturbations. Unfortunately, gradients and scores are unreliable sources of information. It has been shown repeatedly that gradients can be easily masked or obfuscated in various ways, both intentionally and unknowingly (Papernot, McDaniel, Goodfellow, et al. 2017; Athalye, Carlini, and Wagner 2018). Scores (e.g., probabilities) can be equally easily manipulated.

With existing adversarial attacks being so easily averted, judging whether a model is genuinely robust or just insufficiently evaluated is difficult. This problem is the root cause of many premature robustness claims (see research question 3).

Attacking deployed machine learning models is even more difficult. Deployed models are usually black-boxes whose inner workings, such as gradients, cannot be accessed at all. This raises the question whether *robustness through obscurity* may actually be a viable option (see research question 1).

However, both scenarios—gradients and scores being unreliable or unavailable—leave us with one source of information: the actual decisions. A model's decisions are always reliable because any defense that tampers with them by definition jeopardizes the model's accuracy and thus defeats its whole purpose. Similarly, even black-box, real-world machine learning models cannot bluntly hide their decisions because their decisions are how they affect the world. Adversarial attacks that only rely on a model's decisions thus have the potential to attack real-world machine learning systems and might be less prone to simple, non-robust pseudo-defense like gradient masking.

To capture the characteristics of these attacks, we termed them *decision-based adversarial attacks* and introduced a new taxonomy that distinguishes adversarial attacks based on the information or mechanism they depend on. It separates *decision-based attacks* from *score-based attacks* that require meaningful confidence scores (Narodytska et al. 2017; P.-Y. Chen, H. Zhang, et al. 2017), from *transfer-based attacks* that depend on a substitute model from which adversarial perturbations can be transferred (Papernot, McDaniel, Goodfellow, et al. 2017), and from the common *gradient-based attacks* that rely on informative gradients (see section 1.1).[1]

The goal of this study was to test whether *decision-based adversarial attacks* are indeed able to find small adversarial perturbations, attack deployed black-box machine learning models, and break existing, non-robust defense mechanisms. Inspired by the most naive decision-based attacks—noise attacks that randomly sample large perturbations until the input is misclassified (Rauber, Brendel, et al. 2017)—we created the *Boundary Attack*. The Boundary Attack is the first non-trivial decision-based adversarial attack able to attack models directly. Additionally, it can handle various definitions of adversarial examples, including untargeted and targeted attacks. Simply put, it starts with a far-away adversarial example (e.g., by picking a real, correctly-classified example from a different class), then moves towards the decision-boundary using a line search, and finally travels along the decision-boundary (always staying on the adversarial side) us-

[1] Croce and Hein (2019) and our follow-up work (Croce, Rauber, et al. 2020) later introduced a new adversarial attack that exploits the piecewise linearity of standard deep neural network architectures. This attack extends our taxonomy with a fifth attack type (see figure 2, section 3.2). For consistency and following our new nomenclature, I call it *architecture-based attacks* (section 2.2).

ing a random rejection sampling scheme to find ever smaller adversarial perturbations until it eventually converges.

Results

We found that our decision-based adversarial attack is competitive with common gradient-based adversarial attacks in terms of perturbation size in both untargeted and targeted scenarios. We tested our attack on standard computer vision models for MNIST, CIFAR-10, and ImageNet and compared the median minimal L_2 adversarial perturbation size with established gradient-based attacks such as DeepFool (Moosavi-Dezfooli et al. 2016) and the Carlini-Wagner attack (Carlini and Wagner 2017b). Our median perturbation size is always within a factor of two compared to the best attack and often better than at least either DeepFool or Carlini-Wagner.

Running our attack against a model trained with defensive distillation (Papernot, McDaniel, Wu, et al. 2016), a defense known to introduce gradient masking rather than truly increasing the robustness (Carlini and Wagner 2016), confirmed our hypothesis that decision-based adversarial attacks can work well when gradient-based attacks fail because of gradient masking.

To demonstrate that decision-based adversarial attacks pose a realistic threat to real-world models, we attacked celebrity and logo recognition models hosted by Clarifai, a web service that offers (black-box) access to various machine learning and computer vision models. We found that even with the reduced number of queries available in real-world scenarios, the Boundary Attack can find adversarial examples that are practically indistinguishable from the original images.

Discussion

Our results not only confirmed that the Boundary Attack could break defenses *thanks to its restriction* to decisions but also proved that it could quite effectively minimize the adversarial perturbations *despite this limitation*.

> *"It's surprising that something so simple works so well."*
> — Nicholas Carlini, 2017

This was surprising because a priori, there was no reason to assume that traveling along the decision-boundary would not quickly get stuck in local minima, and it asks for caution. The Boundary Attack may still rely on implicit assumptions, local minima may still be a problem in other cases, and non-robust defense mechanisms other than standard gradient masking may still render the attack ineffective. As with any other attack, its ineffectiveness should not be misinterpreted as proof of a model's robustness. The Boundary Attack could, however, have caught many false robustness claims prior to publication. Decision-based attacks in general, and the Boundary Attack in particular, are thus an excellent addition to our arsenal of tools to assess the robustness of machine learning models.

Our experiments also raised new questions regarding the security and safety of machine learning applications. In the past, deployed machine learning models seemed relatively safe from attacks. Their gradients and scores could be hidden from outside attackers, and transferring adversarial perturbations required a good substitute model (or a well-matching architecture and access to the training data, neither of which may be available). Our direct attack against a deployed black-box machine learning model destroyed this illusion of *robustness through obscurity*. Consequently, the security and safety of machine learning applications should be reevaluated.

Given the unique advantages of decision-based adversarial attacks for assessing the robustness of machine learning models and for attacking models in black-box settings, our new fine-grained taxonomy is well justified. Our L_2 Boundary Attack was just the first of its kind. We have since complemented it with the L_0 Pointwise Attack (Schott et al. 2019), and numerous follow-up works have improved its query-efficiency (e.g., Jianbo Chen et al. 2020; H. Li et al. 2020; Maho et al. 2020) or introduced decision-based attacks for other norms (Ilyas et al. 2018).

2.2 Architecture-Based Adversarial Attacks

This section summarizes:

Francesco Croce*, Jonas Rauber*, Matthias Hein (2020). "Scaling up the Randomized Gradient-Free Adversarial Attack Reveals Overestimation of Robustness Using Established Attacks". *International Journal of Computer Vision (IJCV) 128:1028-1046.*

The full publication can be found in the appendix on page 111. * joint first authorship

Motivation

Croce and Hein (2019) proposed a new adversarial attack that exploits the piecewise linearity of standard deep neural network architectures. "The principle of the attack

[is] to solve the minimal adversarial perturbation problem on each linear region as it boils down to a convex [quadratic programming] problem" (Croce, Rauber, et al. 2020). While the attack requires white-box access to the model (including all intermediate layers) to specify the quadratic programming problems, it does not use the model's gradient for gradient descent. Even a flat region without any gradient can be handled just like any other region. Unlike gradient-based attacks, it is thus not susceptible to gradient-masking and suchlike. In this dissertation, we introduce the term *architecture-based attacks* to reflect this difference. This new attack type extends the taxonomy established in the preceding section 2.1.

Croce and Hein (2019) showed that the attack was feasible on small fully-connected neural networks with up to ten layers of 1024 neurons each and that it outperformed the then state-of-the-art Carlini-Wagner attack. Unfortunately, the attack was difficult to scale to larger and more interesting neural networks. It required computing the full Jacobian matrix of all intermediate pre-ReLU activations w.r.t. the network's input for each tiny linear region. This would quickly get out of hand for larger networks, requiring far more memory and computations than realistically available. Repeatedly solving the resulting, potentially gigantic quadratic programming problems increases the computational burden further.

This study aimed to overcome these problems and scale the *Linear Region attack* to larger networks. For this, two great ideas had to come together. First, we replaced the black-box QP solver Gurobi (Gurobi Optimization 2016) used by Croce and Hein (2019) with a custom solver. When repeatedly solving the quadratic programming problems of neighboring linear regions, we are only interested in solutions smaller than our current best adversarial perturbation. Our custom solver for the dual problem can abort as soon as its lower bound on the solution surpasses the best prior solution.

Beyond the immediate advantage of early stopping, controlling the QP solver's inner workings offered another advantage that laid the foundation for our second idea. All instances of the QP problem's constraint matrix within our QP solver's code were either vector-matrix or matrix-vector products. The constraint matrix, in turn, coincides with the stacked Jacobian matrices up to a linear transformation. Explicitly computing the Jacobian matrices thus becomes obsolete. Instead, we can use automatic differentiation algorithms to directly and efficiently compute the vector-Jacobian and Jacobian-vector products using reverse and forward accumulation, respectively.[1]

[1] Reverse-mode automatic differentiation is a generalization of the well-known *backpropagation* algorithm

Thus, these two ideas massively reduced the attack's computational and memory requirements and allowed us to scale to much larger networks, including ResNets (He et al. 2016) with convolutional and pooling layers and residual connections. Finally, with an overhauled region selection strategy, we hoped to improve the attack's effectiveness and efficiency even further.

Results

The quality of an attack cannot easily be reduced to a single number. The optimal attack differs from dataset to dataset, model to model, ϵ-bound to ϵ-bound, and even data point to data point. Robust accuracy, the accuracy remaining despite adversarial perturbations, depends on the size of the allowed perturbations (ϵ-bound) and, of course, on the model. Each attack provides an upper bound on the robust accuracy that we can compare with the best attack in each condition. The difference between each attack's upper bound and the best attack reveals how much the attack overestimates the robust accuracy (at least). We consider the amount of overestimation for 45 different conditions covering three datasets, MNIST, GTS, and CIFAR, with three models (with and without adversarial training) and five ϵ-bounds each. Looking at the average-case and worst-case across conditions helps us understand the quality of an attack.

We found that across all conditions, our attack never overestimates the robust accuracy by more than 5 percentage points whereas all other attacks (including PGD with 10 000 restarts and Carlini-Wagner with 100 000 iterations) overestimate the robust accuracy by more than 50 percentage points in at least one condition. While this striking difference is primarily driven by the differences on MNIST, our attack also shows the best worst-case behavior on all three datasets individually and the best average-case behavior on two of the three datasets.

To demonstrate the scalability of our attack, we tested it on a wide ResNet with 2 883 593 neurons (a 280-fold increase over Croce and Hein (2019)). It included convolutional and pooling layers, residual connections (He et al. 2016), and batch normalization

used to train neural networks. (The gradient is a degenerated vector-Jacobian product.) Forward-mode automatic differentiation is less common in machine learning and, until recently, GPU implementations were relatively scarce. It was available in Theano, but its development ceased in 2017 (Team et al. 2016). Support in TensorFlow was limited to a third-party implementation (Ren 2017), and PyTorch did not support it at all. In 2018, JAX (Bradbury et al. 2018) finally ported autograd's comprehensive automatic differentiation support (Maclaurin et al. 2015) to GPU (which we ended up using). Interestingly, thanks to a "new trick for calculating [Jacobian-vector] products" (Townsend 2017) without specialized code by composing two reverse-mode vector-Jacobian products, TensorFlow and PyTorch have recently retrofitted official support for forward-mode automatic differentiation after all.

(Ioffe et al. 2015), all of which were not previously supported. We found that our attack was competitive with PGD with 1000 restarts and, again, demonstrated the best worst-case behavior, overestimating robust accuracy by at most 1 percentage point (compared to PGD's 3 percentage points).

Discussion

We have already motivated the attack by explaining that it is not susceptible to gradient-masking and suchlike because it does not use the model's gradient for gradient descent. For the same reason, the attack is also "less sensitive to the choice of hyperparameters as no careful selection of the stepsize is required" (Croce, Rauber, et al. 2020). Even though the *maximally black-box* decision-based attacks and the *maximally white-box* architecture-based attacks sit at opposite ends of the spectrum, their advantages over gradient-based attacks thus match in an important aspect. What may seem contradictory, reveals a common misconception about *black-box* and *white-box* attacks that we discuss further in section 3.2.

The risk for severely overestimating the robust accuracy with established gradient-based attacks such as PGD, Carlini-Wagner, and DeepFool demonstrated the practical importance of evaluating adversarial robustness with the Linear Region attack. It is not always an explicit component in the neural network that causes gradient masking. Our results showed that gradient-based attacks may work well on an architecture in one case but can still grossly overestimate the robust accuracy when the same architecture is trained with adversarial training. Such implicit gradient masking limits the interpretability of robustness evaluations with gradient-based attacks. Our attack's reliable worst-case effectiveness demonstrated that it is not subject to such limitations.

Finally, scaling up the original attack by two to three orders of magnitude proved that the attack's alternative approach is feasible on large, interesting models and not limited to a prototype. The new attack's GPU memory requirements are roughly comparable to training the respective neural network. As long as the network's architecture is piecewise linear or can be well approximated with a piecewise linear substitute model, our attack can measure its robustness. Unlike implicit or accidental gradient masking, violations of the piecewise linearity are always explicit. The attack is thus well-suited to reliably estimate the robustness of any piecewise linear model.

2.3 Foolbox Adversarial Attacks Library

This section summarizes:

Jonas Rauber*, Wieland Brendel*, Matthias Bethge (2017). “Foolbox: A Python toolbox to benchmark the robustness of machine learning models”. *Reliable Machine Learning in the Wild Workshop, 34th International Conference on Machine Learning (ICML 2017).*

The full publication can be found in the appendix on page 69. * joint first authorship

Motivation

Adversarial robustness is a young and technologically heterogeneous field. While Python (Van Rossum et al. 1995) has established itself as the field’s de facto standard programming language, new deep learning frameworks emerge constantly, and the dominant framework changes every few years (e.g., Caffe, Theano, TensorFlow, and now PyTorch). Additionally, adversarial attacks and machine learning models offer different interfaces, follow different conventions, and make different assumptions.

These inconsistencies create a huge hurdle for adversarial robustness research. What if an attack assumes pixel values between -1 and 1, while one model requires mean-normalized inputs and the other expects values scaled between 0 and 255? How should one compare one’s own TensorFlow model against someone else’s Theano model using an adversarial attack only implemented for PyTorch? Does the attack expect that the model’s output tensor represents logits or probabilities?

CleverHans (Papernot, Goodfellow, et al. 2016) was the first adversarial attacks library trying to address some of these issues. It was no longer necessary to study each attack’s code individually and to adapt the evaluation to the respective interface. However, one important problem remained: As a TensorFlow library, CleverHans could not solve the problems caused by the different deep learning frameworks. Comparing the robustness of models implemented using different frameworks and ideally adapting to new frameworks remained an open problem. This study aimed to close this gap with a *framework-agnostic* adversarial attacks library (research question 2).

Results

The result of our work was Foolbox, the first framework-agnostic adversarial attacks library. It standardized the interface between attacks and models. This decoupling of framework-agnostic and framework-specific code allowed us to add support for various frameworks without reimplementing the attack algorithms. Foolbox launched with support for five frameworks, PyTorch (Paszke et al. 2019), Keras (Chollet et al. 2015), TensorFlow (Abadi et al. 2016), Theano (Team et al. 2016), and MXNet (T. Chen et al. 2015), and later gained support for Caffe (Jia et al. 2014), JAX (Bradbury et al. 2018), MXNet Gluon (T. Chen et al. 2015), and TensorFlow Eager (Agrawal et al. 2019).

Foolbox also launched with fifteen different adversarial attacks and was constantly updated to include all relevant attacks. Amongst others, Foolbox offered implementations of DeepFool (Moosavi-Dezfooli et al. 2016), the Carlini-Wagner attack (Carlini and Wagner 2017b), the L-BFGS attack (Szegedy et al. 2013), FGSM (Goodfellow et al. 2015), the Local Search Attack (Narodytska et al. 2017), the Basic Iterative Method (Kurakin et al. 2017), PGD (Madry et al. 2018), the Saliency Map Attack (Papernot, McDaniel, Jha, et al. 2016), NewtonFool (Jang et al. 2017), and reference implementations of the Boundary Attack (Brendel, Rauber, and Bethge 2018), the Pointwise Attack (Schott et al. 2019), and the Brendel Bethge attack (Brendel, Rauber, Kümmerer, et al. 2019).

To harmonize the interface of this diverse set of adversarial attacks, Foolbox introduced a declarative API that extended the decoupling to two more factors: the distance measure and the adversarial criterion. The former controls how the size of adversarial perturbations is measured (e.g., using the L_2 norm), while the latter defines the type of adversarial example (e.g., targeted misclassification). A triplet consisting of a model, a distance measure, and an adversarial criterion thus concretely describes an *adversarial problem* that can then be solved by an adversarial attack for different inputs.

The final step to harmonize the adversarial attacks concerned their results. Given the above description of an adversarial problem and a specific input, the obvious robustness measure is the minimal adversarial perturbation size. For those adversarial attacks that instead “maximize the misclassification” given an ϵ-bound on the perturbation size (see section 1.1), Foolbox thus performs an additional line search over that ϵ-bound to find the smallest one for which the attack still succeeds.

Discussion

Foolbox showed that a simple, declarative API for adversarial attacks reduces the entrance barrier for new researchers in the field. Numerous scientists and students profited from easy access to a diverse set of attacks. The extensive support for different deep learning frameworks helped Foolbox's widespread adoption. Most notably, it was the first adversarial attacks library for PyTorch.

Foolbox also laid the foundation for later contributions. The decoupling of attacks and models using a standardized API inspired our new attack taxonomy that distinguishes attacks based on the information or mechanism they depend on (section 2.1). The internal search over ϵ-bounds to transform fixed-ϵ attacks into minimization attacks inspired naive noise attacks. The noise attacks' decision-based approach, in turn, inspired the Boundary Attack (Brendel, Rauber, and Bethge 2018).

2.4 Fast Framework-Agnostic Attacks Using EagerPy

This section summarizes two publications:

Jonas Rauber, Matthias Bethge, Wieland Brendel (2020). "EagerPy: Writing Code That Works Natively with PyTorch, TensorFlow, JAX, and NumPy". *Preprint, arXiv:2008.04175.*

and

Jonas Rauber, Roland Zimmermann, Matthias Bethge, Wieland Brendel (2020). "Foolbox Native: Fast adversarial attacks to benchmark the robustness of machine learning models in PyTorch, TensorFlow, and JAX". *Journal of Open Source Software, 5(53), 2607.*

The full publications can be found in the appendix on page 131 and 143, respectively.

Motivation

New adversarial attacks, new discoveries, and new hardware have increased the need for fast and efficient implementations of adversarial attacks. The Boundary Attack queries thousands or even millions of decisions (Brendel, Rauber, and Bethge 2018). The Linear Region attack executes expensive computations (Croce, Rauber, et al. 2020). And PGD benefits from hundreds of thousands of restarts (Mosbach et al. 2018). At the same time, GPUs have been getting faster and faster, demanding implementations that fully exploit their potential.

This development challenges existing framework-agnostic attack libraries. To communicate with different deep learning frameworks, they require a common interface. To that end, Foolbox 1 (Rauber, Brendel, et al. 2017), ART (Nicolae et al. 2018), and AdvBox (Goodman et al. 2020) have all been built using NumPy (Oliphant 2006). NumPy is the lingua franca for numerical computations in Python and directly supported by all deep learning frameworks. Unfortunately, it runs on the CPU and using it comes at the cost of expensive memory copies between CPU and GPU.

Framework-specific attack libraries do not have this problem. CleverHans (Papernot, Goodfellow, et al. 2016) and AdverTorch (Ding et al. 2019) are two popular adversarial attack libraries implemented specifically for TensorFlow and PyTorch, respectively. The attacks and models use the same framework and thus can communicate maximally efficiently. These libraries, however, forego the benefits of framework-agnostic attacks.

This study aimed to overcome the shortcomings of both types of libraries and unify the framework-agnostic design of Foolbox with the performance of native PyTorch, TensorFlow, or JAX code.

Results

We found that the eager execution APIs in PyTorch, TensorFlow, and JAX have converged sufficiently to build a lightweight abstraction that unifies them. In contrast to PyTorch and JAX, TensorFlow has only recently adopted eager execution, providing this new opportunity (Agrawal et al. 2019). With eager execution, all three frameworks now represent GPU tensors as objects and offer functions or methods to directly (*eagerly*) execute computations on them.

Of course, there remain substantial syntactic and semantic differences between the frameworks. Most notably, the approaches to automatic differentiation range from low-level in-place methods (PyTorch) through mid-level context managers (TensorFlow) to high-level functional transformations (JAX).

Our abstraction has removed all these differences and offers a “unified API that transparently maps to the different underlying frameworks without computational overhead” (Rauber, Bethge, et al. 2020). We published our abstraction as an independent library called *EagerPy* because we expected it to be useful in many other contexts beyond our application to Foolbox.

EagerPy promises native performance in PyTorch, TensorFlow, and JAX, and our reimplementation of Foolbox using EagerPy instead of NumPy confirmed this. Attacks in *Foolbox Native* are as fast as attacks directly implemented in, for example, PyTorch, but support all three frameworks at the same time. In addition, the attacks will automatically inherit support for future frameworks added to EagerPy.

Reimplementing Foolbox with performance in mind implied two other changes. Foolbox Native no longer forces an attack to minimize the perturbation. Instead, it distinguishes *minimization attacks* from *fixed-ϵ* attacks and automatically runs them once or once for each ϵ, respectively, to calculate the robust accuracies. The fixed-ϵ attacks can even be used efficiently for adversarial training. Finally, all attacks in Foolbox Native have been rewritten with real batch support and are now amongst the fastest implementations available.

Discussion

EagerPy unified two goals that previously seemed at odds with each other. Libraries such as Foolbox can now support more than one framework without sacrificing performance and without code duplication. Despite its recent release, other developers have already adopted EagerPy for their libraries (Maria et al. 2014; Whidden 2020). Even deep learning researchers working only with a single framework may benefit from using EagerPy thanks to its other advantages, "such as comprehensive type annotations and consistent support for method chaining" (Rauber, Bethge, et al. 2020).

There is no shortage of adversarial attack libraries. Google's CleverHans (Papernot, Goodfellow, et al. 2016), Borealis AI's AdverTorch (Ding et al. 2019), IBM's ART (Nicolae et al. 2018), Baidu's AdvBox (Goodman et al. 2020), our Foolbox (Rauber, Brendel, et al. 2017), to just name the most important ones. Of course, each new library justifies itself with unique advantages. But the large number of libraries also comes at a cost. Can we expect attack developers to add implementations to all libraries? Certainly not. Will the library maintainers reimplement all attacks? Unlikely. Already today, specific attacks are only available in specific attack libraries. But if one has to use different attack libraries, that puts their whole idea into question. As the field matures, the different attack libraries should consolidate their efforts. Foolbox Native provides a chance to do so because it combines advantages previously split between different attack libraries.

2.5 Robust Analysis by Synthesis

This section summarizes:

Lukas Schott*, Jonas Rauber*, Matthias Bethge, Wieland Brendel (2019). "Towards the first adversarially robust neural network model on MNIST". *Seventh International Conference on Learning Representations (ICLR 2019).*

The full publication can be found in the appendix on page 91. * joint first authorship

Motivation

This study aimed to measure the robustness of a classifier that has learned the distribution of each class rather than just discriminating between the classes. Inspiration for this approach came from the observation that it is possible to create unrecognizable images that DNNs classify as recognizable objects with high confidence (Nguyen et al. 2015). Standard discriminatively trained classifiers do not learn the data distribution and thus cannot easily prevent this behavior. In contrast, generative models can evaluate an input's likelihood and adjust their confidence accordingly.

Using a separate generative model for each class, we can build a classifier. Each class's generative model *analyzes* the input *by synthesizing* its most likely approximation. The input is then classified as the class that best explains the observation. We hypothesized that such an *analysis-by-synthesis* approach should be more robust to adversarial perturbations because an *adversarially perturbed* sample from one class should still be best approximated by that class's generative model.

Results

Our experiments on the MNIST dataset confirmed our hypothesis. For many samples, the minimal adversarial perturbations necessary to mislead our analysis-by-synthesis model were so large that humans could not always unambiguously classify the perturbed samples either. Our model also proved robust against attempts to create unrecognizable images that our model would classify with high confidence. Moreover, the generative analysis-by-synthesis approach allowed us to directly derive instance-specific robustness guarantees that were on par with then state-of-the-art methods to derive guarantees in discriminative classifiers (Hein et al. 2017).

We compared our model with the then state-of-the-art adversarial defense (Madry et al. 2018) and to a standard CNN that binarizes all input pixels to black and white before classifying the image. Repeating our evaluation procedure on Madry et al.'s adversarially trained model confirmed their L_∞ robustness results but revealed that they had substantially overestimated their model's L_2 robustness.[1] In contrast to their results, we showed that their model "is still highly vulnerable to tiny perturbations that are meaningless to humans" (Schott et al. 2019). Using the standard CNN with binarization, we further found that binarization alone is sufficient to achieve substantial L_∞ robustness on the MNIST dataset.

Discussion

Our results indicate that a considerable part of Madry et al.'s increase in L_∞ robustness can be attributed to learning binarization or thresholding.[2] The mostly black and white pixels in the MNIST dataset entail that the L_∞ norm is not suitable to measure or bound adversarial perturbation sizes. From Madry et al.'s L_∞ robustness results alone, it is thus unclear how effective adversarial training with PGD would be in less trivial settings. Overall, our evaluation of Madry et al.'s model demonstrated the limitations of adversarial training in its current form and showed that adversarial robustness on MNIST had not already been achieved (research question 6).

Concerning our own model, our results showed that an analysis-by-synthesis architecture could be robust by design without adversarial training (see research question 7). Minimal adversarial examples that are hard to classify even for humans are an important step forward. Nevertheless, it also became clear that different metrics favor different models and that human-like universal adversarial robustness against perturbations of any kind remains challenging. Besides, the analysis-by-synthesis approach has yet to be scaled up to more complex datasets (see Ju et al. (2020) for a recent step in that direction).

On the meta-level, our study has demonstrated that adversarial robustness can be reliably evaluated if potential pitfalls such as gradient masking and the incentive problem are taken into account, e.g., by adapting and customizing attacks, using decision-based attacks, or splitting development and evaluation between co-authors.

[1] Madry et al. (2018) have since updated their paper's preprint version on arXiv to reflect our findings. In its fourth version (from September 4, 2019), they reversed their interpretation of the results in question and now state that they had overestimated their model's L_2 robustness, most likely due to implicit gradient masking caused by the adversarial training and the learned thresholding.

[2] "the first layer [...] maps the [...] image to three copies thresholded at different values" (Madry et al. 2018)

Chapter 3

Discussion

In the studies summarized in this dissertation, we investigated new methods for evaluating and improving the adversarial robustness of machine learning models.

First, we showed that adversarial attacks could be effective even without access to the attacked model's gradients (section 2.1). This was a crucial finding because all effective attacks to that point relied on gradient descent, either using the true gradient or an approximated or transferred gradient (section 1.1). Our results imply that adversarial attacks are a more realistic threat to machine learning applications than previously assumed (*further discussed in section* 3.1) and that it is—even in white-box scenarios—worthwhile to evaluate a model's robustness with decision-based attacks because they do not suffer from the omnipresent gradient masking problem.

Next, we showed that the latter argument could equally be made for another new type of adversarial attacks that—using the model's gradient to the largest extent possible—sits at the other end of the spectrum (section 2.2). Together, these results imply that the space of adversarial attacks is much broader than previously thought and that the terms *white-box* and *black-box* are inadequate and even misleading when choosing adversarial attacks (*further discussed in section* 3.2).

In the introduction, we identified the incentive problem (research question 4) as well as the practical need for a framework-agnostic adversarial attacks library (section 2.3) with fast attack implementations (section 2.4). However, we have not yet discussed the root cause of both the incentive problem and the importance of good tooling: the empirical evaluation of adversarial robustness cannot be standardized as an explicit, mechanistic process (*further discussed in section 3.3*).

Finally, we designed an alternative neural network architecture and demonstrated its robustness using our new methods for evaluating robustness. This chapter takes a step back from the concrete discussion of this finding (section 2.5) and instead discusses how the underlying goal influences our results' ultimate interpretation (*further discussed in section 3.4*).

3.1 No Robustness Through Obscurity

One shall not count on robustness through obscurity. This is perhaps the most important lesson that machine learning application developers should take from this dissertation. Not all application developers have to worry about attacks. For some applications, it is simply implausible that someone has adversarial interests; for others, even a successful adversarial attack may not pose a problem. However, if adversarial attacks are a *conceptual* threat for the respective application, then they should also be considered a *practical* threat.

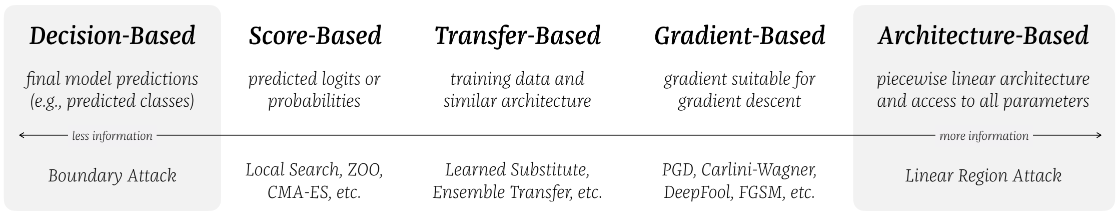
Limitations of the Boundary Attack, like the relatively large number of queries, should not be considered principal limitations that can be exploited to prevent decision-based attacks. Our NeurIPS 2018 Adversarial Vision Challenge (Brendel, Rauber, Kurakin, Papernot, Veliqi, Mohanty, et al. 2020) has demonstrated that the Boundary Attack was just the first of its kind and that decision-based attacks can be made much more query-efficient (e.g., Jianbo Chen et al. 2020; H. Li et al. 2020; Maho et al. 2020).

Moreover, one often overestimates the ability to keep certain information secret. The long history of failed attempts to achieve security through obscurity demonstrates that attackers can often obtain much more information than expected, e.g., through side channels or social engineering. Trying to achieve robustness through obscurity in machine learning applications would be another flawed attempt at security through obscurity.

For the above reasons, a model's worst-case robustness in black-box scenarios may be better captured by its robustness in a white-box setting than by assuming any specific black-box scenario. Even if outside attackers are confined to a black-box scenario, evaluating the robustness in a white-box scenario can thus be advisable.

3.2 The Misconception of White-Box and Black-Box Attacks

Which adversarial attack should I choose? This question gets asked a lot, and it is a legitimate question. The sheer number of adversarial attacks is overwhelming, and the various types of adversarial attacks can be confusing. As if that were not enough, our work introduced two new types of adversarial attacks—decision-based adversarial attacks (section 2.1) and architecture-based adversarial attacks (section 2.2)—that extend the space of adversarial attacks at both ends of the spectrum (figure 2).



Decision-Based
final model predictions (e.g., predicted classes)
Score-Based
predicted logits or probabilities
Transfer-Based
training data and similar architecture
Gradient-Based
gradient suitable for gradient descent
Architecture-Based
piecewise linear architecture and access to all parameters
less information
more information
Boundary Attack
Local Search, ZOO, CMA-ES, etc.
Learned Substitute, Ensemble Transfer, etc.
PGD, Carlini-Wagner, DeepFool, FGSM, etc.
Linear Region Attack

Figure 2: *Extended version of our attack taxonomy (Brendel, Rauber, and Bethge 2018)*

We found that both new types are less susceptible to gradient masking than the common gradient-based attacks. While sections 2.1 and 2.2 explained this observation, they only started to explore its implications, which are therefore discussed in more detail in the following.

Threat scenarios have a clear hierarchy. In a white-box scenario, an attacker has access to all internal information. Gradient- and architecture-based attacks typically require a white-box scenario because gradients are not usually accessible externally. In a black-box scenario, an attacker is limited to the information and access inherently offered by the model. Decision-, score-, and transfer-based attacks are typically considered possible even in a black-box scenario (depending on what is available). This is why the two groups are often called white-box and black-box attacks, respectively.

Unfortunately, this naming convention is misleading because it implies a false hierarchy between attacks. The amount of information an attack requires, uses, and has access to is not indicative of its strength or reliability in either direction. "Black-box" attacks may find tiny perturbations even without gradients (section 2.1). "White-box" attacks may exploit the gradient despite gradient masking and flat regions (section 2.2). An attack's effectiveness depends on its assumptions (coarsely captured by the different types in our taxonomy) and their compatibility with the attacked model and with the threat scenario, but not on whether it is a "white-box" or "black-box" attack.

To avoid the common fallacy that white-box attacks are stronger than black-box attacks, the terms "white-box" and "black-box" should only be used to distinguish threat scenarios—where there is a clear hierarchy—but not to characterize adversarial attacks. A white-box scenario permits all attacks, and all types of attacks should be considered, not just the ones that require a white-box scenario.

3.3 Evaluating Adversarial Robustness Cannot Be Standardized

When a measure becomes a target, it ceases to be a good measure.
— Goodhart's law (Strathern 1997)

Applied to adversarial robustness, Goodhart's law implies that any concrete robustness evaluation procedure with select attacks ceases to be a good measure of adversarial robustness when it becomes the target of defenses.

Unfortunately, targeting the measure is almost an axiom of human behavior. From the scientist who plays the citation game (Biagioli 2016) to the automobile manufacturer that activates emission controls only during testing (Reynaert et al. 2016), humans target the chosen measure, and the once informative proxy loses its value. In the case of adversarial robustness, it is even more absurd because the defense developers—apart from occasional input from reviewers—choose the evaluation procedure themselves. Either way, it follows from Goodhart's law that a standardized evaluation procedure known to the defense will not be a good measure of adversarial robustness.

For this reason, judging a defense's empirical adversarial robustness is ultimately always at the discretion of humans and cannot be fully automized with adversarial attacks (by contrast, robustness guarantees can be objectively verified but are, as of yet, often uninformative). This view is not only supported by our guidelines on evaluating adversarial robustness (Carlini, Athalye, et al. 2019) but also by a recent paper emphasizing the importance of carefully adapting the attacks to the respective defenses and explaining how to do so (Tramèr, Carlini, et al. 2020).

Having said that, there are also recent attempts to standardize the benchmarking of adversarial defenses by imposing restrictions on the models (Croce, Andriushchenko, Sehwag, et al. 2020). It remains to be seen whether those restrictions are sufficient to ensure that their standardized evaluation remains informative in the long run, and the authors themselves already acknowledge that adaptive attacks may improve over their standardized evaluation.

Of course, a non-standardized adaptive evaluation process that is at the discretion of the respective researchers bears a risk that we identified earlier as the *incentive problem* (section 1.2). In the fourth research question, we asked how to overcome this problem and promised to discuss three ideas. The first idea was establishing the guidelines mentioned above (Carlini, Athalye, et al. 2019). It is difficult to assess the extent to which they increased the awareness of reviewers and defense researchers for common pitfalls and best practices. On the one hand, many publications refer to these guidelines to justify their evaluation procedure. On the other hand, insufficiently evaluated defenses still pass the peer-review processes of top-tier conferences to this day (Tramèr, Carlini, et al. 2020). Time will tell whether Tramèr et al.'s more explicit demonstration of the methodology behind adaptive attacks can further mitigate this problem.

Second, we organized the Adversarial Vision Challenge, a competition that pitched proposed defenses against active, opposing evaluators (Brendel, Rauber, Kurakin, Papernot, Veliqi, Salathé, et al. 2018). The goal of this challenge was twofold. On the concrete level, the challenge aimed, among other things, to stimulate new decision-based adversarial attacks. On the meta-level, organizing this two-sided competition aimed at spreading the idea that the development of a defense mechanism and its robust evaluation should be separated and executed by independent parties. To that end, the restriction to decision-based attacks was a severe limitation. Nevertheless, the challenge demonstrated the principles of an active review process.

In the future, machine learning conferences and journals should consider making their own standard peer-review process more adversarial by requiring all proclaimed defenses to release their code publicly at submission time and by asking reviewers and third-parties to perform adaptive attacks against the submissions prior to acceptance.

Until then, we can simulate this separation of concerns by assigning different roles to different co-authors. We followed this third idea when publishing our own defense (Schott et al. 2019), and the longevity of the results suggests that this may have been beneficial.

Finally, in the context of a non-standardized evaluation that leans on individual researchers, having access to the right tools and fast adversarial attack implementations (see section 2.4) is not a mere convenience but can directly impact the researchers' choices and, ultimately, the results. Further elaboration of this point can be found in the fifth research question's derivation on page 25.

3.4 Has the Adversarial Robustness Problem Now Been Solved?

Has the adversarial robustness problem, at least on MNIST, now been solved? This question has been asked when Madry et al. (2018) demonstrated effective adversarial training on MNIST. It has been brought up again when we introduced our analysis-by-synthesis model on MNIST. Rarely, someone answers with an unconditional, satisfying *yes* or *no*. Instead, it prompts a counterquestion: What does it mean to solve adversarial robustness?

One option is to define the adversarial robustness problem as solved if the model achieves high accuracy despite adversarial perturbations up to a certain size. This is what we practically capture through the definitions introduced in section 1.1. Once a metric and an allowed perturbation size have been specified, the objective is mathematically well-defined. Adversarial attacks and robustness guarantees can approximate it from both sides, independent of human perception. We obtain a concrete quantitive measure of a model's adversarial robustness, e.g., 88 % robust accuracy on MNIST for $\epsilon_{L_\infty} = 0.3$ (Madry et al.'s model) or 80 % robust accuracy for $\epsilon_{L_2} = 1.5$ (our analysis-by-synthesis model). Finally, we can interpret the quantitative result to claim that the problem has been solved or not. Given that even human accuracy starts to deteriorate for perturbations with an L_2 norm of 1.5, it seems reasonable to interpret the glass as half full rather than half empty, but the ultimate interpretation is left to the reader.

Another option is to return to our actual motivation for improving a specific model's adversarial robustness. If an application's security hinges on a machine learning model's adversarial robustness, then eliminating that threat is what defines whether the respective robustness problem has been solved. In some cases, it may be enough to prevent imperceptible adversarial perturbations. In others, it can be important to be robust to perturbations up to a certain physical size before it becomes acceptable to make mistakes. Ultimately, the criterion depends on the model's concrete application and not on the machine learning task itself. Whether adversarial robustness on MNIST is solved or not is—from this perspective—inherently *undefined*. It depends on whether we, for example, aim to recognize the postal code on a letter or the digits on a street sign.

If our motivation for adversarial robustness was instead to create a model that mimics human perception to the largest extent possible, it implies yet another definition of *solved*. In this case, it is inevitable to compare our model's perception with our own perception. This is typically done by qualitatively assessing whether minimal adversarial perturbations that mislead the machine learning model are also ambiguous to humans. In such an analysis, our analysis-by-synthesis model showed a substantial improvement over all other models (our results refuted Madry et al.'s original claim in this regard) and so got much closer to resembling human perception, albeit a gap remains. While the analysis-by-synthesis model may not have fully solved adversarial robustness on MNIST, it is currently the machine learning model that is most similar to human digit perception.

This view is not only supported by our experiments but also by results obtained in other labs. Golan et al. (2020) tested different machine learning models on controversial stimuli that generalize the concept of adversarial examples and reveal the models' inductive biases. They found that our analysis-by-synthesis model was the best performing model on controversial stimuli. Additionally, a very recent study (Ju et al. 2020) extended the analysis-by-synthesis approach to more complex image classification tasks, including street sign recognition and SVHN. Independent of whether the adversarial robustness problem has now been solved or not, we can thus conclude that important progress has been made in recent years.

3.5 Conclusions & Outlook

Four years ago, adversarial examples were easily dismissed as practically irrelevant because they could only be created with access to the models' gradients. Any hypothesis for an effective defense mechanism was impossible to confirm because an improvement in robustness could not be distinguished from an impairment of the evaluation methods. Testing a model's robustness with a specific adversarial attack was cumbersome and time-consuming.

Today, we are aware of the threats posed by adversarial attacks. We now have the tools to run adversarial attacks effortlessly. Gradient masking can be identified and circumvented, and reviewers start taking a closer look before accepting dubious defenses. We even see the first genuine improvements in adversarial robustness thanks to adversarial training and the analysis-by-synthesis architecture. The contributions presented in this dissertation are an essential part of this success.

Notwithstanding the progress that has been made, there remains work to be done. By advancing adversarial training and the analysis-by-synthesis architecture, it may be possible to scale these ideas to more complex datasets. The recent extension of the analysis-by-synthesis model is a promising example (Ju et al. 2020). Equipped with the techniques and attack methods to reliably estimate adversarial robustness, also completely new hypotheses for defense mechanisms can now be tested. In addition to defense mechanisms, it seems worthwhile to investigate new architectures and constraints that facilitate better robustness guarantees. There is still a considerable gap between robustness guarantees and adversarial attacks. At the same time, it is important not to lose sight of the underlying goals. Those—and not robust accuracies—ultimately determine where we stand and where we want to go.

Bibliography

Abadi, Martín, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. (2016). "TensorFlow: A System for Large-Scale Machine Learning." In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283. ISBN: 978-1-931971-33-1. arXiv: 1605.08695 (cit. on pp. 20, 39).

Agrawal, Akshay, Akshay Naresh Modi, Alexandre Passos, Allen Lavoie, Ashish Agarwal, Asim Shankar, Igor Ganichev, Josh Levenberg, Mingsheng Hong, Rajat Monga, et al. (2019). "TensorFlow Eager: A multi-stage, Python-embedded DSL for machine learning." In: *Systems for Machine Learning (SysML) 2019* (cit. on pp. 39, 41).

Anil, Cem, James Lucas, and Roger Grosse (2019). "Sorting out Lipschitz function approximation." In: *International Conference on Machine Learning*. PMLR, pp. 291–301. arXiv: 1811.05381 (cit. on p. 22).

Athalye, Anish and Nicholas Carlini (2018). "On the Robustness of the CVPR 2018 White-Box Adversarial Example Defenses." In: arXiv: 1804.03286 (cit. on pp. 21, 25).

Athalye, Anish, Nicholas Carlini, and David A Wagner (2018). "Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples." In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 274–283. arXiv: 1802.00420. URL: http://proceedings.mlr.press/v80/athalye18a.html (cit. on pp. 21, 24, 25, 31).

Bafna, Mitali, Jack Murtagh, and Nikhil Vyas (2018). "Thwarting Adversarial Examples: An L_0-Robust Sparse Fourier Transform." In: *Advances in Neural Information Processing Systems* 31, pp. 10075–10085. arXiv: 1812.05013 (cit. on p. 20).

Barreno, Marco, Blaine Nelson, Russell Sears, Anthony D Joseph, and J D Tygar (2006). "Can Machine Learning Be Secure?" In: *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*. ASIACCS '06. Association for

Computing Machinery, pp. 16–25. ISBN: 1-59593-272-0. DOI: 10.1145/1128817.1128824 (cit. on p. 16).

Biagioli, Mario (2016). “Watch out for cheats in citation game.” In: *Nature* 535.7611, pp. 201–201. DOI: 10.1038/535201a (cit. on p. 48).

Biggio, Battista, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli (2013). “Evasion attacks against machine learning at test time.” In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, pp. 387–402. DOI: 10.1007/978-3-642-40994-3_25 (cit. on p. 16).

Biggio, Battista, Blaine Nelson, and Pavel Laskov (2012). “Poisoning Attacks against Support Vector Machines.” In: *Proceedings of the 29th International Coference on International Conference on Machine Learning*. ICML’12. Omnipress, pp. 1467–1474. ISBN: 978-1-4503-1285-1. arXiv: 1206.6389 (cit. on p. 16).

Bradbury, James, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne (2018). *JAX: composable transformations of Python+NumPy programs*. URL: http://github.com/google/jax (cit. on pp. 36, 39).

Brendel, Wieland and Matthias Bethge (2017). “Comment on ‘Biologically inspired protection of deep networks from adversarial attacks’.” In: arXiv: 1704.01547 (cit. on p. 21).

Brendel, Wieland, Jonas Rauber, and Matthias Bethge (2018). “Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models.” In: *International Conference on Learning Representations*. arXiv: 1712.04248. URL: https://openreview.net/forum?id=SyZI0GWCZ (cit. on pp. 39, 40, 47).

Brendel, Wieland, Jonas Rauber, Matthias Kümmerer, Ivan Ustyuzhaninov, and Matthias Bethge (2019). “Accurate, reliable and fast robustness evaluation.” In: *Advances in Neural Information Processing Systems* 32. arXiv: 1907.01003 (cit. on pp. 19, 39).

Brendel, Wieland, Jonas Rauber, Alexey Kurakin, Nicolas Papernot, Behar Veliqi, Sharada P Mohanty, Florian Laurent, Marcel Salathé, Matthias Bethge, Yaodong Yu, Hongyang Zhang, Susu Xu, Hongbao Zhang, Pengtao Xie, Eric P Xing, Thomas Brunner, Frederik Diehl, Jérôme Rony, Luiz Gustavo Hafemann, Shuyu Cheng, Yinpeng Dong, Xuefei Ning, Wenshuo Li, and Yu Wang (2020). “Adversarial Vision Challenge.” In: *The NeurIPS ’18 Competition*. Springer International Publishing, pp. 129–153. ISBN: 978-3-030-29135-8. DOI: 10.1007/978-3-030-29135-8_5 (cit. on p. 46).

Brendel, Wieland, Jonas Rauber, Alexey Kurakin, Nicolas Papernot, Behar Veliqi, Marcel Salathé, Sharada P Mohanty, and Matthias Bethge (2018). *Adversarial Vision Challenge*.

Competition Proposal. NeurIPS 2018 Competition Track. arXiv: 1808.01976 (cit. on pp. 25, 49).

Buckman, Jacob, Aurko Roy, Colin Raffel, and Ian Goodfellow (2018). "Thermometer Encoding: One Hot Way To Resist Adversarial Examples." In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=S18Su--CW (cit. on p. 20).

Byrd, Richard H, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu (1995). "A Limited Memory Algorithm for Bound Constrained Optimization." In: *SIAM Journal on scientific computing* 16.5, pp. 1190–1208. DOI: 10.1137/0916069 (cit. on p. 16).

Carlini, Nicholas (2020). *A Complete List of All (arXiv) Adversarial Example Papers*. URL: https://nicholas.carlini.com/writing/2019/all-adversarial-example-papers.html (cit. on p. 16).

Carlini, Nicholas, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin (2019). "On Evaluating Adversarial Robustness." In: arXiv: 1902.06705. URL: https://github.com/evaluating-adversarial-robustness/adv-eval-paper (cit. on pp. 25, 48, 49).

Carlini, Nicholas and David A Wagner (2016). "Defensive Distillation is Not Robust to Adversarial Examples." In: arXiv: 1607.04311 (cit. on pp. 21, 33).

Carlini, Nicholas and David A Wagner (2017a). "Magnet and 'efficient defenses against adversarial attacks' are not robust to adversarial examples." In: arXiv: 1711.08478 (cit. on p. 21).

Carlini, Nicholas and David A Wagner (2017b). "Towards Evaluating the Robustness of Neural Networks." In: *38th IEEE Symposium on Security and Privacy*. IEEE, pp. 39–57. DOI: 10.1109/SP.2017.49 (cit. on pp. 17, 33, 39).

Chen, Jianbo, Michael I Jordan, and Martin J Wainwright (2020). "HopSkipJumpAttack: A query-efficient decision-based attack." In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, pp. 1277–1294. DOI: 10.1109/SP40000.2020.00045 (cit. on pp. 34, 46).

Chen, Pin-Yu, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh (2018). "EAD: Elastic-Net Attacks to Deep Neural Networks via Adversarial Examples." In: *AAAI*, pp. 10–17. arXiv: 1709.04114. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16893 (cit. on p. 19).

Chen, Pin-Yu, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh (2017). "ZOO: Zeroth Order Optimization Based Black-Box Attacks to Deep Neural Networks without Training Substitute Models." In: *Proceedings of the 10th ACM Workshop on*

Artificial Intelligence and Security. AISec '17. Association for Computing Machinery, pp. 15–26. ISBN: 978-1-4503-5202-4. DOI: 10.1145/3128572.3140448 (cit. on pp. 31, 32).

Chen, Tianqi, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang (2015). "MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems." In: *LearningSys Workshop at Neural Information Processing Systems 2015*. arXiv: 1512.01274 (cit. on p. 39).

Chen, Xinyun, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song (2017). "Targeted backdoor attacks on deep learning systems using data poisoning." In: arXiv: 1712.05526 (cit. on p. 16).

Chollet, François et al. (2015). *Keras*. URL: https://keras.io (cit. on p. 39).

Croce, Francesco, Maksym Andriushchenko, and Matthias Hein (2019). "Provable Robustness of ReLU networks via Maximization of Linear Regions." In: *Proceedings of Machine Learning Research*. Vol. 89. Proceedings of Machine Learning Research. PMLR, pp. 2057–2066. arXiv: 1810.07481. URL: http://proceedings.mlr.press/v89/croce19a.html (cit. on p. 22).

Croce, Francesco, Maksym Andriushchenko, Vikash Sehwag, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein (2020). "RobustBench: a standardized adversarial robustness benchmark." In: arXiv: 2010.09670 (cit. on p. 48).

Croce, Francesco and Matthias Hein (2019). "A Randomized Gradient-Free Attack on ReLU Networks." In: *Pattern Recognition (GCPR 2018)*. Springer International Publishing, pp. 215–227. ISBN: 978-3-030-12939-2. DOI: 10.1007/978-3-030-12939-2_16 (cit. on pp. 32, 34–36).

Croce, Francesco and Matthias Hein (2020). "Provable robustness against all adversarial l_p-perturbations for $p \geq 1$." In: *International Conference on Learning Representations*. arXiv: 1905.11213. URL: https://openreview.net/forum?id=rklk_ySYPB (cit. on p. 22).

Croce, Francesco, Jonas Rauber, and Matthias Hein (2020). "Scaling up the Randomized Gradient-Free Adversarial Attack Reveals Overestimation of Robustness Using Established Attacks." In: *International Journal of Computer Vision*. ISSN: 1573-1405. DOI: 10.1007/s11263-019-01213-0 (cit. on pp. 32, 35, 37, 40).

Dalvi, Nilesh, Pedro Domingos, Mausam, Sumit Sanghai, and Deepak Verma (2004). "Adversarial Classification." In: *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '04. ACM, pp. 99–108. ISBN: 1-58113-888-1. DOI: 10.1145/1014052.1014066 (cit. on p. 11).

Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei (2009). "ImageNet: A large-scale hierarchical image database." In: *IEEE Conference on Computer Vision* *and Pattern Recognition (CVPR)*. IEEE, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848 (cit. on p. 17).

Dhillon, Guneet S., Kamyar Azizzadenesheli, Jeremy D. Bernstein, Jean Kossaifi, Aran Khanna, Zachary C. Lipton, and Animashree Anandkumar (2018). "Stochastic activation pruning for robust adversarial defense." In: *International Conference on Learning Representations*. arXiv: 1803.01442. URL: https://openreview.net/forum?id=H1uR4GZRZ (cit. on p. 20).

Dieleman, Sander, Jan Schlüter, Colin Raffel, Eben Olson, Søren Kaae Sonderby, et al. (2015). *Lasagne: First release.* DOI: 10.5281/zenodo.27878 (cit. on p. 23).

Ding, Gavin Weiguang, Luyu Wang, and Xiaomeng Jin (2019). "AdverTorch v0.1: An Adversarial Robustness Toolbox based on PyTorch." In: arXiv: 1902.07623 (cit. on pp. 20, 26, 41, 42).

Dong, Yinpeng, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li (2018). "Boosting Adversarial Attacks With Momentum." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. DOI: 10.1109/CVPR.2018.00957 (cit. on p. 19).

Elsayed, Gamaleldin F, Ian Goodfellow, and Jascha Sohl-Dickstein (2019). "Adversarial Reprogramming of Neural Networks." In: *International Conference on Learning Representations*. arXiv: 1806.11146. URL: https://openreview.net/forum?id=Syx_Ss05tm (cit. on p. 16).

Engstrom, Logan, Andrew Ilyas, and Anish Athalye (2018). "Evaluating and understanding the robustness of adversarial logit pairing." In: arXiv: 1807.10272 (cit. on p. 21).

Geirhos, Robert, Carlos R Medina Temme, Jonas Rauber, Heiko H Schütt, Matthias Bethge, and Felix A Wichmann (2018). "Generalisation in humans and deep neural networks." In: *Advances in Neural Information Processing Systems 31*. arXiv: 1808.08750.

Golan, Tal, Prashant C Raju, and Nikolaus Kriegeskorte (2020). "Controversial stimuli: Pitting neural networks against each other as models of human cognition." In: *Proceedings of the National Academy of Sciences* 117.47, pp. 29330–29337. ISSN: 0027-8424. DOI: 10.1073/pnas.1912334117 (cit. on p. 51).

Goodfellow, Ian, Jonathon Shlens, and Christian Szegedy (2015). "Explaining and Harnessing Adversarial Examples." In: *International Conference on Learning Representations*. arXiv: 1412.6572 (cit. on pp. 14, 18, 21, 39).

Goodman, Dou, Hao Xin, Wang Yang, Wu Yuesheng, Xiong Junfeng, and Zhang Huan (2020). *Advbox: a toolbox to generate adversarial examples that fool neural networks*. arXiv: 2001.05574 (cit. on pp. 41, 42).

Gu, Tianyu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg (2019). "BadNets: Evaluating Backdooring Attacks on Deep Neural Networks." In: *IEEE Access* 7, pp. 47230–47244. DOI: 10.1109/ACCESS.2019.2909068 (cit. on p. 16).

Guo, Chuan, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten (2018). "Countering Adversarial Images using Input Transformations." In: *International Conference on Learning Representations*. arXiv: 1711.00117. URL: https://openreview.net/forum?id=SyJ7ClWCb (cit. on p. 20).

Gurobi Optimization, Inc. (2016). *Gurobi Optimizer Reference Manual*. URL: http://www.gurobi.com (cit. on p. 35).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). "Deep Residual Learning for Image Recognition." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778. DOI: 10.1109/CVPR.2016.90 (cit. on p. 36).

Hein, Matthias and Maksym Andriushchenko (2017). "Formal Guarantees on the Robustness of a Classifier against Adversarial Manipulation." In: *Advances in Neural Information Processing Systems 30*, pp. 2266–2276. arXiv: 1705.08475 (cit. on pp. 22, 43).

Hu, Shengyuan, Tao Yu, Chuan Guo, Wei-Lun Chao, and Kilian Q Weinberger (2019). "A New Defense Against Adversarial Images: Turning a Weakness into a Strength." In: *Advances in Neural Information Processing Systems*. Vol. 32, pp. 1635–1646. arXiv: 1910.07629 (cit. on p. 20).

Ilyas, Andrew, Logan Engstrom, Anish Athalye, and Jessy Lin (2018). "Black-box Adversarial Attacks with Limited Queries and Information." In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 2137–2146. arXiv: 1804.08598. URL: http://proceedings.mlr.press/v80/ilyas18a.html (cit. on p. 34).

Ioffe, Sergey and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15, pp. 448–456. arXiv: 1502.03167 (cit. on p. 37).

Jang, Uyeong, Xi Wu, and Somesh Jha (2017). "Objective Metrics and Gradient Descent Algorithms for Adversarial Examples in Machine Learning." In: *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACSAC 2017. Association for

Computing Machinery, pp. 262–277. ISBN: 978-1-4503-5345-8. DOI: 10.1145/3134600.3134635 (cit. on p. 39).

Jia, Yangqing, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell (2014). “Caffe: Convolutional architecture for fast feature embedding.” In: *Proceedings of the 22nd ACM international conference on Multimedia*, pp. 675–678. DOI: 10.1145/2647868.2654889 (cit. on pp. 23, 39).

Ju, An and David A Wagner (2020). “E-ABS: Extending the Analysis-By-Synthesis Robust Classification Model to More Complex Image Domains.” In: *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security*. AISec’20. Association for Computing Machinery, pp. 25–36. ISBN: 978-1-4503-8094-2. DOI: 10.1145/3411508.3421382 (cit. on pp. 44, 51, 52).

Kannan, Harini, Alexey Kurakin, and Ian Goodfellow (2018). “Adversarial logit pairing.” In: arXiv: 1803.06373 (cit. on p. 20).

Kingma, Diederik P and Jimmy Ba (2015). “Adam: A method for stochastic optimization.” In: *International Conference on Learning Representations*. arXiv: 1412.6980 (cit. on p. 17).

Krizhevsky, Alex and Geoffrey Hinton (2009). *Learning multiple layers of features from tiny images*. Technical Report. University of Toronto. URL: http://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf (cit. on p. 21).

Kümmerer, Matthias, Lucas Theis, and Matthias Bethge (2015). “Deep Gaze I: Boosting Saliency Prediction with Feature Maps Trained on ImageNet.” In: *ICLR Workshop*. arXiv: 1411.1045 (cit. on p. 13).

Kurakin, Alexey, Ian Goodfellow, and Samy Bengio (2017). “Adversarial examples in the physical world.” In: *ICLR Workshop*. arXiv: 1607.02533 (cit. on pp. 18, 20, 23, 39).

LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner (1998). “Gradient-based learning applied to document recognition.” In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. DOI: 10.1109/5.726791 (cit. on p. 21).

Li, Huichen, Xiaojun Xu, Xiaolu Zhang, Shuang Yang, and Bo Li (2020). “QEBA: Query-Efficient Boundary-Based Blackbox Attack.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. DOI: 10.1109/CVPR42600.2020.00130 (cit. on pp. 34, 46).

Li, Qiyang, Saminul Haque, Cem Anil, James Lucas, Roger B Grosse, and Jörn-Henrik Jacobsen (2019). “Preventing Gradient Attenuation in Lipschitz Constrained Convolutional Networks.” In: *Advances in neural information processing systems*, pp. 15390–15402. arXiv: 1911.00937 (cit. on p. 22).

Li, Yingzhen, John Bradshaw, and Yash Sharma (2019). “Are Generative Classifiers More Robust to Adversarial Attacks?” In: *Proceedings of the 36th International Conference on Machine Learning*. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 3804–3814. arXiv: 1802.06552. URL: http://proceedings.mlr.press/v97/li19a.html (cit. on p. 20).

Liao, Fangzhou, Ming Liang, Yinpeng Dong, Tianyu Pang, Jun Zhu, and Xiaolin Hu (2018). “Defense Against Adversarial Attacks Using High-Level Representation Guided Denoiser.” In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1778–1787. DOI: 10.1109/CVPR.2018.00191 (cit. on p. 20).

Liu, Yanpei, Xinyun Chen, Chang Liu, and Dawn Song (2017). “Delving into Transferable Adversarial Examples and Black-box Attacks.” In: *International Conference on Learning Representations*. arXiv: 1611.02770. URL: https://openreview.net/forum?id=Sys6GJqxl (cit. on p. 19).

Liu, Yingqi, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang (2018). “Trojaning Attack on Neural Networks.” In: *25nd Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society. DOI: 10.14722/ndss.2018.23291 (cit. on p. 16).

Lowd, Daniel and Christopher Meek (2005). “Adversarial Learning.” In: *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*. KDD ’05. ACM, pp. 641–647. ISBN: 1-59593-135-X. DOI: 10.1145/1081870.1081950 (cit. on p. 14).

Ma, Xingjun, Bo Li, Yisen Wang, Sarah M Erfani, Sudanthi Wijewickrema, Grant Schoenebeck, Michael E Houle, Dawn Song, and James Bailey (2018). “Characterizing Adversarial Subspaces Using Local Intrinsic Dimensionality.” In: *International Conference on Learning Representations*. arXiv: 1801.02613. URL: https://openreview.net/forum?id=B1gJ1L2aW (cit. on p. 20).

Maclaurin, Dougal, David Duvenaud, and Ryan P Adams (2015). “Autograd: Effortless Gradients in NumPy.” In: *ICML 2015 AutoML Workshop*. URL: https://github.com/HIPS/autograd (cit. on p. 36).

Madry, Aleksander, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu (2018). “Towards Deep Learning Models Resistant to Adversarial Attacks.” In: *International Conference on Learning Representations*. eprint: 1706.06083. URL: https://openreview.net/forum?id=rJzIBfZAb (cit. on pp. 18, 21, 22, 24, 26, 39, 43, 44, 50, 51).

Maho, Thibault, Teddy Furon, and Erwan Le Merrer (2020). “SurFree: a fast surrogate-free black-box attack.” In: arXiv: 2011.12807 (cit. on pp. 34, 46).

Maria, Clément, Jean-Daniel Boissonnat, Marc Glisse, and Mariette Yvinec (2014). “The Gudhi Library: Simplicial Complexes and Persistent Homology.” In: *Mathematical Software – ICMS 2014*. Springer Berlin Heidelberg, pp. 167–174. ISBN: 978-3-662-44199-2. DOI: 10.1007/978-3-662-44199-2_28 (cit. on p. 42).

Meng, Dongyu and Hao Chen (2017). “Magnet: a two-pronged defense against adversarial examples.” In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, pp. 135–147. DOI: 10.1145/3133956.3134057 (cit. on p. 20).

Moosavi-Dezfooli, Seyed-Mohsen, Alhussein Fawzi, and Pascal Frossard (2016). “DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2574–2582. DOI: 10.1109/CVPR.2016.282 (cit. on pp. 17, 33, 39).

Mosbach, Marius, Maksym Andriushchenko, Thomas Trost, Matthias Hein, and Dietrich Klakow (2018). “Logit pairing methods can fool gradient-based attacks.” In: *NeurIPS 2018 Workshop on Security in Machine Learning*. arXiv: 1810.12042 (cit. on pp. 18, 40).

Narodytska, Nina and Shiva Prasad Kasiviswanathan (2017). “Simple Black-Box Adversarial Perturbations for Deep Networks.” In: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1310–1318. DOI: 10.1109/CVPRW.2017.172 (cit. on pp. 31, 32, 39).

Nayebi, Aran and Surya Ganguli (2017). “Biologically inspired protection of deep networks from adversarial attacks.” In: arXiv: 1703.09202 (cit. on p. 20).

Nguyen, Anh Mai, Jason Yosinski, and Jeff Clune (2015). “Deep neural networks are easily fooled: High confidence predictions for unrecognizable images.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 427–436. ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7298640 (cit. on pp. 15, 43).

Nicolae, Maria-Irina, Mathieu Sinn, Minh Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, et al. (2018). “Adversarial Robustness Toolbox v1.0.0.” In: arXiv: 1807.01069 (cit. on pp. 20, 26, 41, 42).

Oliphant, Travis (2006). *NumPy: A guide to NumPy*. URL: https://numpy.org (cit. on p. 41).

Pang, Tianyu, Kun Xu, Yinpeng Dong, Chao Du, Ning Chen, and Jun Zhu (2020). “Rethinking Softmax Cross-Entropy Loss for Adversarial Robustness.” In: *International Conference on Learning Representations*. arXiv: 1905.10626. URL: https://openreview.net/forum?id=Byg9A24tvB (cit. on p. 20).

Pang, Tianyu, Kun Xu, Chao Du, Ning Chen, and Jun Zhu (2019). “Improving Adversarial Robustness via Promoting Ensemble Diversity.” In: *Proceedings of the 36th International Conference on Machine Learning*. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 4970–4979. arXiv: 1901.08846. URL: http://proceedings.mlr.press/v97/pang19a.html (cit. on p. 20).

Pang, Tianyu, Kun Xu, and Jun Zhu (2020). “Mixup Inference: Better Exploiting Mixup to Defend Adversarial Attacks.” In: *International Conference on Learning Representations*. arXiv: 1909.11515. URL: https://openreview.net/forum?id=ByxtC2VtPB (cit. on p. 20).

Papernot, Nicolas, Ian Goodfellow, and Patrick McDaniel (2016). “cleverhans v0.1: an adversarial machine learning library.” In: arXiv: 1610.00768 (cit. on pp. 20, 26, 38, 41, 42).

Papernot, Nicolas, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami (2017). “Practical Black-Box Attacks Against Machine Learning.” In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ASIA CCS ’17. ACM, pp. 506–519. ISBN: 978-1-4503-4944-4. DOI: 10.1145/3052973.3053009 (cit. on pp. 19, 23, 24, 31, 32).

Papernot, Nicolas, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami (2016). “The Limitations of Deep Learning in Adversarial Settings.” In: *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, pp. 372–387. DOI: 10.1109/EuroSP.2016.36 (cit. on pp. 19, 39).

Papernot, Nicolas, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami (2016). “Distillation as a defense to adversarial perturbations against deep neural networks.” In: *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, pp. 582–597. DOI: 10.1109/SP.2016.41 (cit. on pp. 20, 33).

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. (2019). “PyTorch: An imperative style, high-performance deep learning library.” In: *Advances in neural information processing systems*, pp. 8026–8037. arXiv: 1912.01703 (cit. on p. 39).

Prakash, Aaditya, Nick Moran, Solomon Garber, Antonella DiLillo, and James Storer (2018). “Deflecting Adversarial Attacks With Pixel Deflection.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. DOI: 10.1109/CVPR.2018.00894 (cit. on p. 20).

Raghunathan, Aditi, Jacob Steinhardt, and Percy Liang (2018). “Certified Defenses against Adversarial Examples.” In: *International Conference on Learning Representa-*

tions. arXiv: 1801.09344. URL: https://openreview.net/forum?id=Bys4ob-Rb (cit. on p. 22).

Rauber, Jonas, Matthias Bethge, and Wieland Brendel (2020). "EagerPy: Writing Code That Works Natively with PyTorch, TensorFlow, JAX, and NumPy." In: arXiv: 2008.04175 (cit. on pp. 41, 42).

Rauber, Jonas, Wieland Brendel, and Matthias Bethge (2017). "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*. arXiv: 1707.04131 (cit. on pp. 20, 32, 41, 42).

Rauber, Jonas, Roland Zimmermann, Matthias Bethge, and Wieland Brendel (2020). "Foolbox Native: Fast adversarial attacks to benchmark the robustness of machine learning models in PyTorch, TensorFlow, and JAX." In: *Journal of Open Source Software* 5.53, p. 2607. DOI: 10.21105/joss.02607 (cit. on p. 20).

Ren, Mengye (2017). *Forward-mode Automatic Differentiation for TensorFlow*. URL: https://github.com/renmengye/tensorflow-forward-ad (cit. on p. 36).

Reynaert, Mathias and James M Sallee (2016). *Corrective Policy and Goodhart's Law: The Case of Carbon Emissions from Automobiles*. Working Paper 22911. National Bureau of Economic Research. DOI: 10.3386/w22911 (cit. on p. 48).

Rony, Jérôme, Luiz G Hafemann, Luiz S Oliveira, Ismail Ben Ayed, Robert Sabourin, and Eric Granger (2019). "Decoupling direction and norm for efficient gradient-based l2 adversarial attacks and defenses." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4322–4330. DOI: 10.1109/CVPR.2019.00445 (cit. on p. 19).

Roth, Kevin, Yannic Kilcher, and Thomas Hofmann (2019). "The Odds are Odd: A Statistical Test for Detecting Adversarial Examples." In: *Proceedings of the 36th International Conference on Machine Learning*. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 5498–5507. arXiv: 1902.04818. URL: http://proceedings.mlr.press/v97/roth19a.html (cit. on p. 20).

Rubinstein, Benjamin I P, Blaine Nelson, Ling Huang, Anthony D Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and J D Tygar (2009). "ANTIDOTE: Understanding and Defending against Poisoning of Anomaly Detectors." In: *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*. IMC '09. Association for Computing Machinery, pp. 1–14. ISBN: 978-1-60558-771-4. DOI: 10.1145/1644893.1644895 (cit. on p. 16).

Samangouei, Pouya, Maya Kabkab, and Rama Chellappa (2018). "Defense-GAN: Protecting Classifiers Against Adversarial Attacks Using Generative Models." In: *In-*

ternational Conference on Learning Representations. arXiv: 1805.06605. URL: https://openreview.net/forum?id=BkJ3ibb0- (cit. on p. 20).

Schott, Lukas, Jonas Rauber, Matthias Bethge, and Wieland Brendel (2019). “Towards the first adversarially robust neural network model on MNIST.” In: *International Conference on Learning Representations*. arXiv: 1805.09190. URL: https://openreview.net/forum?id=S1EHOsC9tX (cit. on pp. 19, 25, 34, 39, 44, 49).

Sen, Sanchari, Balaraman Ravindran, and Anand Raghunathan (2020). “EMPIR: Ensembles of Mixed Precision Deep Networks for Increased Robustness Against Adversarial Attacks.” In: *International Conference on Learning Representations*. arXiv: 2004.10162. URL: https://openreview.net/forum?id=HJem3yHKwH (cit. on p. 20).

Shen, Shiwei, Guoqing Jin, Ke Gao, and Yongdong Zhang (2019). “APE-GAN: Adversarial Perturbation Elimination with GAN.” In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3842–3846. DOI: 10.1109/ICASSP.2019.8683044 (cit. on p. 20).

Sokol, Kacper and Peter A Flach (2019). “Counterfactual Explanations of Machine Learning Predictions: Opportunities and Challenges for AI Safety.” In: *Proceedings of the AAAI Workshop on Artificial Intelligence Safety 2019*. URL: https://openreview.net/forum?id=H1-Y7RedZr (cit. on p. 12).

Song, Yang, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman (2018). “PixelDefend: Leveraging Generative Models to Understand and Defend against Adversarial Examples.” In: *International Conference on Learning Representations*. arXiv: 1710.10766. URL: https://openreview.net/forum?id=rJUYGxbCW (cit. on p. 20).

Strathern, Marilyn (1997). “‘Improving ratings’: audit in the British University system.” In: *European Review* 5.3, pp. 305–321. DOI: 10.1002/(SICI)1234-981X(199707)5:3<305::AID-EURO184>3.0.CO;2-4 (cit. on p. 48).

Szegedy, Christian, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus (2013). “Intriguing properties of neural networks.” In: *International Conference on Learning Representations*. arXiv: 1312.6199. URL: https://openreview.net/forum?id=kklr_MTHMRQjG (cit. on pp. 14–17, 20, 21, 39).

Team, The Theano Development, Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, et al. (2016). “Theano: A Python framework for fast computation of mathematical expressions.” In: arXiv: 1605.02688 (cit. on pp. 23, 36, 39).

Townsend, Jamie (2017). *A new trick for calculating Jacobian vector products*. URL: https://j-towns.github.io/2017/06/12/A-new-trick.html (cit. on p. 36).

Tramèr, Florian, Nicholas Carlini, Wieland Brendel, and Aleksander Madry (2020). “On Adaptive Attacks to Adversarial Example Defenses.” In: *Advances in Neural Information Processing Systems 33*. arXiv: 2002.08347 (cit. on pp. 21, 25, 48, 49).

Tramèr, Florian, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel (2017). “The Space of Transferable Adversarial Examples.” In: arXiv: 1704.03453 (cit. on p. 21).

Tramèr, Florian, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart (2016). “Stealing machine learning models via prediction APIs.” In: *25th USENIX Security Symposium (USENIX Security 16)*, pp. 601–618. arXiv: 1609.02943 (cit. on p. 16).

Van Rossum, Guido et al. (1995). *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam. URL: https://ir.cwi.nl/pub/5008/05008D.pdf (cit. on p. 38).

Verma, Gunjan and Ananthram Swami (2019). “Error Correcting Output Codes Improve Probability Estimation and Adversarial Robustness of Deep Neural Networks.” In: *Advances in Neural Information Processing Systems*. Vol. 32, pp. 8646–8656. URL: https://dl.acm.org/doi/10.5555/3454287.3455063 (cit. on p. 20).

Whidden, Peter (2020). *Tensor Canvas*. URL: https://github.com/PWhiddy/tensor-canvas (cit. on p. 42).

Xiao, Chang, Peilin Zhong, and Changxi Zheng (2020). “Enhancing Adversarial Defense by k-Winners-Take-All.” In: *International Conference on Learning Representations*. arXiv: 1905.10510. URL: https://openreview.net/forum?id=Skgvy64tvr (cit. on p. 20).

Xie, Cihang, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille (2018). “Mitigating Adversarial Effects Through Randomization.” In: *International Conference on Learning Representations*. arXiv: 1711.01991. URL: https://openreview.net/forum?id=Sk9yuql0Z (cit. on p. 20).

Yang, Yuzhe, Guo Zhang, Dina Katabi, and Zhi Xu (2019). “ME-Net: Towards Effective Adversarial Robustness with Matrix Estimation.” In: *Proceedings of the 36th International Conference on Machine Learning*. Vol. 97. Proceedings of Machine Learning Research. PMLR. URL: http://proceedings.mlr.press/v97/yang19e.html (cit. on p. 20).

Yang, Zhuolin, Bo Li, Pin-Yu Chen, and Dawn Song (2019). “Characterizing Audio Adversarial Examples Using Temporal Dependency.” In: *International Conference on Learning Representations*. arXiv: 1809.10875. URL: https://openreview.net/forum?id=r1g4E3C9t7 (cit. on p. 20).

Zantedeschi, Valentina, Maria-Irina Nicolae, and Ambrish Rawat (2017). “Efficient Defenses Against Adversarial Attacks.” In: *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. AISec ’17. Association for Computing Machinery, pp. 39–49. ISBN: 978-1-4503-5202-4. DOI: 10.1145/3128572.3140449 (cit. on p. 20).

Appendix

The publications are ordered chronologically.

Foolbox: A Python toolbox to benchmark the robustness of machine learning models

The following six pages have been published as:

Jonas Rauber*, Wieland Brendel*, Matthias Bethge (2017). “Foolbox: A Python toolbox to benchmark the robustness of machine learning models”. *Reliable Machine Learning in the Wild Workshop, 34th International Conference on Machine Learning* (ICML 2017).

A summary of the motivation, results, and discussion can be found in section 2.3 on page 38. * joint first authorship

Abstract

Even todays most advanced machine learning models are easily fooled by almost imperceptible perturbations of their inputs. *Foolbox* is a new Python package to generate such adversarial perturbations and to quantify and compare the robustness of machine learning models. It is build around the idea that the most comparable robustness measure is the minimum perturbation needed to craft an adversarial example. To this end, Foolbox provides reference implementations of most published adversarial attack methods alongside some new ones, all of which perform internal hyperparameter tuning to find the minimum adversarial perturbation. Additionally, Foolbox interfaces with most popular deep learning frameworks such as PyTorch, Keras, TensorFlow, Theano and MXNet and allows different adversarial criteria such as targeted misclassification and top-k misclassification as well as different distance measures. The code is licensed under the MIT license and is openly available at https://github.com/bethgelab/foolbox. The most up-to-date documentation can be found at http://foolbox.readthedocs.io.

Foolbox: A Python toolbox to benchmark the robustness of machine learning models

Jonas Rauber [*1 2 3] **Wieland Brendel** [*1 2] **Matthias Bethge** [1 2 4 5]

Abstract

Even todays most advanced machine learning models are easily fooled by almost imperceptible perturbations of their inputs. *Foolbox* is a new Python package to generate such adversarial perturbations and to quantify and compare the robustness of machine learning models. It is build around the idea that the most comparable robustness measure is the minimum perturbation needed to craft an adversarial example. To this end, Foolbox provides reference implementations of most published adversarial attack methods alongside some new ones, all of which perform internal hyperparameter tuning to find the minimum adversarial perturbation. Additionally, Foolbox interfaces with most popular deep learning frameworks such as PyTorch, Keras, TensorFlow, Theano and MXNet and allows different adversarial criteria such as targeted misclassification and top-k misclassification as well as different distance measures. The code is licensed under the MIT license and is openly available at `https://github.com/bethgelab/foolbox`. The most up-to-date documentation can be found at `http://foolbox.readthedocs.io`.

In 2013, Szegedy et al. demonstrated that minimal perturbations, often almost imperceptible to humans, can have devastating effects on machine predictions. These so-called *adversarial perturbations* thus demonstrate a striking difference between human and machine perception. As a result, adversarial perturbations have been subject to many studies concerning the generation of such perturbations and strategies to protect machine learning models such as deep neural networks against them.

A practical definition of the robustness R of a model, first used by Szegedy et al. (2013), is the average size of the minimum adversarial perturbation $\rho(\mathbf{x})$ across many samples $\mathbf{x}$,

$$R = \langle \rho(\mathbf{x}) \rangle_{\mathbf{x}} \quad \text{where} \tag{1}$$

$$\rho(\mathbf{x}) = \min_{\boldsymbol{\delta}} d(\mathbf{x}, \mathbf{x}+\boldsymbol{\delta}) \quad \text{s.t.} \quad \mathbf{x}+\boldsymbol{\delta} \text{ is adversarial} \tag{2}$$

and $d(\cdot)$ is some distance measure.

Unfortunately, finding the global minimum adversarial perturbation is close to impossible in any practical setting, and we thus employ heuristic attacks to find a suitable approximation. Such heuristics, however, can fail, in which case we could easily be mislead to believe that a model is robust (Brendel & Bethge, 2017). Our best strategy is thus to employ as many attacks as possible, and to use the minimal perturbation found across all attacks as an approximation to the true global minimum.

At the moment, however, such a strategy is severely obstructed by two problems: first, the code for most known attack methods is either not available at all, or only available for one particular deep learning framework. Second, implementations of the same attack often differ in many details and are thus not directly comparable. Foolbox improves upon the existing Python package *cleverhans* by Papernot et al. (2016b) in three important aspects:

1. It interfaces with most popular machine learning frameworks such as PyTorch, Keras, TensorFlow, Theano, Lasagne and MXNet and provides a straight forward way to add support for other frameworks,
2. it provides reference implementations for more than 15 adversarial attacks with a simple and consistent API, and
3. it supports many different criteria for adversarial examples, including custom ones.

This technical report is structured as follows: In section 1 we provide an overview over Foolbox and demonstrate how

[*]Equal contribution [1]Centre for Integrative Neuroscience, University of Tübingen, Germany [2]Bernstein Center for Computational Neuroscience, Tübingen, Germany [3]International Max Planck Research School for Intelligent Systems, Tübingen, Germany [4]Max Planck Institute for Biological Cybernetics, Tübingen, Germany [5]Institute for Theoretical Physics, University of Tübingen, Germany. Correspondence to: Jonas Rauber <jonas.rauber@bethgelab.org>.

Reliable Machine Learning in the Wild Workshop, 34th International Conference on Machine Learning, Sydney, Australia, 2017.

to benchmark a model and report the result. In section 2 we describe the adversarial attack methods that are implemented in Foolbox and explain the internal hyperparameter tuning.

1. Foolbox Overview

1.1. Structure

Crafting adversarial examples requires five elements: first, a **model** that takes an input (e.g. an image) and makes a prediction (e.g. class-probabilities). Second, a **criterion** that defines what an adversarial is (e.g. misclassification). Third, a **distance measure** that measures the size of a perturbation (e.g. L1-norm). Finally, an **attack algorithm** that takes an input and its label as well as the model, the adversarial criterion and the distance measure to generate an **adversarial perturbation**.

The structure of Foolbox naturally follows this layout and implements five Python modules (models, criteria, distances, attacks, adversarial) summarized below.

Models
`foolbox.models`
This module implements interfaces to several popular machine learning libraries:

- TensorFlow (Abadi et al., 2016)
  `foolbox.models.TensorFlowModel`
- PyTorch (The PyTorch Developers, 2017)
  `foolbox.models.PyTorchModel`
- Theano (Al-Rfou et al., 2016)
  `foolbox.models.TheanoModel`
- Lasagne (Dieleman et al., 2015)
  `foolbox.models.LasagneModel`
- Keras (any backend) (Chollet, 2015)
  `foolbox.models.KerasModel`
- MXNet (Chen et al., 2015)
  `foolbox.models.MXNetModel`

Each interface is initialized with a framework specific representation of the model (e.g. symbolic input and output tensors in TensorFlow or a neural network module in PyTorch). The interface provides the adversarial attack with a standardized set of methods to compute predictions and gradients for given inputs. It is straight-forward to implement interfaces for other frameworks by providing methods to calculate predictions and gradients in the specific framework.

Additionally, Foolbox implements a `CompositeModel` that combines the predictions of one model with the gradient of another. This makes it possible to attack non-differentiable models using gradient-based attacks and allows transfer attacks of the type described by Papernot et al. (2016c).

Criteria
`foolbox.criteria`
A *criterion* defines under what circumstances an [input, label]-pair is considered an adversarial. The following criteria are implemented:

- Misclassification
  `foolbox.criteria.Misclassification`
  Defines adversarials as inputs for which the predicted class is not the original class.
- Top-k Misclassification
  `foolbox.criteria.TopKMisclassification`
  Defines adversarials as inputs for which the original class is not one of the top-k predicted classes.
- Original Class Probability
  `foolbox.criteria.OriginalClassProbability`
  Defines adversarials as inputs for which the probability of the original class is below a given threshold.
- Targeted Misclassification
  `foolbox.criteria.TargetClass`
  Defines adversarials as inputs for which the predicted class is the given target class.
- Target Class Probability
  `foolbox.criteria.TargetClassProbability`
  Defines adversarials as inputs for which the probability of a given target class is above a given threshold.

Custom adversarial criteria can be defined and employed. Some attacks are inherently specific to particular criteria and thus only work with those.

Distance Measures
`foolbox.distances`
Distance measures are used to quantify the size of adversarial perturbations. Foolbox implements the two commonly employed distance measures and can be extended with custom ones:

- Mean Squared Distance
  `foolbox.distances.MeanSquaredDistance`
  Calculates the mean squared error
  $d(\mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_i (x_i - y_i)^2$
  between two vectors $\mathbf{x}$ and $\mathbf{y}$.

- Mean Absolute Distance
  `foolbox.distances.MeanAbsoluteDistance`
  Calculates the mean absolute error
  $d(\mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_i |x_i - y_i|$
  between two vectors $\mathbf{x}$ and $\mathbf{y}$.

- $L\infty$
  `foolbox.distances.Linfinity`
  Calculates the $L\infty$-norm $d(\mathbf{x}, \mathbf{y}) = \max_i |x_i - y_i|$ between two vectors $\mathbf{x}$ and $\mathbf{y}$.

- $L0$
  `foolbox.distances.L0`
  Calculates the $L0$-norm $d(\mathbf{x}, \mathbf{y}) = \sum_i \mathbb{1}_{x_i \neq y_i}$ between two vectors $\mathbf{x}$ and $\mathbf{y}$.

To achieve invariance to the scale of the input values, we normalize each element of $\mathbf{x}, \mathbf{y}$ by the difference between the smallest and largest allowed value (e.g. 0 and 255).

Attacks
`foolbox.attacks`
Foolbox implements a large number of adversarial attacks, see section 2 for an overview. Each attack takes a model for which adversarials should be found and a criterion that defines what an adversarial is. The default criterion is *misclassification*. It can then be applied to a reference input to which the adversarial should be close and the corresponding label. Attacks perform internal hyperparameter tuning to find the minimum perturbation. As an example, our implementation of the fast gradient sign method (FGSM) searches for the minimum step-size that turns the input into an adversarial. As a result there is no need to specify hyperparameters for attacks like FGSM. For computational efficiency, more complex attacks with several hyperparameters only tune some of them.

Adversarial
`foolbox.adversarial`
An instance of the adversarial class encapsulates all information about an adversarial, including which model, criterion and distance measure was used to find it, the original unperturbed input and its label or the size of the smallest adversarial perturbation found by the attack.

An adversarial object is automatically created whenever an attack is applied to an [input, label]-pair. By default, only the actual adversarial input is returned. Calling the attack with `unpack` set to `False` returns the full object instead. Such an adversarial object can then be passed to an adversarial attack instead of the [input, label]-pair, enabling advanced use cases such as pausing and resuming long-running attacks.

1.2. Reporting Benchmark Results

When reporting benchmark results generated with Foolbox the following information should be stated:

- the version number of Foolbox,
- the set of input samples,
- the set of attacks applied to the inputs,
- any non-default hyperparameter setting,
- the criterion and
- the distance metric.

1.3. Versioning System

Each release of Foolbox is tagged with a version number of the type MAJOR.MINOR.PATCH that follows the principles of semantic versioning[1] with some additional precautions for comparable benchmarking. We increment the

1. MAJOR version when we make changes to the API that break compatibility with previous versions.
2. MINOR version when we add functionality or make backwards compatible changes that can affect the benchmark results.
3. PATCH version when we make backwards compatible bug fixes that do not affect benchmark results.

Thus, to compare the robustness of two models it is important to use the same MAJOR.MINOR version of Foolbox. Accordingly, the version number of Foolbox should always be reported alongside the benchmark results, see section 1.2.

2. Implemented Attack Methods

We here give a short overview over each attack method implemented in Foolbox, referring the reader to the original references for more details. We use the following notation:

$\mathbf{x}$	a model input
ℓ	a class label
$\mathbf{x}_0$	reference input
ℓ_0	reference label
$L(\mathbf{x}, \ell)$	loss (e.g. cross-entropy)
$[b_{\text{min}}, b_{\text{max}}]$	input bounds (e.g. 0 and 255)

2.1. Gradient-Based Attacks

Gradient-based attacks linearize the loss (e.g. cross-entropy) around an input $\mathbf{x}$ to find directions $\boldsymbol{\rho}$ to which the model predictions for class ℓ are most sensitive to,

$$L(\mathbf{x} + \boldsymbol{\rho}, \ell) \approx L(\mathbf{x}, \ell) + \boldsymbol{\rho}^\top \nabla_{\mathbf{x}} L(\mathbf{x}, \ell). \qquad (3)$$

Here $\nabla_{\mathbf{x}} L(\mathbf{x}, \ell)$ is referred to as the gradient of the loss w.r.t. the input $\mathbf{x}$.

[1] http://semver.org/

Gradient Attack
`foolbox.attacks.GradientAttack`
This attack computes the gradient $\mathbf{g}(\mathbf{x}_0) = \nabla_\mathbf{x} L(\mathbf{x}_0, \ell_0)$ once and then seeks the minimum step size ϵ such that $\mathbf{x}_0 + \epsilon \mathbf{g}(\mathbf{x}_0)$ is adversarial.

Gradient Sign Attack (FGSM)
`foolbox.attacks.GradientSignAttack`
`foolbox.attacks.FGSM`
This attack computes the gradient $\mathbf{g}(\mathbf{x}_0) = \nabla_\mathbf{x} L(\mathbf{x}_0, \ell_0)$ once and then seeks the minimum step size ϵ such that $\mathbf{x}_0 + \epsilon\,\mathrm{sign}(\mathbf{g}(\mathbf{x}_0))$ is adversarial (Goodfellow et al., 2014).

Iterative Gradient Attack
`foolbox.attacks.IterativeGradientAttack`
Iterative gradient ascent seeks adversarial perturbations by maximizing the loss along small steps in the gradient direction $\mathbf{g}(\mathbf{x}) = \nabla_\mathbf{x} L(\mathbf{x}, \ell_0)$, i.e. the algorithm iteratively updates $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \epsilon \mathbf{g}(\mathbf{x}_k)$. The step-size ϵ is tuned internally to find the minimum perturbation.

Iterative Gradient Sign Attack
`foolbox.attacks.IterativeGradientSignAttack`
Similar to iterative gradient ascent, this attack seeks adversarial perturbations by maximizing the loss along small steps in the ascent direction $\mathrm{sign}(\mathbf{g}(\mathbf{x})) = \mathrm{sign}\left(\nabla_\mathbf{x} L(\mathbf{x}, \ell_0)\right)$, i.e. the algorithm iteratively updates $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \epsilon\,\mathrm{sign}(\mathbf{g}(\mathbf{x}_k))$. The step-size ϵ is tuned internally to find the minimum perturbation.

DeepFool $L2$ Attack
`foolbox.attacks.DeepFoolL2Attack`
In each iteration DeepFool (Moosavi-Dezfooli et al., 2015) computes for each class $\ell \neq \ell_0$ the minimum distance $d(\ell, \ell_0)$ that it takes to reach the class boundary by approximating the model classifier with a linear classifier. It then makes a corresponding step in the direction of the class with the smallest distance.

DeepFool $L\infty$ Attack
`foolbox.attacks.DeepFoolLinfinityAttack`
Like the DeepFool L2 Attack, but minimizes the $L\infty$-norm instead.

L-BFGS Attack
`foolbox.attacks.LBFGSAttack`
L-BFGS-B is a second-order optimiser that we here use to find the minimum of

$$L(\mathbf{x} + \boldsymbol{\rho}, \ell) + \lambda \left\|\boldsymbol{\rho}\right\|_2^2 \quad \text{s.t.} \quad x_i + \rho_i \in [b_{\text{min}}, b_{\text{max}}]$$

where $\ell \neq \ell_0$ is the target class (Szegedy et al., 2013). A line-search is performed over the regularisation parameter $\lambda > 0$ to find the minimum adversarial perturbation. If the target class is not specified we choose ℓ as the class of the adversarial example generated by the gradient attack.

SLSQP Attack
`foolbox.attacks.SLSQPAttack`
Compared to L-BFGS-B, SLSQP allows to additionally specify non-linear constraints. This enables us to skip the line-search and to directly optimise

$$\left\|\boldsymbol{\rho}\right\|_2^2 \quad \text{s.t.} \quad L(\mathbf{x} + \boldsymbol{\rho}, \ell) = l \ \wedge\ x_i + \rho_i \in [b_{\text{min}}, b_{\text{max}}]$$

where $\ell \neq \ell_0$ is the target class. If the target class is not specified we choose ℓ as the class of the adversarial example generated by the gradient attack.

Jacobian-Based Saliency Map Attack
`foolbox.attacks.SaliencyMapAttack`
This targeted attack (Papernot et al., 2016a) uses the gradient to compute a *saliency score* for each input feature (e.g. pixel). This saliency score reflects how strongly each feature can push the model classification from the reference to the target class. This process is iterated, and in each iteration only the feature with the maximum saliency score is perturbed.

2.2. Score-Based Attacks

Score-based attacks do not require gradients of the model, but they expect meaningful scores such as probabilites or logits which can be used to approximate gradients.

Single Pixel Attack
`foolbox.attacks.SinglePixelAttack`
This attack (Narodytska & Kasiviswanathan, 2016) probes the robustness of a model to changes of single pixels by setting a single pixel to white or black. It repeats this process for every pixel in the image.

Local Search Attack
`foolbox.attacks.LocalSearchAttack`
This attack (Narodytska & Kasiviswanathan, 2016) measures the model's sensitivity to individual pixels by applying extreme perturbations and observing the effect on the probability of the correct class. It then perturbs the pixels to which the model is most sensitive. It repeats this process until the image is adversarial, searching for additional critical pixels in the neighborhood of previously found ones.

Approximate L-BFGS Attack
`foolbox.attacks.ApproximateLBFGSAttack`
Same as L-BFGS except that gradients are computed numerically. Note that this attack is only suitable if the input dimensionality is small.

2.3. Decision-Based Attacks

Decision-based attacks rely only on the class decision of the model. They do not require gradients or probabilities.

Boundary Attack
`foolbox.attacks.BoundaryAttack`
Foolbox provides the reference implementation for the Boundary Attack (Brendel et al., 2018). The Boundary Attack is the most effective decision-based adversarial attack to minimize the L2-norm of adversarial perturbations. It finds adversarial perturbations as small as the best gradient-based attacks without relying on gradients or probabilities.

Pointwise Attack
`foolbox.attacks.PointwiseAttack`
Foolbox provides the reference implementation for the Pointwise Attack. The Pointwise Attack is the most effective decision-based adversarial attack to minimize the L0-norm of adversarial perturbations.

Additive Uniform Noise Attack
`foolbox.attacks.AdditiveUniformNoiseAttack`
This attack probes the robustness of a model to i.i.d. uniform noise. A line-search is performed internally to find minimal adversarial perturbations.

Additive Gaussian Noise Attack
`foolbox.attacks.AdditiveGaussianNoiseAttack`
This attack probes the robustness of a model to i.i.d. normal noise. A line-search is performed internally to find minimal adversarial perturbations.

Salt and Pepper Noise Attack
`foolbox.attacks.SaltAndPepperNoiseAttack`
This attack probes the robustness of a model to i.i.d. salt-and-pepper noise. A line-search is performed internally to find minimal adversarial perturbations.

Contrast Reduction Attack
`foolbox.attacks.ContrastReductionAttack`
This attack probes the robustness of a model to contrast reduction. A line-search is performed internally to find minimal adversarial perturbations.

Gaussian Blur Attack
`foolbox.attacks.GaussianBlurAttack`
This attack probes the robustness of a model to Gaussian blur. A line-search is performed internally to find minimal blur needed to turn the image into an adversarial.

Precomputed Images Attack
`foolbox.attacks.PrecomputedImagesAttack`
Special attack that is initialized with a set of expected input images and corresponding adversarial candidates. When applied to an image, it tests the models robustness to the precomputed adversarial candidate corresponding to the given image. This can be useful to test a models robustness against image perturbations created using an external method.

3. Acknowledgements

This work was supported by the Carl Zeiss Foundation (0563-2.8/558/3), the Bosch Forschungsstiftung (Stifterverband, T113/30057/17), the International Max Planck Research School for Intelligent Systems (IMPRS-IS), the German Research Foundation (DFG, CRC 1233, Robust Vision: Inference Principles and Neural Mechanisms) and the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior/Interior Business Center (DoI/IBC) contract number D16PC00003. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/IBC, or the U.S. Government.

References

Abadi, Martín, Agarwal, Ashish, Barham, Paul, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016. URL http://arxiv.org/abs/1603.04467.

Al-Rfou, Rami, Alain, Guillaume, Almahairi, Amjad, Angermüller, Christof, et al. Theano: A python framework for fast computation of mathematical expressions. *CoRR*, abs/1605.02688, 2016. URL http://arxiv.org/abs/1605.02688.

Brendel, W. and Bethge, M. Comment on "Biologically inspired protection of deep networks from adversarial attacks". *arXiv*, abs/1704.01547, 2017. URL http://arxiv.org/abs/1704.01547.

Brendel, Wieland, Rauber, Jonas, and Bethge, Matthias. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SyZI0GWCZ.

Chen, Tianqi, Li, Mu, Li, Yutian, Lin, Min, Wang, Naiyan, Wang, Minjie, Xiao, Tianjun, Xu, Bing, Zhang, Chiyuan, and Zhang, Zheng. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *Neural Information Processing Sys-*

tems, Workshop on Machine Learning Systems, 2015. URL http://arxiv.org/abs/1603.04467.

Chollet, François. Keras. https://github.com/fchollet/keras, 2015.

Dieleman, Sander, Schlüter, Jan, Raffel, Colin, Olson, Eben, Sonderby, Søren Kaae, et al. Lasagne: First release., August 2015. URL http://dx.doi.org/10.5281/zenodo.27878.

Goodfellow, Ian J, Shlens, Jonathon, and Szegedy, Christian. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Moosavi-Dezfooli, Seyed-Mohsen, Fawzi, Alhussein, and Frossard, Pascal. Deepfool: a simple and accurate method to fool deep neural networks. *CoRR*, abs/1511.04599, 2015. URL http://arxiv.org/abs/1511.04599.

Narodytska, Nina and Kasiviswanathan, Shiva Prasad. Simple black-box adversarial perturbations for deep networks. *CoRR*, abs/1612.06299, 2016. URL http://arxiv.org/abs/1612.06299.

Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., and Swami, A. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, pp. 372–387, March 2016a. doi: 10.1109/EuroSP.2016.36.

Papernot, Nicolas, Goodfellow, Ian, Sheatsley, Ryan, Feinman, Reuben, and McDaniel, Patrick. cleverhans v1.0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768*, 2016b.

Papernot, Nicolas, McDaniel, Patrick D., Goodfellow, Ian J., Jha, Somesh, Celik, Z. Berkay, and Swami, Ananthram. Practical black-box attacks against deep learning systems using adversarial examples. *CoRR*, abs/1602.02697, 2016c. URL http://arxiv.org/abs/1602.02697.

Szegedy, Christian, Zaremba, Wojciech, Sutskever, Ilya, Bruna, Joan, Erhan, Dumitru, Goodfellow, Ian J., and Fergus, Rob. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.

The PyTorch Developers. Pytorch. http://pytorch.org, 2017.

Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models

The following twelve pages have been published as:

Wieland Brendel*, Jonas Rauber*, Matthias Bethge (2018). "Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models". *Sixth International Conference on Learning Representations (ICLR 2018).*

A summary of the motivation, results, and discussion can be found in section 2.1 on page 31. * joint first authorship

Abstract

Many machine learning algorithms are vulnerable to almost imperceptible perturbations of their inputs. So far it was unclear how much risk adversarial perturbations carry for the safety of real-world machine learning applications because most methods used to generate such perturbations rely either on detailed model information (*gradient-based attacks*) or on confidence scores such as class probabilities (*score-based attacks*), neither of which are available in most real-world scenarios. In many such cases one currently needs to retreat to *transfer-based attacks* which rely on cumbersome substitute models, need access to the training data and can be defended against. Here we emphasise the importance of attacks which solely rely on the final model decision. Such *decision-based attacks* are (1) applicable to real-world black-box models such as autonomous cars, (2) need less knowledge and are easier to apply than transfer-based attacks and (3) are more robust to simple defences than gradient- or score-based attacks. Previous attacks in this category were limited to simple models or simple datasets. Here we introduce the Boundary Attack, a decision-based attack that starts from a large adversarial perturbation and then seeks to reduce the perturbation while staying adversarial. The attack is conceptually simple, requires close to no hyperparameter tuning, does not rely on substitute models and is competitive with the best gradient-based attacks in standard computer vision tasks like ImageNet. We apply the attack on two black-box algorithms from Clarifai.com. The Boundary Attack in particular and the class of decision-based attacks in general open new avenues to study the robustness of machine learning models and raise new questions regarding the safety of deployed machine learning systems. An implementation of the attack is available as part of Foolbox (https://github.com/bethgelab/foolbox).

Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models

Wieland Brendel[*]**, Jonas Rauber**[*] **& Matthias Bethge**
Werner Reichardt Centre for Integrative Neuroscience,
Eberhard Karls University Tübingen, Germany
`{wieland,jonas,matthias}@bethgelab.org`

Abstract

Many machine learning algorithms are vulnerable to almost imperceptible perturbations of their inputs. So far it was unclear how much risk adversarial perturbations carry for the safety of real-world machine learning applications because most methods used to generate such perturbations rely either on detailed model information (*gradient-based attacks*) or on confidence scores such as class probabilities (*score-based attacks*), neither of which are available in most real-world scenarios. In many such cases one currently needs to retreat to *transfer-based attacks* which rely on cumbersome substitute models, need access to the training data and can be defended against. Here we emphasise the importance of attacks which solely rely on the final model decision. Such *decision-based attacks* are (1) applicable to real-world black-box models such as autonomous cars, (2) need less knowledge and are easier to apply than transfer-based attacks and (3) are more robust to simple defences than gradient- or score-based attacks. Previous attacks in this category were limited to simple models or simple datasets. Here we introduce the Boundary Attack, a decision-based attack that starts from a large adversarial perturbation and then seeks to reduce the perturbation while staying adversarial. The attack is conceptually simple, requires close to no hyperparameter tuning, does not rely on substitute models and is competitive with the best gradient-based attacks in standard computer vision tasks like ImageNet. We apply the attack on two black-box algorithms from Clarifai.com. The Boundary Attack in particular and the class of decision-based attacks in general open new avenues to study the robustness of machine learning models and raise new questions regarding the safety of deployed machine learning systems. An implementation of the attack is available as part of Foolbox (https://github.com/bethgelab/foolbox).

	Gradient-based Model M	**Transfer-based** Training Data T	**Score-based** Detailed Model Prediction Y (e.g. probabilities or logits)	**Decision-based** Final Model Prediction Y_{max} (e.g. max class label)
				less information →
Untargeted Flip to any label	FGSM, DeepFool	FGSM Transfer	Local Search	★ **this work (Boundary Attack)**
Targeted Flip to target label	L-BFGS-B, Houdini, JSMA, Carlini & Wagner, Iterative Gradient Descent	Ensemble Transfer	ZOO	★

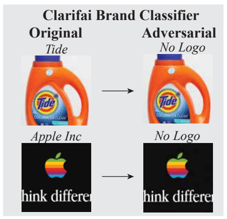


Figure 1: (Left) Taxonomy of adversarial attack methods. The Boundary Attack is applicable to real-world ML algorithms because it only needs access to the final decision of a model (e.g. class-label or transcribed sentence) and does not rely on model information like the gradient or the confidence scores. (Right) Application to the Clarifai Brand Recognition Model.

[*]Equal contribution.

1 Introduction

Many high-performance machine learning algorithms used in computer vision, speech recognition and other areas are susceptible to minimal changes of their inputs (Szegedy et al., 2013). As a concrete example, a modern deep neural network like VGG-19 trained on object recognition might perfectly recognize the main object in an image as a *tiger cat*, but if the pixel values are only slightly perturbed in a specific way then the prediction of the very same network is drastically altered (e.g. to *bus*). These so-called adversarial perturbations are ubiquitous in many machine learning models and are often imperceptible to humans. Algorithms that seek to find such adversarial perturbations are generally denoted as *adversarial attacks*.

Adversarial perturbations have drawn interest from two different sides. On the one side, they are worrisome for the integrity and security of deployed machine learning algorithms such as autonomous cars or face recognition systems. Minimal perturbations on street signs (e.g. turning a stop-sign into a 200 km/h speed limit) or street lights (e.g. turning a red into a green light) can have severe consequences. On the other hand, adversarial perturbations provide an exciting spotlight on the gap between the sensory information processing in humans and machines and thus provide guidance towards more robust, human-like architectures.

Adversarial attacks can be roughly divided into three categories: *gradient-based*, *score-based* and *transfer-based* attacks (cp. Figure 1). Gradient-based and score-based attacks are often denoted as white-box and oracle attacks respectively, but we try to be as explicit as possible as to what information is being used in each category[1]. A severe problem affecting attacks in all of these categories is that they are surprisingly straight-forward to defend against:

- **Gradient-based attacks.** Most existing attacks rely on detailed model information including the gradient of the loss w.r.t. the input. Examples are the Fast-Gradient Sign Method (FGSM), the Basic Iterative Method (BIM) (Kurakin et al., 2016), DeepFool (Moosavi-Dezfooli et al., 2015), the Jacobian-based Saliency Map Attack (JSMA) (Papernot et al., 2015), Houdini (Cisse et al., 2017) and the Carlini & Wagner attack (Carlini & Wagner, 2016a).

  Defence: A simple way to defend against gradient-based attacks is to mask the gradients, for example by adding non-differentiable elements either implicitly through means like defensive distillation (Papernot et al., 2016) or saturated non-linearities (Nayebi & Ganguli, 2017), or explicitly through means like non-differentiable classifiers (Lu et al., 2017).

- **Score-based attacks.** A few attacks are more agnostic and only rely on the predicted scores (e.g. class probabilities or logits) of the model. On a conceptual level these attacks use the predictions to numerically estimate the gradient. This includes black-box variants of JSMA (Narodytska & Kasiviswanathan, 2016) and of the Carlini & Wagner attack (Chen et al., 2017) as well as generator networks that predict adversarials (Hayes & Danezis, 2017).

  Defence: It is straight-forward to severely impede the numerical gradient estimate by adding stochastic elements like dropout into the model. Also, many robust training methods introduce a sharp-edged plateau around samples (Tramer et al., 2017) which not only masks gradients themselves but also their numerical estimate.

- **Transfer-based attacks.** Transfer-based attacks do not rely on model information but need information about the training data. This data is used to train a fully observable substitute model from which adversarial perturbations can be synthesized (Papernot et al., 2017a). They rely on the empirical observation that adversarial examples often transfer between models. If adversarial examples are created on an ensemble of substitute models the success rate on the attacked model can reach up to 100% in certain scenarios (Liu et al., 2016).

  Defence: A recent defence method against transfer attacks (Tramer et al., 2017), which is based on robust training on a dataset augmented by adversarial examples from an ensemble of substitute models, has proven highly successful against basically all attacks in the 2017 Kaggle Competition on Adversarial Attacks[2].

[1] For example, the term *oracle* does not convey what information is used by attacks in this category.

[2] `https://www.kaggle.com/c/nips-2017-defense-against-adversarial-attack`

The fact that many attacks can be easily averted makes it often extremely difficult to assess whether a model is truly robust or whether the attacks are just too weak, which has lead to premature claims of robustness for DNNs (Carlini & Wagner, 2016b; Brendel & Bethge, 2017).

This motivates us to focus on a category of adversarial attacks that has so far received fairly little attention:

- **Decision-based attacks.** Direct attacks that solely rely on the final decision of the model (such as the top-1 class label or the transcribed sentence).

The delineation of this category is justified for the following reasons: First, compared to score-based attacks decision-based attacks are much more relevant in real-world machine learning applications where confidence scores or logits are rarely accessible. At the same time decision-based attacks have the potential to be much more robust to standard defences like gradient masking, intrinsic stochasticity or robust training than attacks from the other categories. Finally, compared to transfer-based attacks they need much less information about the model (neither architecture nor training data) and are much simpler to apply.

There currently exists no effective decision-based attack that scales to natural datasets such as ImageNet and is applicable to deep neural networks (DNNs). The most relevant prior work is a variant of transfer attacks in which the training set needed to learn the substitute model is replaced by a synthetic dataset (Papernot et al., 2017b). This synthetic dataset is generated by the adversary alongside the training of the substitute; the labels for each synthetic sample are drawn from the black-box model. While this approach works well on datasets for which the intra-class variability is low (such as MNIST) it has yet to be shown that it scales to more complex natural datasets such as CIFAR or ImageNet. Other decision-based attacks are specific to linear or convex-inducing classifiers (Dalvi et al., 2004; Lowd & Meek, 2005; Nelson et al., 2012) and are not applicable to other machine learning models. The work by (Biggio et al., 2013) basically stands between transfer attacks and decision-based attacks in that the substitute model is trained on a dataset for which the labels have been observed from the black-box model. This attack still requires knowledge about the data distribution on which the black-box models was trained on and so we don't consider it a pure decision-based attack. Finally, some naive attacks such as a line-search along a random direction away from the original sample can qualify as decision-based attacks but they induce large and very visible perturbations that are orders of magnitude larger than typical gradient-based, score-based or transfer-based attacks.

Throughout the paper we focus on the threat scenario in which the adversary aims to change the decision of a model (either targeted or untargeted) for a particular input sample by inducing a minimal perturbation to the sample. The adversary can observe the final decision of the model for arbitrary inputs and it knows at least one perturbation, however large, for which the perturbed sample is adversarial.

The contributions of this paper are as follows:

- We emphasise decision-based attacks as an important category of adversarial attacks that are highly relevant for real-world applications and important to gauge model robustness.
- We introduce the first effective decision-based attack that scales to complex machine learning models and natural datasets. The Boundary Attack is (1) conceptually surprisingly simple, (2) extremely flexible, (3) requires little hyperparameter tuning and (4) is competitive with the best gradient-based attacks in both targeted and untargeted computer vision scenarios.
- We show that the Boundary Attack is able to break previously suggested defence mechanisms like defensive distillation.
- We demonstrate the practical applicability of the Boundary Attack on two black-box machine learning models for brand and celebrity recognition available on Clarifai.com.

1.1 NOTATION

Throughout the paper we use the following notation: $\boldsymbol{o}$ refers to the original input (e.g. an image), $\boldsymbol{y} = F(\boldsymbol{o})$ refers to the full prediction of the model $F(\cdot)$ (e.g. logits or probabilities), y_{max} is the predicted label (e.g. class-label). Similarly, $\tilde{\boldsymbol{o}}$ refers to the adversarially perturbed image, $\tilde{\boldsymbol{o}}^k$ refers to the perturbed image at the k-th step of an attack algorithm. Vectors are denoted in bold.

2 Boundary Attack

The basic intuition behind the boundary attack algorithm is depicted in Figure 2: the algorithm is *initialized* from a point that is already adversarial and then performs a random walk along the boundary between the adversarial and the non-adversarial region such that (1) it stays in the adversarial region and (2) the distance towards the target image is reduced. In other words we perform rejection sampling with a suitable *proposal distribution* $\mathcal{P}$ to find progressively smaller adversarial perturbations according to a given *adversarial criterion* $c(.)$. The basic logic of the algorithm is described in Algorithm 1, each individual building block is detailed in the next subsections.

```
Data: original image $\mathbf{o}$, adversarial criterion $c(.)$, decision of model $d(.)$
Result: adversarial example $\tilde{\boldsymbol{o}}$ such that the distance $d(\boldsymbol{o}, \tilde{\boldsymbol{o}}) = \|\boldsymbol{o} - \tilde{\boldsymbol{o}}\|_2^2$ is minimized
initialization: $k = 0$, $\tilde{\boldsymbol{o}}^0 \sim \mathcal{U}(0,1)$ s.t. $\tilde{\boldsymbol{o}}^0$ is adversarial;
while $k <$ maximum number of steps do
    draw random perturbation from proposal distribution $\boldsymbol{\eta}_k \sim \mathcal{P}(\tilde{\boldsymbol{o}}^{k-1})$;
    if $\tilde{\boldsymbol{o}}^{k-1} + \boldsymbol{\eta}_k$ is adversarial then
        set $\tilde{\boldsymbol{o}}^k = \tilde{\boldsymbol{o}}^{k-1} + \boldsymbol{\eta}_k$;
    else
        set $\tilde{\boldsymbol{o}}^k = \tilde{\boldsymbol{o}}^{k-1}$;
    end
    $k = k + 1$
end
```

Algorithm 1: Minimal version of the Boundary Attack.

2.1 Initialisation

The Boundary Attack needs to be initialized with a sample that is already adversarial[3]. In an untargeted scenario we simply sample from a maximum entropy distribution given the valid domain of the input. In the computer vision applications below, where the input is constrained to a range of $[0, 255]$ per pixel, we sample each pixel in the initial image $\tilde{\mathbf{o}}^{\mathbf{0}}$ from a uniform distribution $\mathcal{U}(0, 255)$. We reject samples that are not adversarial. In a targeted scenario we start from any sample that is classified by the model as being from the target class.

2.2 Proposal distribution

The efficiency of the algorithm crucially depends on the proposal distribution $\mathcal{P}$, i.e. which random directions are explored in each step of the algorithm. The optimal proposal distribution will generally depend on the domain and / or model to be attacked, but for all vision-related problems tested here a very simple proposal distribution worked surprisingly well. The basic idea behind this proposal distribution is as follows: in the k-th step we want to draw perturbations $\boldsymbol{\eta}^k$ from a maximum entropy distribution subject to the following constraints:

1. The perturbed sample lies within the input domain,
$$\tilde{o}_i^{k-1} + \eta_i^k \in [0, 255]. \tag{1}$$
2. The perturbation has a relative size of δ,
$$\left\|\boldsymbol{\eta}^k\right\|_2 = \delta \cdot d(\mathbf{o}, \tilde{\mathbf{o}}^{\mathbf{k-1}}). \tag{2}$$
3. The perturbation reduces the distance of the perturbed image towards the original input by a relative amount ϵ,
$$d(\mathbf{o}, \tilde{\mathbf{o}}^{\mathbf{k-1}}) - d(\mathbf{o}, \tilde{\mathbf{o}}^{\mathbf{k-1}} + \boldsymbol{\eta}^k) = \epsilon \cdot d(\mathbf{o}, \tilde{\mathbf{o}}^{\mathbf{k-1}}). \tag{3}$$

[3]Note that here *adversarial* does not mean that the decision of the model is wrong—it might make perfect sense to humans—but that the perturbation fulfills the adversarial criterion (e.g. changes the model decision).

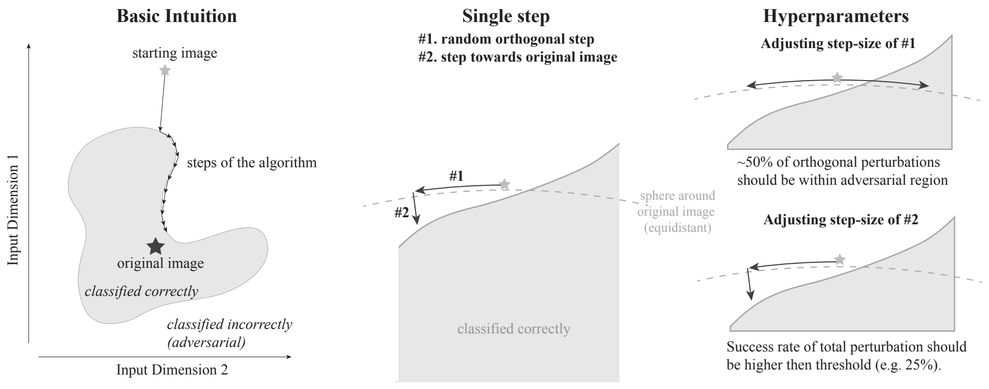

Basic Intuition
starting image
steps of the algorithm
original image
classified correctly
classified incorrectly (adversarial)
Input Dimension 1
Input Dimension 2
Single step
#1. random orthogonal step
#2. step towards original image
#1
#2
sphere around original image (equidistant)
classified correctly
Hyperparameters
Adjusting step-size of #1
~50% of orthogonal perturbations should be within adversarial region
Adjusting step-size of #2
Success rate of total perturbation should be higher then threshold (e.g. 25%).

Figure 2: (Left) In essence the Boundary Attack performs rejection sampling along the boundary between adversarial and non-adversarial images. (Center) In each step we draw a new random direction by (#1) drawing from an iid Gaussian and projecting on a sphere, and by (#2) making a small move towards the target image. (Right) The two step-sizes (orthogonal and towards the original input) are dynamically adjusted according to the local geometry of the boundary.

In practice it is difficult to sample from this distribution, and so we resort to a simpler heuristic: first, we sample from an iid Gaussian distribution $\boldsymbol{\eta}_i^k \sim \mathcal{N}(0,1)$ and then rescale and clip the sample such that (1) and (2) hold. In a second step we project $\boldsymbol{\eta}^k$ onto a sphere around the original image $\boldsymbol{o}$ such that $d(\boldsymbol{o}, \tilde{\boldsymbol{o}}^{k-1} + \boldsymbol{\eta}^k) = d(\boldsymbol{o}, \tilde{\boldsymbol{o}}^{k-1})$ and (1) hold. We denote this as the *orthogonal perturbation* and use it later for hyperparameter tuning. In the last step we make a small movement towards the original image such that (1) and (3) hold. For high-dimensional inputs and small δ, ϵ the constraint (2) will also hold approximately.

2.3 Adversarial criterion

A typical criterion by which an input is classified as *adversarial* is misclassification, i.e. whether the model assigns the perturbed input to some class different from the class label of the original input. Another common choice is targeted misclassification for which the perturbed input has to be classified in a given target class. Other choices include top-k misclassification (the top-k classes predicted for the perturbed input do not contain the original class label) or thresholds on certain confidence scores. Outside of computer vision many other choices exist such as criteria on the word-error rates. In comparison to most other attacks, the Boundary Attack is extremely flexible with regards to the adversarial criterion. It basically allows any criterion (including non-differentiable ones) as long as for that criterion an initial adversarial can be found (which is trivial in most cases).

2.4 Hyperparameter adjustment

The Boundary Attack has only two relevant parameters: the length of the total perturbation δ and the length of the step ϵ towards the original input (see Fig. 2). We adjust both parameters dynamically according to the local geometry of the boundary. The adjustment is inspired by Trust Region methods. In essence, we first test whether the orthogonal perturbation is still adversarial. If this is true, then we make a small movement towards the target and test again. The orthogonal step tests whether the step-size is small enough so that we can treat the decision boundary between the adversarial and the non-adversarial region as being approximately linear. If this is the case, then we expect around 50% of the orthogonal perturbations to still be adversarial. If this ratio is much lower, we reduce the step-size δ, if it is close to 50% or higher we increase it. If the orthogonal perturbation is still adversarial we add a small step towards the original input. The maximum size of this step depends on the angle of the decision boundary in the local neighbourhood (see also Figure 2). If the success rate is too small we decrease ϵ, if it is too large we increase it. Typically, the closer we get to the original image, the flatter the decision boundary becomes and the smaller ϵ has to be to still make progress. The attack is converged whenever ϵ converges to zero.

3 Comparison with other attacks

We quantify the performance of the Boundary Attack on three different standard datasets: MNIST (LeCun et al., 1998), CIFAR-10 (Krizhevsky & Hinton, 2009) and ImageNet-1000 (Deng et al., 2009). To make the comparison with previous results as easy and transparent as possible, we here use the same MNIST and CIFAR networks as Carlini & Wagner (2016a)[4]. In a nutshell, both the MNIST and CIFAR model feature nine layers with four convolutional layers, two max-pooling layers and two fully-connected layers. For all details, including training parameters, we refer the reader to (Carlini & Wagner, 2016a). On ImageNet we use the pretrained networks VGG-19 (Simonyan & Zisserman, 2014), ResNet-50 (He et al., 2015) and Inception-v3 (Szegedy et al., 2015) provided by Keras[5].

We evaluate the Boundary Attack in two settings: an (1) *untargeted setting* in which the adversarial perturbation flips the label of the original sample to any other label, and a (2) *targeted setting* in which the adversarial flips the label to a specific target class. In the untargeted setting we compare the Boundary Attack against three gradient-based attack algorithms:

- **Fast-Gradient Sign Method (FGSM).** FGSM is among the simplest and most widely used untargeted adversarial attack methods. In a nutshell, FGSM computes the gradient $\boldsymbol{g} = \nabla_o \mathcal{L}(\boldsymbol{o}, c)$ that maximizes the loss $\mathcal{L}$ for the true class-label c and then seeks the smallest ϵ for which $\boldsymbol{o} + \epsilon \cdot \boldsymbol{g}$ is still adversarial. We use the implementation in Foolbox 0.10.0 (Rauber et al., 2017).
- **DeepFool.** DeepFool is a simple yet very effective attack. In each iteration it computes for each class $\ell \neq \ell_0$ the minimum distance $d(\ell, \ell_0)$ that it takes to reach the class boundary by approximating the model classifier with a linear classifier. It then makes a corresponding step in the direction of the class with the smallest distance. We use the implementation in Foolbox 0.10.0 (Rauber et al., 2017).
- **Carlini & Wagner.** The attack by Carlini & Wagner (Carlini & Wagner, 2016a) is essentially a refined iterative gradient attack that uses the Adam optimizer, multiple starting points, a tanh-nonlinearity to respect box-constraints and a max-based adversarial constraint function. We use the original implementation provided by the authors with all hyperparameters left at their default values[4].

To evaluate the success of each attack we use the following metric: let $\boldsymbol{\eta}_{A,M}(\boldsymbol{o}_i) \in \mathbb{R}^N$ be the adversarial perturbation that the attack A finds on model M for the i-th sample $\boldsymbol{o}_i$. The total score $\mathcal{S}_A$ for A is the median squared L2-distance across all samples,

$$\mathcal{S}_A(M) = \underset{i}{\text{median}} \left(\frac{1}{N} \left\| \boldsymbol{\eta}_{A,M}(\boldsymbol{o}_i) \right\|_2^2 \right). \tag{4}$$

For MNIST and CIFAR we evaluate 1000 randomly drawn samples from the validation set, for ImageNet we use 250 images.

3.1 Untargeted attack

In the untargeted setting an adversarial is any image for which the predicted label is different from the label of the original image. We show adversarial samples synthesized by the Boundary Attack for each dataset in Figure 3. The score (4) for each attack and each dataset is as follows:

				ImageNet		
	Attack Type	MNIST	CIFAR	VGG-19	ResNet-50	Inception-v3
FGSM	gradient-based	4.2e-02	2.5e-05	1.0e-06	1.0e-06	9.7e-07
DeepFool	gradient-based	4.3e-03	5.8e-06	1.9e-07	7.5e-08	5.2e-08
Carlini & Wagner	gradient-based	2.2e-03	7.5e-06	5.7e-07	2.2e-07	7.6e-08
Boundary (ours)	decision-based	3.6e-03	5.6e-06	2.9e-07	1.0e-07	6.5e-08

[4]`https://github.com/carlini/nn_robust_attacks` (commit 1193c79)
[5]`https://github.com/fchollet/keras` (commit 1b5d54)

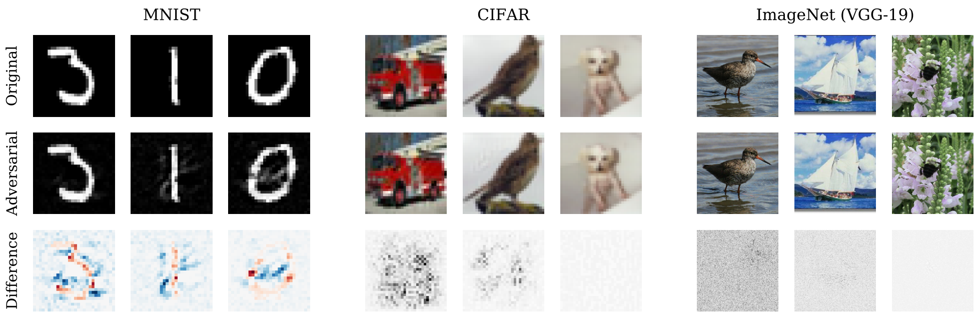


Figure 3: Adversarial examples generated by the Boundary Attack for an MNIST, CIFAR and ImageNet network. For MNIST, the difference shows positive (blue) and negative (red) changes. For CIFAR and ImageNet, we take the norm across color channels. All differences have been scaled up for improved visibility.

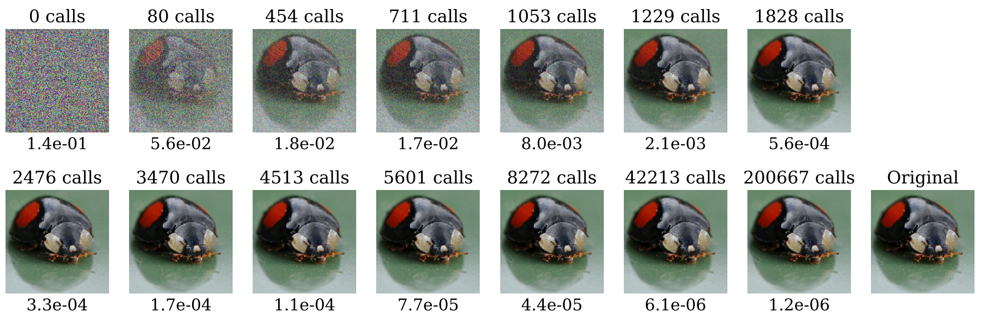


Figure 4: Example of an untargeted attack. Here the goal is to synthesize an image that is as close as possible (in L2-metric) to the original image while being misclassified (the original image is correctly classified). For each image we report the total number of model calls (predictions) until that point (above the image) and the mean squared error between the adversarial and the original (below the image).

Despite its simplicity the Boundary Attack is competitive with gradient-based attacks in terms of the minimal adversarial perturbations and very stable against the choice of the initial point (Figure 5). This finding is quite remarkable given that gradient-based attacks can fully observe the model whereas the Boundary Attack is severely restricted to the final class prediction. To compensate for this lack of information the Boundary Attack needs many more iterations to converge. As a rough measure for the run-time of an attack independent of the quality of its implementation we tracked the number of forward passes (predictions) and backward passes (gradients) through the network requested by each of the attacks to find an adversarial for ResNet-50: averaged over 20 samples and under the same conditions as before, DeepFool needs about 7 forward and 37 backward passes, the Carlini & Wagner attack requires 16.000 forward *and* the same number of backward passes, and the Boundary Attack uses 1.200.000 forward passes but zero backward passes. While that (unsurprisingly) makes the Boundary Attack more expensive to run it is important to note that the Boundary Attacks needs much fewer iterations if one is only interested in imperceptible perturbations, see figures 4 and 6.

3.2 TARGETED ATTACK

We can also apply the Boundary Attack in a targeted setting. In this case we initialize the attack from a sample of the target class that is correctly identified by the model. A sample trajectory from the starting point to the original sample is shown in Figure 7. After around 10^4 calls to the model

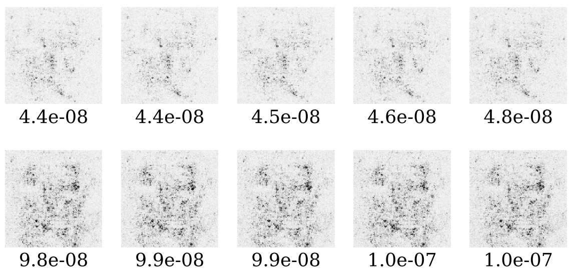


Figure 5: Adversarial perturbation (difference between the adversarial and the original image) for ten repetitions of the Boundary Attack on the **same image**. There are basically two different minima with similar distance (first row and second row) to which the Boundary Attack converges.

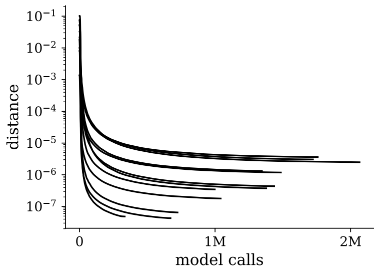


Figure 6: Distance between adversarial and original image over number of model calls for 12 **different images** (until convergence). Very few steps are already sufficient to get almost imperceptible perturbations.

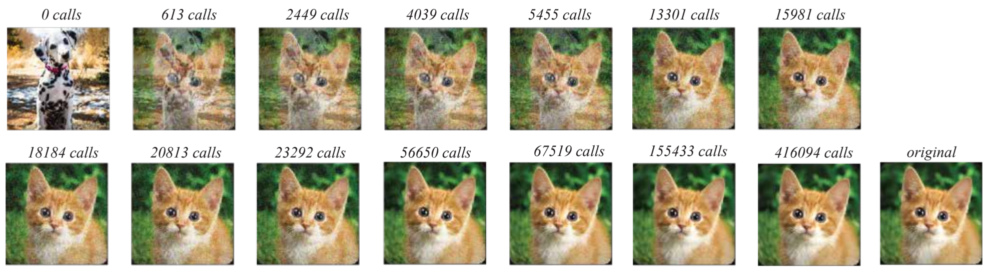


Figure 7: Example of a targeted attack. Here the goal is to synthesize an image that is as close as possible (in L2-metric) to a given image of a tiger cat (2nd row, right) but is classified as a dalmatian dog. For each image we report the total number of model calls (predictions) until that point.

the perturbed image is already clearly identified as a cat by humans and contains no trace of the Dalmatian dog, as which the image is still classified by the model.

In order to compare the Boundary Attack to Carlini & Wagner we define the target target label for each sample in the following way: on MNIST and CIFAR a sample with label ℓ gets the target label $\ell + 1$ modulo 10. On ImageNet we draw the target label randomly but consistent across attacks. The results are as follows:

	Attack Type	MNIST	CIFAR	VGG-19
Carlini & Wagner	gradient-based	4.8e-03	3.0e-05	5.7e-06
Boundary (ours)	decision-based	6.5e-03	3.3e-05	9.9e-06

4 The importance of decision-based attacks to evaluate model robustness

As discussed in the introduction, many attack methods are straight-forward to defend against. One common nuisance is gradient masking in which a model is implicitely or explicitely modified to yield masked gradients. An interesting example is the saturated sigmoid network (Nayebi & Ganguli, 2017) in which an additional regularization term leads the sigmoid activations to saturate, which in turn leads to vanishing gradients and failing gradient-based attacks (Brendel & Bethge, 2017).

Another example is defensive distillation (Papernot et al., 2016). In a nutshell defensive distillation uses a temperature-augmented softmax of the type

$$softmax(x,T)_i = \frac{e^{x_i/T}}{\sum_j e^{x_j/T}} \tag{5}$$

and works as follows:

1. Train a teacher network as usual but with temperature T.
2. Train a distilled network—with the same architecture as the teacher—on the softmax outputs of the teacher. Both the distilled network and the teacher use temperature T.
3. Evaluate the distilled network at temperature $T = 1$ at test time.

Initial results were promising: the success rate of gradient-based attacks dropped from close to 100% down to 0.5%. It later became clear that the distilled networks only appeared to be robust because they masked their gradients of the cross-entropy loss (Carlini & Wagner, 2016b): as the temperature of the softmax is decreased at test time, the input to the softmax increases by a factor of T and so the probabilities saturate at 0 and 1. This leads to vanishing gradients of the cross-entropy loss w.r.t. to the input on which gradient-based attacks rely. If the same attacks are instead applied to the logits the success rate recovers to almost 100% (Carlini & Wagner, 2016a).

Decision-based attacks are immune to such defences. To demonstrate this we here apply the Boundary Attack to two distilled networks trained on MNIST and CIFAR. The architecture is the same as in section 3 and we use the implementation and training protocol by (Carlini & Wagner, 2016a) which is available at `https://github.com/carlini/nn_robust_attacks`. Most importantly, we do not operate on the logits but provide only the class label with maximum probability to the Boundary Attack. The results are as follows:

		MNIST		CIFAR	
	Attack Type	standard	distilled	standard	distilled
FGSM	gradient-based	4.2e-02	fails	2.5e-05	fails
Boundary (ours)	decision-based	3.6e-03	4.2e-03	5.6e-06	1.3e-05

The size of the adversarial perturbations that the Boundary Attack finds is fairly similar for the distilled and the undistilled network. This demonstrates that defensive distillation does not significantly increase the robustness of network models and that the Boundary Attack is able to break defences based on gradient masking.

5 Attacks on real-world applications

In many real-world machine learning applications the attacker has no access to the architecture or the training data but can only observe the final decision. This is true for security systems (e.g. face identification), autonomous cars or speech recognition systems like Alexa or Cortana.

In this section we apply the Boundary Attack to two models of the cloud-based computer vision API by Clarifai[6]. The first model identifies brand names in natural images and recognizes over 500 brands. The second model identifies celebrities and can recognize over 10.000 individuals. Multiple identifications per image are possible but we only consider the one with the highest confidence score. It is important to note that Clarifai does provide confidence scores for each identified class (but not for all possible classes). However, in our experiments we do not provide this confidence score to the Boundary Attack. Instead, our attack only receives the name of the identified object (e.g. *Pepsi* or *Verizon* in the brand-name detection task).

We selected several samples of natural images with clearly visible brand names or portraits of celebrities. We then make a square crop and resize the image to 100×100 pixels. For each sample we make sure that the brand or the celebrity is clearly visible and that the corresponding Clarifai

[6]`www.clarifai.com`

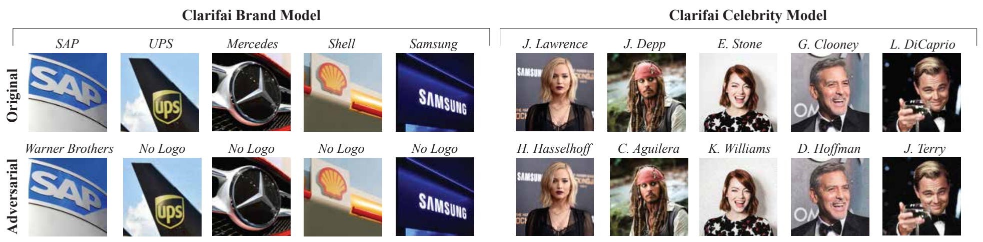


Figure 8: Adversarial examples generated by the Boundary Attack for two black-box models by Clarifai for brand-detection (left side) and celebrity detection (right side).

model correctly identifies the content. The adversarial criterion was misclassification, i.e. Clarifai should report a different brand / celebrity or None on the adversarially perturbed sample.

We show five samples for each model alongside the adversarial image generated by the Boundary Attack in Figure 8. We generally observed that the Clarifai models were more difficult to attack than ImageNet models like VGG-19: while for some samples we did succeed to find adversarial perturbations of the same order ($1e^{-7}$) as in section 3 (e.g. for *Shell* or *SAP*), most adversarial perturbations were on the order of $1e^{-2}$ to $1e^{-3}$ resulting in some slightly noticeable noise in some adversarial examples. Nonetheless, for most samples the original and the adversarial image are close to being perceptually indistinguishable.

6 Discussion & Outlook

In this paper we emphasised the importance of a mostly neglected category of adversarial attacks—*decision-based attacks*—that can find adversarial examples in models for which only the final decision can be observed. We argue that this category is important for three reasons: first, attacks in this class are highly relevant for many real-world deployed machine learning systems like autonomous cars for which the internal decision making process is unobservable. Second, attacks in this class do not rely on substitute models that are trained on similar data as the model to be attacked, thus making real-world applications much more straight-forward. Third, attacks in this class have the potential to be much more robust against common deceptions like gradient masking, intrinsic stochasticity or robust training.

We also introduced the first effective attack in this category that is applicable to general machine learning algorithms and complex natural datasets: the *Boundary Attack*. At its core the Boundary Attack follows the decision boundary between adversarial and non-adversarial samples using a very simple rejection sampling algorithm in conjunction with a simple proposal distribution and a dynamic step-size adjustment inspired by Trust Region methods. Its basic operating principle—starting from a large perturbation and successively reducing it—inverts the logic of essentially all previous adversarial attacks. Besides being surprisingly simple, the Boundary attack is also extremely flexible in terms of the possible adversarial criteria and performs on par with gradient-based attacks on standard computer vision tasks in terms of the size of minimal perturbations.

The mere fact that a simple constrained iid Gaussian distribution can serve as an effective proposal perturbation for each step of the Boundary attack is surprising and sheds light on the brittle information processing of current computer vision architectures. Nonetheless, there are many ways in which the Boundary attack can be made even more effective, in particular by learning a suitable proposal distribution for a given model or by conditioning the proposal distribution on the recent history of successful and unsuccessful proposals.

Decision-based attacks will be highly relevant to assess the robustness of machine learning models and to highlight the security risks of closed-source machine learning systems like autonomous cars. We hope that the Boundary attack will inspire future work in this area.

Acknowledgments

This work was supported by the Carl Zeiss Foundation (0563-2.8/558/3), the Bosch Forschungsstiftung (Stifterverband, T113/30057/17), the International Max Planck Research School for Intelligent Systems (IMPRS-IS), the German Research Foundation (DFG, CRC 1233, Robust Vision: Inference Principles and Neural Mechanisms) and the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior/Interior Business Center (DoI/IBC) contract number D16PC00003. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/IBC, or the U.S. Government.

References

Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 387–402. Springer, 2013.

Wieland Brendel and Matthias Bethge. Comment on "biologically inspired protection of deep networks from adversarial attacks". *CoRR*, abs/1704.01547, 2017. URL http://arxiv.org/abs/1704.01547.

Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. *CoRR*, abs/1608.04644, 2016a. URL http://arxiv.org/abs/1608.04644.

Nicholas Carlini and David A. Wagner. Defensive distillation is not robust to adversarial examples. *CoRR*, abs/1607.04311, 2016b. URL http://arxiv.org/abs/1607.04311.

Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. *CoRR*, abs/1708.03999, 2017. URL http://arxiv.org/abs/1708.03999.

Moustapha Cisse, Yossi Adi, Natalia Neverova, and Joseph Keshet. Houdini: Fooling deep structured prediction models. *CoRR*, abs/1707.05373, 2017. URL http://arxiv.org/abs/1707.05373.

Nilesh Dalvi, Pedro Domingos, Mausam, Sumit Sanghai, and Deepak Verma. Adversarial classification. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pp. 99–108, New York, NY, USA, 2004. ACM. ISBN 1-58113-888-1. doi: 10.1145/1014052.1014066. URL http://doi.acm.org/10.1145/1014052.1014066.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. IEEE, 2009.

Jamie Hayes and George Danezis. Machine learning as an adversarial service: Learning black-box adversarial examples. *CoRR*, abs/1708.05207, 2017. URL http://arxiv.org/abs/1708.05207.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Technical report, University of Toronto*, 2009.

Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *CoRR*, abs/1607.02533, 2016. URL http://arxiv.org/abs/1607.02533.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. *CoRR*, abs/1611.02770, 2016. URL `http://arxiv.org/abs/1611.02770`.

Daniel Lowd and Christopher Meek. Adversarial learning. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, pp. 641–647, New York, NY, USA, 2005. ACM. ISBN 1-59593-135-X. doi: 10.1145/1081870.1081950. URL `http://doi.acm.org/10.1145/1081870.1081950`.

Jiajun Lu, Theerasit Issaranon, and David A. Forsyth. Safetynet: Detecting and rejecting adversarial examples robustly. *CoRR*, abs/1704.00103, 2017. URL `http://arxiv.org/abs/1704.00103`.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. *CoRR*, abs/1511.04599, 2015. URL `http://arxiv.org/abs/1511.04599`.

Nina Narodytska and Shiva Prasad Kasiviswanathan. Simple black-box adversarial perturbations for deep networks. *CoRR*, abs/1612.06299, 2016. URL `http://arxiv.org/abs/1612.06299`.

Aran Nayebi and Surya Ganguli. Biologically inspired protection of deep networks from adversarial attacks. *CoRR*, abs/1703.09202, 2017. URL `http://arxiv.org/abs/1703.09202`.

Blaine Nelson, Benjamin I. P. Rubinstein, Ling Huang, Anthony D. Joseph, Steven J. Lee, Satish Rao, and J. D. Tygar. Query strategies for evading convex-inducing classifiers. *J. Mach. Learn. Res.*, 13:1293–1332, May 2012. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=2188385.2343688`.

Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. *CoRR*, abs/1511.07528, 2015. URL `http://arxiv.org/abs/1511.07528`.

Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pp. 582–597. IEEE, 2016.

Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pp. 506–519. ACM, 2017a.

Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, pp. 506–519, New York, NY, USA, 2017b. ACM. ISBN 978-1-4503-4944-4. doi: 10.1145/3052973.3053009. URL `http://doi.acm.org/10.1145/3052973.3053009`.

Jonas Rauber, Wieland Brendel, and Matthias Bethge. Foolbox v0.8.0: A python toolbox to benchmark the robustness of machine learning models. *CoRR*, abs/1707.04131, 2017. URL `http://arxiv.org/abs/1707.04131`.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL `http://arxiv.org/abs/1409.1556`.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013. URL `http://arxiv.org/abs/1312.6199`.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015. URL `http://arxiv.org/abs/1512.00567`.

Florian Tramer, Alexey Kurakin, Nicolas Papernot, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *CoRR*, abs/1705.07204, May 2017. URL `http://arxiv.org/abs/1705.07204`.

Towards the first adversarially robust neural network model on MNIST

The following 17 pages have been published as:

Lukas Schott*, Jonas Rauber*, Matthias Bethge, Wieland Brendel (2019). “Towards the first adversarially robust neural network model on MNIST”. *Seventh International Conference on Learning Representations (ICLR 2019).*

A summary of the motivation, results, and discussion can be found in section 2.5 on page 43. * joint first authorship

Abstract

Despite much effort, deep neural networks remain highly susceptible to tiny input perturbations and even for MNIST, one of the most common toy datasets in computer vision, no neural network model exists for which the adversarial perturbations are large and make semantic sense to humans. We show that the widely recognized and by far most successful defense by Madry et al. (1) overfits on the L_∞ metric (it’s highly susceptible to L_2 and L_0 perturbations), (2) classifies unrecognizable images with high certainty, (3) a simple defense based on binarization performs almost as well and (4) its adversarial perturbations make little sense to humans. These results suggest that MNIST is far from being solved in terms of adversarial robustness. We present a novel robust classification model that performs *analysis by synthesis* using learned class-conditional data distributions. We derive robustness guarantees and go to great length to empirically evaluate our model using maximally effective adversarial attacks by (a) applying decision-based, score-based, gradient-based and transfer-based attacks for several different L_p norms, (b) by designing a new attack that exploits the structure of our defended model and (c) by devising a novel decision-based attack that seeks to minimize the number of perturbed pixels (L_0). The results suggest that this approach yields state-of-the-art robustness on MNIST against L_0, L_2 and L_∞ perturbations and we demonstrate that most adversarial examples are strongly perturbed towards the perceptual boundary between the original and the adversarial class.

TOWARDS THE FIRST ADVERSARIALLY ROBUST NEURAL NETWORK MODEL ON MNIST

Lukas Schott[1-3*], Jonas Rauber[1-3*], Matthias Bethge[1,3,4†] & Wieland Brendel[1,3†]

[1]Centre for Integrative Neuroscience, University of Tübingen
[2]International Max Planck Research School for Intelligent Systems
[3]Bernstein Center for Computational Neuroscience Tübingen
[4]Max Planck Institute for Biological Cybernetics
[*]Joint first authors
[†]Joint senior authors
firstname.lastname@bethgelab.org

ABSTRACT

Despite much effort, deep neural networks remain highly susceptible to tiny input perturbations and even for MNIST, one of the most common toy datasets in computer vision, no neural network model exists for which adversarial perturbations are large and make semantic sense to humans. We show that even the widely recognized and by far most successful L_∞ defense by Madry et al. (1) has lower L_0 robustness than undefended networks and is still highly susceptible to L_2 perturbations, (2) classifies unrecognizable images with high certainty, (3) performs not much better than simple input binarization and (4) features adversarial perturbations that make little sense to humans. These results suggest that MNIST is far from being solved in terms of adversarial robustness. We present a novel robust classification model that performs *analysis by synthesis* using learned class-conditional data distributions. We derive bounds on the robustness and go to great length to empirically evaluate our model using maximally effective adversarial attacks by (a) applying decision-based, score-based, gradient-based and transfer-based attacks for several different L_p norms, (b) by designing a new attack that exploits the structure of our defended model and (c) by devising a novel decision-based attack that seeks to minimize the number of perturbed pixels (L_0). The results suggest that our approach yields state-of-the-art robustness on MNIST against L_0, L_2 and L_∞ perturbations and we demonstrate that most adversarial examples are strongly perturbed towards the perceptual boundary between the original and the adversarial class.

1 INTRODUCTION

Deep neural networks (DNNs) are strikingly susceptible to *minimal adversarial perturbations* (Szegedy et al., 2013), perturbations that are (almost) imperceptible to humans but which can switch the class prediction of DNNs to basically any desired target class.

One key problem in finding successful defenses is the difficulty of reliably evaluating model robustness. It has been shown time and again (Athalye et al., 2018; Athalye & Carlini, 2018; Brendel & Bethge, 2017) that basically all defenses previously proposed did not increase model robustness but prevented existing attacks from finding minimal adversarial examples, the most common reason being masking of the gradients on which most attacks rely. The few verifiable defenses can only guarantee robustness within a small linear regime around the data points (Hein & Andriushchenko, 2017; Raghunathan et al., 2018).

The only defense currently considered effective (Athalye et al., 2018) is a particular type of adversarial training (Madry et al., 2018). On MNIST, as of today this method is able to reach an accuracy of 88.79% for adversarial perturbations with an L_∞ norm bounded by $\epsilon = 0.3$ (Zheng et al., 2018). In other words, if we allow an attacker to perturb the brightness of each pixel by up to 0.3 (range $[0, 1]$), then he can only trick the model on $\approx 10\%$ of the samples. This is a great success, but does the model really learn more causal features to classify MNIST? We here demonstrate that this is not the case: For one, the defense by Madry et al. (SOTA on L_∞) has lower L_0 robustness than undefended networks and is still highly susceptible in the L_2 metric. Second, the robustness results by Madry et al. can also be achieved with a simple input quantization because of the binary nature of single pixels in MNIST (which are typically either completely black or white) (Schmidt et al., 2018). Third, it is straight-forward to find unrecognizable images that are classified as a digit with high certainty. Finally, the minimum adversarial examples we find for the defense by Madry et al. make little to no sense to humans.

Taken together, even MNIST cannot be considered solved with respect to adversarial robustness. By "solved" we mean a model that reaches at least 99% accuracy (see accuracy-vs-robustness trade-off (Tsipras et al., 2018; Bubeck et al., 2018)) and whose adversarial examples carry semantic meaning to humans (by which we mean that they start looking like samples that could belong to either class). Hence, despite the fact that MNIST is considered "too easy" by many and a mere toy example, finding adversarially robust models on MNIST is still an open problem.

A potential solution we explore in this paper is inspired by unrecognizable images (Nguyen et al., 2015) or *distal adversarials*. Distal adversarials are images that do not resemble images from the training set but which typically look like noise while still being classified by the model with high confidence. It seems difficult to prevent such images in feedforward networks as we have little control over how inputs are classified that are far outside of the training domain. In contrast, generative models can learn the distribution of their inputs and are thus able to gauge their confidence accordingly. By additionally learning the image distribution within each class we can check that the classification makes sense in terms of the image features being present in the input (e.g. an image of a bus should contain actual bus features). Following this line of thought from an information-theoretic perspective, one arrives at the well-known concept of Bayesian classifiers. We here introduce a fine-tuned variant based on variational autoencoders (Kingma & Welling, 2013) that combines robustness with high accuracy.

In summary, the contributions of this paper are as follows:

- We show that MNIST is unsolved from the point of adversarial robustness: the SOTA defense of Madry et al. (2018) is still highly vulnerable to tiny perturbations that are meaningless to humans.
- We introduce a new robust classification model and derive instance-specific robustness guarantees.
- We develop a strong attack that leverages the generative structure of our classification model.
- We introduce a novel decision-based attack that minimizes L_0.
- We perform an extensive evaluation of our defense across many attacks to show that it surpasses SOTA on L_0, L_2 and L_∞ and features many adversarials that carry semantic meaning to humans.

We have evaluated the proposed defense to the best of our knowledge, but we are aware of the (currently unavoidable) limitations of evaluating robustness. We will release the model architecture and trained weights as a friendly invitation to fellow researchers to evaluate our model independently.

2 Related Work

The many defenses against adversarial attacks can roughly be subdivided into four categories:

- **Adversarial training:** The training data is augmented with adversarial examples to make models more robust (Madry et al., 2018; Szegedy et al., 2013; Tramèr et al., 2017; Ilyas et al., 2017).
- **Manifold projections:** An input sample is projected onto a learned data manifold (Samangouei et al., 2018; Ilyas et al., 2017; Shen et al., 2017; Song et al., 2018).
- **Stochasticity:** Certain inputs or hidden activations are shuffled or randomized (Prakash et al., 2018; Dhillon et al., 2018; Xie et al., 2018).
- **Preprocessing:** Inputs or hidden activations are quantized, projected into a different representation or are otherwise preprocessed (Buckman et al., 2018; Guo et al., 2018; Kabilan et al., 2018).

There has been much work showing that basically all defenses suggested so far in the literature do not substantially increase robustness over undefended neural networks (Athalye et al., 2018; Brendel &

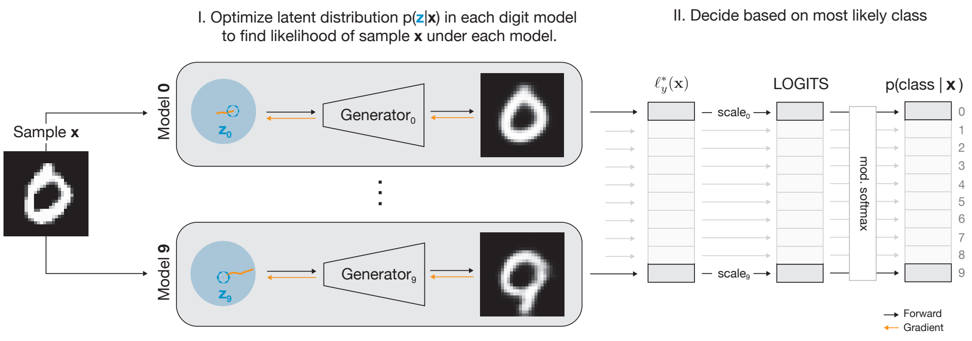

I. Optimize latent distribution p(z|x) in each digit model to find likelihood of sample x under each model.
II. Decide based on most likely class
Sample x
Model 0
Model 9
Generator₀
Generator₉
z₀
z₉
ℓ*y(x)
LOGITS
p(class | x)
scale₀
scale₉
mod. softmax
0
1
2
3
4
5
6
7
8
9
Forward
Gradient

Figure 1: Overview over model architecture. In a nutshell: I) for each sample $\mathbf{x}$ we compute a lower bound on the log-likelihood (ELBO) under each class using gradient descent in the latent space. II) A class-dependent scalar weighting of the class-conditional ELBOs forms the final class prediction.

Bethge, 2017). The only widely accepted exception according to Athalye et al. (2018) is the defense by Madry et al. (2018) which is based on data augmentation with adversarials found by iterative projected gradient descent with random starting points. However, as we see in the results section, this defense is limited to the metric it is trained on (L_∞) and it is straight-forward to generate small adversarial perturbations that carry little semantic meaning for humans.

Some other defenses have been based on generative models. Typically these defenses use the generative model to project onto the (learned) manifold of "natural" inputs. This includes in particular DefenseGAN (Samangouei et al., 2018), Adversarial Perturbation Elimination GAN (Shen et al., 2017) and Robust Manifold Defense (Ilyas et al., 2017), all of which project an image onto the manifold defined by a generator network G. The generated image is then classified by a discriminator in the usual way. A similar idea is used by PixelDefend (Song et al., 2018) which uses an autoregressive probabilistic method to learn the data manifold. Other ideas in similar directions include the use of denoising autoencoders (Liao et al., 2017) as well as MagNets (Meng & Chen, 2017), which projects or rejects inputs depending on their distance to the data manifold. All of these proposed defenses except for the defense by Ilyas et al. (2017) have been tested by Athalye et al. (2018); Athalye & Carlini (2018); Carlini & Wagner (2017) and others, and shown to be ineffective. It is straight-forward to understand why: For one, many adversarials still look like normal data points to humans. Second, the classifier on top of the projected image is as vulnerable to adversarial examples as before. Hence, for any data set with a natural amount of variation there will almost always be a certain perturbation against which the classifier is vulnerable and which can be induced by the right inputs.

We here follow a different approach by modeling the input distribution within each class (instead of modeling a single distribution for the complete data), and by classifying a new sample according to the class under which it has the highest likelihood. This approach, commonly referred to as a Bayesian classifier, gets away without any additional and vulnerable classifier. A very different but related approach is the work by George et al. (2017) which suggested a generative compositional model of digits to solve cluttered digit scenes like Captchas (adversarial robustness was not evaluated).

3 Model Description

Intuitively, we want to learn a causal model of the inputs (Schölkopf, 2017). Consider a cat: we want a model to learn that cats have four legs and two pointed ears, and then use this model to check whether a given input can be generated with these features. This intuition can be formalized as follows. Let $(\mathbf{x}, y)$ with $\mathbf{x} \in \mathbb{R}^N$ be an input-label datum. Instead of directly learning a posterior $p(y|\mathbf{x})$ from inputs to labels we now learn generative distributions $p(\mathbf{x}|y)$ and classify new inputs using Bayes formula,

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})} \propto p(\mathbf{x}|y)p(y). \tag{1}$$

The label distribution $p(y)$ can be estimated from the training data. To learn the class-conditional sample distributions $p(\mathbf{x}|y)$ we use variational autoencoders (VAEs) (Kingma & Welling, 2013). VAEs estimate the log-likelihood $\log p(\mathbf{x})$ by learning a probabilistic generative model $p_\theta(\mathbf{x}|\mathbf{z})$

with latent variables $\mathbf{z} \sim p(\mathbf{z})$ and parameters θ (see Appendix A.3 for the full derivation). For class-conditional VAEs we can derive a lower bound on the log-likelihood $\log p(\mathbf{x}|y)$ as

$$\log p(\mathbf{x}|y) \geq \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x},y)} \left[\log p_\theta(\mathbf{x}|\mathbf{z}, y)\right] - \mathcal{D}_{KL} \left[q_\phi(\mathbf{z}|\mathbf{x}, y)||p(\mathbf{z})\right] =: \ell_y(\mathbf{x}), \tag{2}$$

where $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbb{1})$ is a simple normal prior and $q_\phi(\mathbf{z}|\mathbf{x}, y)$ is the variational posterior with parameters ϕ. The first term on the RHS is basically a reconstruction error while the second term on the RHS is the mismatch between the variational and the true posterior. The term on the RHS is the so-called evidence lower bound (ELBO) on the log-likelihood (Kingma & Welling, 2013). We implement the conditional distributions $p_\theta(\mathbf{x}|\mathbf{z}, y)$ and $q_\phi(\mathbf{z}|\mathbf{x}, y)$ as normal distributions for which the means are parametrized as DNNs (all details and hyperparameters are reported in Appendix A.7).

Our *Analysis by Synthesis* model (ABS) is illustrated in Figure 1. It combines several elements to simultaneously achieve high accuracy and robustness against adversarial perturbations:

- **Class-conditional distributions:** For each class y we train a variational autoencoder VAE$_y$ on the samples of class y to learn the class-conditional distribution $p(\mathbf{x}|y)$. This allows us to estimate a lower bound $\ell_y(\mathbf{x})$ on the log-likelihood of sample $\mathbf{x}$ under each class y.
- **Optimization-based inference:** The variational inference $q_\phi(\mathbf{z}|\mathbf{x}, y)$ is itself a neural network susceptible to adversarial perturbations. We therefore only use variational inference during training and perform “exact” inference over $p_\theta(\mathbf{x}|\mathbf{z}, y)$ during evaluation. This “exact” inference is implemented using gradient descent in the latent space (with fixed posterior width) to find the optimal $\mathbf{z}_y$ which maximizes the lower bound on the log-likelihood for each class:
$$\ell^*_y(\mathbf{x}) = \max_{\mathbf{z}} \; \log p_\theta(\mathbf{x}|\mathbf{z}, y) - \mathcal{D}_{KL} \left[\mathcal{N}(\mathbf{z}, \sigma_q \mathbb{1})||\mathcal{N}(\mathbf{0}, \mathbb{1})\right]. \tag{3}$$
Note that we replaced the expectation in equation 2 with a maximum likelihood sample to avoid stochastic sampling and to simplify optimization. To avoid local minima we evaluate 8000 random points in the latent space of each VAE, from which we pick the best as a starting point for a gradient descent with 50 iterations using the Adam optimizer (Kingma & Ba, 2014).
- **Classification and confidence:** Finally, to perform the actual classification, we scale all $\ell^*_y(\mathbf{x})$ with a factor α, exponentiate, add an offset η and divide by the total evidence (like in a softmax),
$$p(y|\mathbf{x}) = \left(e^{\alpha \ell^*_y(\mathbf{x})} + \eta\right) / \sum\nolimits_c \left(e^{\alpha \ell^*_c(\mathbf{x})} + \eta\right). \tag{4}$$
We introduced η for the following reason: even on points far outside the data domain, where all likelihoods $q(\mathbf{x}, y) = e^{\alpha \ell^*_y(\mathbf{x})} + \eta$ are small, the standard softmax ($\eta = 0$) can lead to sharp posteriors $p(y|\mathbf{x})$ with high confidence scores for one class. This behavior is in stark contrast to humans, who would report a uniform distribution over classes for unrecognizable images. To model this behavior we set $\eta > 0$: in this case the posterior $p(y|\mathbf{x})$ converges to a uniform distribution whenever the maximum $q(\mathbf{x}, y)$ gets small relative to η . We chose η such that the median confidence $p(y|\mathbf{x})$ is 0.9 for the predicted class on clean test samples. Furthermore, for a better comparison with cross-entropy trained networks, the scale α is trained to minimize the cross-entropy loss. We also tested this graded softmax in standard feedforward CNNs but did not find any improvement with respect to unrecognizable images.
- **Binarization (*Binary ABS* only):** The pixel intensities of MNIST images are almost binary. We exploit this by projecting the intensity b of each pixel to 0 if $b < 0.5$ or 1 if $b \geq 0.5$ during testing.
- **Discriminative finetuning (*Binary ABS* only):** To improve the accuracy of the *Binary ABS* model we multiply $\ell^*_y(\mathbf{x})$ with an additional class-dependent scalar γ_y. The scalars are learned discriminatively (see A.7) and reach values in the range $\gamma_y \in [0.96, 1.06]$ for all classes y.

On important ingredient for the robustness of the ABS model is the Gaussian posterior in the reconstruction term which ensures that small changes in the input (in terms of L2) can only entail small changes to the posterior likelihood and thus to the model decision.

4 Tight Estimates of the Lower Bound for Adversarial Examples

The decision of the model depends on the likelihood in each class, which for clean samples is mostly dominated by the posterior likelihood $p(\mathbf{x}|\mathbf{z})$. Because we chose this posterior to be Gaussian, the class-conditional likelihoods can only change gracefully with changes in $\mathbf{x}$, a property which allows us to derive lower bounds on the model robustness. To see this, note that equation 3 can be written as,

$$\ell_c^*(\mathbf{x}) = \max_{\mathbf{z}} \; -\mathcal{D}_{KL}\left[\mathcal{N}(\mathbf{z}, \sigma_q \mathbb{1}) || \mathcal{N}(\mathbf{0}, \mathbb{1})\right] - \frac{1}{2\sigma^2} \left\|\mathbf{G}_c(\mathbf{z}) - \mathbf{x}\right\|_2^2 + C, \tag{5}$$

where we absorbed the normalization constants of $p(\mathbf{x}|\mathbf{z})$ into C and $\mathbf{G}_c(\mathbf{z})$ is the mean of $p(\mathbf{x}|\mathbf{z}, c)$. Let y be the ground-truth class and let $\mathbf{z}_{\mathbf{x}}^*$ be the optimal latent for the clean sample $\mathbf{x}$ for class y. We can then estimate a lower bound on $\ell_y^*(\mathbf{x} + \boldsymbol{\delta})$ for a perturbation $\boldsymbol{\delta}$ with size $\epsilon = \|\boldsymbol{\delta}\|_2$ (see derivation in Appendix A.4),

$$\ell_y^*(\mathbf{x} + \boldsymbol{\delta}) \geq \ell_y^*(\mathbf{x}) - \frac{1}{\sigma^2}\epsilon \left\|\mathbf{G}_y(\mathbf{z}_{\mathbf{x}}^*) - \mathbf{x}\right\|_2 - \frac{1}{2\sigma^2}\epsilon^2 + C. \tag{6}$$

Likewise, we can derive an upper bound of $\ell_y^*(\mathbf{x} + \boldsymbol{\delta})$ for all other classes $c \neq y$ (see Appendix A.5),

$$\ell_c^*(\mathbf{x} + \boldsymbol{\delta}) \leq -\mathcal{D}_{KL}\left[\mathcal{N}(\mathbf{0}, \sigma_q \mathbb{1}) || \mathcal{N}(\mathbf{0}, \mathbb{1})\right] + C - \begin{cases} \frac{1}{2\sigma^2}(d_c - \epsilon)^2 & \text{if } d_c \geq \epsilon \\ 0 & \text{else} \end{cases}. \tag{7}$$

for $d_c = \min_z \left\|\mathbf{G}_c(\mathbf{z}) - \mathbf{x}\right\|_2$. Now we can find ϵ for a given image $\mathbf{x}$ by equating $(7) = (6)$,

$$\epsilon_x = \min_{c \neq y} \max \left\{0, \frac{d_c + \ell_y^*(\mathbf{x}) - \mathcal{D}_{KL}\left[\mathcal{N}(\mathbf{0}, \sigma_q \mathbb{1}) || \mathcal{N}(\mathbf{0}, \mathbb{1})\right]}{2(d_c + \left\|\mathbf{G}_y(\mathbf{z}_{\mathbf{x}}^*) - \mathbf{x}\right\|_2)}\right\}. \tag{8}$$

Note that one assumption we make is that we can find the global minimum of $\left\|\mathbf{G}_c(\mathbf{z}) - \mathbf{x}\right\|_2^2$. In practice we generally find a very tight estimate of the global minimum (and thus the lower bound) because we optimize in a smooth and low-dimensional space and because we perform an additional brute-force sampling step. We provide quantitative values for ϵ in section 7.

5 Adversarial Attacks

Reliably evaluating model robustness is difficult because each attack only provides an upper bound on the size of the adversarial perturbations (Uesato et al., 2018). To make this bound as tight as possible we apply many different attacks and choose the best one for each sample and model combination (using the implementations in Foolbox v1.3 (Rauber et al., 2017) which often perform internal hyperparameter optimization). We also created a novel decision-based L_0 attack as well as a customized attack that specifically exploits the structure of our model. Nevertheless, we cannot rule out that more effective attacks exist and we will release the trained model for future testing.

Latent Descent attack This novel attack exploits the structure of the ABS model. Let $\mathbf{x}_t$ be the perturbed sample $\mathbf{x}$ in iteration t. We perform variational inference $p(\mathbf{z}|\mathbf{x}_t, y) = \mathcal{N}(\boldsymbol{\mu}_y(\mathbf{x}_t), \sigma_q \boldsymbol{I})$ to find the most likely class $\tilde{y}$ that is different from the ground-truth class. We then make a step towards the maximum likelihood posterior $p(\mathbf{x}|\mathbf{z}, \tilde{y})$ of that class which we denote as $\tilde{\mathbf{x}}_{\tilde{y}}$,

$$\mathbf{x}_t \mapsto (1 - \epsilon)\mathbf{x}_t + \epsilon \tilde{\mathbf{x}}_{\tilde{y}}. \tag{9}$$

We choose $\epsilon = 10^{-2}$ and iterate until we find an adversarial. For a more precise estimate we perform a subsequent binary search of 10 steps within the last ϵ interval. Finally, we perform another binary search between the adversarial and the original image to reduce the perturbation as much as possible.

Decision-based attacks We use several decision-based attacks because they do not rely on gradient information and are thus insensitive to gradient masking or missing gradients. In particular, we apply the *Boundary Attack* (Brendel et al., 2018), which is competitive with gradient-based attacks in minimizing the L_2 norm, and introduce the *Pointwise Attack*, a novel decision-based attack that greedily minimizes the L_0 norm. It first adds salt-and-pepper noise until the image is misclassified and then repeatedly iterates over all perturbed pixels, resetting them to the clean image if the perturbed image stays adversarial. The attack ends when no pixel can be reset anymore. We provide an implementation of the attack in Foolbox (Rauber et al., 2017). Finally, we apply two simple noise attacks, the *Gaussian Noise* attack and the *Salt&Pepper Noise* attack as baselines.

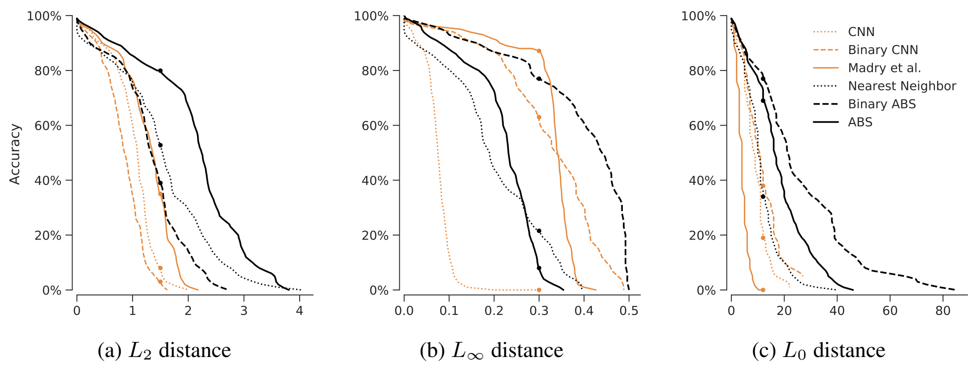


Figure 2: Accuracy-distortion plots for each distance metric and all models. In (b) we see that a threshold at 0.3 favors Madry et al. while a threshold of 0.35 would have favored the Binary ABS.

Transfer-based attacks Transfer attacks also don't rely on gradients of the target model but instead compute them on a substitute: given an input $\mathbf{x}$ we first compute adversarial perturbations $\boldsymbol{\delta}$ on the substitute using different gradient-based attacks (L_2 and L_∞ Basic Iterative Method (BIM), Fast Gradient Sign Method (FGSM) and L_2 Fast Gradient Method) and then perform a line search to find the smallest ϵ for which $\mathbf{x} + \epsilon\boldsymbol{\delta}$ (clipped to the range [0, 1]) is still an adversarial for the target model.

Gradient-based attacks We apply the Momentum Iterative Method (MIM) (Dong et al., 2017) that won the NIPS 2017 adversarial attack challenge, the Basic Iterative Method (BIM) (Kurakin et al., 2016) (also known as Projected Gradient Descent (PGD))—for both the L_2 and the L_∞ norm—as well as the Fast Gradient Sign Method (FGSM) (Goodfellow et al., 2014) and its L_2 variant, the Fast Gradient Method (FGM). For models with input binarization (Binary CNN, Binary ABS), we obtain gradients using the straight-through estimator (Bengio et al., 2013).

Score-based attacks We additionally run all attacks listed under *Gradient-based attacks* using numerically estimated gradients (possible for all models). We use a simple coordinate-wise finite difference method (NES estimates (Ilyas et al., 2018) performed comparable or worse) and repeat the attacks with different values for the step size of the gradient estimator.

Postprocessing (binary models only) For models with input binarization (sec. 6) we postprocess all adversarials by setting pixel intensities either to the corresponding value of the clean image or the binarization threshold (0.5). This reduces the perturbation size without changing model decisions.

6 Experiments

We compare our ABS model as well as two ablations—ABS with input binarization during test time (Binary ABS) and a CNN with input binarization during train and test time (Binary CNN)—against three other models: the SOTA L_∞ defense (Madry et al., 2018)[1], a Nearest Neighbour (NN) model (as a somewhat robust but not accurate baseline) and a vanilla CNN (as an accurate but not robust baseline), see Appendix A.7. We run all attacks (see sec. 5) against all applicable models.

For each model and L_p norm, we show how the accuracy of the models decreases with increasing adversarial perturbation size (Figure 2) and report two metrics: the median adversarial distance (Table 1, left values) and the model's accuracy against bounded adversarial perturbations (Table 1, right values). The median of the perturbation sizes (Table 1, left values) is robust to outliers and summarizes most of the distributions quite well. It represents the perturbation size for which the particular model achieves 50% accuracy and does not require the choice of a threshold. Clean samples that are already misclassified are counted as adversarials with a perturbation size equal to 0, failed attacks as ∞. The commonly reported model accuracy on bounded adversarial perturbations, on the other hand, requires a metric-specific threshold that can bias the results. We still report it (Table 1, right values) for completeness and set $\epsilon_{L_2} = 1.5$, $\epsilon_{L_\infty} = 0.3$ and $\epsilon_{L_0} = 12$ as thresholds.

[1]We used the trained model provided by the authors: https://github.com/MadryLab/mnist_challenge

	CNN	Binary CNN	Nearest Neighbor	Madry et al.	Binary ABS	ABS
Clean	99.1%	98.5%	96.9%	98.8%	99.0%	99.0%
L_2-metric ($\epsilon = 1.5$)						
Transfer Attacks	1.1 / 14%	1.4 / 38%	5.4 / 90%	3.7 / 94%	2.5 / 86%	4.6 / 94%
Gaussian Noise	5.2 / 96%	3.4 / 92%	∞ / 91%	5.4 / 96%	5.6 / 89%	10.9 / 98%
Boundary Attack	1.2 / 21%	3.3 / 84%	2.9 / 73%	1.4 / 37%	6.0 / 91%	2.6 / 83%
Pointwise Attack	3.4 / 91%	1.9 / 71%	3.5 / 89%	1.9 / 71%	3.1 / 86%	4.6 / 94%
FGM	1.4 / 48%	1.4 / 50%		∞ / 96%		
FGM w/ GE	1.4 / 42%	2.8 / 51%	3.7 / 79%	∞ / 88%	1.9 / 68%	3.5 / 89%
DeepFool	1.2 / 18%	1.0 / 11%		9.0 / 91%		
DeepFool w/ GE	1.3 / 30%	0.9 / 5%	1.6 / 55%	5.1 / 90%	1.4 / 41%	2.4 / 83%
L2 BIM	1.1 / 13%	1.0 / 11%		4.8 / 88%		
L2 BIM w/ GE	1.1 / 37%	∞ / 50%	1.7 / 62%	3.4 / 88%	1.6 / 63%	3.1 / 87%
Latent Descent Attack					2.6 / 97%	2.7 / 85%
All L_2 Attacks	1.1 / 8%	0.9 / 3%	1.5 / 53%	1.4 / 35%	1.3 / 39%	**2.3** / 80%
L_∞-metric ($\epsilon = 0.3$)						
Transfer Attacks	0.08 / 0%	0.44 / 85%	0.42 / 78%	0.39 / 92%	0.49 / 88%	0.34 / 73%
FGSM	0.10 / 4%	0.43 / 77%		0.45 / 93%		
FGSM w/ GE	0.10 / 21%	0.42 / 71%	0.38 / 68%	0.47 / 89%	0.49 / 85%	0.27 / 34%
L_∞ DeepFool	0.08 / 0%	0.38 / 74%		0.42 / 90%		
L_∞ DeepFool w/ GE	0.09 / 0%	0.37 / 67%	0.21 / 26%	0.53 / 90%	0.46 / 78%	0.27 / 39%
BIM	0.08 / 0%	0.36 / 70%		0.36 / 90%		
BIM w/ GE	0.08 / 37%	∞ / 70%	0.25 / 43%	0.46 / 89%	0.49 / 86%	0.25 / 13%
MIM	0.08 / 0%	0.37 / 71%		0.34 / 90%		
MIM w/ GE	0.09 / 36%	∞ / 69%	0.19 / 26%	0.36 / 89%	0.46 / 85%	0.26 / 17%
All L_∞ Attacks	0.08 / 0%	0.34 / 64%	0.19 / 22%	0.34 / 88%	**0.44** / 77%	0.23 / 8%
L_0-metric ($\epsilon = 12$)						
Salt&Pepper Noise	44.0 / 91%	44.0 / 88%	161.0/ 88%	13.5 / 56%	146.0/ 94%	165.0/ 94%
Pointwise Attack 10x	9.0 / 19%	11.0 / 39%	10.0 / 34%	4.0 / 0%	22.0 / 77%	16.5 / 69%
All L_0 Attacks	9.0 / 19%	11.0 / 38%	10.0 / 34%	4.0 / 0%	**21.5** / 77%	16.5 / 69%

Table 1: Results for different models, adversarial attacks and distance metrics. Each entry shows the median adversarial distance across all samples (left value, black) as well as the model's accuracy against adversarial perturbations bounded by the thresholds $\epsilon_{L_2} = 1.5$, $\epsilon_{L_\infty} = 0.3$ and $\epsilon_{L_0} = 12$ (right value, gray). *"w/ GE"* indicates attacks that use numerical gradient estimation.

7 RESULTS

Minimal Adversarials Our robustness evaluation results of all models are reported in Table 1 and Figure 2. All models except the Nearest Neighbour classifier perform close to 99% accuracy on clean test samples. We report results for three different norms: L_2, L_∞ and L_0.

- For L_2 our ABS model outperforms all other models by a large margin.
- For L_∞, our Binary ABS model is state-of-the-art in terms of median perturbation size. In terms of accuracy (perturbations < 0.3), Madry et al. seems more robust. However, as revealed by the accuracy-distortion curves in Figure 2, this is an artifact of the specific threshold (Madry et al. is optimized for 0.3). A slightly larger one (e.g. 0.35) would strongly favor the Binary ABS model.
- For L_0, both ABS and Binary ABS are much more robust than all other models. Interestingly, the model by Madry et al. is the least robust, even less than the baseline CNN.

In Figure 3 we show adversarial examples. For each sample we show the minimally perturbed L_2 adversarial found by any attack. Adversarials for the baseline CNN and the Binary CNN are almost

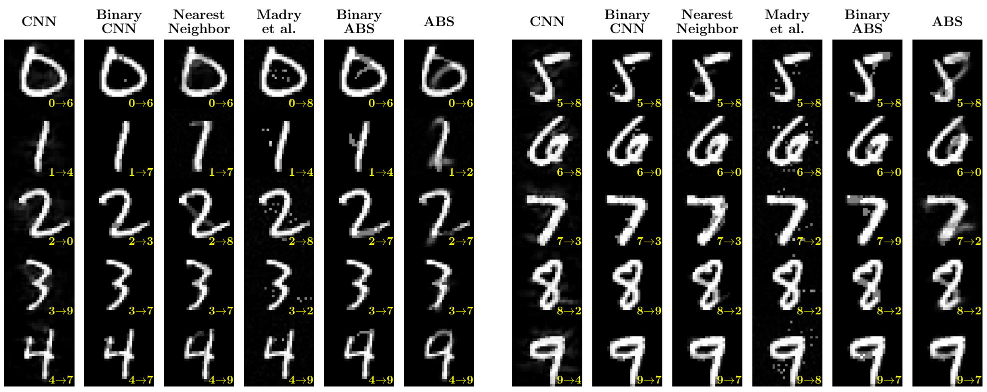


Figure 3: Adversarial examples for the ABS models are perceptually meaningful: For each sample (randomly chosen from each class) we show the minimally perturbed L_2 adversarial found by any attack. Our ABS models have clearly visible and often semantically meaningful adversarials. Madry et al. requires perturbations that are clearly visible, but their semantics are less clear.

imperceptible. The Nearest Neighbour model, almost by design, exposes (some) adversarials that interpolate between two numbers. The model by Madry et al. requires perturbations that are clearly visible but make little semantic sense to humans. Finally, adversarials generated for the ABS models are semantically meaningful for humans and are sitting close to the perceptual boundary between the original and the adversarial class. For a more thorough comparison see appendix Figures 5, 6 and 7.

Lower bounds on Robustness For the ABS models and the L_2 metric we estimate a lower bound of the robustness. The lower bound for the mean perturbation[2] for the MNIST test set is $\epsilon = 0.690 \pm 0.005$ for the ABS and $\epsilon = 0.601 \pm 0.005$ for the binary ABS. We estimated the error by using different random seeds for our optimization procedure and standard error propagation over 10 runs. With adversarial training Hein & Andriushchenko (2017) achieve a mean L_2 robustness guarantee of $\epsilon = 0.48$ while reaching 99% accuracy. In the L_{inf} metric we find a median robustness of 0.06.

Distal Adversarials We probe the behavior of CNN, Madry et al. and our ABS model outside the data distribution. We start from random noise images and perform gradient ascent to maximize the output probability of a fixed label until $p(y|\mathbf{x}) \geq 0.9$ (as computed by the modified softmax from equation (8)). The results are visualized in Figure 4. Standard CNNs and Madry et al. provide high confidence class probabilities for unrecognizable images. Our ABS model does not provide high confidence predictions in out-of-distribution regions.

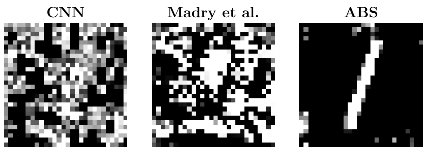


Figure 4: Images of ones classified with a probability above 90%.

8 Discussion & Conclusion

In this paper we demonstrated that, despite years of work, we as a community failed to create neural networks that can be considered robust on MNIST from the point of human perception. In particular, we showed that even today's best defense is susceptible to small adversarial perturbations that make little to no semantic sense to humans. We presented a new approach based on *analysis by synthesis* that seeks to explain its inference by means of the actual image features. We performed an extensive analysis to show that minimal adversarial perturbations in this model are large across all tested L_p norms and semantically meaningful to humans. Note that our architecture derives its robustness from its design and does not require any additionally training with adversarial examples.

We acknowledge that it is not easy to reliably evaluate a model's adversarial robustness and most defenses proposed in the literature have later been shown to be ineffective. In particular, the structure

[2]The mean instead of the median is reported to allow for a comparison with (Hein & Andriushchenko, 2017).

of the ABS model prevents the computation of gradients which might give the model an unfair advantage. We put a lot of effort into an extensive evaluation of adversarial robustness using a large collection of powerful attacks, including one specifically designed to be particularly effective against the ABS model (the *Latent Descent* attack), and we will release the model architecture and trained weights as a friendly invitation to fellow researchers to evaluate our model.

Looking at the results of individual attacks (Table 1) we find that there is no single attack that works best on all models, thus highlighting the importance for a broad range of attacks. Without the Boundary Attack, for example, Madry et al. would have looked more robust to L_2 adversarials than it is. For similar reasons Figure 6b of Madry et al. (2018) reports a median L_2 perturbation size larger than 5, compared to the 1.4 achieved by the Boundary Attack. Moreover,the combination of all attacks of one metric (*All L_2 / L_∞ / L_0 Attacks*) is often better than any individual attack, indicating that different attacks are optimal on different samples.

Our conceptual implementation of the ABS model with one VAE per class neither scales efficiently to more classes nor to more complex datasets (a preliminary experiment on CIFAR10 provided only 54% test accuracy). However, first experiments on two class CIFAR indicate that the proposed model is also robust on CIFAR (we reach a median L2 robustness of 2.6 compared to 0.8 for a vanilla CNN, see Appendix A.1) for details). To increase the accuracy, there are many ways in which the ABS model can be improved, ranging from better and faster generative models (e.g. flow-based) to better training procedures.

In a nutshell, we demonstrated that MNIST is still not solved from the point of adversarial robustness and showed that our novel approach based on analysis by synthesis has great potential to reduce the vulnerability against adversarial attacks and to align machine perception with human perception.

Acknowledgments

This work has been funded, in part, by the German Federal Ministry of Education and Research (BMBF) through the Bernstein Computational Neuroscience Program Tübingen (FKZ: 01GQ1002) as well as the German Research Foundation (DFG CRC 1233 on “Robust Vision”). The authors thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting L.S. and J.R.; J.R. acknowledges support by the Bosch Forschungsstiftung (Stifterverband, T113/30057/17); W.B. was supported by the Carl Zeiss Foundation (0563-2.8/558/3); M.B. acknowledges support by the Centre for Integrative Neuroscience Tübingen (EXC 307); W.B. and M.B. were supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior / Interior Business Center (DoI/IBC) contract number D16PC00003.

References

Anish Athalye and Nicholas Carlini. On the robustness of the cvpr 2018 white-box adversarial example defenses. *arXiv preprint arXiv:1804.03286*, 2018.

Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

W. Brendel and M. Bethge. Comment on “biologically inspired protection of deep networks from adversarial attacks”. *arXiv preprint arXiv:1704.01547*, 2017.

Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=SyZI0GWCZ`.

Sébastien Bubeck, Eric Price, and Ilya Razenshteyn. Adversarial examples from computational constraints. *arXiv preprint arXiv:1805.10204*, 2018.

Jacob Buckman, Aurko Roy, Colin Raffel, and Ian Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=S18Su--CW`.

Nicholas Carlini and David Wagner. Magnet and" efficient defenses against adversarial attacks" are not robust to adversarial examples. *arXiv preprint arXiv:1711.08478*, 2017.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Guneet S. Dhillon, Kamyar Azizzadenesheli, Jeremy D. Bernstein, Jean Kossaifi, Aran Khanna, Zachary C. Lipton, and Animashree Anandkumar. Stochastic activation pruning for robust adversarial defense. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=H1uR4GZRZ`.

Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Xiaolin Hu, Jianguo Li, and Jun Zhu. Boosting adversarial attacks with momentum. arxiv preprint. *arXiv preprint arXiv:1710.06081*, 2017.

Dileep George, Wolfgang Lehrach, Ken Kansky, Miguel Lázaro-Gredilla, Christopher Laan, Bhaskara Marthi, Xinghua Lou, Zhaoshi Meng, Yi Liu, Huayan Wang, Alex Lavin, and D. Scott Phoenix. A generative vision model that trains with high data efficiency and breaks text-based captchas. *Science*, 358(6368), 2017. ISSN 0036-8075. doi: 10.1126/science.aag2612. URL `http://science.sciencemag.org/content/358/6368/eaag2612`.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

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

Matthias Hein and Maksym Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. In *Advances in Neural Information Processing Systems 30*, pp. 2266–2276. Curran Associates, Inc., 2017.

Andrew Ilyas, Ajil Jalal, Eirini Asteri, Constantinos Daskalakis, and Alexandros G Dimakis. The robust manifold defense: Adversarial training using generative models. *arXiv preprint arXiv:1712.09196*, 2017.

Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. *arXiv preprint arXiv:1804.08598*, 2018.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Vishaal Munusamy Kabilan, Brandon Morris, and Anh Nguyen. Vectordefense: Vectorization as a defense to adversarial examples. *arXiv preprint arXiv:1804.08529*, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

Fangzhou Liao, Ming Liang, Yinpeng Dong, Tianyu Pang, Jun Zhu, and Xiaolin Hu. Defense against adversarial attacks using high-level representation guided denoiser. *arXiv preprint arXiv:1712.02976*, 2017.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rJzIBfZAb`.

Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 135–147. ACM, 2017.

Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

Aaditya Prakash, Nick Moran, Solomon Garber, Antonella DiLillo, and James Storer. Deflecting adversarial attacks with pixel deflection. *arXiv preprint arXiv:1801.08926*, 2018.

Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=Bys4ob-Rb`.

Jonas Rauber, Wieland Brendel, and Matthias Bethge. Foolbox: A python toolbox to benchmark the robustness of machine learning models. *arXiv preprint arXiv:1707.04131*, 2017. URL `http://arxiv.org/abs/1707.04131`.

Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-GAN: Protecting classifiers against adversarial attacks using generative models. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=BkJ3ibb0-`.

Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. Adversarially robust generalization requires more data. *CoRR*, abs/1804.11285, 2018. URL `http://arxiv.org/abs/1804.11285`.

Bernhard Schölkopf. Causal learning, 2017. URL `https://icml.cc/Conferences/2017/Schedule?showEvent=931`. Thirty-fourth International Conference on Machine Learning.

Shiwei Shen, Guoqing Jin, Ke Gao, and Yongdong Zhang. Ape-gan: Adversarial perturbation elimination with gan. *arXiv preprint arXiv:1707.05474*, 2017.

Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate 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`.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. There is no free lunch in adversarial robustness (but there are unexpected benefits). *arXiv preprint arXiv:1805.12152*, 2018.

Jonathan Uesato, Brendan O'Donoghue, Pushmeet Kohli, and Aaron van den Oord. Adversarial risk and the dangers of evaluating against weak attacks. In *Proceedings of the 35th International Conference on Machine Learning*, 2018. URL `http://proceedings.mlr.press/v80/uesato18a.html`.

Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=Sk9yuql0Z`.

Tianhang Zheng, Changyou Chen, and Kui Ren. Distributionally adversarial attack. *arXiv preprint arXiv:1808.05537*, 2018.

A Appendix

A.1 Two Class CIFAR

We estimate the robustness of our ABS model on two class CIFAR (airplane vs. automobile). Preliminary results suggest that our robustness is not limited to MNIST.

In order to adapt to CIFAR, we modified the ABS slightly by modifying encoder and decoder to fit (32x32x3) CIFAR images. We also increased the number of dimensions in the latent space form 8 to 20.

Model	CNN	ABS
Accuracy	97.1%	89.7%
Median L_2 distance	0.8 (with BIM)	2.5 (with Latent Descent attack)

Table 2: Accuracy and estimated robustness on two class CIFAR.

A.2 Figures

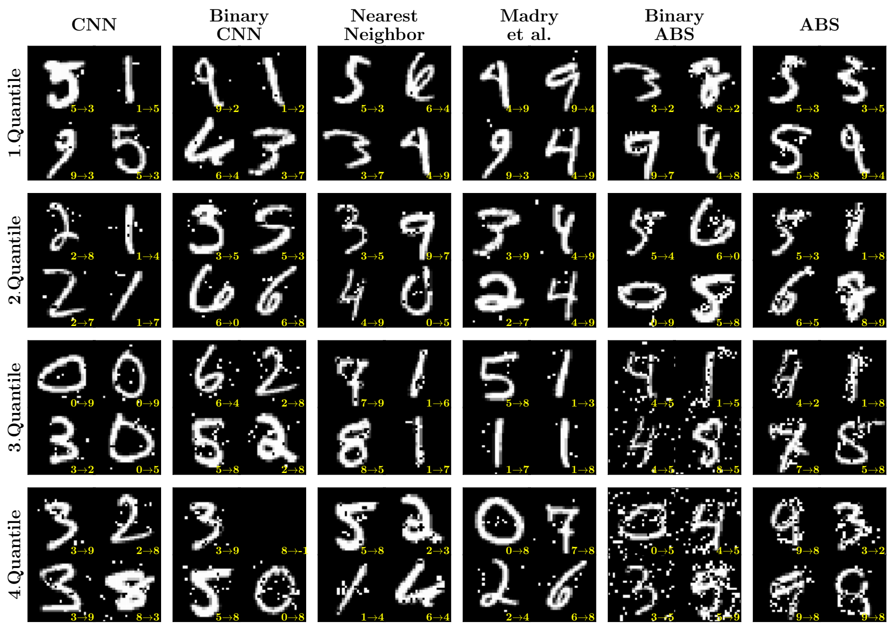


Figure 5: L_0 error quantiles: We always choose the minimally perturbed L_0 adversarial found by any attack for each model. For an unbiased selection, we then randomly sample images within four error quantiles $(0 - 25\%,\ 25 - 50\%,\ 50 - 75\%,\text{ and } 75 - 100\%)$. Where 100% corresponds to the maximal (over samples) minimum (over attacks) perturbation found for each model.

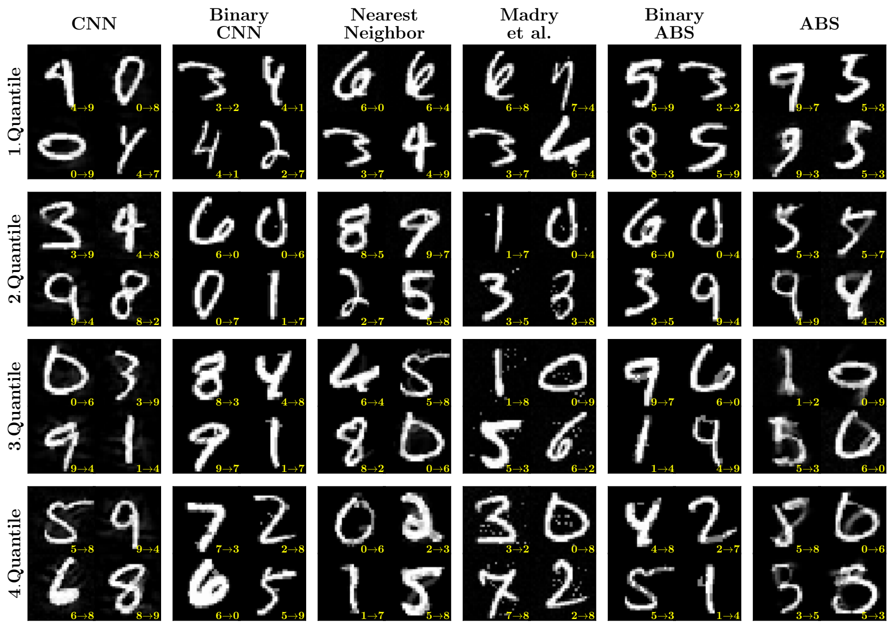


Figure 6: L_2 error quantiles: We always choose the minimally perturbed L_2 adversarial found by any attack for each model. For an unbiased selection, we then randomly sample 4 images within four error quantiles $(0-25\%,\ 25-50\%,\ 50-75\%,$ and $75-100\%)$.

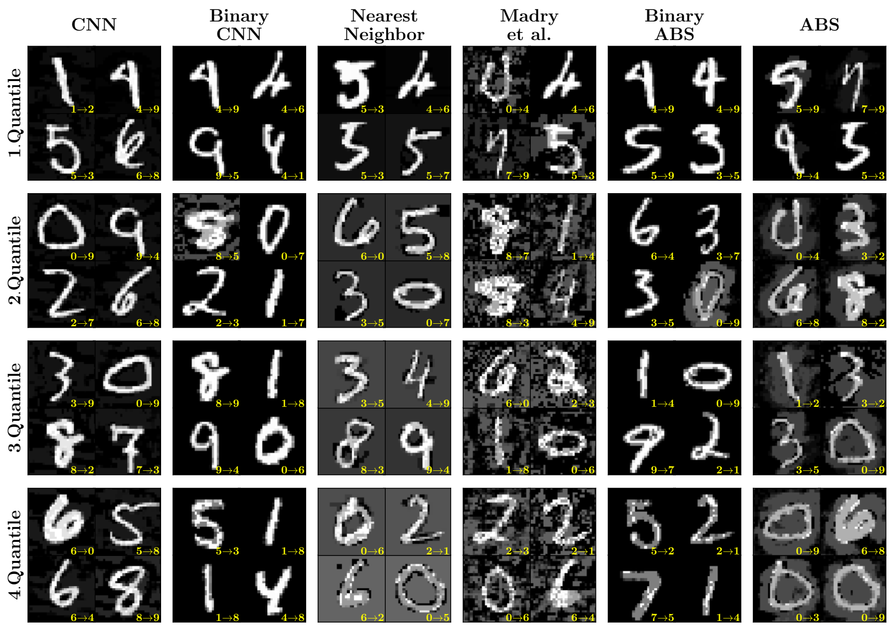
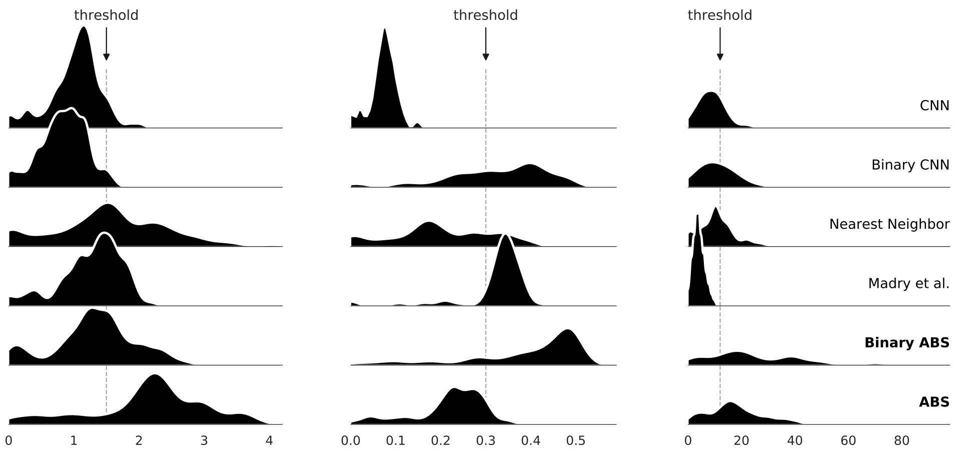


Figure 7: L_∞ error quantiles: We always choose the minimally perturbed L_∞ adversarial found by any attack for each model. For an unbiased selection, we then randomly sample images within four error quantiles $(0-25\%,\ 25-50\%,\ 50-75\%,$ and $75-100\%)$.



(a) L_2 distance (b) L_∞ distance (c) L_0 distance

Figure 8: Distribution of minimal adversarials for each model and distance metric. In (b) we see that a threshold at 0.3 favors Madry et al. while a threshold of 0.35 would have favored the Binary ABS.

A.3 DERIVATION I

Derivation of the ELBO in equation 2.

$$\log p_\theta(\mathbf{x}) = \log \int \mathbf{dz}\, p_\theta(\mathbf{x}|\mathbf{z}) p(\mathbf{z}),$$

where $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbb{1})$ is a simple normal prior. Based on the idea of importance sampling using a variational posterior $q_\phi(\mathbf{z}|\mathbf{x})$ with parameters ϕ and using Jensen's inequality we arrive at

$$\begin{aligned}
&= \log \int \mathbf{dz}\, \frac{q_\phi(\mathbf{z}|\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})} p_\theta(\mathbf{x}|\mathbf{z}) p(\mathbf{z}),\\
&= \log \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\frac{p_\theta(\mathbf{x}|\mathbf{z}) p(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} \right],\\
&\geq \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\log \frac{p_\theta(\mathbf{x}|\mathbf{z}) p(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} \right],\\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x}|\mathbf{z}) + \log \frac{p(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} \right],\\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x}|\mathbf{z}) \right] - \mathcal{D}_{KL}\left[q_\phi(\mathbf{z}|\mathbf{x}) || p(\mathbf{z}) \right].
\end{aligned}$$

This lower bound is commonly referred to as ELBO.

A.4 DERIVATION II: LOWER BOUND FOR L_2 ROBUSTNESS ESTIMATION

Derivation of equation 6. Starting from equation 3 we find that for a perturbation $\boldsymbol{\delta}$ with size $\epsilon = \|\boldsymbol{\delta}\|_2$ of sample $\mathbf{x}$ the lower bound $\ell_y^*(\mathbf{x}+\boldsymbol{\delta})$ can itself be bounded by,

$$\begin{aligned}
\ell_y^*(\mathbf{x}+\boldsymbol{\delta}) &= \max_{\mathbf{z}} -\mathcal{D}_{KL}\left[\mathcal{N}(\mathbf{z}, \sigma_q \mathbb{1}) || \mathcal{N}(\mathbf{0}, \mathbb{1})\right] - \frac{1}{2\sigma^2} \left\|\mathbf{G}_y(\mathbf{z}) - \mathbf{x} - \boldsymbol{\delta}\right\|_2^2 + C,\\
&\geq -\mathcal{D}_{KL}\left[\mathcal{N}(\mathbf{z}_\mathbf{x}^*, \sigma_q \mathbb{1}) || \mathcal{N}(\mathbf{0}, \mathbb{1})\right] - \frac{1}{2\sigma^2} \left\|\mathbf{G}_y(\mathbf{z}_\mathbf{x}^*) - \mathbf{x} - \boldsymbol{\delta}\right\|_2^2 + C,
\end{aligned}$$

where $\mathbf{z}_\mathbf{x}^*$ is the optimal latent vector for the clean sample $\mathbf{x}$ for class y,

$$\begin{aligned}
&= \ell_y^*(\mathbf{x}) + \frac{1}{\sigma^2} \boldsymbol{\delta}^\top (\mathbf{G}_y(\mathbf{z}_\mathbf{x}^*) - \mathbf{x}) - \frac{1}{2\sigma^2}\epsilon^2 + C,\\
&\geq \ell_y^*(\mathbf{x}) - \frac{1}{\sigma^2}\epsilon \left\|\mathbf{G}_y(\mathbf{z}_\mathbf{x}^*) - \mathbf{x}\right\|_2 - \frac{1}{2\sigma^2}\epsilon^2 + C. \qquad (10)
\end{aligned}$$

A.5 DERIVATION III: UPPER BOUND FOR L_2 ROBUSTNESS ESTIMATION

Derivation of equation 7.

$$\begin{aligned}
\ell_c^*(\mathbf{x}+\boldsymbol{\delta}) &= \max_{\mathbf{z}} -\mathcal{D}_{KL}\left[\mathcal{N}(\mathbf{z}, \sigma_q \mathbb{1}) || \mathcal{N}(\mathbf{0}, \mathbb{1})\right] - \frac{1}{2\sigma^2} \left\|\mathbf{G}_y(\mathbf{z}) - \mathbf{x} - \boldsymbol{\delta}\right\|_2^2 + C,\\
&\leq -\mathcal{D}_{KL}\left[\mathcal{N}(\mathbf{0}, \sigma_q \mathbb{1}) || \mathcal{N}(\mathbf{0}, \mathbb{1})\right] + C - \min_{\mathbf{z}} \frac{1}{2\sigma^2} \left\|\mathbf{G}_c(\mathbf{z}) - \mathbf{x} - \boldsymbol{\delta}\right\|_2^2,\\
&\leq -\mathcal{D}_{KL}\left[\mathcal{N}(\mathbf{0}, \sigma_q \mathbb{1}) || \mathcal{N}(\mathbf{0}, \mathbb{1})\right] + C - \min_{\mathbf{z},\boldsymbol{\delta}} \frac{1}{2\sigma^2} \left\|\mathbf{G}_c(\mathbf{z}) - \mathbf{x} - \boldsymbol{\delta}\right\|_2^2,\\
&= -\mathcal{D}_{KL}\left[\mathcal{N}(\mathbf{0}, \sigma_q \mathbb{1}) || \mathcal{N}(\mathbf{0}, \mathbb{1})\right] + C - \begin{cases} \frac{1}{2\sigma^2}(d_c - \epsilon)^2 & \text{if } d_c \geq \epsilon \\ 0 & \text{else} \end{cases}. \qquad (11)
\end{aligned}$$

for $d_c = \min_z \|\mathbf{G}_c(\mathbf{z}) - \mathbf{x}\|_2$. The last equation comes from the solution of the constrained optimization problem $\min_d (d-\epsilon)^2 d$ s.t. $d > d_c$. Note that a tighter bound might be achieved by assuming single $\boldsymbol{\delta}$ for upper and lower bound.

A.6 L_∞ ROBUSTNESS ESTIMATION

We proceed in the same way as for L_2. Starting again from

$$\ell_c^*(\mathbf{x}) = \max_{\mathbf{z}} \; -\mathcal{D}_{KL}\left[\mathcal{N}(\mathbf{z}, \sigma_q \mathbb{1}) || \mathcal{N}(\mathbf{0}, \mathbb{1})\right] - \frac{1}{2\sigma^2} \left\|\mathbf{G}_c(\mathbf{z}) - \mathbf{x}\right\|_2^2 + C, \qquad (12)$$

let y be the predicted class and let $\mathbf{z}^*_\mathbf{x}$ be the optimal latent for the clean sample $\mathbf{x}$ for class y. We can then estimate a lower bound on $\ell^*_y(\mathbf{x}+\boldsymbol{\delta})$ for a perturbation $\boldsymbol{\delta}$ with size $\epsilon = \|\boldsymbol{\delta}\|_\infty$,

$$\begin{aligned}\ell^*_y(\mathbf{x}+\boldsymbol{\delta}) &= \max_{\mathbf{z}} -\mathcal{D}_{KL}\left[\mathcal{N}(\mathbf{z},\sigma_q\mathbb{1})||\mathcal{N}(\mathbf{0},\mathbb{1})\right] - \frac{1}{2\sigma^2}\left\|\mathbf{G}_y(\mathbf{z}) - \mathbf{x} - \boldsymbol{\delta}\right\|_2^2 + C,\\ &\geq -\mathcal{D}_{KL}\left[\mathcal{N}(\mathbf{z}^*_\mathbf{x},\sigma_q\mathbb{1})||\mathcal{N}(\mathbf{0},\mathbb{1})\right] - \frac{1}{2\sigma^2}\left\|\mathbf{G}_y(\mathbf{z}^*_\mathbf{x}) - \mathbf{x} - \boldsymbol{\delta}\right\|_2^2 + C,\end{aligned}$$

where $\mathbf{z}^*_\mathbf{x}$ is the optimal latent for the clean sample $\mathbf{x}$ for class y.

$$\begin{aligned}&= \ell^*_y(\mathbf{x}) + \frac{1}{\sigma^2}\boldsymbol{\delta}^\top(\mathbf{G}_y(\mathbf{z}^*_\mathbf{x}) - \mathbf{x}) - \frac{1}{2\sigma^2}\|\boldsymbol{\delta}\|_2^2 + C,\\ &\geq \ell^*_y(\mathbf{x}) + C + \frac{1}{2\sigma^2}\min_{\boldsymbol{\delta}}\left(2\boldsymbol{\delta}^\top(\mathbf{G}_y(\mathbf{z}^*_\mathbf{x}) - \mathbf{x}) - \|\boldsymbol{\delta}\|_2^2\right),\\ &= \ell^*_y(\mathbf{x}) + C + \frac{1}{2\sigma^2}\sum_i \min_{\delta_i}\left(2\delta_i[\mathbf{G}_y(\mathbf{z}^*_\mathbf{x}) - \mathbf{x}]_i - \delta_i^2\right),\\ &= \ell^*_y(\mathbf{x}) + C + \frac{1}{2\sigma^2}\sum_i \begin{cases}[\mathbf{G}_y(\mathbf{z}^*_\mathbf{x}) - \mathbf{x}]_i^2 & \text{if } |[\mathbf{G}_y(\mathbf{z}^*_\mathbf{x}) - \mathbf{x}]_i| \leq \epsilon\\ \epsilon\,|[\mathbf{G}_y(\mathbf{z}^*_\mathbf{x}) - \mathbf{x}]_i| & \text{else}\end{cases}. \end{aligned} \tag{13}$$

Similarly, we can estimate an upper bound on $\ell^*_c(\mathbf{x}+\boldsymbol{\delta})$ on all other classes $c \neq y$,

$$\begin{aligned}\ell^*_c(\mathbf{x}+\boldsymbol{\delta}) &\leq -\mathcal{D}_{KL}\left[\mathcal{N}(\mathbf{0},\sigma_q\mathbb{1})||\mathcal{N}(\mathbf{0},\mathbb{1})\right] + C - \min_{\mathbf{z}}\frac{1}{2\sigma^2}\left\|\mathbf{G}_c(\mathbf{z}) - \mathbf{x} - \boldsymbol{\delta}\right\|_2^2,\\ &\leq -\mathcal{D}_{KL}\left[\mathcal{N}(\mathbf{0},\sigma_q\mathbb{1})||\mathcal{N}(\mathbf{0},\mathbb{1})\right] + C - \min_{\mathbf{z},\boldsymbol{\delta}}\frac{1}{2\sigma^2}\left\|\mathbf{G}_c(\mathbf{z}) - \mathbf{x} - \boldsymbol{\delta}\right\|_2^2,\\ &= -\mathcal{D}_{KL}\left[\mathcal{N}(\mathbf{0},\sigma_q\mathbb{1})||\mathcal{N}(\mathbf{0},\mathbb{1})\right] + C - \min_{\mathbf{z}}\frac{1}{2\sigma^2}\sum_i\min_{\delta_i}\left([\mathbf{G}_c(\mathbf{z}) - \mathbf{x}]_i - \delta_i\right)^2,\\ &= -\mathcal{D}_{KL}\left[\mathcal{N}(\mathbf{0},\sigma_q\mathbb{1})||\mathcal{N}(\mathbf{0},\mathbb{1})\right] + C\\ &\quad - \min_{\mathbf{z}}\frac{1}{2\sigma^2}\sum_i\begin{cases}0 & \text{if } |[\mathbf{G}_y(\mathbf{z}^*_\mathbf{x}) - \mathbf{x}]_i| \leq \epsilon\\ ([\mathbf{G}_y(\mathbf{z}^*_\mathbf{x}) - \mathbf{x}]_i - \epsilon)^2 & \text{if } [\mathbf{G}_y(\mathbf{z}^*_\mathbf{x}) - \mathbf{x}]_i > \epsilon\\ ([\mathbf{G}_y(\mathbf{z}^*_\mathbf{x}) - \mathbf{x}]_i + \epsilon)^2 & \text{if } [\mathbf{G}_y(\mathbf{z}^*_\mathbf{x}) - \mathbf{x}]_i < \epsilon\end{cases}.\end{aligned} \tag{14}$$

In this case there is no closed-form solution for the minimization problem on the RHS (in terms of the minimum of $\|\mathbf{G}_c(\mathbf{z}) - \mathbf{x}\|_2$) but we can still compute the solution for each given ϵ which allows us perform a line search along ϵ to find the point where equation 13 = equation 14.

A.7 Model & training details

Hyperparameters and training details for the ABS model The binary ABS and ABS have the same weights and architecture: The encoder has 4 layers with kernel sizes= $[5, 4, 3, 5]$, strides= $[1, 2, 2, 1]$ and feature map sizes= $[32, 32, 64, 2{*}8]$. The first 3 layers have ELU activation functions (Clevert et al., 2015), the last layer is linear. All except the last layer use Batch Normalization (Ioffe & Szegedy, 2015). The Decoder architecture has also 4 layers with kernel sizes= $[4, 5, 5, 3]$, strides= $[1, 2, 2, 1]$ and feature map sizes= $[32, 16, 16, 1]$. The first 3 layers have ELU activation functions, the last layer has a sigmoid activation function, and all layers except the last one use Batch Normalization.

We trained the VAEs with the Adam optimizer (Kingma & Ba, 2014). We tuned the dimension L of the latent space of the class-conditional VAEs (ending up with $L = 8$) to achieve 99% test error; started with a high weight for the KL-divergence term at the beginning of training (which was gradually decreased from a factor of 10 to 1 over 50 epochs); estimated the weighting $\boldsymbol{\gamma} = [1, 0.96, 1.001, 1.06, 0.98, 0.96, 1.03, 1, 1, 1]$ of the lower bound via a line search on the training accuracy. The parameters maximizing the test cross entropy[3] and providing a median confidence of $p(y|x) = 0.9$ for our modified softmax (equation 8) are $\eta = 0.000039$ and $\alpha = 440$. For our latent prior, we chose $\sigma_q = 1$ and for the posterior width we choose $\sigma = 1/\sqrt{2}$

Hyperparameters for the CNNs The CNN and Binary CNN share the same architecture but have different weights. The architecture has kernel sizes $= [5, 4, 3, 5]$, strides $= [1, 2, 2, 1]$, and feature map sizes $= [20, 70, 256, 10]$. All layers use ELU activation functions and all layers except the last one apply Batch Normalization. The CNNs are both trained on the cross entropy loss with the Adam optimizer (Kingma & Ba, 2014). The parameters maximizing the test cross entropy and providing a median confidence of $p(y|x) = 0.9$ of the CNN for our modified softmax (equation 8) are $\eta = 143900$ and $\alpha = 1$.

[3]Note that this solely scales the probabilities and does not change the classification accuracy.

Hyperparameters for Madry et al. We adapted the pre-trained model provided by Madry et al[4]. Basically the architecture contains two convolutional, two pooling and two fully connected layers. The network is trained on clean and adversarial examples minimizing the cross cross-entropy loss. The parameters maximizing the test cross entropy and providing a median confidence of $p(y|x) = 0.9$ for our modified softmax (equation 8) are $\eta = 60$ and $\alpha = 1$.

Hyperparameters for the Nearest Neighbour classifier For a comparison with neural networks, we imitate logits by replacing them with the negative minimal distance between the input and all samples within each class. The parameters maximizing the test cross entropy and providing a median confidence of $p(y|x) = 0.9$ for our modified softmax (equation 8) are $\eta = 0.000000000004$ and $\alpha = 5$.

[4]`https://github.com/MadryLab/mnist_challenge`

Scaling up the randomized gradient-free adversarial attack reveals over-estimation of robustness using established attacks

The following 19 pages have been published as:

Francesco Croce*, Jonas Rauber*, Matthias Hein (2020). “Scaling up the Randomized Gradient-Free Adversarial Attack Reveals Overestimation of Robustness Using Established Attacks”. *International Journal of Computer Vision (IJCV) 128:1028-1046.*

A summary of the motivation, results, and discussion can be found in section 2.2 on page 34. * joint first authorship

Abstract

Modern neural networks are highly non-robust against adversarial manipulation. A significant amount of work has been invested in techniques to compute lower bounds on robustness through formal guarantees and to build provably robust models. However, it is still difficult to get guarantees for larger networks or robustness against larger perturbations. Thus attack strategies are needed to provide tight upper bounds on the actual robustness. We significantly improve the randomized gradient-free attack for ReLU networks (Croce and Hein in GCPR, 2018), in particular by scaling it up to large networks. We show that our attack achieves similar or significantly smaller robust accuracy than state-of-the-art attacks like PGD or the one of Carlini and Wagner, thus revealing an overestimation of the robustness by these state-of-the-art methods. Our attack is not based on a gradient descent scheme and in this sense gradient-free, which makes it less sensitive to the choice of hyperparameters as no careful selection of the stepsize is required.

International Journal of Computer Vision (2020) 128:1028–1046
https://doi.org/10.1007/s11263-019-01213-0

Scaling up the Randomized Gradient-Free Adversarial Attack Reveals Overestimation of Robustness Using Established Attacks

Francesco Croce[1] · **Jonas Rauber**[1] · **Matthias Hein**[1]

Received: 3 March 2019 / Accepted: 12 August 2019 / Published online: 3 October 2019

Abstract
Modern neural networks are highly non-robust against adversarial manipulation. A significant amount of work has been invested in techniques to compute lower bounds on robustness through formal guarantees and to build provably robust models. However, it is still difficult to get guarantees for larger networks or robustness against larger perturbations. Thus attack strategies are needed to provide tight upper bounds on the actual robustness. We significantly improve the randomized gradient-free attack for ReLU networks (Croce and Hein in GCPR, 2018), in particular by scaling it up to large networks. We show that our attack achieves similar or significantly smaller robust accuracy than state-of-the-art attacks like PGD or the one of Carlini and Wagner, thus revealing an overestimation of the robustness by these state-of-the-art methods. Our attack is not based on a gradient descent scheme and in this sense gradient-free, which makes it less sensitive to the choice of hyperparameters as no careful selection of the stepsize is required.

Keywords Adversarial attacks · Adversarial robustness · White-box attacks · Gradient-free attacks

1 Introduction

Recent work has shown that state-of-the-art neural networks are non-robust (Szegedy et al. 2014; Goodfellow et al. 2015), in the sense that a small adversarial change of a (even with high confidence) correctly classified input leads to a wrong decision again potentially with high confidence. While Szegedy et al. (2014), Goodfellow et al. (2015) have brought up this problem in object recognition tasks, the problem itself has been discussed for some time in the area of email spam classification (Dalvi et al. 2004; Lowd and Meek 2005). However, since machine learning is nowadays used as a component for automated decision making in safety critical systems e.g. autonomous driving or medical diagnosis systems, fixing this problem should have high priority as it potentially can lead to fatal failures beyond the eminent security issue (Liu et al. 2017).

While a lot of research has been done on attacks and defenses (Papernot et al. 2016; Liu et al. 2017; Kurakin et al. 2017; Yuan et al. 2019) it has been shown that all existing defense strategies can be broken again (Carlini and Wagner 2017a; Athalye et al. 2018), with two exceptions. The first one are methods which provide provable guarantees on the robustness of a network (Katz et al. 2017; Tjeng et al. 2019; Hein and Andriushchenko 2017; Raghunathan et al. 2018; Wong and Kolter 2018; Mirman et al. 2018; Weng et al. 2018; Schott et al. 2019; Croce et al. 2019) and which have proposed new ways of training (Wong and Kolter 2018; Mirman et al. 2018) or of regularizing neural networks (Hein and Andriushchenko 2017; Croce et al. 2019) to make them more robust. While this area has made huge progress it is still difficult to provide such guarantees for medium-sized networks (Wong and Kolter 2018; Mirman et al. 2018). Then the only way to evaluate robustness for large networks is still to use successful attacks which thus provide, for every clean input, an upper bound on the norm of the minimal perturbation necessary to change the class. In fact, this is an approach to the problem of estimating robustness symmetric to formal certificates, which are lower bounds on the actual robustness. The first attack scheme based on L-BFGS has been proposed in Szegedy et al. (2014), afterwards research has produced a variety of adversarial attacks of growing

Communicated by Thomas Brox.

Francesco Croce, Jonas Rauber: shared first author.

✉ Francesco Croce
francesco91.croce@gmail.com

[1] Department of Computer Science, University of Tübingen, Tübingen, Germany

effectiveness (Goodfellow et al. 2015; Huang et al. 2016b; Moosavi-Dezfooli et al. 2016). However, it has been recognized that simple attacks often fail when they face a defense created against the specific attacks but which can be easily broken again using other more powerful techniques (Carlini and Wagner 2017a; Athalye et al. 2018). Apart from these white-box attacks (model is known at attack time), also several black-box attacks have been proposed Liu et al. (2017), Narodytska and Kasiviswanathan (2016) and Brendel et al. (2018).

The second exception is adversarial training with a relatively powerful attack (Madry et al. 2018) based on projected gradient descent (PGD). This defense technique could not be broken even using the state-of-the-art attack of Carlini and Wagner (2017b), Carlini and Wagner (2017a), Athalye et al. (2018).

In this paper we extend the white-box attack scheme proposed in Croce and Hein (2018), originally designed to attack fully-connected neural networks using ReLU type activation function. It is well known that these networks result in continuous piecewise affine functions (Arora et al. 2018), that is the domain is decomposed into linear regions given as polytopes on which the classifier is affine. The principle of the attack of Croce and Hein (2018) is then to solve the minimal adversarial perturbation problem on each linear region as it boils down to a convex optimization problem. In Croce and Hein (2018) they report that the attack outperforms the DeepFool attack (Moosavi-Dezfooli et al. 2016) and the state-of-the-art Carlini–Wagner attack (CW) (Carlini and Wagner 2017b) by up to 9% relative improvement in the norm of the smallest perturbation δ needed to change the classifier decision. However, the attack has been limited to small fully-connected neural networks with up to 10,000 neurons. In this paper we show that this attack can also be applied to convolutional, residual and dense networks with piecewise affine activation functions as well as max and average pooling layers and scales to networks consisting of more than 2.5 million neurons and achieving state-of-the-art performance.

The main contributions of this paper are (1) an upscaling of the attack to large networks so that it can be applied to standard networks for CIFAR-10 and (2) supporting now the most common types of layers, e.g. convolutional and residual ones. The key for the upscaling is a very fast solver for the dual of the quadratic program which has to be solved for finding minimal l_2 perturbations. The employed accelerated gradient descent scheme achieves quickly medium accuracy, which is enough for our purposes. Moreover, we use the fact that the solver just needs matrix-vector products of the constraint matrix and thus the explicit computation of the constraint matrix is not needed. This leads to a small memory footprint so that we can use this solver directly on the GPU. Finally, compared to Croce and Hein (2018) we have designed a more efficient sampling scheme of the next region to be checked. All these speed-ups together allow us now to attack networks as long as they basically fit into GPU memory. In this paper the largest network has over 2.8 million neurons, which is 280 times more than in Croce and Hein (2018). We show that in most of the cases our attack performs at least as good as the best attack among PGD (Madry et al. 2018), DeepFool (Moosavi-Dezfooli et al. 2016) and CW (Carlini and Wagner 2017b). In particular our algorithm works well across architectures, datasets and training schemes, being always the most successful or very close to the best of the competitors. Notably, we show especially for models trained with adversarial training (Madry et al. 2018) against the l_∞-norm and provably robust models (Croce et al. 2019; Wong and Kolter 2018) that the other attacks overestimate, partially by large margin, the robustness wrt to the l_2-norm.

We thus recommend our attack if a reliable estimation of the real robustness of a network is needed, as our attack not only performs well on average but does not, unlike the established state-of-the-art attacks (PGD, DeepFool, CW), lead to gross overestimation of the robustness of the network in some cases.

2 Piecewise Affine Formulation of ReLU Networks

It has been noted in Arora et al. (2018), Croce and Hein (2018) that ReLU networks, that is networks that use only the ReLU activation function, result in continuous piecewise affine classifiers in the form $f : \mathbb{R}^d \rightarrow \mathbb{R}^K$, where d is the input dimension and K the number of classes. This implies that there exists a *finite* set of polytopes Q_r, with $r = 1, \dots, R$, such that on each polytope the classifier is an affine function, that is there exists $W \in \mathbb{R}^{K\times d}$ and $b \in \mathbb{R}^K$ such that $f(x) = Wx + b$ holds for $x \in Q_r$. Note that, although we here focus on ReLU, the same property hold for any piecewise affine activation function, like Leaky ReLU. In the following we generalize the construction from Croce and Hein (2018) done for fully-connected networks to the case of other layer types, so that it extends to convolutional networks (CNNs), ResNets (He et al. 2016) and DenseNets (Huang et al. 2016a).

We can write ReLU networks with L hidden layers as the composition of $L+1$ functions, each standing for one of the layers (including the output layer), $f^{(1)}, \dots, f^{(L+1)}$. Note that we consider the application of the activation function as a stand-alone layer (called ReLU layer). Denoting with n_j, for $j = 1, \dots, L+1$, the number of units in layer j (in particular $n_{L+1} = K$ and we assume $n_0 = d$), we can define, for $j = 1, \dots, L+1$,

$$f^{(j)} : \mathbb{R}^{n_0} \times \mathbb{R}^{n_1} \times \dots \times \mathbb{R}^{n_{j-1}} \longrightarrow \mathbb{R}^{n_j} \tag{1}$$

and the output of the j-th layer is obtained as

$$x^{(j)} = f^{(j)}(x, x^{(1)}, \ldots, x^{(j-1)}), \tag{2}$$

where x is the input of the network. Then, the final output of the classifier f is given by

$$f(x) \equiv x^{(L+1)} = f^{(L+1)}(x, x^{(1)}, \ldots, x^{(L)}). \tag{3}$$

If we make explicit the relation between each $x^{(j)}$ and the input x we can recover the formulation of classifier f as a function from $\mathbb{R}^d$ to $\mathbb{R}^K$. While this definition of a classifier differs from the usual recursive formulation, it allows to handle also connections between non-consecutive layers [as it happens for residual networks (He et al. 2016)]. Finally, the class c which is assigned to x is given by

$$c = \underset{r=1,\ldots,K}{\arg\max}\, f_r(x) = \underset{r=1,\ldots,K}{\arg\max}\, x_r^{(L+1)}.$$

Every layer has one of the following types: dense, convolutional, skip connection, ReLU (or leaky ReLU), avg-pooling, max-pooling, batch normalization. We now show how it is possible to rewrite each of them, $f^{(j)}$, $j = 1, \ldots, L+1$, as an affine function

$$\begin{aligned} &\mathcal{A}^{(j)} : \mathbb{R}^{n_0} \times \mathbb{R}^{n_1} \times \ldots \times \mathbb{R}^{n_{j-1}} \longrightarrow \mathbb{R}^{n_j}, \\ &\mathcal{A}^{(j)}(t) = A^{(j)}t + a^{(j)}, \end{aligned} \tag{4}$$

with $A^{(j)} \in \mathbb{R}^{n_j \times N_j}$, $a^{(j)} \in \mathbb{R}^{n_j}$, $N_j = \sum_{i=0}^{j-1} n_i$, in the linear region $Q(x)$ corresponding to the input x (see below for the definition). For simplicity in the following we call $y = (x, x^{(1)}, \ldots, x^{(j-1)})$ the fixed input of the layer $f^{(j)}$ we are considering (see Eq. 2).

First, let us notice that dense, convolutional, skip connection and batch normalization (at inference time) layers are already affine operations, which means $\mathcal{A}^{(j)} \equiv f^{(j)}$.

Second, ReLU layers apply the function $\sigma(t) = \max\{0, t\}$ componentwise to the output of the previous layer. Thus, they can be, noticing that y is defined so that the last n_j components correspond to $x^{(j-1)}$, replaced by linear functions explicitly represented by the matrices $\Sigma \in \mathbb{R}^{n_j \times N_j}$ defined as

$$\Sigma = \begin{pmatrix} 0 & \ldots & 0 & h(x_1^{(j-1)}) & 0 & \ldots & 0 \\ 0 & \ldots & 0 & 0 & h(x_2^{(j-1)}) & \ldots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \ldots & 0 & 0 & 0 & \ldots & h(x_{n_j}^{(j-1)}) \end{pmatrix},$$

with

$$h : \mathbb{R} \longrightarrow \mathbb{R}, \quad h(t) = \begin{cases} 0 \text{ if } t < 0 \\ 1 \text{ else} \end{cases}.$$

Then, the desired affine function is $\mathcal{A}^{(j)}(t) = \Sigma t$. Since Σ depends on the input of the layer, $A^{(j)}$ and $\mathcal{A}^{(j)}$ are not shared by all the input points.

Third, average pooling computes the mean over certain subsets of the input vector. For example, the average of the first four entries of y is obtained, introducing

$$a = \left(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, 0, \ldots, 0\right) \in \mathbb{R}^{N_j},$$

as $\langle a, y\rangle$. Then, since we have n_j pools of p elements, it is sufficient to create n_j vectors similar to a, with entries equal to $1/p$ in the positions of the elements we want to average and zero else. We use then these vectors as rows of the matrix $A^{(j)}$, getting $\mathcal{A}^{(j)}(t) = A^{(j)}t$. We notice that $\mathcal{A}^{(j)}$ does not depend on the input y (as avg-pooling is already an affine function).

Finally, the construction of $A^{(j)}$ for max-pooling layers is analogue (as these layers return the maximum instead of the mean). The main difference is that in this case $A^{(j)}$ may change as y does. In fact, going back to our example, if we want to extract the maximum among the first four entries of y and assume that it is realized by the second component, we can set $a = (0, 1, 0, 0, \ldots, 0) \in \mathbb{R}^{N_j}$. If the position of the maximum changes also the vector a changes. Again, $\langle a, y\rangle$ returns the value we are interested in. If $p_1, \ldots, p_{n_j}$ are the positions of the maxima for each of the n_j pools, we can then build $A^{(j)} \in \mathbb{R}^{n_j \times N_j}$ as

$$A_{rs}^{(j)} = \begin{cases} 1 \text{ if } s = p_r \\ 0 \text{ else} \end{cases}, \quad s = 1, \ldots, N_j, \quad r = 1, \ldots, n_j,$$

so that $\mathcal{A}^{(j)}(t) = A^{(j)}t$. Please notice that, similarly to the case of ReLU layers, avg- and max-pooling layers usually involve only the output of the immediately preceding layer.

Once we have computed the affine functions $\mathcal{A}^{(j)}$ for every $j = 1, \ldots, L+1$ we can explicitly derive recursively the affine functions $\mathcal{V}^{(j)} : \mathbb{R}^d \longrightarrow \mathbb{R}^{n_j}$, represented by the matrices $V^{(j)} \in \mathbb{R}^{n_j \times d}$ and vectors $v^{(j)} \in \mathbb{R}^{n_j}$, satisfying the conditions

$$f^{(j)}(x, x^{(1)}, \ldots, x^{(j-1)}) \equiv \mathcal{V}^{(j)}(x) = V^{(j)}x + v^{(j)}. \tag{5}$$

Let us start with $j = 1$, that is the first layer. Then $V^{(1)}$ and the $v^{(1)}$ are the linear function and the bias which define $\mathcal{A}^{(1)}$, namely $A^{(1)}$ and $a^{(1)}$.

Assuming now that $\mathcal{V}^{(l)}$ are available for $l = 1, \ldots, j-1$, we get $\mathcal{V}^{(j)}$ combining Eqs. (2) and (5) and the definition of $\mathcal{A}^{(j)}$, so that

$$\mathcal{V}^{(j)}(x) = \mathcal{A}^{(j)}(x, \mathcal{V}^{(1)}(x), \ldots, \mathcal{V}^{(j-1)}(x)), \tag{6}$$

which is affine as a composition of linear and affine functions is affine again.

It still remains to compute the polytope $Q(x)$ containing x on which all the previous affine approximations hold exactly. First, note that $\mathcal{A}^{(j)}$ is independent of its input y (and thus from the input x of the network as well) if j is either a dense, convolutional, residual, avg-pooling or batch normalization layer, meaning that $\mathcal{A}^{(j)}$ is equivalent to $f^{(j)}$ on the whole input space. Thus these layers do not contribute to the definition of $Q(x)$. We are left to define where the linear reformulations of ReLU and max-pooling layers hold. As we noticed above, these kinds of layers only take into account the output of the immediately previous layer. Therefore, while considering layer j, we are allowed to act like the only input of $f^{(j)}$ was $x^{(j-1)}$.

Let $f^{(j)}$ be a ReLU layer and notice that the matrix Σ computed for $x^{(j-1)}$ is the same for any vector whose components have the same sign as those of $x^{(j-1)}$. Defining δ elementwise as

$$\delta_r = \operatorname{sgn}(x_r^{(j-1)}), \quad r = 1, \ldots, n_{j-1},$$

with the convention $\operatorname{sgn}(0) = 1$[1], we then get the set

$$S^{(j)}(x^{(j-1)}) = \{z \in \mathbb{R}^{n_{j-1}} \mid \operatorname{sgn}(z_r) = \delta_r, \\ r = 1, \ldots, n_{j-1}\}$$

containing the points of $\mathbb{R}^{n_{j-1}}$ which lead to the same matrix Σ as $x^{(j-1)}$. We note that the condition $\operatorname{sgn}(z_r) = \delta_r$ is equivalent to $z_r\delta_r \geq 0$ and that we are interested in the intersection of $S^{(j)}$ and the domain of layer j. With (5), we define the polytope on which $f^{(j)}$ is affine,

$$\begin{aligned} Q^{(j)}(x) &= \{z \in \mathbb{R}^d \bigm| \delta_r \mathcal{V}^{(j-1)}(z) \geq 0,\ r = 1, \ldots, n_{j-1}\} \\ &= \Big\{z \in \mathbb{R}^d \bigm| \delta_r (V^{(j-1)}z + v^{(j-1)}) \geq 0, \\ &\qquad r = 1, \ldots, n_{j-1}\Big\}. \end{aligned} \tag{7}$$

The set $Q^{(j)}(x)$ defines the region of the input space containing x and where $\mathcal{A}^{(j)}(x)$ and thus $f^{(j)}$ is an affine function. If $f^{(j)}$ is instead a max-pooling layer, we can see that $A^{(j)}$ is preserved as long as the maximum within each pool is realized at the same position. We can denote the n_j pools as the sets $P^1, \ldots, P^{n_j}$, whose elements are the indices of the components of the input (of the max-pooling layer) involved in the pool. Moreover we define for every $r = 1, \ldots, n_j$

$$p^r_{max} = \operatorname*{arg\,max}_{i \in P^r} x_i^{(j-1)},$$

that is the index of the component of $x^{(j-1)}$ attaining the maximum for each pool P^r. Then, for $r = 1, \ldots, n_j$,

$$S_r^{(j)}(x^{(j-1)}) = \left\{z \in \mathbb{R}^{n_{j-1}} \mid z_{p^r_{max}} \geq z_i,\ \forall i \in P^r\right\}$$

is the set of the vectors in $\mathbb{R}^{n_{j-1}}$ preserving the position of the maximum computed at $x^{(j-1)}$ for pool P^r. Similar to what has been done for ReLU layers, we define

$$\begin{aligned} Q_r^{(j)}(x) &= \left\{z \in \mathbb{R}^d \mid \mathcal{V}_{p^r_{max}}^{(j-1)}(z) \geq \mathcal{V}_i^{(j-1)}(z),\ \forall i \in P^r\right\} \\ &= \Big\{z \in \mathbb{R}^d \mid \left(V_{p^r_{max}}^{(j-1)} - V_i^{(j-1)}\right) z \\ &\quad + v_{p^r_{max}}^{(j-1)} - v_i^{(j-1)} \geq 0,\ \forall i \in P^r\Big\}, \end{aligned} \tag{8}$$

so that, finally, $Q^{(j)}(x) = \bigcap_{r=1}^{n_j} Q_r^{(j)}(x)$ is the subset of the input space containing x on which $f^{(j)}$ is an affine function.

Note that $Q^{(j)}(x) = \mathbb{R}^d$ if j is neither a ReLU nor a max-pooling layer. The polytope $Q(x)$ on which $f^{(L+1)}$ (and all layers below) is affine is given by

$$Q(x) = \bigcap_{j=1}^{L+1} Q^{(j)}(x).$$

In the following we refer to $Q(x)$ as the *linear region* of x. Note also that the intersection of $Q(x)$ with any other polytope is still a polytope (e.g. this is necessary when the input domain of a classifier is a subset of $\mathbb{R}^d$). Note that the explicit storage of the matrices $V^{(j)}$ is not possible for large networks and high input dimension as one needs $O(Nd)$ memory. In Sect. 4 it will turn out that our attack algorithm only requires matrix-vector products $V^{(j)}x$ which can be done without computing $V^{(j)}$ explicitly and thus we can do the whole attack on the GPU as long as the network itself fits into GPU memory.

3 Minimal Adversarial Perturbation Inside a Linear Region

Classifiers based on neural networks have been shown to be vulnerable to *adversarial samples*, that is they misclassify inputs which are almost indistinguishable from an original, correctly recognized test image (Szegedy et al. 2014; Goodfellow et al. 2015). The minimal adversarial perturbation δ wrt an l_p-norm is defined as the solution of the following optimization problem

[1] The case $x_r^{(j-1)} = 0$ implies that the region on which the affine approximation holds has dimension smaller than that of the input space. Setting $\operatorname{sgn}(0) = 1$ we consider a polytope which contains as a face the hyperplane defined by the condition $x_r^{(j-1)} = 0$.

$$\min_{\delta \in \mathbb{R}^d} \|\delta\|_p \quad \text{s.th.} \quad \max_{l \neq c} f_l(x+\delta) \geq f_c(x+\delta), \qquad x+\delta \in C, \tag{9}$$

with C being a set of constraints the input of f has to satisfy (in the following we assume that C is a polytope), e.g. images scaled to be in $[0,1]^d$, $x \in \mathbb{R}^d$ is the original point and c the class assigned to x by f (we assume x is correctly classified by f). The l_p-norm of δ measures the difference between original and adversarial inputs (changing p leads to adversarial samples with different properties). In practice, one often uses $p = 2$ or $p = \infty$. We concentrate for simplicity in this paper on $p = 2$, even though the framework allows to handle any p-norm given that a fast solver is available for the following linearized problem (11). Note that (9) represents an untargeted attack, that is we just want that the decision changes but we do not want to achieve that $x + \delta$ is classified as a particular class.

The optimization problem (9) is in general non-convex and NP-hard (Katz et al. 2017). However, as shown in Croce and Hein (2018), one can solve it efficiently inside every linear region of the classifier, that is if we add to (9) the constraint $x+\delta \in Q(y)$, where $Q(y)$ is the linear region which contains the point $y \in \mathbb{R}^d$. In fact, recalling Sect. 2, we introduce for $l \neq c$ the vectors δ_l as the solutions of the $K-1$ convex problems (note that we assume that C is a polytope)

$$\begin{aligned} \min_{\delta \in \mathbb{R}^d} \|\delta\|_p \quad \text{s.th.} \quad & \left\langle V_l^{(L+1)} - V_c^{(L+1)}, x+\delta \right\rangle \\ & + v_l^{(L+1)} - v_c^{(L+1)} \geq 0, \\ & x+\delta \in C \cap Q(y). \end{aligned} \tag{10}$$

Then, the solution of Problem (9) restricted to the linear region $Q(y)$ is $\underset{\{\delta_l : l \neq c\}}{\arg\min} \|\delta_l\|_p$. While we are mainly interested in untargeted attacks, we would like to highlight that targeted attacks against any of the classes $s \neq c$ are easily possible by solving instead the following problem:

$$\begin{aligned} \min_{\delta \in \mathbb{R}^d} \|\delta\|_p \quad \text{s.th.} \quad & \left\langle V_s^{(L+1)} - V_r^{(L+1)}, x+\delta \right\rangle \\ & + v_s^{(L+1)} - v_r^{(L+1)} \geq 0, \ \forall r \neq s, \\ & x+\delta \in C \cap Q(y). \end{aligned} \tag{11}$$

Please note that if one would solve (10) for all possible linear regions and take the smallest perturbation, then this the exact solution of (9). However, due to the extremely large number of linear regions this is infeasible in practice. Thus we use a randomized scheme for selecting the next linear region which is described in Sect. 4 together with a description of the particular solver for the resulting quadratic program in (10) for the choice of $p = 2$.

4 Generation of Adversarial Samples Through Randomized Local Search

In the following we present an improved selection scheme of the linear regions compared to the one in Croce and Hein (2018). The observation motivating our scheme is that the decision surface dividing areas of the input space assigned to different classes extends continuously across neighboring linear regions. If a point, say y, lying on the decision boundary is available, it is highly likely to find in its vicinity other points, again on the decision boundary between two classes, closer than y to the target image x. However, as pointed out in Croce and Hein (2018) it is very difficult to determine neighboring regions as a large number of the constraints defining the polytope are active at the solution of (10). In this case the neighboring region is not unique and checking all of them is infeasible and inefficient.

Thus we sample random points (more details below) in a small ball centered around the currently best point y, that is realizing the smallest adversarial perturbation found so far, and then solve (10) in the corresponding linear region until we find a better adversarial sample.

Moreover, we save the activation patterns of the linear regions we have explored. Before checking a point and its corresponding linear region we compare the activation pattern to the ones of the points which we have already visited. If the activation patterns agree it means that the two points belong to the same region and then we can skip checking it again.

Algorithm 1 shows our overall attack for a general l_p-norm trying to solve the optimization problem (9) for the minimal adversarial perturbation. In the experiments we use either $N = 400$ or $N = 500$, that is we check 400 resp. 500 linear regions. Please note that Algorithm 1 requires to be fed with a feasible point δ_{WS} of (9). There are several possibilities e.g. an adversarial sample of a fast attack like DeepFool as has been used in Croce and Hein (2018). In this paper we prefer to be independent of another attack. Thus we are using the following scheme to choose M starting points. At x we rank the classes $\{1, \dots, K\}$ according to the components of corresponding classifier output $f(x)$ in descending order ρ, where ρ_1 is the class which is assigned to x. We choose the M classes $\rho_2, \dots, \rho_{M+1}$ in the ranking and compute the point z_j in the training set correctly classified by f in class ρ_j which is closest to x for $j = 2, \dots, M+1$. In order to be speed up the attack we do for each z_j a binary search on $[x, z_j]$ and identify the point u_j which is closest to x but is classified differently from x and use $\delta_{WS}^{(j)} = u_j - x$, $j = 2, \dots, M+1$, as starting perturbations for Algorithm 1.

Algorithm 1: Our attack

```
Input  : $x$ original image, $\delta_{WS}$ starting perturbation, $\gamma, N, p$
Output: $\delta$ adversarial perturbation
1 $\delta \leftarrow \delta_{WS}, u \leftarrow \|\delta\|_p$
2 for $j = 1, \dots, N$ do
3     $y \leftarrow$ sampled according to (14)
4     if region containing $y$ has not been checked already then
5         computation of $Q(y)$
6         $\delta_{temp} \leftarrow$ solution of Problem (10) on $Q(y)$
7         if $\|\delta_{temp}\|_p < u$ then $\delta \leftarrow \delta_{temp}, u \leftarrow \|\delta\|_p$
8     end
9 end
```

At each of the N iterations we sample a point around the current best (smallest l_p-norm) feasible point $y := x + \delta$ of (9). The following sampling scheme is biased towards x, where $q \in [\frac{1}{2}, 1]$ is a parameter controlling the bias towards x ($q = \frac{1}{2}$ no bias, $q = 1$ maximal bias) and $\gamma > 0$ is a parameter controlling how localized our search is (the larger γ, the more localized). We provide an analysis of the influence of these parameters in Sect. 5.5. We sample (i) uniformly a point $y^\perp$ from the intersection of the unit sphere $\mathcal{S}^d$ centered in y and the hyperplane containing y with normal vector δ, and (ii) an angle $\theta \in [-\pi, \pi]$ given by

$$\begin{aligned} &X_1 \text{ r.v. : } \mathbb{P}(X_1 = 1) = q,\ \mathbb{P}(X_1 = -1) = 1 - q, \\ &X_2 \sim \mathcal{U}[0, \pi], \\ &\theta = X_1 X_2, \end{aligned} \tag{12}$$

where $\mathcal{U}[0, \pi]$ is the uniform distribution on the interval $[0, \pi]$. We define $\delta^\perp = y^\perp - y$. Note that by construction $\|\delta^\perp\|_2 = 1$. Finally,

$$\begin{aligned} &\delta_{\text{new}} = \cos(\theta)\delta^\perp - \sin(\theta)\frac{\delta}{\|\delta\|_2}, \\ &r_{\text{new}} = \|\delta\|_2\, X_3^\gamma \quad \text{with } X_3 \sim \mathcal{U}[0, 1] \end{aligned} \tag{13}$$

give direction and step size to produce the next point y_{new} whose linear region will be checked, defined as

$$y_{\text{new}} = y + r_{\text{new}}\delta_{\text{new}}. \tag{14}$$

Note that the larger γ the more biased y_{new} will be towards y. On the other hand our sampling scheme makes a difference between the half-sphere centered at y with pole at $x = y - \delta$ versus the half-sphere with pole at $y + \delta$. If $q = \frac{1}{2}$ samples from both half-sphere are equally probable, whereas if $q = 1$ one samples just from the half-sphere pointing towards x. At first sight it might look strange that we do not choose $q = 1$, as points sampled from the half-sphere pointing away from x have larger distance from x than y. However, experiments on a small subset of points show that a value of $q = 0.8$ leads to best results even though the difference to $q = 1$ is not large and thus we fix it to $q = 0.8$ for all experiments. Moreover, we use $\gamma = 6$ or $\gamma = 9$ for all experiments, noting anyway that the attack is not very sensitive to this value as γ in the range between 3 and 9 lead to very similar results. In Sect. 5.5 we provide a detailed analysis of the influence of these two parameters on the performance of our scheme.

If C is a polytope e.g. $C = [0, 1]^d$, then the optimization problems (10) and (11) are equivalent to linear programs (LP) for $p = \infty$ and $p = 1$ and equivalent to a quadratic program (QP) for $p = 2$. The main cost of the attack is to solve the optimization problem. Next we describe an efficient scalable way of solving (10) for $p = 2$, avoiding the explicit calculation of the linear regions.

4.1 A Scalable and Efficient Solver for the Quadratic Program

Let us suppose $x + \delta$ is our current best found solution, then we would like that the solution of (10) produces a new δ' which satisfies $\|\delta'\|_2 < \|\delta\|_2$. This implies that as soon as we have a certificate that the optimal value of (10) is larger than $\|\delta\|_2$ then we can stop the solver as checking this region will not yield an improvement. Thus we work with the dual of (10) as the dual objective is always a lower bound on the primal objective. As soon as we have found dual parameters realizing a larger dual objective than $\|\delta\|_2$ we can stop.

In the following we describe first how we solve the generic resulting dual QP using accelerated gradient descent together with coordinate descent in a subset of the variables. Then we describe how this algorithm for solving the QP can be efficiently implemented on the GPU without having to ever to compute the constraint matrix. Note that in Croce and Hein (2018) we used the commercial package Gurobi for solving the QP on the CPU. Now we present an own implementation fully running on the GPU which is roughly three orders of magnitude faster than then our old implementation on the CPU and which allows us to deal with fully-connected, convolutional and residual layers.

Solving the Dual Problem As we are mainly interested in applications in computer vision we specialize to the case $C = [0, 1]^d$ in (10), which can then be formulated as

$$\min_{z \in \mathbb{R}^d} \|z - x\|_2^2 \quad \text{s.th.} \quad Az \leq b, \quad z \in [0, 1]^d. \tag{15}$$

Note that the formulation is different from (10) but can be transformed into each other using $\delta = z - x$. The chosen formulation of the optimization problem in (15) is better adapted to the componentwise constraints imposed by C. The primal problem is strongly convex and thus has a unique solution. We derive the dual problem as

$$\max_{\alpha,\beta\in\mathbb{R}^d,\mu\in\mathbb{R}^m} \quad q(\mu,\alpha,\beta) \quad \text{s.th.} \quad \alpha\geq 0,\ \beta\geq 0,\ \mu\geq 0, \tag{16}$$

where

$$q(\mu,\alpha,\beta) = -\frac{1}{2}\left\|A^T\mu+\alpha-\beta\right\|_2^2 + \left\langle A^T\mu+\alpha-\beta, y\right\rangle - \langle\alpha,\mathbf{1}\rangle - \langle\mu,b\rangle$$

and the inequalities in (16) are componentwise. The correspondence between the primal variable z and the dual optimal variables α, β, μ is given by

$$z = y - A^T\mu - \alpha + \beta. \tag{17}$$

Note however that even for dual feasible α, β and μ, the primal variable z need not to be feasible. The KKT conditions are

$$\alpha_i(x_i-1)=0, \quad \beta_i x_i = 0, \quad \mu_i((Ax)_i - b_i) = 0.$$

This implies $\alpha_i\beta_i = 0$. Solving for α, β yields

$$\alpha = \max\{0, y - A^T\mu - \mathbf{1}\}, \beta = \max\{0, A^T\mu - y\}. \tag{18}$$

Thus for fixed μ we can directly find the optimal values of α and β. The dual problem is also a quadratic program but it is not necessarily strongly convex as AA^T does not need to be positive definite. However, the gradient

$$\begin{aligned}\nabla_\mu q &= -AA^T\mu + A(y-\alpha+\beta) - b\\ \nabla_\alpha q &= A^T\mu + \alpha - \beta + y - \mathbf{1}\\ \nabla_\beta q &= -(A^T\mu+\alpha-\beta) - y\end{aligned}$$

is Lipschitz continuous and the Lipschitz constant L can be upper bounded as,

$$L \leq \max\left\{\left\|A^TA\right\|^2, \left\|A^TA\right\|, 1\right\}. \tag{19}$$

We estimate $\|A^TA\|^2$ via the power method with 20 iterations, which is enough to get already a quite accurate estimate. We solve the QP itself with accelerated projected gradient descent (Nesterov 1983; Beck and Teboulle 2009; Chambolle and Pock 2011) in μ by setting α and β to their optimal values for given μ as in (18) which can be seen as a mixture of a coordinate descent in α, β and accelerated projected gradient descent in μ. Note that in all steps we never need the matrix A explicitly, but just matrix vector products $A^T\mu$ or Az if we want to compute feasibility of the current primal variable z. Even for the computation of $\|A^TA\|$ we use the power method which also only requires matrix vector products. The only caveat is a good pre-conditioning of the problem, which can be achieved by normalizing the rows a_i, $i=1,\dots,N$ of A to have unit norm (with corresponding rescaling of b). One can compute them via matrix vector products $a_i = A^Te_i$, but this would require too many of them. We discuss how this can be resolved in the next section and how the whole QP solver can be ported to the GPU.

4.2 Solving the QP Efficiently on the GPU Without Explicit Computation of the Constraint Matrix *A*

As discussed at the end of the previous section, the QP solver via accelerated gradient descent does not require the explicit computation of A as long as there is a way to compute matrix vector products $A^T\mu$ and Az efficiently. While in Croce and Hein (2018) the matrix A has been explicitly computed on the CPU, this is no longer feasible for larger networks as the memory consumption is $O(Nd)$, where d is the input dimension and N the total number of neurons. Even if one uses sparse matrix formats e.g. in the case of convolutional layers, this does not help to reduce the required memory significantly if the network is deep. Moreover, also the computation of the hyperplanes requires a computational cost equivalent to d forward passes of the network.

Thus a major improvement of this paper compared to Croce and Hein (2018) is the transfer of all computations from the CPU to the GPU which is only possible if the matrix A is not explicitly computed as the GPU memory would not suffice for this. The major insight to do this is that accelerated gradient descent only requires matrix-vector products of the form $A^T\mu$ and Az. Note that A contains basically the concatenated matrices $V^{(j)}$ from (7) and (8). However, we note that according to (5) it holds

$$f^{(j)}(x, x^{(1)},\dots,x^{(j-1)}) = \mathcal{V}^{(j)}(x) = V^{(j)}x + v^{(j)},$$

and thus $V^{(j)}$ is nothing else than the Jacobian $Jf^{(j)}$ of $f^{(j)}$ with respect to x and

$$v^{(j)} = f^{(j)}(x, x^{(1)},\dots,x^{(j-1)}) - V^{(j)}x.$$

Suppose for simplicity that

$$f^{(j)}(x) = g_j(g_{j-1}(\dots(g_1(x))\dots)).$$

Then the Jacobian $Jf^{(j)}$ of $f^{(j)}$ at x is given by the chain rule as

$$\begin{aligned}V^{(j)} &= Jf^{(j)}\big|_x\\ &= Jg_j\big|_{g_{j-1}(x)} Jg_{j-1}\big|_{g_{j-2}(x)} \cdots Jg_1\big|_x.\end{aligned}$$

Note that $V^{(j)}u$ can be evaluated as

$$V^{(j)}u = Jf^{(j)}\big|_x u$$

$$= Jg_j\big|_{g_{j-1}(x)}\Big(J_{g-1}\big|_{g_{j-2}(x)}\Big(\cdots\Big(Jg_1\big|_x u\Big)\cdots\Big)\Big).$$

In the same way we can compute $w^T V^{(j)}$ as

$$w^T V^{(j)} = w^T Jf^{(j)}\big|_x$$
$$= \Big(\cdots\Big(w^T Jg_j\big|_{g_{j-1}(x)}\Big) Jg_{j-1}\big|_{g_{j-2}(x)}\cdots\Big) Jg_1\big|_x.$$

Thus calculating $V^{(j)}u$ requires a single forward pass through the network and $w^T V^{(j)}$ requires a forward pass for computing the values $g_j(x)$ and then a backward pass through the network. More general, the computation of the Jacobian-vector products can be done via automatic differentiation (forward-mode resp. backward-mode automatic differentiation). Finally, to calculate the above expressions efficiently we still need a fast way to compute $Jg_k\big|_y v$ and $z^T Jg_k\big|_y$ for primitive functions g_k e.g. if g_k is a convolution, then $Jg_k\big|_y v$ can be computed as well as a convolution and $z^T Jg_k\big|_y$ as the transposed convolution. Fortunately, modern implementations of automatic differentiation already come with a large collection of primitive functions and corresponding rules for $Jg_k(y)v$ and $z^T Jg_k(y)$. Thus, we can directly and efficiently compute them on the GPU without computing the Jacobians itself. Thus our QP solver does not require much more memory than the network itself which allows it to scale to large networks.

Note that for pre-conditioning of A it would make sense to rescale the rows of A to have unit norm (one has to rescale correspondingly also the vector b). While every row vector a_i of A can be obtained as $a_i = e_i^T A$ and thus also just via matrix-vector products, doing this for every row is prohibitively expensive. Thus we use the fact that the norms of the row vectors corresponding to the same hidden layer have quite similar norms (typically we see increasing norms as one moves from lower to upper layers). Thus we just sample a small number of rows (in our case 10) of each layer, compute their norms, take the mean of them and use the inverse of that as a rescaling factor for that layer. While this coarse pre-conditioning scheme is worse than if one rescales every row individually, it is significantly better than not doing any rescaling at all. There is one exception: we upscale the constraint of the decision boundary, as we have found that this leads to faster feasibility of this constraint which is the most important one of all the constraints.

Moreover, we do not need an accurate solution of (10) and thus we have found that in practice 500 iterations of the accelerated gradient descent scheme suffice to get a reasonable solution. As the primal variable z in (17) obtained from the dual variables need not be feasible, we explicitly check if the output z is an adversarial sample. If not then we check via a small line search $x + \alpha(z - x)$, where $\alpha \geq 1$, if it is an adversarial sample as long as $\alpha \|z - x\|_2 < \|\delta\|$, where $\|\delta\|$ is the norm of the perturbation of the currently best adversarial sample $x + \delta$. Finally, this leads to a scheme which is more than three orders of magnitude faster than that in Croce and Hein (2018).

5 Experiments

In this section we show that our attack often outperforms the state-of-the-art methods to compute upper bounds on the robust accuracy of a model, which is defined for a given $\epsilon > 0$, as the minimal accuracy that the classifier can achieve if each test sample is allowed to be perturbed within a p-norm ball of radius ϵ in order to achieve a misclassification. The smaller the found robust accuracy the stronger is the attack and the less robust is the network. We focus here on the l_2-attack. The code for our attack is publicly available.[2]

We show that current state-of-the-art attacks sometimes overestimate the robustness of classifiers. In fact, with our attack we are often able to achieve smaller robust accuracy than our competitors, and even when we do not we never overestimate the robust test accuracy more than 5.0% compared to the minimal one found by the competitors. In contrast, all the other attacks have cases where they achieve a robust accuracy at least 50.4% larger than that provided by our method (see Table 1). Thus if one just evaluates robustness using the competing attacks, one would consider models robust which are in fact quite non-robust. Our technique does not show a similar weakness in any setting, pointing out how our algorithm is, on one side, able to recover in general small adversarial perturbations and, on the other side, less susceptible to changes in the characteristics of the network. Interestingly, we notice that l_2- gradient-based methods suffer especially when attacking models trained with l_∞-adversarial training.

We consider three datasets: MNIST, German Traffic Sign (GTS) (Stallkamp et al. 2012) and CIFAR-10 (Krizhevsky et al. 2014) (all images are scaled in $[0, 1]^d$). On each of them three models are trained, the plain model (*plain*), one with l_2-adversarial training (called *l_2-at*) and one with l_∞-adversarial training (*l_∞-at*) (we use the adversarial training scheme of Madry et al. (2018) that is based on the Projected Gradient Descent attack). More details about architectures and training are provided below.

We compare our attack against: Projected Gradient Descent on the loss function (PGD) (Madry et al. 2018), Carlini–Wagner l_2-attack (CW) (Carlini and Wagner 2017b) and DeepFool (DF) (Moosavi-Dezfooli et al. 2016). We use two versions of PGD: PGD-1 uses a single starting point, while PGD-10k exploits 10,000 restarts, randomly sampled in the l_2-ball of radius ϵ around the original image. This large number of restarts is motivated by a recent paper which could

[2] https://github.com/jonasrauber/linear-region-attack.

Table 1 Performances of different attacks

Model	PGD-1	PGD-10k	CW-10k	CW-100k	DF	Ours
Average difference to the best l_2 robust accuracy						
MNIST	0.2367	0.1011	0.1701	0.1681	0.3135	**0.00051**
GTS	0.0361	0.0237	0.0177	0.0172	0.0643	**0**
CIFAR-10	0.0693	0.0515	0.0045	**0.00037**	0.0812	0.0060
Maximum difference to the best l_2 robust accuracy						
MNIST	0.7800	0.5040	0.6200	0.6120	0.9000	**0.00500**
GTS	0.1600	0.1260	0.0440	0.0420	0.1140	**0**
CIFAR-10	0.2280	0.2040	0.0180	0.0180	0.1220	**0.00140**

For each dataset, attack and threshold ϵ, we compute the differences between the robust accuracies estimated by an attack and the best one among those of all the attacks. We here report, given dataset and attack, the mean (top) and the maximum (bottom) of these differences across the thresholds. We can see that our attack has the smallest average distance from the best on two of three datasets and always achieves the best maximal distance. Notably, on GTS both mean and maximum for our attack are 0, which means that it gets the lowest robust accuracy for every model and ϵ

In bold the best, for each dataset, average or maximum "difference to the best l_2 robust accuracy" (lower is better)

break a certain defense only when using 10,000 restarts of PGD (Mosbach et al. 2018). For both PGD versions we set $k = 40$ iterations and, if ϵ is the threshold at which we want to evaluate robust accuracy, we use a step size of $\epsilon/4$. Similarly, we evaluate CW in the implementation of Papernot et al. (2017) with 40 binary search steps and either 10,000 (CW-10k) or 100,000 iterations (CW-100k). We use the DF implementation as in Rauber et al. (2017).

Since the objective of PGD is only to find out if there exists an adversarial sample with norm less than the threshold ϵ, it provides directly the robust accuracy at ϵ (and must be rerun for each threshold ϵ). On the contrary, CW, DeepFool and our attack try to find the minimal adversarial perturbation as in (9). After running these attacks, we compute the robust accuracy for a given threshold as the fraction of points whose adversarial examples are farther, in l_2-distance, than ϵ. Note that we only check correctly classified points for all methods. The obtained values of robust accuracy achieved for all attacks and different thresholds are reported in Tables 2 (MNIST), 3 (GTS) and 4 (CIFAR-10).

In order to thoroughly evaluate the effectiveness of an attack, it is necessary to assess average and worst case performance. In this way one can see whether it overfits to some particular model, dataset or training scheme. For every dataset we compute for all thresholds ϵ the difference between the robust accuracy provided by every attack and the minimal robust accuracy across all the attacks for the fixed threshold. Thus the worse the performance a method achieves, the larger the difference is. In Table 1 we report for each attack the mean and the maximal distance from the best accuracy, across the three models and five thresholds ϵ, for each of the three datasets. It can be directly seen from this table that our attack has at the same time the best *worst case performance* for all three datasets and the best *average performance* in two of three datasets with only a tiny difference in the case it is worse.

In particular, we can see how on MNIST the second best attack (PGD-10k) is on average 10.11% worse than the minimal robust accuracy, compared to 0.51% for our attack, while in the worst case it returns a robust accuracy 50.4% larger than the minimal one versus 5.0% for our method. On GTS, both average and maximal difference are 0.0% for our attack, meaning that it always achieves the minimal robust accuracy among all competing methods. Although on CIFAR-10 we cannot match the average result of CW attack, our attack has nevertheless the best *worst case performance*, highlighting the quality of our approach.

5.1 Main Experiments: Details

MNIST For MNIST we use the same architecture as in Madry et al. (2018), consisting in 2 convolutional layers of 16 and 32 filters, each followed by max-pooling, and 2 dense layers. In particular, the plain and l_∞- trained models are the *natural* and *secret* models of "MNIST Adversarial Examples Challenge",[3] based on Madry et al. (2018). For l_2-*at* we adapted the code of Madry et al. (2018) to perform adversarial training using the PGD attack wrt the l_2-norm with $\epsilon = 2$ and 40 iterations. The clean accuracy of the models can be found in Table 2 in the row corresponding to $\epsilon = 0$. Moreover, we use $M = 5$ different starting points for our attack (corresponding to five classes) and $N = 500$ as the maximum number of linear regions checked, or equivalently iterations

[3] https://github.com/MadryLab/mnist_challenge.

Table 2 Robustness of MNIST models

Model	ϵ	PGD-1	PGD-10k	CW-10k	CW-100k	DF	Ours
l_2 robust accuracy on MNIST							
plain	0.0	0.984					
	0.5	0.928	**0.926**	**0.926**	**0.926**	0.936	**0.926**
	1.0	0.508	**0.472**	0.474	0.474	0.586	0.474
	1.5	0.168	0.106	0.088	0.088	0.198	**0.078**
	2.0	0.106	0.028	0.006	0.006	0.018	**0.002**
	2.5	0.078	0.014	**0.000**	**0.000**	**0.000**	**0.000**
l_2-at	0.0	0.986					
	1.0	0.930	0.930	**0.926**	**0.926**	0.938	**0.926**
	1.5	0.838	**0.834**	0.846	0.848	0.872	**0.834**
	2.0	0.698	**0.672**	0.706	0.706	0.790	0.680
	2.5	0.468	**0.366**	0.466	0.464	0.674	0.416
	3.0	0.192	**0.096**	0.170	0.172	0.542	0.112
l_∞-at	0.0	0.984					
	1.0	0.924	0.878	0.888	0.888	0.948	**0.736**
	1.5	0.886	0.748	0.774	0.776	0.932	**0.258**
	2.0	0.812	0.536	0.652	0.644	0.918	**0.032**
	2.5	0.758	0.248	0.552	0.538	0.904	**0.004**
	3.0	0.658	0.064	0.480	0.468	0.848	**0.000**

We report upper bounds on the robust accuracy, that is the fraction of points in the test set which are still correctly classified when any perturbation of l_2-norm smaller than or equal to ϵ is allowed in order to achieve a misclassification (a smaller robust accuracy means a stronger attack). The statistics are computed on the first 500 points of the MNIST test set

For each threshold and model we highlight in bold the lowest robust accuracy, which indicates the strongest attack

in Algorithm 1, for each starting point. Moreover we set the parameter $\gamma = 6$ in Algorithm 1.

In Table 2 we report the robust accuracy, computed on 500 points of the test set, for the three models when the l_2-norm of the perturbations is bounded by ϵ. We see that in most of the cases our attack achieves the best performance. In particular, on the l_∞-trained model all the other gradient-based methods suggest that the classifier is highly robust, while our attack shows that this is not the case, as it turns out to be just slightly less vulnerable to adversarial examples than the *plain* model (e.g. at $\epsilon = 2.0$ the best of other attacks reduces accuracy only to 53.6% while our technique brings it down to 3.2%).

Notably, in Schott et al. (2019) the same l_∞-*at* model was tested and, taking the pointwise best output among those of 11 attacks of various nature, the authors could decrease robust accuracy no more than 35% with $\epsilon = 1.5$. On the other hand we see that our attack alone, without even testing all the possible 9 target classes, yields an upper bound on robust accuracy for the same ϵ of 25.8%, which is almost 10% less than the current state-of-the-art (Brendel et al. 2018).

GTS In this case the models are CNNs with 2 convolutional layers (16 and 32 feature maps) with stride 2, which replaces max-pooling for downsizing, and 2 dense layers. Adversarial training is based on 40 iterations of PGD attack, with $\epsilon = 0.5$ for l_2-*at* and $\epsilon = 4/255$ for l_∞-*at*. Since GTS has 43 classes, we run our algorithm with $M = 15$ starting points, 500 linear regions each and $\gamma = 9$.

Table 3 shows how the upper bounds on robust accuracy, computed on the first 500 images of the test set, obtained through our technique are always smaller than those by the competitors, apart from 4 cases out of 15 where the PGD results can only match ours. We notice that, although in some cases the difference is not extremely large, in 3 of 15 settings our attack reduces the robust accuracy at least by 2% compared to the best result of the other methods, with a maximum of 4.2% for $\epsilon = 1.25$ for the l_∞-trained model (robust accuracy of 11.4% by CW-100k vs. 7.2% for our attack).

CIFAR-10 Since CIFAR-10 represents a more difficult classification task, we use for it a deeper and wider architecture, made of 8 convolutional layers (with number of filters increasing from 96 to 384) and 2 dense layers, which contains overall more than 375000 units. We perform adversarial training again with the PGD attack, with 10 iterations, $\epsilon = 80/255$ and $\epsilon = 4/255$ for l_2- and l_∞-robust training respec-

Table 3 Robustness of GTS models

Model	ϵ	PGD-1	PGD-10k	CW-10k	CW-100k	DF	Ours
l_2 robust accuracy on GTS							
plain	0.0	0.946					
	0.1	0.746	0.746	0.754	0.754	0.788	**0.740**
	0.2	0.568	0.562	0.566	0.566	0.628	**0.550**
	0.4	0.360	0.348	0.334	0.334	0.408	**0.316**
	0.6	0.298	0.274	0.214	0.214	0.292	**0.178**
	0.8	0.268	0.234	0.124	0.124	0.210	**0.108**
l_2-at	0.0	0.908					
	0.1	**0.818**	**0.818**	0.826	0.820	0.826	**0.818**
	0.2	0.708	**0.704**	0.706	0.710	0.728	**0.704**
	0.4	0.488	0.472	0.496	0.496	0.538	**0.468**
	0.6	0.328	**0.320**	0.322	0.322	0.378	**0.320**
	0.8	0.222	0.218	0.224	0.222	0.284	**0.212**
l_∞-at	0.0	0.904					
	0.25	0.690	**0.686**	0.692	0.692	0.718	**0.686**
	0.5	0.460	0.446	0.468	0.466	0.500	**0.444**
	0.75	0.302	0.288	0.300	0.300	0.338	**0.280**
	1.0	0.212	0.200	0.214	0.214	0.246	**0.194**
	1.25	0.164	0.130	0.116	0.114	0.172	**0.072**

We report upper bounds on the robust accuracy, that is the fraction of points in the test set which are still correctly classified when any perturbation of l_2-norm smaller than or equal to ϵ is allowed in order to achieve a misclassification (a smaller robust accuracy means a stronger attack). The statistics are computed on the first 500 points of the GTS test set

For each threshold and model we highlight in bold the lowest robust accuracy, which indicates the strongest attack

tively. We here run our attack with 400 iterations and 3 starting points, fixing $\gamma = 9$.

The statistics over the first 500 points of the test set are summarized in Table 4. Although with this dataset we see that the best performances are achieved by different methods in many situations, we can nevertheless notice that our attack clearly outperforms PGD and DF and is at most 1.4% off from the best robust accuracy. CW attack performs here very well but it still has a slightly worse performance in the *worst case* setting, as we can see in Table 1.

Moreover, these CIFAR-10 networks are less robust than those trained on MNIST and GTS, so that the task of crafting small adversarial examples is easier than previously. This implies that even weak attackers can succeed in finding good, maybe almost optimal, adversarial perturbations.

5.2 Testing Provably Robust Models

In this section we test classifiers trained to be provably robust, that is it is possible to compute for a large fraction of the test points if there exists or not an adversarial perturbation with norm smaller than a fixed threshold. This means that non-trivial *lower* bounds on the robust accuracy are provided. For what concerns *upper* bounds, we have mostly to rely, especially for the l_2 case, on the adversarial examples provided by the attacks. Then, using powerful attacks allows also to correctly assess the tightness of the lower bounds or equivalently the effectiveness of the verification methods.

We consider the models presented in Croce et al. (2019), that is CNNs with 2 convolutional layers of 16 and 32 filters and a hidden fully-connected layer of 100 units. These are trained with the techniques of either (Croce et al. 2019) (called MMR) or (Wong and Kolter 2018; Wong et al. 2018) (KW) to be robust wrt the l_2-norm at $\epsilon_{\text{train}} = 0.3$ for MNIST and $\epsilon_{\text{train}} = 0.1$ for CIFAR-10, wrt the l_∞-norm at $\epsilon_{\text{train}} = 0.1$ for MNIST and $\epsilon_{\text{train}} = 2/255$. We decide to test the l_2 robustness of all the models with thresholds ϵ larger than those used for l_2 robust training since at those levels the uncertainty on robust accuracy is limited as tight bounds on it are available (see Croce et al. 2019). We run our attack for 500 regions and 5 starting points. In Tables 5 (MNIST) and 6 (CIFAR-10) we report similarly to the previous section the upper bounds on the robust accuracy, computed with 500 test points, provided the different attacks (we here use PGD-1k with 1000 restarts instead of the weaker version with a single restart).

Table 4 Robustness of CIFAR-10 models

Model	ϵ	PGD-1	PGD-10k	CW-10k	CW-100k	DF	Ours
l_2 robust accuracy on CIFAR-10							
plain	0.0	0.892					
	0.1	0.686	**0.676**	0.694	0.694	0.722	0.690
	0.15	0.546	**0.536**	0.554	0.552	0.626	0.550
	0.2	0.440	**0.422**	0.434	0.432	0.512	0.434
	0.3	0.256	0.234	**0.216**	**0.216**	0.338	0.220
	0.4	0.182	0.146	0.094	**0.092**	0.208	0.098
l_2-at	0.0	0.812					
	0.25	0.658	**0.656**	0.660	0.660	0.670	**0.656**
	0.5	0.496	0.488	0.482	0.482	0.538	**0.478**
	0.75	0.382	0.362	**0.324**	**0.324**	0.422	**0.324**
	1.0	0.358	0.322	0.212	**0.204**	0.300	0.216
	1.25	0.336	0.302	**0.114**	**0.114**	0.224	0.124
l_∞-at	0.0	0.794					
	0.25	0.646	**0.644**	0.646	0.646	0.670	**0.644**
	0.5	0.488	**0.484**	**0.484**	**0.484**	0.530	0.488
	0.75	0.390	0.368	**0.332**	**0.332**	0.414	0.334
	1.0	0.352	0.332	**0.226**	0.228	0.326	0.228
	1.25	0.348	0.324	**0.120**	**0.120**	0.242	0.130

We report upper bounds on the robust accuracy, that is the fraction of points in the test set which are still correctly classified when any perturbation of l_2-norm smaller than or equal to ϵ is allowed in order to achieve a misclassification (a smaller robust accuracy means a stronger attack). The statistics are computed on the first 500 points of the CIFAR-10 test set
For each threshold and model we highlight in bold the lowest robust accuracy, which indicates the strongest attack

For both datasets we see that our attack outperforms, often significantly, the competitors, with the only exception being the largest value of ϵ on the model trained with KW technique wrt l_∞-norm on MNIST. Moreover, note that similar to Table 2, the largest differences (over 22% between the upper bounds on robust accuracies of PGD-100k and our attack) are reached for the classifier trained on MNIST with adversarial training from Madry et al. (2018) wrt l_∞.

5.3 Attacking Large Models

In order to show the scalability of our approach to large models, we here attack the networks from "CIFAR-10 Adversarial Examples Challenge"[4] trained on CIFAR-10 with either plain or l_∞-adversarial training (Madry et al. 2018) (called *naturally trained* and *secret* in the original challenge). The architecture used is a residual convolutional network consisting of a convolutional layer, five residual blocks and a fully-connected layer, derived from the "w32-10 wide" variant of the TensorFlow model repository, with 2.883.593 units. In order to apply our algorithm we had to replace the per image normalization, which is not an affine operation on the input, with the following step: for each input image, we subtract the mean of its entries and divide it by a constant (0.21, which is an approximation of the average standard deviation across the images of the training set). Note that this small variation does not affect the performance of the classifier while allows the network to result in a piecewise affine function.

In Table 7 we report the robust accuracy, on the first 100 test points, given by the three methods (in this case we use PGD with 1000 but not 10,000 restarts as it would be computationally too expensive). We omit CW since with the default parameters it fails to provide meaningful results. For our method we use 5 starting points. While DeepFool is always worse than the others, PGD and our attack perform similarly, although the largest gap (3%, achieved at $\epsilon = 1.25$ for the l_∞-*at* model) is in favour of our method.

5.4 Runtime Comparison

We analyze the runtime the different attacks take to return results on 500 test points on the *plain* model on CIFAR-10 of Sect. 5.1 using a single GPU. Note that CW, DeepFool and our method aim at finding the minimal adversarial perturbation within a limited budget of iterations while PGD takes as

[4] https://github.com/MadryLab/cifar10_challenge.

Table 5 Provably robust MNIST models

Model	ϵ	PGD-1k	PGD-10k	CW-10k	CW-100k	DF	Ours
l_2 robust accuracy on MNIST							
l_∞-MMR-*at*	0.0	0.988					
	1.0	0.828	0.816	0.854	0.854	0.868	**0.704**
	1.5	0.488	0.428	0.642	0.642	0.682	**0.250**
	2.0	0.310	0.270	0.414	0.412	0.642	**0.048**
	2.5	0.222	0.180	0.196	0.194	0.238	**0.004**
	3.0	0.136	0.116	0.074	0.070	0.084	**0.000**
l_∞-KW	0.0	0.982					
	1.0	0.924	0.910	0.924	0.924	0.926	**0.854**
	1.5	0.674	0.600	0.834	0.834	0.898	**0.478**
	2.0	0.226	0.176	0.664	0.662	0.844	**0.148**
	2.5	0.030	0.020	0.454	0.454	0.784	**0.018**
	3.0	0.002	**0.000**	0.264	0.264	0.644	0.002
l_2-MMR-*at*	0.0	0.986					
	1.0	0.848	0.848	0.850	0.850	0.868	**0.842**
	1.5	0.608	0.606	0.622	0.622	0.682	**0.576**
	2.0	0.286	0.270	0.312	0.312	0.462	**0.238**
	2.5	0.050	0.048	0.090	0.090	0.238	**0.044**
	3.0	0.016	0.012	0.032	0.030	0.084	**0.010**
l_2-KW	0.0	0.988					
	1.0	0.916	0.916	0.914	0.914	0.928	**0.912**
	1.5	0.722	0.716	0.740	0.740	0.826	**0.692**
	2.0	0.392	0.366	0.438	0.438	0.690	**0.298**
	2.5	0.214	0.202	0.166	0.166	0.478	**0.078**
	3.0	0.172	0.152	0.046	0.046	0.292	**0.012**

We report upper bounds on the robust accuracy, that is the fraction of points in the test set which are still correctly classified when any perturbation of l_2-norm smaller than or equal to ϵ is allowed in order to achieve a misclassification (a smaller robust accuracy means a stronger attack). The statistics are computed on the first 500 points of the MNIST test set

For each threshold and model we highlight in bold the lowest robust accuracy, which indicates the strongest attack

input a thresholds ϵ and looks for a manipulation with norm smaller than it, but does not try to minimize it. This means that, in order to build Table 4 one has to run PGD once for each value ϵ. Conversely, for the other attacks a single run is sufficient to compute the robust accuracy at every threshold.

We compare the runtime of the attacks in the setting used for the experiment in Table 4, and report the total time needed to run the attacks on 500 different points on a single GPU: PGD-10k takes around 18 h for a single threshold. CW-100k needs 55 h in total and our method takes 150 h (using 3 starting points), while the fastest but also weakest attack is DeepFool with a runtime of less than 1 min.

5.5 Choosing Parameters

In order to choose a proper parameter q for the sampling scheme in Equation (12) we run our attack on the MNIST *plain* model, already introduced in Sect. 5.1, with $q \in \{0.6, 0.7, 0.8, 0.9, 1.0\}$ and with the scheme proposed in Croce and Hein (2018), where the next linear region to check is chosen by sampling uniformly a direction from the current best solution (corresponding to $q = 0.5$). In Fig. 1 we show the development of median (left), maximum (center) and mean (right) of the l_2-norms of the adversarial perturbations found as a function of the explored linear regions. We can see that the final values of the statistics do not differ significantly. Moreover, we repeat the previous experiment, this time varying the value of γ in Eq. (13). In particular, we test $\gamma = 1, \dots, 9$ and report in Fig. 2 median (left), maximum (center) and mean (right) of the l_2-norms of the adversarial perturbations found as a function of the explored linear regions. We can see that, while for $\gamma = 1$ the results are much worse and for $\gamma = 2$ the convergence to the final solu-

Table 6 Provably robust CIFAR-10 models

Model	ϵ	PGD-1k	PGD-10k	CW-10k	CW-100k	DF	Ours
l_2 Robust accuracy on CIFAR-10							
l_∞-MMR-*at*	0.0	0.638					
	0.25	0.504	0.504	0.490	0.490	0.498	**0.484**
	0.5	0.332	0.330	0.340	0.340	0.348	**0.314**
	0.75	0.180	0.174	0.176	0.174	0.210	**0.154**
	1.0	0.066	0.064	0.070	0.070	0.096	**0.056**
	1.25	0.036	0.034	0.032	0.032	0.050	**0.028**
l_∞-KW	0.0	0.532					
	0.25	0.390	0.390	0.376	0.376	0.374	**0.364**
	0.5	0.238	0.236	0.218	0.218	0.236	**0.216**
	0.75	0.132	0.130	0.128	0.128	0.146	**0.104**
	1.0	0.060	0.060	0.064	0.064	0.082	**0.036**
	1.25	0.018	0.018	0.032	0.032	0.036	**0.014**
l_2-MMR-*at*	0.0	0.618					
	0.25	0.418	0.418	0.404	0.404	0.412	**0.398**
	0.5	0.270	0.266	0.264	0.262	0.284	**0.252**
	0.75	0.146	0.144	0.146	0.146	0.174	**0.128**
	1.0	0.076	0.076	0.094	0.094	0.104	**0.064**
	1.25	0.032	0.032	0.050	0.050	0.054	**0.024**
l_2-KW	0.0	0.614					
	0.25	0.492	0.492	**0.478**	**0.478**	0.480	**0.478**
	0.5	0.384	0.384	0.374	0.374	0.376	**0.360**
	0.75	0.266	0.266	0.262	0.262	0.284	**0.246**
	1.0	0.172	0.172	0.176	0.176	0.190	**0.152**
	1.25	0.094	0.092	0.108	0.108	0.122	**0.082**

We report upper bounds on the robust accuracy, that is the fraction of points in the test set which are still correctly classified when any perturbation of l_2-norm smaller than or equal to ϵ is allowed in order to achieve a misclassification (a smaller robust accuracy means a stronger attack). The statistics are computed on the first 500 points of the CIFAR-10 test set

For each threshold and model we highlight in bold the lowest robust accuracy, which indicates the strongest attack

tion is significantly slower, the algorithm appears to perform similarly with γ between 3 and 9.

We also run the experiments on the GTS *plain* model. In Fig. 3 one can see that higher values of q lead to faster convergence to the final solutions (we fix $\gamma = 9$). In Fig. 4 we test different values of γ between 1 and 9 keeping constant $q = 0.8$. We notice that all the runs achieve similar performance, even for small values of γ differently from what happens on MNIST. This observation, together with the fact that about 15 regions are sufficient for the results to be almost indistinguishable from the final ones, suggests that this model is easier to attack than the one on MNIST.

Thus we choose to set $q = 0.8$ as for smaller values the runs converge slightly more slowly, while for $q = 1.0$ the maximum appears to be marginally suboptimal (anyway we want to highlight that the results are in the end almost identical).

Finally, from these ablation studies one can also appreciate the stability of the method with respect to the random part inherent to the algorithm. In fact, with the exception of the case $\gamma = 1$ on MNIST, in all the runs obtained varying the parameters q and γ, median, maximum and mean converge to the same or very similar values, meaning that sampling different points and then possibly checking different regions does not lead to inconsistent results.

6 Visualizing the Decision Boundary

While our attack runs, almost at each iteration an image lying on the decision boundary, that is the classifier outputs assigns the same (up to a tolerance) probability for the input to belong to different classes, is available. In fact, unless the linear region to which the current solution belongs does

Table 7 Large models

Model	ϵ	PGD-1k	DF	Ours
l_2 robust accuracy of large networks on CIFAR-10				
plain	0.0	0.96		
	0.05	**0.78**	0.83	**0.78**
	0.075	0.61	0.75	**0.60**
	0.1	**0.43**	0.63	0.44
	0.15	**0.18**	0.42	**0.18**
	0.2	0.08	0.26	**0.07**
l_∞-at	0.0	0.85		
	0.25	0.72	0.75	**0.71**
	0.5	**0.53**	0.62	0.54
	0.75	**0.36**	0.51	0.37
	1.0	0.23	0.44	**0.22**
	1.25	0.15	0.37	**0.12**

We report here the robust accuracy, that is an upper bound on the fraction of points in the test set which are correctly classified when any perturbation of l_2-norm smaller than or equal to ϵ is allowed (a smaller robust accuracy means a stronger attack). The statistics are computed on the first 100 points of the CIFAR-10 test set

For each threshold and model we highlight in bold the lowest robust accuracy, which indicates the strongest attack

not intersect the decision boundary, the solution of problem (10) is attained when the first constraint holds as an equality.

In Fig. 5 we show some of these intermediate solutions found while crafting an adversarial example. The first three rows are obtained attacking the plain models reported in the Sect. 5, while for the fourth to sixth row we used respectively the l_∞-*at* network on MNIST and the l_2-*at* classifiers on GTS and CIFAR-10. For every row, the first image is the starting point of our method and belongs to the training set of the respective dataset, while the second image is the point we get through the initial binary search on the segment joining the starting point and the target image for which we want to provide an adversarial perturbation (represented in the last image of each row). We also report the l_2-distance between each image and the target image, which is equivalent to the l_2-norm of the adversarial manipulation found at that iteration of the algorithm.

We can see that, apart from the starting image and the target image, all the images lie on the decision boundary. Furthermore, in many cases, although the distance from the

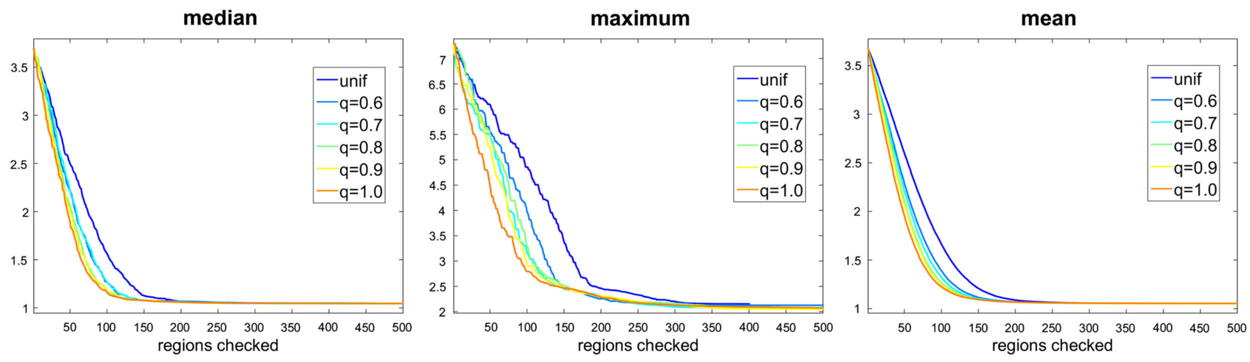


Fig. 1 Progression of our attack for different sampling schemes on MNIST. We show median (left), maximum (center) and mean (right) of the norms of the adversarial perturbations found by our attack as a function of the explored linear regions. We repeat the experiments for different values of q (see Eq. 12), represented in different colors, and with the uniform sampling scheme ($q = 0.5$) from Croce and Hein (2018) as a comparison (Color figure online)

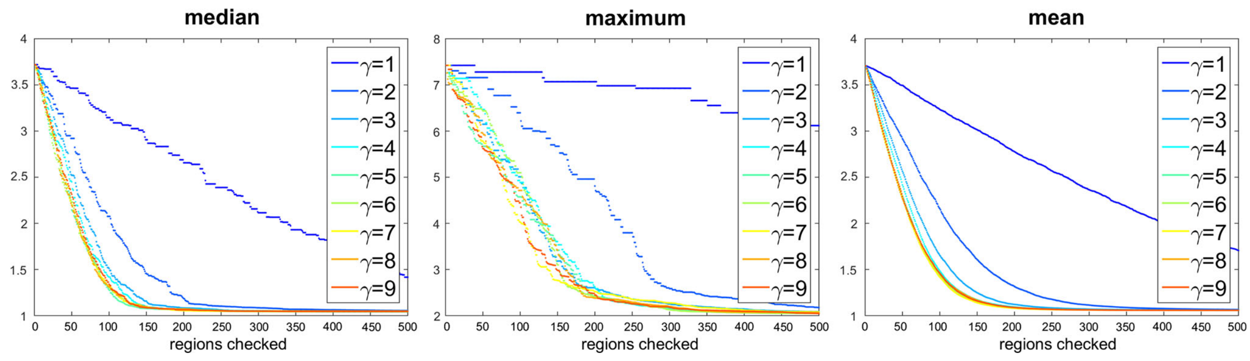


Fig. 2 Progression of our attack for different values of the parameter γ on MNIST. We show median (left), maximum (center) and mean (right) of the norms of the adversarial perturbations found by our attack as a function of the explored linear regions. We repeat the experiments for $\gamma = 1, \ldots, 9$ (see Eq. 13) represented in different colors (Color figure online)

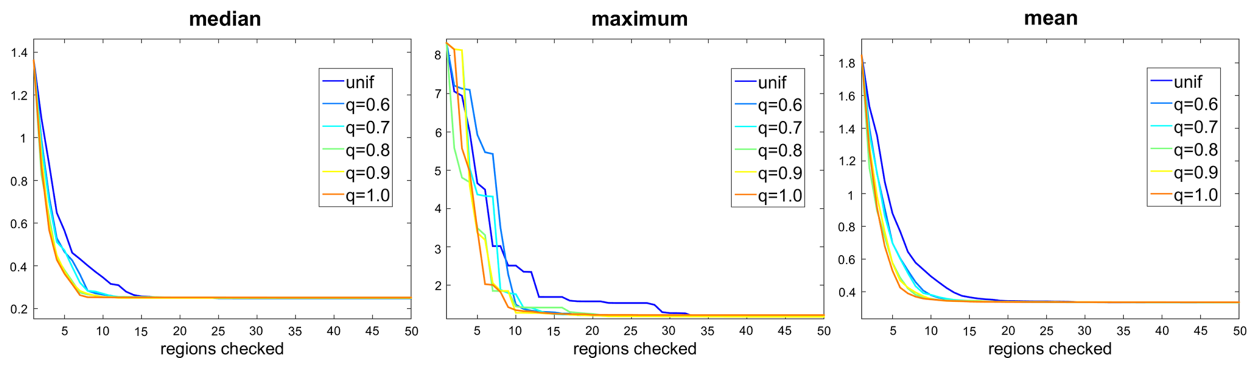


Fig. 3 Progression of our attack for different sampling schemes on GTS. We show median (left), maximum (center) and mean (right) of the norms of the adversarial perturbations found by our attack as a function of the explored linear regions. We repeat the experiments for different values of q (see Eq. 12), represented in different colors, and with the uniform sampling scheme ($q = 0.5$) from Croce and Hein (2018) as a comparison (Color figure online)

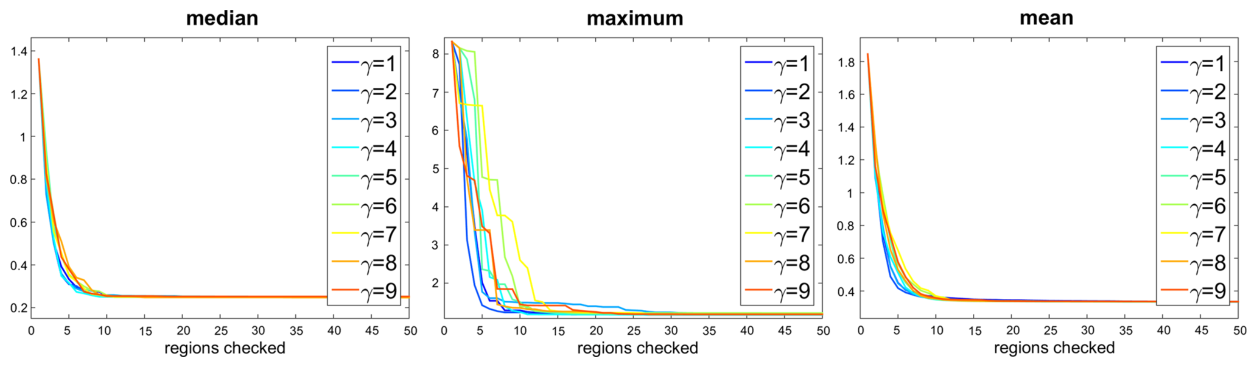


Fig. 4 Progression of our attack for different values of the parameter γ on GTS. We show median (left), maximum (center) and mean (right) of the norms of the adversarial perturbations found by our attack as a function of the explored linear regions. We repeat the experiments for $\gamma = 1, \ldots, 9$ (see Equation (13)) represented in different colors (Color figure online)

target image is notable, they are clearly assignable to a specific class, meaning that the decision boundary is still wrong showing that there is still quite some way to go if we want to achieve robustness with respect to human perception of these images.

We can also check how large the linear regions are. The first polytope $Q(y)$ our attack checks is the one containing the point y of the linear search performed as initial step of the attack between the image from the training set and the target image. We show the image y and the solution of (10) on $Q(y)$. Both images are contained in $Q(y)$ and both lie on the decision boundary. In Fig. 6 we show these two images for some cases for the GTS models. It is interesting that, although the number of polytopes is extremely large, they are still wide enough to contain images of such different appearance and with significant l_2-distance.

7 Conclusion

We extended the white-box gradient-free adversarial attack of Croce and Hein (2018) by (i) deriving a new, scalable QP solver, (ii) solving the QP problem efficiently on GPU without computing the constraint matrix explicitly, (iii) adding support for more layer types, and (iv) introducing a new attack scheme to select regions. Taken together, these improvements allowed us to attack larger and more complex neural networks in less time and finding better adversarial examples. We demonstrated the importance of evaluating robustness with our attack by showing that all the established methods for producing adversarial examples have at least one case where they estimate a robust accuracy at least 50% higher (in absolute value) than that given by the best attack, while our attack is never farther than 5%. This means that, while most of the attacks perform well on average, for all

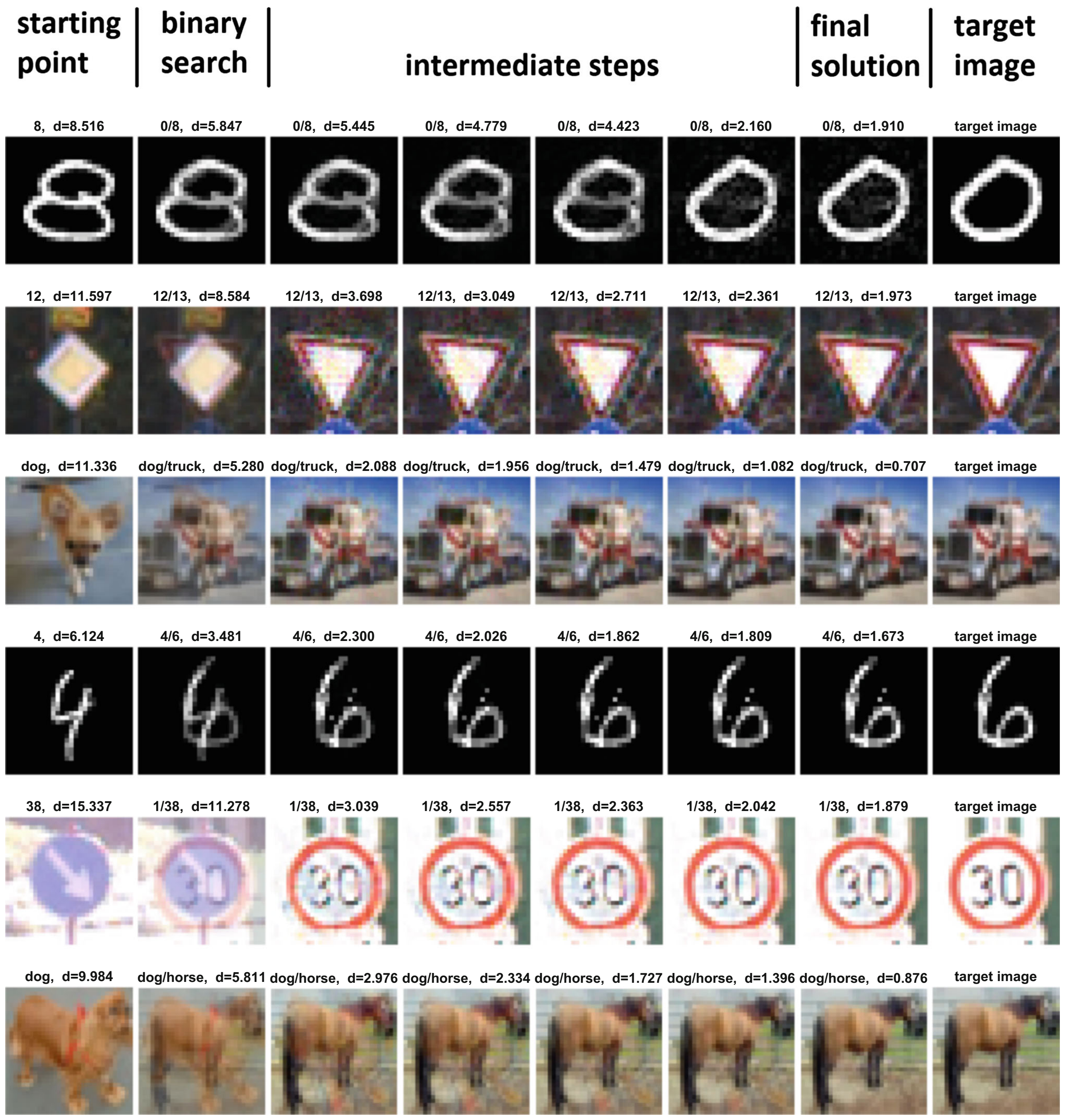


Fig. 5 Progression of our attack. In each row, the first image is from the training set, the second is obtained with the linear search towards the target image (last image) for which we create an adversarial example. The other images are intermediate adversarial images found by our attack (the seventh is the final output). Apart from the starting image and the target image, all are on the decision boundary, that is between the classes indicated on top of each picture (0/8 means it is on the decision boundary between class 0 and 8). We also report the l_2-distance between each image and the target image (d). First three rows: non-robust plain model, last three rows: l_∞ (first) and l_2 (second, third) adversarially trained models on MNIST, GTS and CIFAR-10

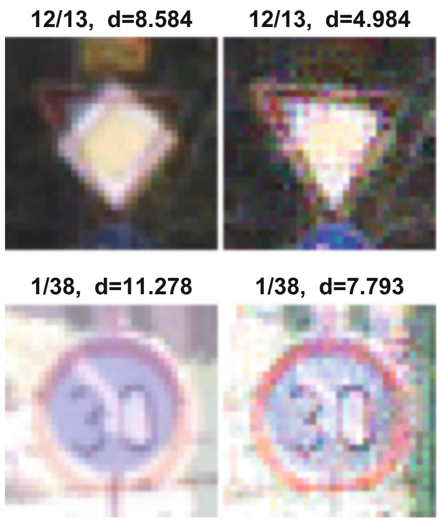


Fig. 6 The linear regions can be large. For the same cases reported in Fig. 5 for GTS we show here the image got by the initial linear search, say y, and that obtained by solving (10) on the first region $Q(y)$. This means that the two images of each row belong to the same linear region even though their appearance is quite different. This shows that some of the linear regions cover quite large parts of the input space

of them except ours there exist situations where they heavily overestimate the adversarial robustness.

Acknowledgements F. C. and M. H. acknowledge support from the BMBF through the Tübingen AI Center (FKZ: 01IS18039A) and by the DFG via Grant 389792660 as part of TRR 248 and the Excellence Cluster "Machine Learning-New Perspectives for Science". J. R. acknowledges support from the Bosch Research Foundation (Stifterverband, T113/30057/17) and the International Max Planck Research School for Intelligent Systems (IMPRS-IS).

References

Arora, R., Basuy, A., Mianjyz, P., & Mukherjee, A. (2018). Understanding deep neural networks with rectified linear unit. In *ICLR*.
Athalye, A., Carlini, N., & Wagner, D. A. (2018). Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *ICML*.
Beck, A., & Teboulle, M. (2009). A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, *2*, 183–202.
Brendel, W., Rauber, J., & Bethge, M. (2018). Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *ICLR*.
Carlini, N., & Wagner, D. (2017a). Adversarial examples are not easily detected: Bypassing ten detection methods. In *ACM workshop on artificial intelligence and security*.
Carlini, N., & Wagner, D. (2017b). Towards evaluating the robustness of neural networks. In *IEEE symposium on security and privacy*.
Chambolle, A., & Pock, T. (2011). A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, *40*(1), 120–145.
Croce, F., Andriushchenko, M., & Hein, M. (2019). Provable robustness of ReLU networks via maximization of linear regions. In *AISTATS*.
Croce, F., & Hein, M. (2018). A randomized gradient-free attack on ReLU networks. In *GCPR*.
Dalvi, N., Domingos, P., Mausam, S., & Verma, D. (2004). Adversarial classification. In *KDD*.
Goodfellow, I. J., Shlens, J., & Szegedy, C. (2015). Explaining and harnessing adversarial examples. In *ICLR*.
He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *CVPR* (pp. 770–778).
Hein, M., & Andriushchenko, M. (2017). Formal guarantees on the robustness of a classifier against adversarial manipulation. In *NIPS*.
Huang, G., Liu, Z., & Weinberger, K. Q. (2016a). Densely connected convolutional networks. In *CoRR*, abs/1608.06993.
Huang, R., Xu, B., Schuurmans, D., & Szepesvari, C. (2016b). Learning with a strong adversary. In *ICLR*.
Katz, G., Barrett, C., Dill, D., Julian, K., & Kochenderfer, M. (2017). Reluplex: An efficient SMT solver for verifying deep neural networks. In *CAV*.
Krizhevsky, A., Nair, V., & Hinton, G. (2014). Cifar-10 (canadian institute for advanced research). https://www.cs.toronto.edu/~kriz/cifar.html.
Kurakin, A., Goodfellow, I. J., & Bengio, S. (2017). Adversarial examples in the physical world. In *ICLR workshop*.
Liu, Y., Chen, X., Liu, C., & Song, D. (2017). Delving into transferable adversarial examples and black-box attacks. In *ICLR*.
Lowd, D., & Meek, C. (2005). Adversarial learning. In *KDD*.
Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Valdu, A. (2018). Towards deep learning models resistant to adversarial attacks. In *ICLR*.
Mirman, M., Gehr, T., & Vechev, M. (2018). Differentiable abstract interpretation for provably robust neural networks. In *ICML*.
Moosavi-Dezfooli, S.-M., Fawzi, A., & Frossard, P. (2016). Deepfool: A simple and accurate method to fool deep neural networks. In *CVPR* (pp. 2574–2582).
Mosbach, M., Andriushchenko, M., Trost, T., Hein, M., & Klakow, D. (2018). Logit pairing methods can fool gradient-based attacks. In *NeurIPS 2018 workshop on security in machine learning*. arXiv:1810.12042.
Narodytska, N., & Kasiviswanathan, S. P. (2016). Simple black-box adversarial perturbations for deep networks. In *CVPR 2017 Workshops*.
Nesterov, Y. E. (1983). A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Mathematics Doklady*, *27*(2), 372–376.
Papernot, N., Carlini, N., Goodfellow, I., Feinman, R., Faghri, F., & Matyasko, A., et al. (2017). cleverhans v2.0.0: An adversarial machine learning library. preprint arXiv:1610.00768.
Papernot, N., McDonald, P., Wu, X., Jha, S., & Swami, A. (2016). Distillation as a defense to adversarial perturbations against deep networks. In *IEEE symposium on security & privacy*.
Raghunathan, A., Steinhardt, J., & Liang, P. (2018). Certified defenses against adversarial examples. In *ICLR*.
Rauber, J., Brendel, W., & Bethge, M. (2017). Foolbox: A python toolbox to benchmark the robustness of machine learning models. In *ICML reliable machine learning in the wild workshop*.
Schott, L., Rauber, J., Bethge, M., & Brendel, W. (2019). Towards the first adversarially robust neural network model on MNIST. In *ICLR*.
Stallkamp, J., Schlipsing, M., Salmen, J., & Igel, C. (2012). Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, *32*, 323–332.
Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., & Goodfellow, I., et al. (2014). Intriguing properties of neural networks. In *ICLR* (pp. 2503–2511).

Tjeng, V., Xiao, K., & Tedrake, R. (2019). Evaluating robustness of neural networks with mixed integer programming. preprint arXiv:1711.07356v3.

Weng, T., Zhang, H., Chen, H., Song, Z., Hsieh, C., & Daniel, L., et al. (2018). Towards fast computation of certified robustness for ReLU networks. In *ICML*.

Wong, E., & Kolter, J. Z. (2018). Provable defenses against adversarial examples via the convex outer adversarial polytope. In *ICML*.

Wong, E., Schmidt, F., Metzen, J. H., & Kolter, J. Z. (2018). Scaling provable adversarial defenses. In *NeurIPS*.

Yuan, X., He, P., Zhu, Q., Bhat, R. R., & Li, X. (2019). Adversarial examples: Attacks and defenses for deep learning. *IEEE Transactions on Neural Networks and Learning Systems*, *30*, 2805–2824.

EagerPy: Writing Code That Works Natively with PyTorch, TensorFlow, JAX, and NumPy

The following nine pages have been published as:

Jonas Rauber, Matthias Bethge, Wieland Brendel (2020). "EagerPy: Writing Code That Works Natively with PyTorch, TensorFlow, JAX, and NumPy". *Preprint, arXiv:*2008.04175.

A summary of the motivation, results, and discussion can be found in section 2.4 on page 40.

Abstract

EagerPy is a Python framework that lets you write code that automatically works natively with PyTorch, TensorFlow, JAX, and NumPy. Library developers no longer need to choose between supporting just one of these frameworks or reimplementing the library for each framework and dealing with code duplication. Users of such libraries can more easily switch frameworks without being locked in by a specific 3rd party library. Beyond multi-framework support, EagerPy also brings comprehensive type annotations and consistent support for method chaining to any framework. The latest documentation is available online at https://eagerpy.jonasrauber.de and the code can be found on GitHub at https://github.com/jonasrauber/eagerpy.

EagerPy: Writing Code That Works Natively with PyTorch, TensorFlow, JAX, and NumPy

Jonas Rauber[1,2] JONAS.RAUBER@BETHGELAB.ORG
Matthias Bethge[1,3,†] MATTHIAS.BETHGE@BETHGELAB.ORG
Wieland Brendel[1,3,†] WIELAND.BRENDEL@BETHGELAB.ORG

[1] *Tübingen AI Center, University of Tübingen, Germany*
[2] *International Max Planck Research School for Intelligent Systems, Tübingen, Germany*
[3] *Bernstein Center for Computational Neuroscience Tübingen, Germany*
[†] *joint senior authors*

Abstract

EagerPy is a Python framework that lets you write code that automatically works natively with PyTorch, TensorFlow, JAX, and NumPy. Library developers no longer need to choose between supporting just one of these frameworks or reimplementing the library for each framework and dealing with code duplication. Users of such libraries can more easily switch frameworks without being locked in by a specific 3rd party library. Beyond multi-framework support, EagerPy also brings comprehensive type annotations and consistent support for method chaining to any framework. The latest documentation is available online at `https://eagerpy.jonasrauber.de` and the code can be found on GitHub at `https://github.com/jonasrauber/eagerpy`.

Keywords: Eager Execution, PyTorch, TensorFlow, JAX, NumPy, Python

1. Introduction

The recent advances in deep learning go hand in hand with the development of more and more deep learning frameworks. These frameworks provide high-level yet efficient APIs for automatic differentiation and GPU acceleration and make it possible to implement extremely complex and powerful deep learning models with relatively little and simple code.

Originally, many of the popular frameworks like Theano (Team et al., 2016), Caffe (Jia et al., 2014), MXNet (Chen et al., 2015), TensorFlow (Abadi et al., 2016), and CNTK (Seide and Agarwal, 2016) used a graph-based approach. The user first defines a static data flow graph that can then be efficiently differentiated, compiled, and executed on GPUs. Knowing the whole computation graph ahead of time is useful for achieving high performance. It can, however, make it difficult to debug models and to implement dynamic models with changing graphs such as RNNs.

More recently, eager execution of deep learning models has become the dominant approach in deep learning research. Instead of building a static data flow graph ahead of time, eager execution frameworks provide a define-by-run API that builds dynamic, temporary graphs on the fly. The first popular implementations of this approach were Torch (Collobert et al., 2011), Chainer (Tokui et al., 2015), and DyNet (Neubig et al., 2017). Using the define-by-run approach, they made it much easier to debug models and to implement dynamic computation graphs such as RNNs. Originally, this came at the cost of lower performance or the need to use less popular programming languages. This changed when PyTorch (Paszke et al., 2019) combined the advantages of the different eager execution frameworks, that is it combined high performance—competitive to graph-based frameworks—with an easy-to-use define-by-run *Python* API. With the introduction of TensorFlow Eager (Agrawal et al., 2019) and the switch to eager execution in TensorFlow 2, eager execution is now being used by the two dominant deep learning frameworks, PyTorch and TensorFlow.

Despite these similarities between PyTorch and TensorFlow 2, it is not easily possible to write framework-agnostic code that directly works with both frameworks. At the semantic level, the most fundamental difference lies in the APIs for automatic differentiation. In PyTorch, gradients are requested using an in-place `requires_grad_()` call, zeroed out using the `zero_grad()` function, backpropagated by calling `backward()` and finally read using the `.grad` attribute. TensorFlow offers a more high-level `GradientTape` context manager to track gradients and a `tape.gradient` function to query gradients. Beyond that, the APIs of PyTorch and TensorFlow 2 differ a lot at the syntactic level, e.g. in how they name parameters (e.g. `dim` vs. `axis`), classes (e.g. `CrossEntropyLoss` vs. `CategoricalCrossentropy`), and functions (e.g. `sum` vs. `reduce_sum`), and whether they support method chaining.

EagerPy resolves these differences between PyTorch and TensorFlow 2 by providing a single unified API that transparently maps to the different underlying frameworks without computational overhead. This is similar to how Keras (Chollet et al., 2015) unified the graph-based APIs of TensorFlow 1 and Theano. The difference is that EagerPy focuses on eager execution instead of graph building. In addition, EagerPy's approach is very transparent and allows users to easily combine framework-agnostic EagerPy code with framework-specific code. This makes it possible to gradually adopt EagerPy for individual functions.

Supporting additional eager execution frameworks in EagerPy is just a matter of specifying the necessary translations. EagerPy therefore also comes with support for JAX (Bradbury et al., 2018), a relatively new framework that has recently gotten a lot of traction thanks to its functional design, NumPy-compatible API and innovative features such as automatic vectorization. In fact, EagerPy's approach to unify the different APIs for automatic differentiation borrows a lot from the high-level functional automatic differentiation API in JAX. Finally, EagerPy also supports NumPy (Oliphant, 2006) as yet another backend, though of course NumPy neither supports automatic differentiation nor GPU acceleration.

EagerPy thus makes it possible to write framework-agnostic code that works natively with PyTorch, TensorFlow, JAX, and NumPy. In a first step, developers of new libraries profit from this because they no longer need to choose between supporting just one of these frameworks or reimplementing their library for each framework and dealing with code duplication. In a second step, the users of these libraries profit because they can more easily switch frameworks without being locked in by a specific 3rd party library.

Beyond that, even users of only a single framework can benefit from EagerPy because it brings all of EagerPy's API improvements such as comprehensive type annotations and consistent support for method chaining to each supported framework.

2. Design & Implementation

EagerPy is build with four design goals in mind. The two key goals are to provide a unified API for eager execution (Section 2.1) and to maintain the native performance of the underlying frameworks (Section 2.2). These two key goals define what EagerPy is and are core to its design. The two additional goals, a fully chainable API (Section 2.3) and comprehensive type checking support (Section 2.4), make EagerPy easier and safer to work with than the underlying framework-specific APIs. Despite these changes and improvements, we try to not unnecessarily sacrifice familiarity. Whenever it makes sense, the EagerPy API follows the standards set by NumPy, PyTorch, and JAX.

2.1 Unified API

To achieve syntactic consistency, we define an abstract `Tensor` class with the appropriate methods and an instance variable holding the native tensor, and then implement a specific subclass for each supported framework. For many operations such as `sum` or `log` this is as simple as calling the underlying framework, for others it is slightly more work. The most difficult part is unifying the automatic differentiation APIs. PyTorch uses a low-level autograd API that allows but also requires precise control over the backpropagation (see Section 1 for some details). TensorFlow uses a slightly higher-level API based on gradient tapes. And JAX uses a very high-level API based on differentiating functions. To unify them, EagerPy mimics JAX's high-level functional API and reimplements it in PyTorch and TensorFlow. EagerPy exposes it through its `value_and_grad_fn()` function (Appendix C).

Being able to write code that automatically works with all supported frameworks requires not only syntactic but also semantic unification. To guarantee this, EagerPy comes with a huge test suite that verifies the consistency between the different framework-specific subclasses. It is automatically run on all pull-requests and needs to pass before new code can be merged. The test suite also acts as the ultimate reference for which operations and which parameter combinations are supported. This avoids inconsistencies between documentation and implementation and in practice results in a test-driven development process.

2.2 Native Performance

Without EagerPy, code that wants to interface with different deep learning frameworks has to go through NumPy. This requires expensive memory copies between CPU (NumPy) and GPU (PyTorch, TensorFlow, JAX) and vice versa. Furthermore, many computations are then only executed on CPU. To avoid this, EagerPy just keeps references to the original native framework-specific tensors (e.g. the PyTorch tensor on GPU) and delegates all operations to the respective framework. This introduces virtually no computational overhead.

2.3 Fully Chainable API

Many operations such as `sum` or `square` take a tensor and return one. Often, these operations are applied sequentially, e.g. `square`, `sum`, and `sqrt` to compute the L_2 norm. In EagerPy, all operations are available as methods on the tensor object. This makes it possible to chain the operations in their natural order: `x.square().sum().sqrt()`. In contrast, NumPy, for example, requires an inverted order of operations: `np.sqrt(np.square(x).sum())`.

2.4 Type Checking

In Python 3.5, the Python syntax was extended to support type annotations (van Rossum et al., 2015). Even with type annotations, Python remains a dynamically typed programming language and all type annotations are currently ignored during runtime. They can however be checked by static code analyzers before running the code.

EagerPy comes with comprehensive type annotations of all parameters and return values and checks them using Mypy (Lehtosalo et al., 2016). This helps us catch bugs in EagerPy that would otherwise stay undetected. EagerPy users can further benefit from this by type annotating their own code and thus automatically checking it against EagerPy's function signatures. This is particularly useful because TensorFlow, NumPy, and JAX do not currently provide type annotations themselves.

3. Examples

Listing 1 shows a generic EagerPy `norm` function that can be called with a native tensor from any framework and returns its norm, again as a native tensor from the same framework. More examples with detailed explanations can be found in Appendix A and Appendix B.

```python
import eagerpy as ep

def norm(x):
    x = ep.astensor(x)  # native tensor to EagerPy tensor
    result = x.square().sum().sqrt()
    return result.raw  # EagerPy tensor to native tensor
```

Listing 1: A framework-agnostic `norm` function

4. Use Cases

Foolbox (Rauber et al., 2017) is a highly popular adversarial attacks library (more than 220 citations and 1.500 stars on GitHub) that has long supported different deep learning frameworks through a common NumPy interface. With Foolbox 3.0 aka Foolbox Native (Rauber et al., 2020), it has been completely reimplemented using EagerPy. It now achieves native performance while still supporting different frameworks using a single code base.

While EagerPy was specifically created with Foolbox in mind, it is now being adopted by other libraries as well. GUDHI (Maria et al., 2014), for example, is a library for computational topology. It uses EagerPy to support automatic differentiation in PyTorch, TensorFlow, and JAX without code duplication. Moreover, EagerPy also makes it easy to share framework-agnostic reference implementations of algorithms (Rauber and Bethge, 2020).

5. Conclusion

EagerPy provides a unified API to PyTorch, TensorFlow, JAX, and NumPy without sacrificing performance. Automatic tests guarantee consistency across frameworks. Automatic deployments encourage rapid releases. Comprehensive type annotations help detecting bugs early. Consistent support for method chaining enables beautiful code. And being the foundation of the popular Foolbox library ensures continuous development.

Acknowledgments

J.R. acknowledges support from the Bosch Research Foundation (Stifterverband, grant T113/30057/17) and the International Max Planck Research School for Intelligent Systems (IMPRS-IS). This work was supported by the German Federal Ministry of Education and Research (BMBF): Tübingen AI Center, FKZ: 01IS18039A, and by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior/Interior Business Center (DoI/IBC) contract number D16PC00003. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/IBC, or the U.S. Government.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: A system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

Akshay Agrawal, Akshay Naresh Modi, Alexandre Passos, Allen Lavoie, Ashish Agarwal, Asim Shankar, Igor Ganichev, Josh Levenberg, Mingsheng Hong, Rajat Monga, et al. TensorFlow Eager: A multi-stage, Python-embedded DSL for machine learning. In *Systems for Machine Learning (SysML) 2019*, 2019.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems. In *LearningSys Workshop at Neural Information Processing Systems 2015*, 2015.

François Chollet et al. Keras. `https://keras.io`, 2015.

Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A Matlab-like environment for machine learning. In *BigLearn, NeurIPS workshop*, 2011.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678, 2014.

Jukka Lehtosalo et al. Mypy: Optional static typing for python, 2016. URL `https://github.com/python/mypy`.

Clément Maria, Jean-Daniel Boissonnat, Marc Glisse, and Mariette Yvinec. The gudhi library: Simplicial complexes and persistent homology. In Hoon Hong and Chee Yap, editors, *Mathematical Software – ICMS 2014*, pages 167–174, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. ISBN 978-3-662-44199-2.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. DyNet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*, 2017.

Travis Oliphant. NumPy: A guide to NumPy. USA: Trelgol Publishing, 2006. URL `http://www.numpy.org/`.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.

Jonas Rauber and Matthias Bethge. Fast differentiable clipping-aware normalization and rescaling. *arXiv preprint arXiv:2007.07677*, 2020. URL `https://github.com/jonasrauber/clipping-aware-rescaling`.

Jonas Rauber, Wieland Brendel, and Matthias 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 `https://arxiv.org/abs/1707.04131`.

Jonas Rauber, Roland Zimmermann, Matthias Bethge, and Wieland Brendel. Foolbox Native: Fast adversarial attacks to benchmark the robustness of machine learning models in PyTorch, TensorFlow, and JAX. Manuscript in preparation, 2020. URL `https://foolbox.jonasrauber.de`.

Frank Seide and Amit Agarwal. CNTK: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2135–2135, 2016.

The Theano Development Team, Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 2016.

Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. Chainer: a next-generation open source framework for deep learning. In *LearningSys Workshop at Neural Information Processing Systems 2015*, 2015.

Guido van Rossum, Jukka Lehtosalo, and Łukasz Langa. Type hints. PEP 484, Python Software Foundation, 2015. URL `https://www.python.org/dev/peps/pep-0484/`.

Appendix A. Converting Between EagerPy and Native Tensors

A native tensor could be a PyTorch GPU or CPU tensor (Listing 2), a TensorFlow tensor (Listing 3), a JAX array (Listing 4), or a NumPy array (Listing 5).

```
import torch
x = torch.tensor([1., 2., 3., 4., 5., 6.])
```

Listing 2: A native PyTorch tensor

```
import tensorflow as tf
x = tf.constant([1., 2., 3., 4., 5., 6.])
```

Listing 3: A native TensorFlow tensor

```
import jax.numpy as np
x = np.array([1., 2., 3., 4., 5., 6.])
```

Listing 4: A native JAX array

```
import numpy as np
x = np.array([1., 2., 3., 4., 5., 6.])
```

Listing 5: A native NumPy array

No matter which native tensor you have, it can always be turned into the appropriate EagerPy tensor using `ep.astensor`. This will automatically wrap the native tensor with the correct EagerPy tensor class. The original native tensor can always be accessed using the `.raw` attribute. A full example is shown in Listing 6.

```
# x should be a native tensor (see above)
# for example:
import torch
x = torch.tensor([1., 2., 3., 4., 5., 6.])

# Any native tensor can easily be turned into an EagerPy tensor
import eagerpy as ep
x = ep.astensor(x)

# Now we can perform any EagerPy operation
x = x.square()

# And convert the EagerPy tensor back into a native tensor
x = x.raw
# x will now again be a native tensor (e.g. a PyTorch tensor)
```

Listing 6: Converting between EagerPy and native tensors

Especially in functions, it is common to convert all inputs to EagerPy tensors. This could be done using individual calls to `ep.astensor`, but using `ep.astensors` this can be written even more compactly (Listing 7).

```
# x, y should be a native tensors (see above)
# for example:
import torch
x = torch.tensor([1., 2., 3.])
y = torch.tensor([4., 5., 6.])

import eagerpy as ep
x, y = ep.astensors(x, y)  # works for any number of inputs
```

Listing 7: Converting multiple native tensors at once

Appendix B. Implementing Generic Framework-Agnostic Functions

Using the conversion functions shown in Appendix A, we can already define a simple framework-agnostic function (Listing 8). This function can be called with a native tensor from any framework and it will return the norm of that tensor, again as a native tensor from that framework (Listing 9, Listing 10).

```
import eagerpy as ep

def norm(x):
    x = ep.astensor(x)
    result = x.square().sum().sqrt()
    return result.raw
```

Listing 8: A simple framework-agnostic `norm` function

```
import torch
norm(torch.tensor([1., 2., 3.]))
# tensor(3.7417)
```

Listing 9: Calling the `norm` function using a PyTorch tensor

```
import tensorflow as tf
norm(tf.constant([1., 2., 3.]))
# <tf.Tensor: shape=(), dtype=float32, numpy=3.7416575>
```

Listing 10: Calling the `norm` function using a TensorFlow tensor

If we would call the function in Listing 8 with an EagerPy tensor, the `ep.astensor` call would simply return its input. The `result.raw` call in the last line would however still extract the underlying native tensor. Often it is preferably to implement a generic function that not only transparently handles any native tensor but also EagerPy tensors, that is the return type should always match the input type. This is particularly useful in libraries like Foolbox that allow users to work with EagerPy and native tensors. To achieve that, EagerPy comes with two derivatives of the above conversion functions: `ep.astensor_` and `ep.astensors_`. Unlike their counterparts without an underscore, they return an additional inversion function that restores the input type. If the input to `astensor_` is a native tensor, `restore_type` will be identical to `.raw`, but if the original input was an EagerPy tensor, `restore_type` will not call `.raw`. With that, we can write generic framework-agnostic functions that work transparently for any input (Listing 11, Listing 12).

```
import eagerpy as ep

def norm(x):
    x, restore_type = ep.astensor_(x)
    result = x.square().sum().sqrt()
    return restore_type(result)
```

Listing 11: An improved framework-agnostic `norm` function

```
import eagerpy as ep

def example(x, y, z):
    (x, y, z), restore_type = ep.astensors_(x, y, z)
    result = (x + y) * z
    return restore_type(result)
```

Listing 12: Converting and restoring multiple inputs using `ep.astensors_`

Appendix C. Automatic Differentiation in EagerPy

EagerPy uses a functional approach to automatic differentiation. You first define a function that will then be differentiated with respect to its inputs. This function is then passed to `ep.value_and_grad` to evaluate both the function and its gradient (Listing 13). More generally, you can also use `ep.value_aux_and_grad` if your function has additional auxiliary outputs and `ep.value_and_grad_fn` if you want the gradient function without immediately evaluating it at some point x.

```
import torch
x = torch.tensor([1., 2., 3.])

# The following code works for any framework, not just Pytorch!

import eagerpy as ep
x = ep.astensor(x)

def loss_fn(x):
    # this function takes and returns an EagerPy tensor
    return x.square().sum()

print(loss_fn(x))
# PyTorchTensor(tensor(14.))

print(ep.value_and_grad(loss_fn, x))
# (PyTorchTensor(tensor(14.)), PyTorchTensor(tensor([2., 4., 6.])))
```

Listing 13: Using `ep.value_and_grad` for automatic differentiation in EagerPy

Foolbox Native: Fast adversarial attacks to benchmark the robustness of machine learning models in PyTorch, TensorFlow, and JAX

The following three pages have been published as:

Jonas Rauber, Roland Zimmermann, Matthias Bethge, Wieland Brendel (2020). "Foolbox Native: Fast adversarial attacks to benchmark the robustness of machine learning models in PyTorch, TensorFlow, and JAX". *Journal of Open Source Software, 5(53), 2607.*

A summary of the motivation, results, and discussion can be found in section 2.4 on page 40.

Abstract

Machine learning has made enormous progress in recent years and is now being used in many real-world applications. Nevertheless, even state-of-the-art machine learning models can be fooled by small, maliciously crafted perturbations of their input data. Foolbox is a popular Python library to benchmark the robustness of machine learning models against these adversarial perturbations. It comes with a huge collection of state-of-the-art adversarial attacks to find adversarial perturbations and thanks to its framework-agnostic design it is ideally suited for comparing the robustness of many different models implemented in different frameworks. Foolbox 3 aka Foolbox Native has been rewritten from scratch to achieve native performance on models developed in PyTorch, TensorFlow, and JAX, all with one codebase without code duplication.

The Journal of Open Source Software

Foolbox Native: Fast adversarial attacks to benchmark the robustness of machine learning models in PyTorch, TensorFlow, and JAX

Jonas Rauber[1, 2], **Roland Zimmermann**[1, 2], **Matthias Bethge**[*1, 3], **and Wieland Brendel**[1, 3]

1 Tübingen AI Center, University of Tübingen, Germany **2** International Max Planck Research School for Intelligent Systems, Tübingen, Germany **3** Bernstein Center for Computational Neuroscience Tübingen, Germany

DOI: 10.21105/joss.02607

Software

- Review
- Repository
- Archive

Editor: Yuan Tang

Reviewers:

- @GregaVrbancic
- @ethanwharris

Submitted: 10 August 2020
Published: 27 September 2020

Summary

Machine learning has made enormous progress in recent years and is now being used in many real-world applications. Nevertheless, even state-of-the-art machine learning models can be fooled by small, maliciously crafted perturbations of their input data. Foolbox is a popular Python library to benchmark the robustness of machine learning models against these adversarial perturbations. It comes with a huge collection of state-of-the-art adversarial attacks to find adversarial perturbations and thanks to its framework-agnostic design it is ideally suited for comparing the robustness of many different models implemented in different frameworks. Foolbox 3 aka Foolbox Native has been rewritten from scratch to achieve native performance on models developed in PyTorch (Paszke et al., 2019), TensorFlow (Abadi et al., 2016), and JAX (Bradbury et al., 2018), all with one codebase without code duplication.

Statement of need

Evaluating the adversarial robustness of machine learning models is crucial to understanding their shortcomings and quantifying the implications on safety, security, and interpretability. Foolbox Native is the first adversarial robustness toolbox that is both fast and framework-agnostic. This is important because modern machine learning models such as deep neural networks are often computationally expensive and are implemented in different frameworks such as PyTorch and TensorFlow. Foolbox Native combines the framework-agnostic design of the original Foolbox (Rauber, Brendel, & Bethge, 2017) with real batch support and native performance in PyTorch, TensorFlow, and JAX, all using a single codebase without code duplication. To achieve this, all adversarial attacks have been rewritten from scratch and now use EagerPy (Rauber et al., 2020) instead of NumPy (Oliphant, 2006) to interface *natively* with the different frameworks.

This is great for both users and developers of adversarial attacks. Users can efficiently evaluate the robustness of different models in different frameworks using the same set of state-of-the-art adversarial attacks, thus obtaining comparable results. Attack developers do not need to choose between supporting just one framework or reimplementing their new adversarial attack multiple times and dealing with code duplication. In addition, they both benefit from the comprehensive type annotations (Rossum, Lehtosalo, & Langa, 2015) in Foolbox Native to catch bugs even before running their code.

*joint senior authors

The combination of being framework-agnostic and simultaneously achieving native performance sets Foolbox Native apart from other adversarial attack libraries. The most popular alternative to Foolbox is CleverHans[1]. It was the first adversarial attack library and has traditionally focused solely on TensorFlow (plans to make it framework-agnostic *in the future* have been announced). The original Foolbox was the second adversarial attack library and the first one to be framework-agnostic. Back then, this was achieved at the expense of performance. The adversarial robustness toolbox ART[2] is another framework-agnostic adversarial attack library, but it is conceptually inspired by the original Foolbox and thus comes with the same performance trade-off. AdverTorch[3] is a popular adversarial attack library that was inspired by the original Foolbox but improved its performance by focusing soley on PyTorch. Foolbox Native is our attempt to improve the performance of Foolbox without sacrificing the framework-agnostic design that is crucial to consistently evaluate the robustness of different machine learning models that use different frameworks.

Use Cases

Foolbox was designed to make adversarial attacks easy to apply even without expert knowledge. It has been used in numerous scientific publications and has already been cited more than 220 times. On GitHub it has received contributions from several developers and has gathered more than 1.500 stars. It provides the reference implementations of various adversarial attacks, including the Boundary Attack (Brendel, Rauber, & Bethge, 2018), the Pointwise Attack (Schott, Rauber, Bethge, & Brendel, 2019), clipping-aware noise attacks (Rauber & Bethge, 2020), the Brendel Bethge Attack (Brendel, Rauber, Kümmerer, Ustyuzhaninov, & Bethge, 2019), and the HopSkipJump Attack (Chen, Jordan, & Wainwright, 2020), and is under active development since 2017.

Acknowledgements

J.R. acknowledges support from the Bosch Research Foundation (Stifterverband, T113/30057/17) and the International Max Planck Research School for Intelligent Systems (IMPRS-IS). This work was supported by the German Federal Ministry of Education and Research (BMBF): Tübingen AI Center, FKZ: 01IS18039A, and by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior/Interior Business Center (DoI/IBC) contract number D16PC00003. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/IBC, or the U.S. Government.

We thank all contributors to Foolbox, in particular Behar Veliqi, Evgenia Rusak, Jianbo Chen, Rene Bidart, Jerome Rony, Ben Feinstein, Eric R Meissner, Lars Holdijk, Lukas Schott, Carl-Johann Simon-Gabriel, Apostolos Modas, William Fleshman, Xuefei Ning, and many others.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., et al. (2016). TensorFlow: A system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)* (pp. 265–283).

[1] https://github.com/tensorflow/cleverhans
[2] https://github.com/Trusted-AI/adversarial-robustness-toolbox
[3] https://github.com/BorealisAI/advertorch

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., & Wanderman-Milne, S. (2018). JAX: Composable transformations of Python+NumPy programs. Retrieved from http://github.com/google/jax

Brendel, W., Rauber, J., & Bethge, M. (2018). Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *International conference on learning representations*. Retrieved from https://openreview.net/forum?id=SyZI0GWCZ

Brendel, W., Rauber, J., Kümmerer, M., Ustyuzhaninov, I., & Bethge, M. (2019). Accurate, reliable and fast robustness evaluation. In *Advances in neural information processing systems 32*.

Chen, J., Jordan, M. I., & Wainwright, M. J. (2020). HopSkipJumpAttack: A query-efficient decision-based attack. In *2020 ieee symposium on security and privacy (sp)* (pp. 1277–1294). IEEE. doi:10.1109/SP40000.2020.00045

Oliphant, T. (2006). NumPy: A guide to NumPy. USA: Trelgol Publishing. Retrieved from http://www.numpy.org/

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., et al. (2019). PyTorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems* (pp. 8026–8037).

Rauber, J., & Bethge, M. (2020). Fast differentiable clipping-aware normalization and rescaling. *arXiv preprint arXiv:2007.07677*. Retrieved from https://github.com/jonasrauber/clipping-aware-rescaling

Rauber, J., Bethge, M., & Brendel, W. (2020). EagerPy: Writing code that works natively with PyTorch, TensorFlow, JAX, and NumPy. *arXiv preprint arXiv:2008.04175*. Retrieved from https://eagerpy.jonasrauber.de

Rauber, J., Brendel, W., & Bethge, M. (2017). 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*. Retrieved from https://arxiv.org/abs/1707.04131

Rossum, G. van, Lehtosalo, J., & Langa, Ł. (2015). *Type hints* (PEP No. 484). Python Software Foundation. Retrieved from https://www.python.org/dev/peps/pep-0484/

Schott, L., Rauber, J., Bethge, M., & Brendel, W. (2019). Towards the first adversarially robust neural network model on MNIST. In *International conference on learning representations*. Retrieved from https://openreview.net/forum?id=S1EHOsC9tX

Acknowledgments

The work presented in this dissertation would not have been possible without the amazing people who accompanied me along the way.

Matthias Bethge first took me on as a young student when I just started my master's degree. From the first day, he provided an environment where I felt welcome, empowered, and valued. I am grateful for his trust, for the inspiration and freedom he provided, for all his advice, mentoring, and support over the years, for sharing his enthusiasm and energy, and for the great time we had together.

Wieland Brendel was the one who showed and taught me how to turn ideas into results. I am grateful for his trust, for all the time we spent iterating ideas, for his patience when things took longer than expected, for his constant support and guidance, and for the chance to work together on so many exciting and impactful projects.

I also want to thank all my other co-authors for many productive collaborations. I want to give special thanks to *Lukas Schott*, *Robert Geirhos*, *Francesco Croce*, and *Matthias Hein*, without whom our joint papers would not exist.

I am grateful to all members of the *Bethge lab*—and the *Schölkopf group* when they generously hosted us for a year—for providing a great environment, for teaching me many new things and explaining the latest papers, and for exciting discussions and the fun we had. Special thanks to *Heike König* for her constant support, always closing the gap between my wishful thinking and the hard reality of the university's administration, and to *Judith Lam* for her constant cheerfulness and for always helping out.

I thank the *Bosch Research Foundation* for its generous funding, and its members as well as my advisory committee—*Matthias* and *Wieland*, *Michael Black*, and *Matthias Hein*—for listening to my progress and providing valuable feedback. Furthermore, I thank *Zeynep Akata*, *Alois Knoll*, and *Matthias* for reviewing my dissertation.

I am grateful to my friends for making my time in Tübingen so enjoyable and for many lasting memories. Special thanks to *Timothy Gebhard*, *Mara Weis*, and *Santiago Cadena* for always having an open ear and to *Felix Riese* for sharing his valuable tips.

Finally, I am deeply grateful to my parents *Michaela* and *Lothar* for their unlimited support and everything they did for me.