Optimization and Optimizers for Adversarial Robustness

Hengyue Liang[1*], Buyun Liang[2], Le Peng[2], Ying Cui[3], Tim Mitchell[4] and Ju Sun[2*]

[1]Department of Electrical & Computer Engineering, University of Minnesota.
[2]Department of Computer Science & Engineering, University of Minnesota.
[3]Department of Industrial & Systems Engineering, University of Minnesota.
[4]Department of Computer Science, Queens College, City University of New York.

*Corresponding author(s). E-mail(s): liang656@umn.edu; jusun@umn.edu;
Contributing authors: liang664@umn.edu; peng0347@umn.edu; yingcui@umn.edu;
tmitchell@qc.cuny.edu;

Abstract

Empirical evaluation of deep learning models against adversarial perturbations entails solving nontrivial constrained optimization problems. Existing numerical algorithms commonly used in practice to solve these problems predominantly rely on using projected gradient methods and mostly handle perturbations modeled by $\boldsymbol{\ell_1}$, $\boldsymbol{\ell_2}$ and $\boldsymbol{\ell_\infty}$ distance metrics. In this paper, we introduce a novel algorithmic framework that blends a general-purpose constrained-optimization solver `PyGRANSO` **W**ith **C**onstraint-**F**olding (PWCF), which can add more reliability and generality to the state-of-the-art (SOTA) algorithms (e.g., `AutoAttack`). Regarding *reliability*, PWCF provide solutions with stationarity measures to assess the solution quality, and is generally free from delicate hyperparameter tuning. For *generality*, PWCF can handle much more general perturbation models (e.g., modeled by any piece-wise differentiable metric) which are inaccessible to the existing project gradient methods. With PWCF, we further explore the distinct solution patterns found by various combinations of losses, perturbation models, and optimization algorithms used in robustness evaluation, and discuss the possible implications of these patterns on the current robustness evaluation and adversarial training.

Keywords: deep learning, deep neural networks, adversarial robustness, adversarial attack, adversarial training, minimal distortion radius, constrained optimization, sparsity

1 Introduction

In visual recognition, deep neural networks (DNNs) are not robust against perturbations that are easily discounted by human perception—either adversarially constructed or naturally occurring [1–9]. A popular way to formalize robustness is by finding perturbations via solving the following *max-form* constrained optimization problem [10, 11]:

$$\begin{aligned} &\max_{\boldsymbol{x}'} \ \ell\left(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x}')\right) \\ &\text{s.t. } d\left(\boldsymbol{x}, \boldsymbol{x}'\right) \leq \varepsilon\,, \quad \boldsymbol{x}' \in [0,1]^n \end{aligned} \tag{1}$$

Here, f_θ is the DNN model; $\boldsymbol{x}'$ is a perturbed version of $\boldsymbol{x}$ with an allowable perturbation radius ε measured by the distance metric d; $\boldsymbol{x}'$ is also encoded as a valid image by the box constraint (n is the number of pixels). Another popular formalism of robustness is by means of a *min-form*

constrained optimization problem:

$$\begin{aligned} &\min_{\boldsymbol{x}'} \; d\left(\boldsymbol{x}, \boldsymbol{x}'\right) \\ &\text{s.t.} \; \max_{i \neq y} f^i_{\boldsymbol{\theta}}(\boldsymbol{x}') \geq f^y_{\boldsymbol{\theta}}(\boldsymbol{x}') \;,\; \boldsymbol{x}' \in [0,1]^n \end{aligned} \tag{2}$$

where y is the true class of $\boldsymbol{x}$, and is first introduced in [1], earlier than Form. (1). Early works assume the distance metric d in both Form. (1) and (2) to be the ℓ_p norm ball, where $p = 1, 2, \infty$ are popular choices [2, 11]. Recent works have also modeled nontrivial transformations using metrics other than ℓ_p norms [3–9, 12], to capture visually realistic perturbations and generate adversaries with more varieties.

Form. (1) is usually associated with an attacker in the adversarial setup, where a hacker takes advantage of Form. (1) to make $f_{\boldsymbol{\theta}}$ misclassify $\boldsymbol{x}$, as solutions to Form. (1) lead to worst-case perturbations to fool $f_{\boldsymbol{\theta}}$. Thus, it is popular to perform robust evaluation (RE) via Form. (1)—generating perturbations over a evaluation dataset and reporting the classification accuracy over these generated perturbations (robust accuracy). Form. (1) also naturally motivates *adversarial training* (AT) in the following min-max formulation [2, 10, 11]:

$$\min_{\boldsymbol{\theta}} \; \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}} \max_{\boldsymbol{x}' \in \Delta(\boldsymbol{x})} \ell\left(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x}')\right) \tag{3}$$

where $\Delta(\boldsymbol{x}) = \{\boldsymbol{x}' \in [0,1]^n : d(\boldsymbol{x}, \boldsymbol{x}') \leq \varepsilon\}$, in contrast to the classical goal of supervised learning:

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}} \, \ell\left(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x})\right) \tag{4}$$

as a disciplined framework to achieve adversarial robustness (AR). Form. (2) is also a popular choice of RE in terms of providing robust accuracy[1] [14–16], as any solution will produce a $\boldsymbol{x}'$ that causes $f_{\boldsymbol{\theta}}$ to mis-classify its label. More importantly, solutions to Form. (2) also return *robustness radius* corresponding to each sample. This makes Form. (2) suitable in testing the limitations of a model—less robust samples can be identified by smaller robustness radius, but is often overlooked in existing literatures.

Despite the popularity of Form. (1) and (2), solving them is not easy. For Form. (1), the objective is non-concave for typical choices of loss ℓ and model $f_{\boldsymbol{\theta}}$; for non-ℓ_p metrics d, $\boldsymbol{x}'$ often belongs to a complicated non-convex set. In practice, there are two major lines of algorithms: **1) direct numerical maximization** that takes (sub-)differentiable ℓ and $f_{\boldsymbol{\theta}}$, and tries direct maximization, e.g., using gradient-based methods [11, 15]. This often only produces a suboptimal solution and can lead to overoptimistic RE. **2) upper-bound maximization** that constructs tractable upper bounds for the margin loss:

$$\ell_{\text{ML}} = \max_{i \neq y} f^i_{\boldsymbol{\theta}}(\boldsymbol{x}') - f^y_{\boldsymbol{\theta}}(\boldsymbol{x}') \tag{5}$$

and then optimizes against the upper bounds [17–22]. Improving the tightness of the upper bound while maintaining tractability remains an active area of research. It is also worth mentioning that since $\ell_{\text{ML}} \geq 0$ implies an attack success, family **2)** can also be used to RE by providing an underestimate of robust accuracy as well [17–27]. For Form. (2), for small-scale, restricted $f_{\boldsymbol{\theta}}$ and selected d, the problem can be solved exactly by mixed integer programming [28–30]. For general $f_{\boldsymbol{\theta}}$ and selected d, lower bounds of the robustness radius can be computed [31–34]. But in practice, Form. (2) is heuristically solved by gradient-based methods or iterative linearization [1, 14, 16, 35–38] to retain an upper bound of the robustness radius, see Section 2.

Although performing RE with numerical methods is widely used in practice [39], there are two major limitations in the existing evaluation methods:

- **Lack of reliability:** existing numerical methods typically do not assess the solution quality and there are little access to the trustworthiness of the solution found. Fig. 1 shows that the default iteration budget used in the RE package `AutoAttack` mostly leads to pre-mature termination of the optimization process, and it is indeed tricky to set the termination iterations as they vary from sample to sample.
- **Lack of generality:** existing projected gradient methods are mainly applied to problems where d is ℓ_1, ℓ_2 or ℓ_∞ norm, while methods to deal with other distance metric d are lacking. In fact, the popular RE benchmark

[1] One can also perform AT using Form. (2) via bi-level optimization; see, e.g., [13].

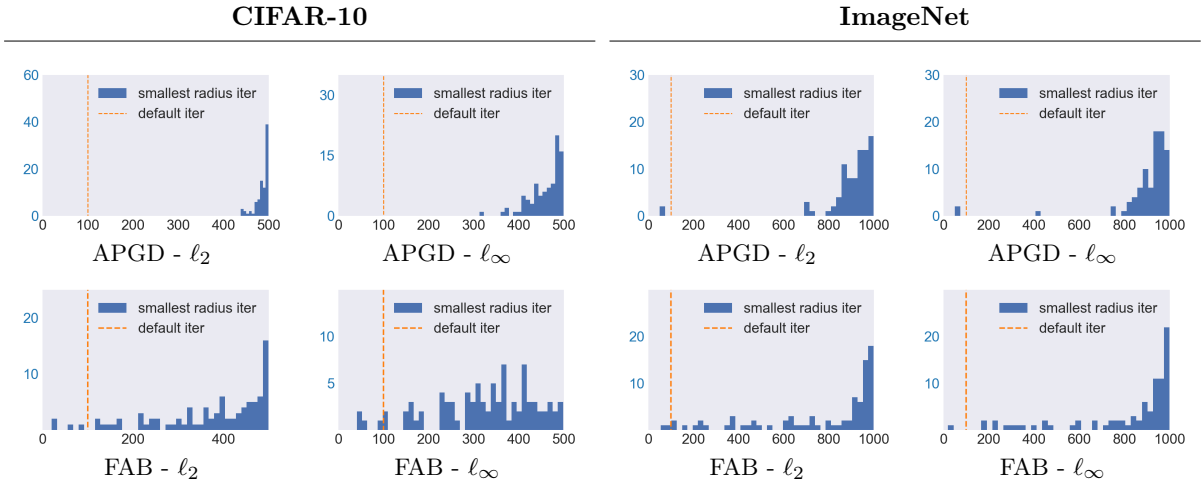


Fig. 1: Histogram of iterations where APGD (to solve Form. (1)) and FAB (to solve Form. (2)) in `AutoAttack` find the best objective values (blue bars) for 100 sample images. The dashed orange line in each figure depicts the termination iteration. We set the termination iteration to be 500 for Cifar-10 images and 1000 for ImageNet images. In each plot, we can conclude that 1) the iterations where the best results are achieved by APGD and FAB vary from sample to sample; 2) the best results are achieved mostly after the default maximal iteration used in `AutoAttack`; 3) the maximal iterations we set in these experiments are probably still not sufficient for most samples.

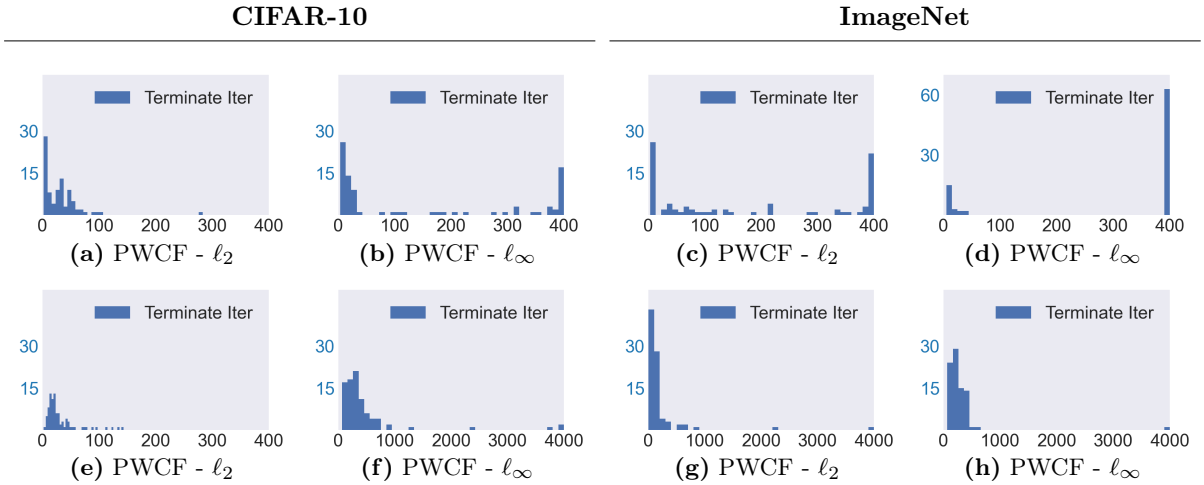


Fig. 2: Histogram of iterations where PWCF terminates in solving Form. (1)—**(a)-(d)**, and Form. (2)—**(e)-(h)**. We also set a maximal termination iteration 400 for (**(a)-(d)**) and 4000 for (**(e)-(h)**) in this test.

`robustbench` [39] only has ℓ_2 and ℓ_∞ leaderboards; and the most studied adversarial attack model is the ℓ_∞ attack [40].

Both limitations are important to address, as reliability relates to how much we can trust the results, and generality determines the usage—to test the robustness of the models with more varieties of attacks, such as real-world-like perturbations, there is a need for solvers to Form. (1) and (2) that can handle more varieties of distance metrics d.

Our contributions In this paper, we focus on numerical optimization of Form. (1) and (2)[2].

[2]Both formulations have their targeted versions: e.g., replacing $\max f^i_{\boldsymbol{\theta}}(\boldsymbol{x}')$ by $f^i_{\boldsymbol{\theta}}(\boldsymbol{x}')$ in Form. (2), and similarly in Form. (1) using margin loss Eq. (5), which are indeed simpler versions

We first provide a general solver that can handle any piece-wise differentiable distance metric d, and has a principled stopping criterion to assess the solution quality:

1. We adapt the constrained optimization solver `PyGRANSO` [41, 42] **W**ith **C**onstraint-**F**olding (PWCF), which is crucial for boosting the speed and solution quality of `PyGRANSO`—see Section 3.1.
2. As `PyGRANSO` is equipped with a rigorous line-search rule and stopping criterion, PWCF terminates the optimization by assessing the constraint violations and stationarity, thus providing solution quality guarantees—see Section 3.3.
3. We show that PWCF can not only perform comparably to the state-of-the-art (SOTA) RE packages on ℓ_1, ℓ_2, and ℓ_∞ attacks, e.g., `AutoAttack` [15], serving as a *reliable complement* to them, but also handle general distance metric d *other than the popular but limited* ℓ_1, ℓ_2, *and* ℓ_∞—beyond the reach of existing numerical methods, see Section 4.

Then we compare the solutions to Form. (1) and (2) from different solvers and discuss the implications:

1. Different combinations of distance metric d, loss ℓ and optimization solver used in solving Form. (1) and (2) can induce different sparsity patterns. In terms of numerical RE, combining solutions with different patterns is complementary; any evaluation based on a small set of algorithms may not be sufficient, see Section 5 and Section 6.1.
2. The robust accuracy used in RE via solving Form. (1) may be a bad robustness metric. Instead, performing RE via Form. (2) can be more beneficial, see Section 7.1.
3. Due to the pattern difference in solving Form. (1) with numerical methods, the common practice of adversarial training via Form. (3) may not achieve adversarial robustness, see Section 7.2.

of the untargetted Form. (2) and Form. (1). Thus, we will only focus on the untargetted formulations in this paper.

2 Technical background

2.1 Numerical maximization of Form. (1)

Form. (1) is often solved by the projected gradient descent (PGD)[3] method. The basic update reads

$$\boldsymbol{x}'_{new} = \mathcal{P}_{\Delta(\boldsymbol{x})}\left(\boldsymbol{x}'_{old} + t\nabla\ell(\boldsymbol{x}'_{old})\right) \tag{6}$$

where $\mathcal{P}_{\Delta(\boldsymbol{x})}$ is the projection operator onto the feasible set $\Delta(\boldsymbol{x})$. When $\Delta(\boldsymbol{x}) = \{\boldsymbol{x}' \in [0,1]^n : \|\boldsymbol{x}' - \boldsymbol{x}\|_p \leq \varepsilon\}$ with $p = 1, \infty$, $\mathcal{P}_{\Delta(\boldsymbol{x})}$ takes simple forms. For $p = 2$, sequential projection onto the box first and then the norm ball at least finds a feasible solution (see our clarification of these projections in Section A). Therefore, PGD is viable for these cases. For *other choices of* p and *general non-*ℓ_p *metrics* d where analytical projected gradients are hard to derive, existing algorithms do not apply. For practical PGD methods, previous works have shown that the solution quality is sensitive to hyperparameters tuning, e.g., step-size schedule and iteration budget [15, 43, 44]. The SOTA PGD variants, APGD-CE and APGD-DLR, try to make the tuning process automatic by combining a heuristic adaptive step-size schedule and momentum acceleration under fixed iteration budget [15]—both are built into the popular `AutoAttack` package[4]. Another non-PGD method, Square Attack [45], which is based on gradient-free random search, is also included in `AutoAttack`, but the performance is not as effective as APGDs.

2.2 Numerical minimization of Form. (2)

The difficulty in optimizing Form. (2) lies in dealing with the highly nonlinear constraint $\max_{i \neq y} f^i_{\boldsymbol{\theta}}(\boldsymbol{x}') \geq f^y_{\boldsymbol{\theta}}(\boldsymbol{x}')$. There are two lines of ideas to circumvent it: **1) penalty methods** turn the constraint into a penalty term added to the objective [1, 37]. The resulting box-constrained problems can then be handled by classical optimization methods, such as L-BFGS [46] or PGD. But penalty methods do not guarantee to return feasible solutions to the original problem; **2) iterative linearization** linearizes the constraint at

[3] It should be "ascent" instead of "descent" due to the maximization, but we follow the `AutoAttack` package.

[4] Package website: https://github.com/fra31/auto-attack.

each step, leading to simple solutions to the projection (onto the intersection of the linearized decision boundary and the $[0,1]^n$ box) for particular choices of d (i.e.,ℓ_1, ℓ_2 and ℓ_∞), see, e.g., [14, 35]. However, for general metric d, this projection does not have closed form solutions. In [16, 38], the problem is reformulated as:

$$\begin{aligned} &\min_{\boldsymbol{x}',t} \; t \\ \text{s.t. } &\max_{i\neq y} f^i_{\boldsymbol{\theta}}(\boldsymbol{x}') \geq f^y_{\boldsymbol{\theta}}(\boldsymbol{x}') \\ &d(\boldsymbol{x},\, \boldsymbol{x}') \leq t\,,\; \boldsymbol{x}' \in [0,1]^n \end{aligned} \tag{7}$$

so that the perturbation (determined by $\boldsymbol{x}'$) and the radius (determined by t) are decoupled. Then penalty methods and iterative linearization are combined to solve Form. (7). The fast adaptive boundary (FAB) attack [14] included in the `AutoAttack` package belongs to family **2)**.

2.3 `PyGRANSO` for constrained optimization

In principle, as instances of general **n**on**l**inear **opt**imization (NLOPT) problems [46, 47]

$$\begin{aligned} &\min_{\boldsymbol{x}} \; g(\boldsymbol{x}) \\ \text{s.t. } &c_i(\boldsymbol{x}) \leq 0,\; \forall\, i \in \mathcal{I} \\ &h_j(\boldsymbol{x}) = 0,\; \forall\, j \in \mathcal{E} \end{aligned} \tag{8}$$

where $g(\cdot)$ is the objective function; c_i's and h_j's are inequality and equality constraints, respectively. Form. (1) and (2) can be solved by general-purpose NLOPT solvers such as `Knitro` [48], `IPOPT` [49], and `GENO` [50, 51]. However, there are two caveats: **1)** these solvers only handle continuously differentiable g, c_i's and h_j's, while non-differentiable ones are common in Form. (1) and (2), e.g., when d is ℓ_∞ distance, or $f_{\boldsymbol{\theta}}$ uses non-differentiable activations; **2)** they require analytical gradients of g, c_i's and h_j's, which are impractical to derive when DNN models $f_{\boldsymbol{\theta}}$ are involved.

`PyGRANSO` [41, 42][5] is a recent `PyTorch`-port of the powerful MATLAB package `GRANSO` [41] that can handle general NLOPT problems of Form. (8) with non-differentiable g, c_i's, and h_j's. It only requires them to be *almost everywhere differentiable* [52–54], which covers a much wider range than almost all forms of Form. (1) and (2) proposed so far in the literature. `GRANSO` employs a quasi-Newton sequential quadratic programming (BFGS-SQP) to solve Form. (8), and features a rigorous adaptive step-size rule via line search and a principled stopping criterion inspired by gradient sampling [55]. See a sketch of the algorithm in Section B and [41] for details. `PyGRANSO` equips `GRANSO` with auto-differentiation and GPU computing powered by `PyTorch`—crucial for deep learning problems. The stopping criterion is controlled by tolerance level of the total constraint violation and stationarity—the former determines how strict the constraints are enforced, while the latter measures how 'flat' the local optimization landscape is. Thus, the solution quality can be transparently controlled.

2.4 Min-max optimization for practical adversarial training

For a finite training set, Form. (3) becomes

$$\min_{\boldsymbol{\theta}} \frac{1}{N}\sum_{i=1}^{N} \max_{\boldsymbol{x}'_i \in \Delta(\boldsymbol{x}_i)} \ell\left(\boldsymbol{y}_i, f_{\boldsymbol{\theta}}(\boldsymbol{x}'_i)\right) \tag{9}$$

$$\iff \min_{\boldsymbol{\theta}} \max_{\boldsymbol{x}'_i \in \Delta(\boldsymbol{x}_i)\ \forall i} \frac{1}{N}\sum_{i=1}^{N} \ell\left(\boldsymbol{y}_i, f_{\boldsymbol{\theta}}(\boldsymbol{x}'_i)\right) \tag{10}$$

i.e., in the form of

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{x}'_i \in \Delta(\boldsymbol{x}_i)\ ,\forall i} \phi(\boldsymbol{\theta}, \{\boldsymbol{x}_i\}) \tag{11}$$

Form. (9) is often solved by iterating between the (separable) inner maximization and a subgradient update for the outer minimization. The latter takes a subgradient of $h(\boldsymbol{\theta}) \doteq \max_{\boldsymbol{x}'_i \in \Delta(\boldsymbol{x}_i)\ \forall i} \phi(\boldsymbol{\theta}, \{\boldsymbol{x}_i\})$ from the subdifferential $\partial_{\boldsymbol{\theta}} \phi(\boldsymbol{\theta}, \{\boldsymbol{x}^*_i\})$ ($\boldsymbol{x}^*_i$ are maximizers for the inner maximization), justified by the celebrated Danskin's theorem [56–58]. However, if the numerical solutions to the inner maximization are substantially suboptimal[6], the subgradient we take can be misleading—see our discussion in Section C. In practice, people typically do not strive to find the optimal solution to the inner

[5] Package webpage: https://ncvx.org/

[6] [11] argues using numerical evidence that the maximization landscapes are typically benign in practice and gradient-based methods often find solutions with function values close to the global optimal.

maximization strictly, but early stop the optimization process within a few iterations, or when a successful 'attack' sample is found.

3 A generic solver for Form. (1) and (2): PyGRANSO With Constraint-Folding (PWCF)

Although PyGRANSO can serve as a promising solver for Form. (1) and (2), naive deployment to solve the above two formulations can suffer from slow convergence or low-quality solutions due to numerical issues. Thus, we introduce PyGRANSO **W**ith **C**onstraint-**F**olding (PWCF)—essential to substantially speed up the optimization process and improve the solution quality of PyGRANSO.

3.1 General techniques

The following techniques are developed from solving Form. (1) and (2), but are conceptually general and should be considered for general NLOPT problems when using PyGRANSO.

3.1.1 Reduce the number of constraints: constraint-folding

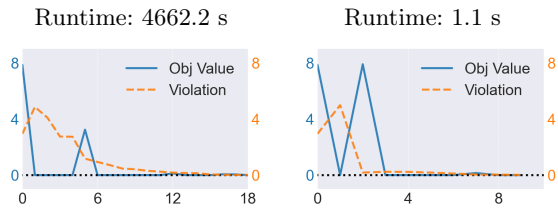
The natural image constraint $\boldsymbol{x}' \in [0,1]^n$ is a set of n box constraints. The reformulation described in Section 3.2.1 and Section 3.2.2 will introduce another $\Theta(n)$ box constraints. Although all these are simple linear constraints, the $\Theta(n)$-growth is daunting: for natural images, n is the number of pixels that can easily get into hundreds of thousands. Typical NLOPT problems become much more difficult when the number of constraints grows, e.g., leading to slow convergence for numerical algorithms.

To combat this, we introduce constraint-folding to reduce the number of constraints into a single one. To see how this is possible, first note that any equality constraint $h(\boldsymbol{x}) = 0$ or inequality constraint $c(\boldsymbol{x}) \leq 0$ can be reformulated as

$$\begin{aligned} h(\boldsymbol{x}) = 0 &\iff |h(\boldsymbol{x})| \leq 0 \,, \\ c(\boldsymbol{x}) \leq 0 &\iff \max\{c(\boldsymbol{x}), 0\} \leq 0 \end{aligned} \tag{12}$$

We can fold them together as

$$\mathcal{F}(|h(\boldsymbol{x})|, \max\{c(\boldsymbol{x}), 0\}) \leq 0 \tag{13}$$



(a) n box constraints **(b)** folded constraints

Fig. 3: Examples of the PyGRANSO optimization trajectory to solve Form. (1) with ℓ_2 ($\varepsilon = 0.5$) as d, and a clipped margin loss ℓ (described in Section 3.2.4) on a CIFAR-10 image. **(a)** the constraint is in the original form of n linear box constraints $\boldsymbol{x}' \in [0,1]^n$; **(b)** the box constraints are folded by ℓ_2 norm into a single one. The horizontal axis denotes the iteration number. Here an optimal solution is found when the objective value reaches -0.01 and the constraint violation reaches 0. Although it takes PyGRANSO similar number of iterations to reach an optimal solution for both cases, it consumes significantly less time with the constraint-folding technique than without.

where $\mathcal{F} : \mathbb{R}_+^2 \mapsto \mathbb{R}_+$ ($\mathbb{R}_+ \doteq \{t : t \geq 0\}$) can be any function satisfying $\mathcal{F}(\boldsymbol{z}) = 0 \implies \boldsymbol{z} = \boldsymbol{0}$, e.g., any ℓ_p ($p \geq 1$) norm.

It is easy to verify the equivalence of Eq. (13) and the original constraints in Eq. (12). The folding technique can be used to a subset or all constraints; or to group and fold the constraints according to their physical meanings, respectively. We note that folding or aggregating constraints is not a new idea and has been popular in engineering design, e.g., [59] uses ℓ_∞ folding and its log-sum-exponential approximation to deal with numerous design constraints, also see [60–63]. However, applying folding into NLOPT problems in machine learning seems rare, potentially because producing non-differentiable constraint(s) due to the folding seems counterproductive.

In our experiments, we always use $\mathcal{F} = \|\cdot\|_2$ for constraint folding. Fig. 3 depicts the benefit of the constraint folding technique by an example of Form. (1), where time efficiency is greatly improved and the solution quality is maintained. Note that *PWCF minimize the negative loss ($-\ell$) when solving Form. (1)* and thus the objective value curves are descending. Constraint-folding can significantly improve the optimization performance due to the fact that: **1)** when forming the penalty function

Algorithm 1 Selection of $\boldsymbol{x}_r^*$ and s_r^* in the two-stage process

Require: Intermediate optimization results $\boldsymbol{x}_r^i$ and corresponding solver states s_r^i.

if Any $\boldsymbol{x}_r^i$ is feasible in Form. (8) **then**
  Set $\boldsymbol{x}_r^*$ to be the $\boldsymbol{x}_r^i$ with the least objective value.
else
  Set $\boldsymbol{x}_r^*$ to be the $\boldsymbol{x}_r^i$ with the least constraint violation.
end if
Set s_r^* corresponds to $\boldsymbol{x}_r^*$.
return $\boldsymbol{x}_r^*$ and s_r^*.

with a large amount of constraints in `PyGRANSO`, the searching directions are biased towards staying feasible, thus slowing down the progress towards minimizing the objective function; **2)** the two QPs ((B12) and (B16)) become very expensive when the number of constraints p is large. The number of optimization variables in both QPs is proportional to p, and thus reducing p (by constraint-folding) can significantly reduce the cost per iteration.

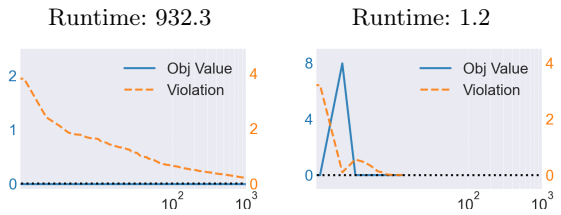
3.1.2 Two stage optimization

Numerical methods can be trapped at poor local minima for highly nonlinear problems. Running the optimization algorithm multiple times with different random initialization points seems to be an effective way to counter, and thus improve the final solution quality. E.g., each method in `AutoAttack` [15] has 5 random restarts by default. We apply the same technique for PWCF, but in a two-stage fashion:

1. **Stage 1:** Optimize the problems in PWCF form with n_k different random initializations for n_r iterations; collect the solver state s_r^i[7] and the final solution $\boldsymbol{x}_r^i$ of each run, where $i = 1, \cdots, n_k$ and $r = 1, \cdots, n_r$. Determine the best intermediate result $\boldsymbol{x}_r^*$ and corresponding state s_r^* following Algorithm 1.
2. **Stage 2:** Continue optimization process with $\boldsymbol{x}_r^*$ and s_r^* until the stopping criterion is met.

While n_k and n_r are empirically picked, the purpose is to improve the time efficiency of PWCF by not wasting time searching in sub-optimal plateaus:

[7]The solver state s_r^i will be the inverse Hessian matrix if using BFGS algorithm, or the latest updates if using L-BFGS algorithm in PWCF.



(a) original ℓ_∞ **(b)** reformulated ℓ_∞

Fig. 4: Examples of the `PyGRANSO` optimization trajectory to solve Form. (1) with ℓ_∞ ($\varepsilon = 0.03$), and a clipped margin loss ℓ (described in Section 3.2.4) on a CIFAR-10 image. **(a)** uses the original form $\|\boldsymbol{x}' - \boldsymbol{x}\|_\infty \leq \varepsilon$ as the constraint; **(b)** uses the reformulated and folded constraint. The horizontal axis denotes the iteration number. In this example, optimization terminates when constraint violation turns 0. After reformulate and fold the ℓ_∞ constraints, the optimization process runs much faster in terms of both time and iteration needed.

when the number of iteration is large, PWCF tends to refine the solution within a local plateau due to the line-search update rule—see (a) and (b) in Fig. 6 later for an example, where the objective value decreases much more slowly in the later stage than in the early 20 iterations. For the experiments in Section 4, we use $n_k = 20$ and $n_r = 400$ for Form. (1) experiments and $n_k = 50$ and $n_r = 4000$ for Form. (2) experiments. These numbers are set on the conservative side to ensure that PWCF returns solutions with sufficient quality.

3.2 Techniques specific to Form. (1) and (2)

The following techniques are developed to improve the performance of solving Form. (1) and (2). According to our experiments, they should be applied whenever encountered for these two problems.

3.2.1 Avoid sparse subgradients: reformulating ℓ_∞ constraints

The BFGS-SQP algorithm inside `PyGRANSO` relies on the subgradients of the objective and the constraint functions to approximate the (inverse) Hessian and to compute the search direction. Hence, when the subgradients are sparse, updating all optimization variables may take many iterations,

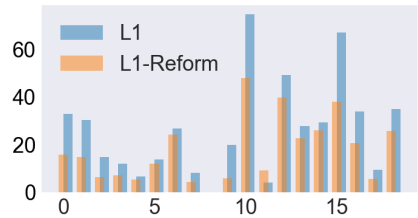


Fig. 5: Minimal radius find by solving Form. (2) with ℓ_1 norm as constraint (original form) and solving Form. (15) (reformulated version) on 18 CIFAR-10 images. The horizontal axis denotes the sample index. All solutions are feasible, and thus the lower the radius, the better the solution quality.

leading to slow convergence. For the ℓ_∞ metric,

$$\partial_{\boldsymbol{z}} \|\boldsymbol{z}\|_\infty = \text{conv}\{\boldsymbol{e}_k \,\text{sign}(z_k) : z_k = \|\boldsymbol{z}\|_\infty \,\forall\, k\}$$

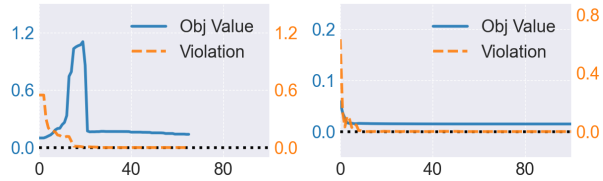
where $\boldsymbol{e}_k$'s are the standard basis vectors, conv denotes the convex hull and $\text{sign}(z_k) = z_k/|z_k|$ if $z_k \neq 0$, else $[-1, 1]$. The subgradient thus contains no more than $n_k = |\{k : z_k = \|\boldsymbol{z}\|_\infty\}|$ nonzeros and is thus sparse when n_k is small. To avoid this issue, we propose the following reformulation which should be applied whenever ℓ_∞ is present in the constraints using `PyGRANSO`:

$$\|\boldsymbol{x} - \boldsymbol{x}'\|_\infty \leq \varepsilon \iff -\varepsilon\mathbf{1} \leq \boldsymbol{x} - \boldsymbol{x}' \leq \varepsilon\mathbf{1} \quad (14)$$

where $\mathbf{1} \in \mathbb{R}^n$ is the all-one vector. Then, the resulted n box constraints can be folded into 1 single constraint as introduced in Section 3.1.1 to improve its efficiency. By reformulation and folding, PWCF can greatly accelerate the optmization process, see Fig. 4.

3.2.2 Decouple the update direction and the radius: reformulating ℓ_1 and ℓ_∞ objectives

It is not surprising that to solve Form. (2) with ℓ_∞ as d, it is more effective to solve the reformulated version as Form. (7)—the reason is to avoid sparse subgradients, similar to Section 3.2.1. We also find that solving the reformulated version below is more



(a) min t **(b)** min $t \cdot \sqrt{n}$

Fig. 6: Examples of PWCF optimization trajectory to solve Form. (7) with ℓ_∞ norm as d on a Cifar10 image **(a)** without rescaling and **(b)** with rescaling. The horizontal axis is the number of iterations. The objective value shown in **(b)** is scaled back to the original value t for fair comparisons with **(a)**. In Figure **(a)**, the optimization terminates around iteration 60 due to line-search failure (indicating bad numeric conditions met, instead of terminating by the stopping criterion), and the final solution has a much higher (worse) objective value than **(b)**. Also note in both **(a)** and **(b)**, PWCF makes the most progress within a few iterations (< 20), then refines the objective value with minor improvements afterwards.

effective when d in Form. (2) is the ℓ_1 norm:

$$\begin{aligned} &\min_{\boldsymbol{x},\boldsymbol{t}} \ \mathbf{1} \cdot \boldsymbol{t} \\ \text{s. t. } &\max_{i \neq y} f^i_{\boldsymbol{\theta}}(\boldsymbol{x}') \geq f^y_{\boldsymbol{\theta}}(\boldsymbol{x}') \\ &|\boldsymbol{x}_i - \boldsymbol{x}'_i| \leq t_i\,, \quad i = 1, 2, \cdots, n \\ &\boldsymbol{x}' \in [0, 1]^n \end{aligned} \quad (15)$$

The newly introduced box constraint $|\boldsymbol{x}_i - \boldsymbol{x}'_i| \leq t_i$ is then folded by ℓ_2 norm into a single constraint as Section 3.1.1. Fig. 5 compares the minimal radius found by solving Form. (2) and Form. (15) with ℓ_1 norm on 18 CIFAR-10 images against the same DNN model. Solving Form. (15) clearly outperforms solving Form. (2) as it finds much smaller radius in all but one samples.

3.2.3 Numerical re-scaling to balance objective and constraints

The step-size rule in `PyGRANSO` (see line 5 in Algorithm 3, Section B) can only downplay the contribution from the objective and gradually push the solution towards feasibility. Thus, if the scale of the objective value is too small compared to the constraint violation initially, numeric problems

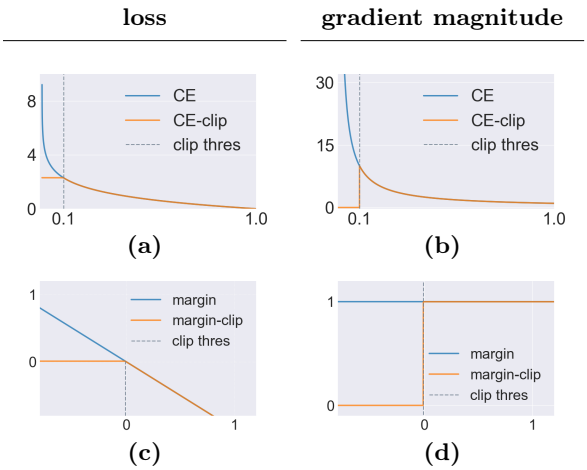


Fig. 7: Visualizations of the loss clipping. **(a):** cross-entropy loss and its clipped version. The depicted clipping threshold is used in CIFAR-10 experiments. **(b):** the gradient magnitude of **(a)**. The horizontal axes of **(a)** and **(b)** are the network output value $f^y_{\boldsymbol{\theta}}(\boldsymbol{x}')$ after the softmax regularization following the common practice. **(c):** margin loss and its clipped version. **(d):** the gradient magnitude of **(c)**. The horizontal axes of **(c)** and **(d)** are the value $\max_{i\neq y} f^i_{\boldsymbol{\theta}}(\boldsymbol{x}') - f^y_{\boldsymbol{\theta}}(\boldsymbol{x}')$ *before* the softmax regularization as is defined by the raw decision boundary in Eq. (5).

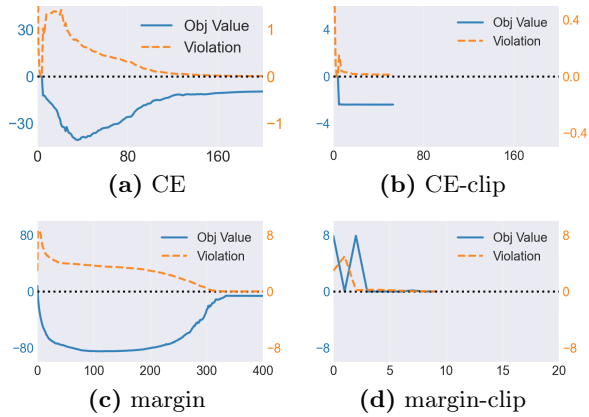


Fig. 8: PWCF optimization trajectory comparison among using cross-entropy (CE) loss, margin loss and their clipped versions in solving Form. (1) with a CIFAR-10 image. The horizontal axis denotes the number of iterations. For both CE and margin loss without clipping (**(a)** and **(c)**), PWCF progresses slowly towards feasibility (dashed orange curve), while with clipping (**(b)** and **(d)**), PWCF finds an optimal and feasible solution within only a few iterations.

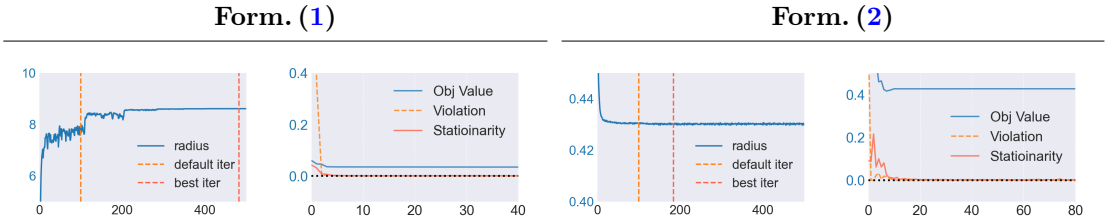
arise—PyGRANSO will try hard to combat the violation, while the objective hardly decrease. This only happens when solving Form. (7) for the ℓ_∞ case, where the radius t in the objective is a scalar of 10^{-2} while the folded constraints are the ℓ_2 norm of a $n-$dimensional vector with an order of 10^2 if random initialization is used. To combat this, we simply re-balance the objective by a constant scalar—we minimize $t\cdot\sqrt{n}$ instead of t, and PWCF can perform as effectively as in other cases, see an example in Fig. 6.

3.2.4 Loss clipping in solving Form. (1) with PWCF to generate attack

When solving Form. (1) with the popular cross-entropy (CE) and margin losses (both unbounded, see Fig. 7), the objective value can easily dominate the constraint violation during the maximization process. Since PyGRANSO tries to balance the objective value and constraint violation when making progress, it can persistently prioritize optimizing the objective over constraint satisfaction, resulting in slow progress in finding a feasible solution. To solve this numerical difficulty, we propose using the clipped margin loss ℓ_{ML} with maximum value 0.01, as any $\ell_{ML} \geq 0$ indicates a successful attack. For the same reason, we use clipped CE loss with maximal value at 2.4 for CIFAR-10 dataset and 4.7 for ImageNet-100 dataset in PWCF. The CE threshold is justified, as the attack success must have happened when the true logit output less than $1/K$ (after softmax normalization is applied), where K is the number of classes. The corresponding critical value is $-\log 1/K$—when $K = 10$ (the number of total classes for CIFAR-10 dataset), this value is 2.4, and when $K = 100$ (for ImageNet-100 dataset), this value is 4.605. Loss clipping significantly speeds up the optimization process in solving Form. (1), see an example in Fig. 8.

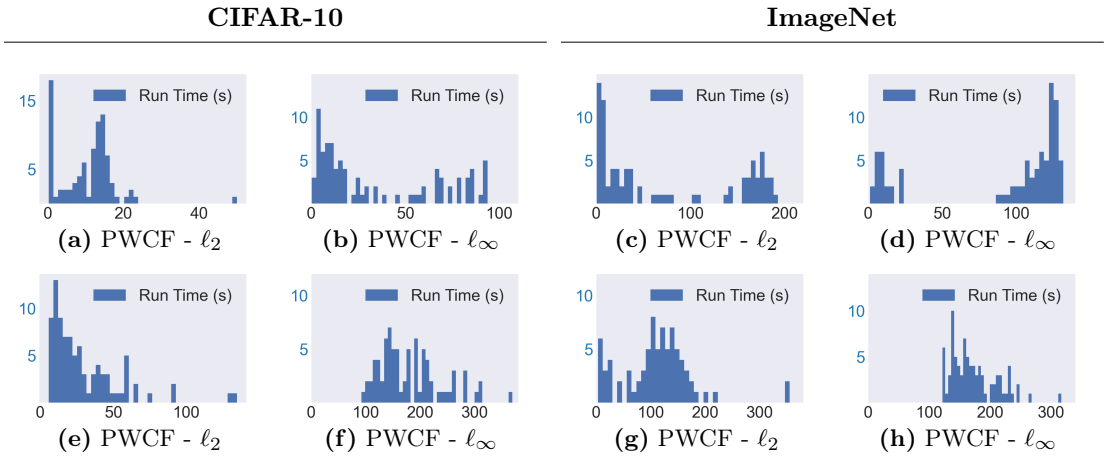
3.3 Summary of PWCF to solve Form. (1) and Form. (2)

We here summarize PWCF to solve Form. (1) and (2) in Algorithm 2. Then we provide information on PWCF's reliability and running speed when used to solve these two RE problems.



(a) APGD **(b)** PWCF - margin loss **(c)** FAB **(d)** PWCF

Fig. 9: Examples of optimization trajectories of APGD, FAB and PWCF solving Form. (1) and (2) of a CIFAR-10 Image. The blue curves in **(a)**-**(d)** denote the objective value. In **(a)** and **(c)**, the dashed orange lines are the default termination iteration used in current RE pracice [39] and the dashed red lines are the iteration where the actual best feasible solution found. In **(b)** and **(d)**, the dashed orange lines denote the constraint violation value of PWCF and the red lines represent the stationarity. In this example, we set the stopping stationarity value to be 10^{-8} (much stricter than necessary), and thus the optimization does not stop within n_r. We can observe from **(b)** and **(d)** that PWCF optimization process is stable around iteration 20 and only improves subtly afterwords.



(a) PWCF - ℓ_2 **(b)** PWCF - ℓ_∞ **(c)** PWCF - ℓ_2 **(d)** PWCF - ℓ_∞

(e) PWCF - ℓ_2 **(f)** PWCF - ℓ_∞ **(g)** PWCF - ℓ_2 **(h)** PWCF - ℓ_∞

Fig. 10: Histogram of optimization time for PWCF to solve Form. (1): **(a)**-**(d)**, and Form. (2): **(e)**-**(h)** with the proposed Algorithm 2.

Table 1: Statistical summary of Fig. 10.

Time (s)	Min	Max	Median
(a)	0.269	50.02	12.37
(b)	0.253	93.62	17.60
(c)	0.377	193.2	36.26
(d)	1.081	131.9	115.5
(e)	5.778	134.5	21.58
(f)	91.67	369.2	167.5
(g)	2.965	354.8	112.9
(h)	121.5	317.0	161.7

3.3.1 Reliability

As mentioned in Section 1, popular numerical methods terminate with empirically set iteration budgets which can lead to early stopping at sub-optimal points, see Fig. 1. We observe that the default iteration budget used in the `AutoAttack` package *does terminate* the optimization process early. On the contrary, PWCF automatically terminates when the stopping criterion is met: Fig. 2 shows the termination iterations for PWCF in solving Form. (1) and (2) with both constraint violation and stationarity tolerance to be 10^{-2}. Fig. 9 depicts examples of the optimization trajectories of APGD, FAB and PWCF, which again confirms the reliable termination of PWCF. Note that Fig. 9 also suggests that even for some samples, PWCF terminates by the maximal iterations set as shown in Fig. 2, the optimizations are also

Table 2: **Comparison between PWCF and SOTA attack methods on ℓ_1, ℓ_2 and ℓ_∞ attacks (Form. (1)). Attack(ε)** denotes the type of adversarial attacks and their ε's. For each method, we report the models' clean and robust accuracy (numbers are in (%))—*lower robust accuracy* indicates *more effective* attacks. We tested APGD and PWCF using both cross-entropy (**CE**) and margin (**M**) loss. **CE+M** column shows the robust accuracy achieved by combining the adversarial samples found by using CE and M; **APGD+PWCF** shows the robust accuracy achieved by combining both APGD and PWCF using both CE and M loss. We highlight the best performance achieved by each individual solver with underlines, and highlight the performance achieved by the combination of all solvers and losses in **bold**.

			APGD			PWCF(ours)			Square	APGD+
Dataset	**Attack(ε)**	**Clean**	**CE**	**M**	**CE+M**	**CE**	**M**	**CE+M**	**M**	**PWCF**
CIFAR10	$\ell_1(12)$	73.29	0.97	0.00	0.00	17.93	0.01	0.01	2.28	**0.00**
	$\ell_2(0.5)$	94.61	81.81	81.06	80.92	81.99	81.02	80.87	87.9	**80.77**
	$\ell_\infty(0.03)$	90.81	69.44	67.71	67.33	88.71	68.20	68.17	71.6	**67.26**
ImageNet100	$\ell_2(4.7)$	75.04	42.44	44.06	40.86	42.50	43.52	40.60	63.1	**40.46**
	$\ell_\infty(0.016)$	75.04	46.78	47.54	45.20	73.92	47.72	47.72	59.9	**45.12**

Algorithm 2 PWCF to solve Form. (1) and (2)

Require: Original input $\boldsymbol{x}$, a pretrained network $f_{\boldsymbol{\theta}}$ and a distance metric d.
Require: Loss ℓ if solving Form. (1).
if Solving Form. (2) and d is either ℓ_1 or ℓ_∞ **then**
  Reformualte the problem as Section 3.2.2 and Section 3.2.3.
end if
if Solving Form. (1) **then**
  Clip loss function ℓ as Section 3.2.4.
end if
if Any constraint takes ℓ_∞ norm **then**
  Reformulate the constraint as Section 3.2.1.
end if
Perform constraint folding as Section 3.1.1.
Perform the two stage optimization as Section 3.1.2.
return $\boldsymbol{x}'$

close to convergence and the returned result is of good quality.

3.3.2 Running cost

We here provide statistical summaries of the running time of PWCF in solving the RE problems Form. (1) and (2) for reference. Our computation environment uses a AMD Milan 7763 64-core processor and a NVIDIA A100 GPU, connected with Mellanox HDR-100 InfiniBand network. Fig. 10 shows the running time of the experiment presented in Fig. 2, and the corresponding min, max, and median values are summarized in Table 1.

4 Performance of PWCF on RE problems

4.1 Compare PWCF with existing ℓ_1, ℓ_2 and ℓ_∞ solutions

We first compare PWCF with the existing SOTA numeric algorithms in solving Form. (1) and (2) with ℓ_1, ℓ_2 and ℓ_∞ norm as d to verify the effectiveness of our PWCF.

4.1.1 PWCF offers competitive and complementary performance in solving Form. (1)

We select some recent and publicly available adversarially trained models by ℓ_1, ℓ_2, and ℓ_∞ attacks on

Table 3: **Comparison between PWCF(ours) and FAB in solving Form. (2) with ℓ_1, ℓ_2 and ℓ_∞ norm as the metric d.** We experimented with 88 fixed images from CIFAR-10 and ImageNet-100 dataset with each d. We report the **Mean**, **Median** and **standard deviation (STD)** of the minimal perturbation radius—*lower* radius means *more effective* minimization. The columns under **Difference** are calculated based on the *sample-wise* radius difference (PWCF radius minus FAB), where **Mean** and **Median** ≤ 0 indicates PWCF performs better than FAB on average.

		FAB			**PWCF (ours)**			**Difference**		
Dataset	**Metric** d	**Mean**	**Median**	**STD**	**Mean**	**Median**	**STD**	**Mean**	**Median**	**STD**
CIFAR10	ℓ_1	13.92	10.50	12.63	12.02	7.29	11.46	**-1.89**	**-0.81**	5.24
	ℓ_2	1.02	0.90	0.71	1.00	0.88	0.69	**-0.019**	**-0.015**	0.250
	ℓ_∞	0.0298	0.0245	0.0220	0.0298	0.0252	0.0224	**0.0008**	**-0.00008**	0.007
ImageNet100	ℓ_1	435.4	400.6	303.9	408.1	390.6	284.7	**-27.31**	**-13.46**	70.55
	ℓ_2	6.75	6.81	3.82	6.71	6.88	3.76	**-0.042**	**-0.035**	0.758
	ℓ_∞	0.028	0.028	0.016	0.029	0.029	0.016	**0.0009**	**0.00002**	0.002

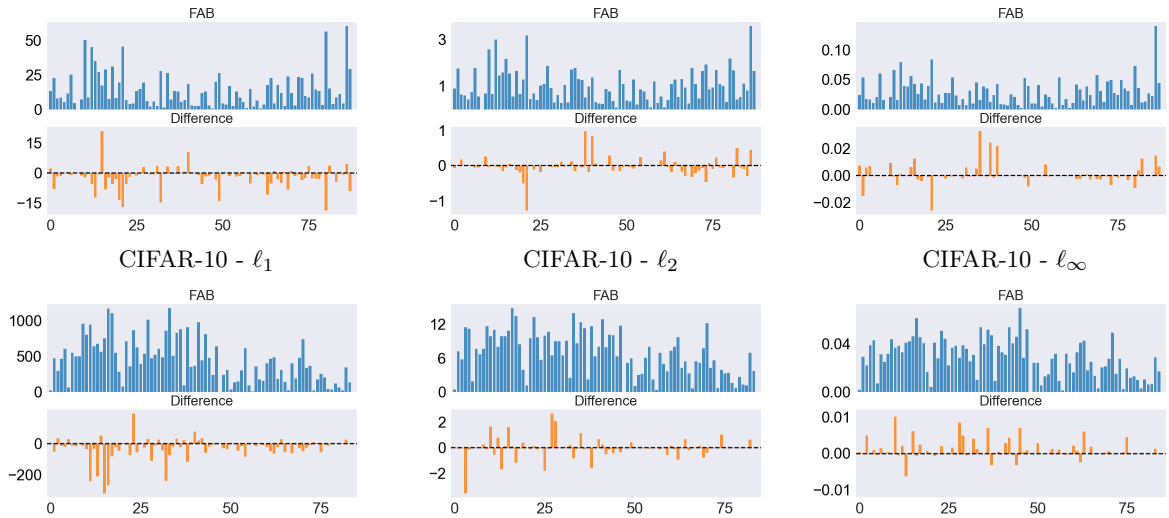


Fig. 11: Comparison of the minimum radius for every tested sample between PWCF(ours) and FAB (details of Table 3). 1st row: CIFAR-10; 2nd row: ImageNet-100. In each subfigure, the vertical axis depicts the radius, and the horizontal axis is the image indices.

CIFAR-10[8][9], and by perceptual attack[10] on ImageNet [12][11], and compare the robust accuracy by

[8]For ℓ_1 experiment, we pick the model 'L1.pt'from https://github.com/locuslab/robust_union/tree/master/CIFAR10, which is adversarially trained by ℓ_1-attack.

[9]For ℓ_2 ad ℓ_∞ experiments, We pick the models 'L2-Extra.pt' and 'Linf-Extra.pt' from https://github.com/deepmind/deepmind-research/tree/master/adversarial_robustness, with the WRN-70-16 network architecuture.

[10]See Section 4.2 for details.

[11]We use the 'pat_alexnet_0.5.pt' from https://github.com/cassidylaidlaw/perceptual-advex, where the author tested and showed its ℓ_2- and ℓ_∞- robustness in the original work.

solving Form. (1) with PWCF and APGD[12] methods from `AutoAttack` [15]. The bound ε are set to follow the common practice of RE[13].

From Table 2, we can conclude that **1)** PWCF performs strongly and comparably to APGD on ℓ_1, ℓ_2 and ℓ_∞ attacks, especially when using *margin loss* as the objective. The weak performance of PWCF on ℓ_1 and ℓ_∞ attacks using CE loss is likely due to the bad numerical scaling of the loss itself: the gradient magnitude of CE grows faster as maximization proceeds whereas we typically prefer a diminishing update step for better numerical outcomes in optimization problems, see Fig. 7 for a visualization of the losses. APGD [15] has an explicit step-size rule different from the vanilla PGD algorithms to boost the attack performance under CE loss, while we did not perform any special tuning for these cases. In contrast, the gradient scale is always 1 using the margin loss. **2)** Combining all successful attack samples found by APGD or PWCF, using CE and margin loss as the objective (column CE+M under APGD and PWCF in Table 2) can reduce the robust accuracy lower than the robust accuracy achieved by any single attack, and combining all attack samples found by APGD and PWCF using CE and margin loss (APGD+PWCF in Table 2) achieves the lowest robust accuracy—PWCF and APGD are complementary. Note that [44] also remarks that the diversity of solutions matters much more than the superiority of individual solvers, which is the reason why `AutoAttack` includes Square Attack—a zero$^{\text{th}}$-order black-box attack method that does not perform strongly itself as shown in Table 2. We will provide our extensive discussion on the diversity below in Section 5, based on the solution patterns found by different methods.

4.1.2 PWCF provides competitive solutions to Form. (2)

We take adversarially trained models using perceptual attacks on CIFAR-10[14] and ImageNet[15] and compare the performance by solving Form. (2) with PWCF and Fast Adaptive Boundary (FAB) attack from `AutoAttack` package [14]. Table 3 and Fig. 11 summarize the minimal radius found by each method. From the column Mean and Median in Table 3, We conclude that: PWCF performs on average **1)** better than FAB in solving Form. (2) with ℓ_1 and ℓ_2 as the distance metric d; **2)** comparable to FAB in the ℓ_∞ case.

Fig. 12: Minimal robustness radius (vertical axis) found by solving Form. (2) using PWCF on CIFAR-10 images. The horizontal axis shows the image indices. Red depicts the result for $\ell_{1.5}$ and blue for ℓ_8.

4.2 PWCF can solve Form. (1) and (2) with other general distance metric d

As highlighted in Section 2, a major limitation of the existing numerical methods is that *they can only handle limited choice of* d. In contrast, PWCF stands out as a convenient choice when d is other general distance metrics. In what follows, we will show this by using PWCF to solve Form. (1) and (2) with $\ell_{1.5}$ and ℓ_8 norm as distance metric d, as well as d being the perceptual distance that involves another DNN. To our best knowledge, there is no prior work that is capable of handling general ℓ_p distances; [12] has proposed 3 algorithms to solve Form. (1) with the perceptual distance, and we will compare our PWCF with them in the following sections.

4.2.1 Solving Form. (2) with general d

We first apply PWCF to solve Form. (2) with $\ell_{1.5}$ and ℓ_8 norms. Due to the lack of existing methods

[12] We implement the margin loss on top of `AutoAttack`.

[13] E.g., the ε of ℓ_2 and ℓ_∞ for CIFAR-10 are chosen from https://robustbench.github.io/; ℓ_1 for CIFAR-10 is chosen from https://github.com/locuslab/robust_union; ℓ_2 and ℓ_∞ for ImageNet-100 are from [12].

[14] We use model 'pat_self_0.5.pt' from https://github.com/cassidylaidlaw/perceptual-advex.

[15] The same ImageNet model used in Table 2.

Table 4: Performance comparison of different methods in solving Form. (1) with LPIPS distance on ImageNet-100 evaluation set, using (clipped) cross-entropy and margin losses respectively. **Viol.** reports the ratio of final solutions that violate the constraint; **Att. Succ.** is the ratio of all *feasible and successful attack samples* divided by *total number of samples*—*higher* indicates more *effective* attack performance.

	cross-entropy loss		**margin loss**	
Method	**Viol.** (%) ↓	**Att. Succ.** (%) ↑	**Viol.** (%) ↓	**Att. Succ.** (%) ↑
Fast-LPA	73.8	3.54	41.6	56.8
LPA	**0.00**	80.5	**0.00**	97.0
PPGD	5.44	25.5	**0.00**	38.5
PWCF (ours)	0.62	93.6	**0.00**	100

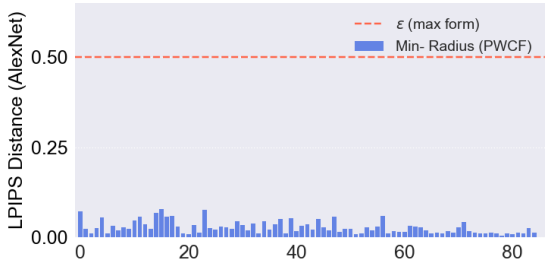


Fig. 13: Minimal robustness radius (vertical axis) found by solving Form. (2) using PWCF with LPIPS distance on ImageNet-100 images. The horizontal axis is the image index. The red dashed line is the proposed bound ε used to solve Form. (1) (perceptual attack) in [12], which is much larger than each radii found by solving Form. (2) with PWCF.

to compare, we directly show in Fig. 12 the minimal radius[16] found by PWCF on 100 CIFAR-10 images.

Similarly, there is no existing work considers solving Form. (2) with d to be the perceptual metric (LPIPS distance, which is first introduced in [64]):

$$\begin{aligned} & d(\boldsymbol{x}, \boldsymbol{x}') \doteq \|\phi(\boldsymbol{x}) - \phi(\boldsymbol{x}')\|_2 \\ \text{where} \quad & \phi(\boldsymbol{x}) \doteq [\, \widehat{g}_1(\boldsymbol{x}), \ldots, \widehat{g}_L(\boldsymbol{x}) \,] \end{aligned} \tag{16}$$

where $\widehat{g}_1(\boldsymbol{x}), \ldots, \widehat{g}_L(\boldsymbol{x})$ are the vectorized intermediate feature maps from pretrained DNNs. Therefore, we also directly show in Fig. 13 the minimal radius found[17] on 100 ImageNet images by PWCF. Fig. 12 and Fig. 13 can show that PWCF solves Form. (2) with general distance metric d with high quality.

4.2.2 Solving Form. (1) with $\ell_{1.5}$ and ℓ_8 norms

Since no existing work has evaluated DNN's robustness under $\ell_{1.5}$ and ℓ_8 attacks, we employ a sample-adaptive ε to evaluate the performance of PWCF: we use the same DNN model as Fig. 12 and solve Form. (1) with 1.2 times the minimal robust radius found in Fig. 12 for each sample. In our experiment, PWCF achieves 8% robust accuracy for $\ell_{1.5}$ and 1% for ℓ_8 attack—both are close to 0%, indicating PWCF's effectiveness. It is worth mentioning again that we only intent to show *the ability of PWCF in handling general metric d* in solving Form. (1), but do not strive to set the most reasonable perturbation radii, or to stress the attack rates.

4.2.3 Solving Form. (1) with perceptual metric

Table 4 summarizes the results of solving Form. (1) with the perceptual distance as metric d on ImageNet-100 dataset[18], reporting both attack success rate and the constraint violation rate. We compare PWCF with the three methods proposed in [12] on this problem: Perceptual Projected Gradient Descent (PPGD) which is based on iterative linearization and projection; Lagrangian Perceptual Attack (LPA) which is a penalty method with iterative projections to the feasible set; and a variant of LPA without the projection (Fast-LPA). We

[16] Using the same adversarially pretrained model on CIFAR10 dataset as in Table 3

[17] Using the same adversarially pretrained model on ImageNet dataset as in Table 3

[18] Using the same adversarially pretrained model as in Fig. 13

Fig. 14: A 'fish' image example from the Imagenet-100 evaluation dataset.

use $\varepsilon = 0.5$ as the original work [12]. According to Fig. 13, the per-sample robustness radius are much smaller than this ε—effective solvers are expected to achieve 100% attack success rate with 0% violation rate. As in Table 4, PWCF with margin loss is the only one that meets this standard and is thus the clear winner.

5 Different combinations of ℓ, d, and the solvers prefer different patterns

In addition to the effectiveness of PWCF in solving Form. (1) and (2), we also observe that using different combinations of **1)** distance metric d, **2)** solver, and **3)** loss ℓ can lead to different sparsity patterns in the final solutions, which we will demonstrate in this section in the following two ways:

1. **Visualization of an example image:** we take a 'fish' image (Fig. 14) from the ImageNet-100 validation set, employ various combinations of losses ℓ, distance metrics d and solvers to Form. (1) and (2), and visualize the final solutions in terms of the error image $\boldsymbol{x}' - \boldsymbol{x}$, and the histogram of the element-wise error magnitude $|\boldsymbol{x}' - \boldsymbol{x}|$ to display the pattern difference.
2. **Statistic measure:** we evaluate the *the sparsity measure* $\|\boldsymbol{x}' - \boldsymbol{x}\|_1 / \|\boldsymbol{x}' - \boldsymbol{x}\|_2$ to quantify the statistical difference of the patterns—the higher the value, the denser the pattern. Fig. 17 and Fig. 18 display the histograms of the sparsity measure over the error images derived by solving Form. (1) and Form. (2) respectively, where we randomly sample and fix 500 ImageNet-100 images from the validation set for comparison.

Contrary to the ℓ_1 norm that induces sparsity, the ℓ_∞ norm promotes dense perturbations with comparable entry-wise magnitudes [65] and the ℓ_2 norm promotes dense perturbations whose entries follow power-law distributions. These varying sparsity patterns due to the norms are evident when **1)** we compare the solutions in Fig. 15 and Fig. 16 under the same solver and loss but with different norms, where the shape and value range of the element-wise error magnitude histograms are very different; **2)** the sparsity measures display a shift from left to right along the horizontal axis in Fig. 17 and Fig. 18. In addition to the norms, we also highlight other patterns induced by loss ℓ and numerical solvers on top of these. Although we cannot coherently explain how patterns are induced by the complex interplay of the loss ℓ, distance metric d, and the numerical solver, their presence seems prevalent.

- **Margin and cross-entropy losses have different sparsity patterns in solving Form. (1)** Columns 'cross-entropy' and 'margin' of PWCF in Fig. 15 depict the pattern difference with clear divergences in the error histograms, e.g., the error values of PWCF-ℓ_2-margin are more concentrated towards 0 compared to PWCF-ℓ_2-cross-entropy. The sparsity measures in Fig. 17 can further confirm the existence of the difference due to the loss used in solving Form. (1), both for APGD and PWCF.
- **`PyGRANSO` solutions have more variety in sparsity than APGD** For the same ℓ_p and loss used to solve Form. (1), Fig. 17 shows that PWCF's solutions have wider spread in the sparsity measure than APGD. The same observation can be found in the min- form as well between PWCF and FAB (Fig. 18 for Form. (2)).

Here we provide a conceptual explanation on why this can happen. We take the ℓ_1 distance as an example, i.e., $\|\boldsymbol{x} - \boldsymbol{x}'\|_1 \leq \varepsilon$ and ignore the box constraint. The ℓ_1 norm is a famous sparsity promoter in statistics and machine learning [66, 67], and hence the expected perturbation $\boldsymbol{\delta} = \boldsymbol{x}' - \boldsymbol{x}$ should be sparse. For simplicity, we take the loss as 0/1 classification error $\ell(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x}')) = \mathbb{1}\{\max_i f^i_{\boldsymbol{\theta}}(\boldsymbol{x}') \neq y\}$. Note that ℓ is maximized whenever $f^i_{\boldsymbol{\theta}}(\boldsymbol{x}') > f^y_{\boldsymbol{\theta}}(\boldsymbol{x}')$ for a certain $i \neq y$, so that $\boldsymbol{x}'$ crosses the local decision boundary between the i-th and y-th classes—see Fig. 19. In practice, people set a substantially larger perturbation radius than the robustness radius—which can be

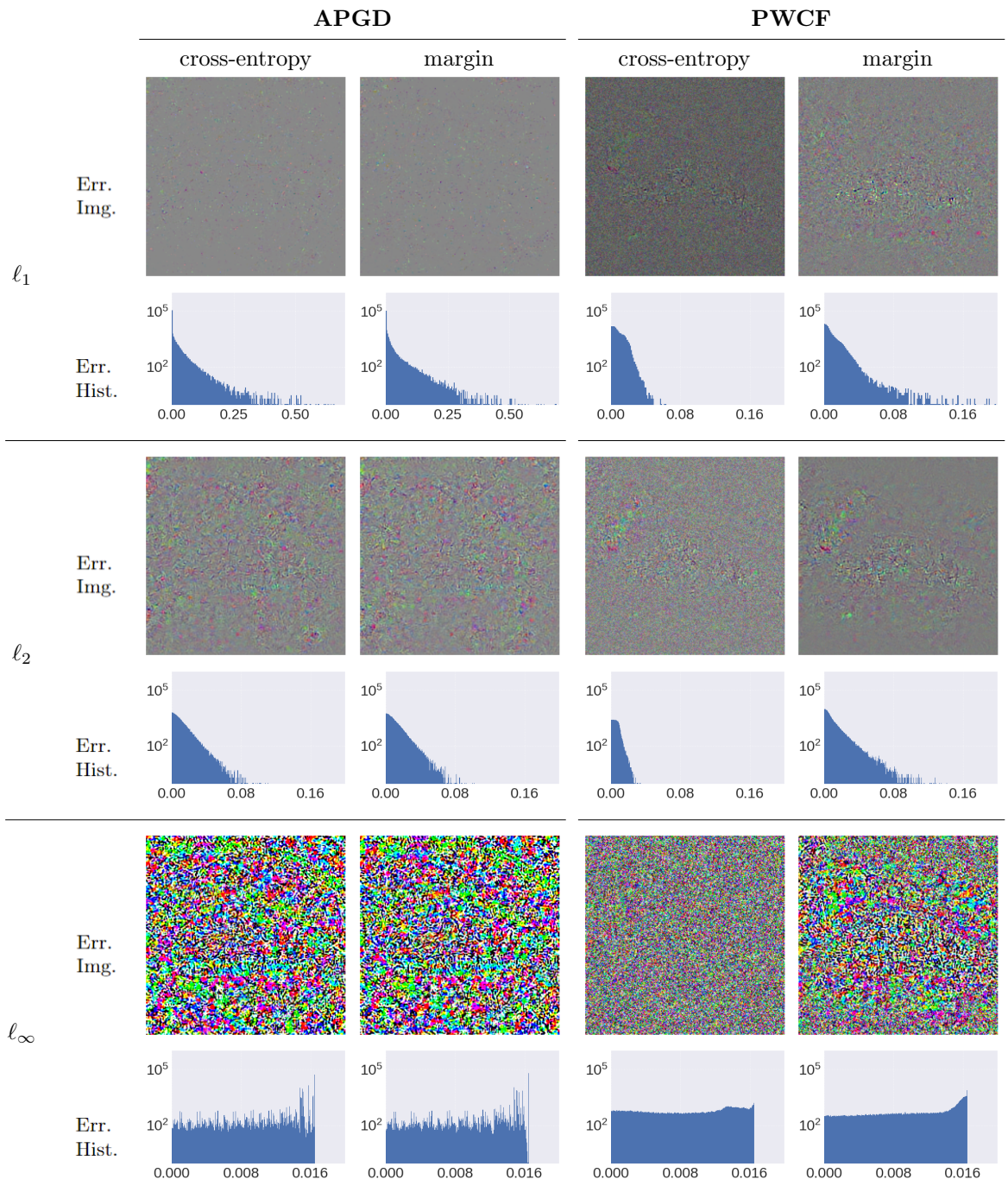


Fig. 15: Visualization of the error images by solving Form. (1) with different losses (cross-entropy and margin), different distance metrics d (ℓ_1, ℓ_2 and ℓ_∞) and different solvers (APGD and PWCF). Within each group by distance metric d, the top row shows of the error image $\boldsymbol{x}' - \boldsymbol{x}$, which has been normalized to range $[0, 1]$ for better visualization purpose; the bottom row shows the histogram of the element-wise error magnitude $|\boldsymbol{x}' - \boldsymbol{x}|$, where the horizontal axis is the pixel value magnitude and the vertical axis is the count.

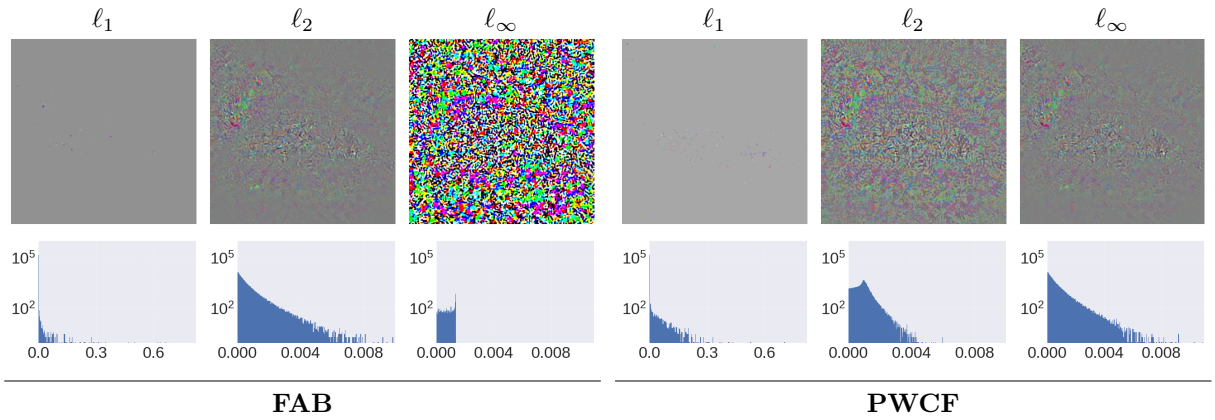


Fig. 16: Visualizations of the error images ($\boldsymbol{x}' - \boldsymbol{x}$, top row) and the histogram of element-wise error magnitude $|\boldsymbol{x}' - \boldsymbol{x}|$, bottom row) by solving Form. (2). Note that the comparison between FAB and PWCF may not be as straightforward as Fig. 15 due to the fact that the radius ε found under Form. (2) are likely different, resulting in different ranges of error magnitude values. However, the shape of the histograms can still reveal the pattern differences.

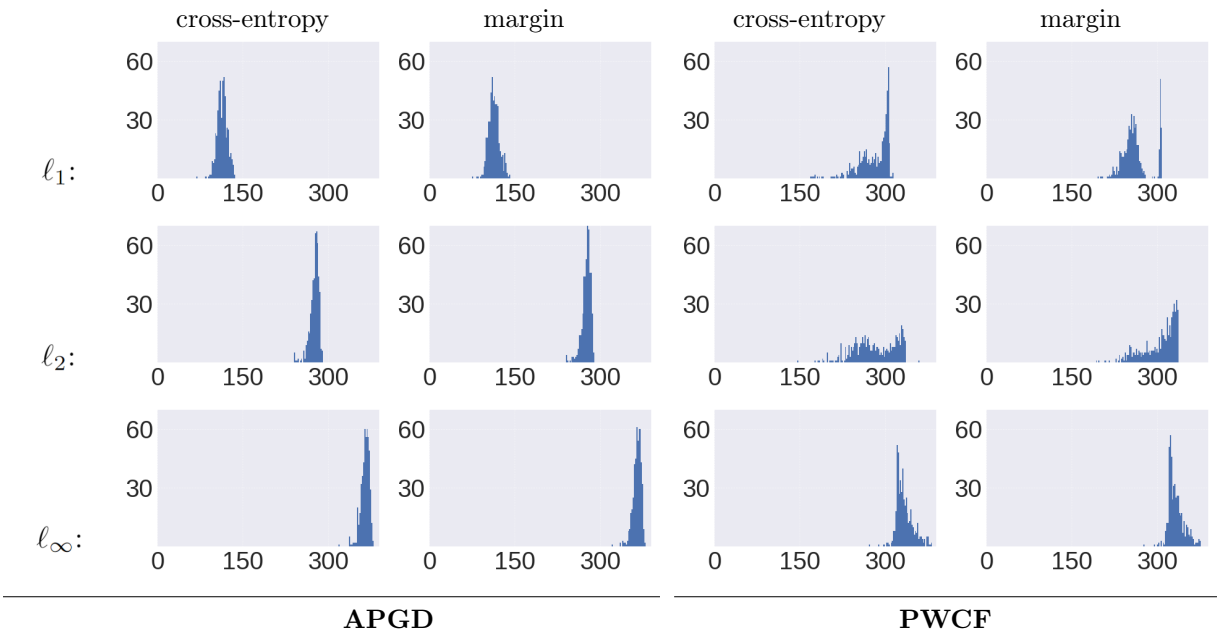


Fig. 17: Histograms of the sparsity measure $\|\boldsymbol{x}' - \boldsymbol{x}\|_1 / \|\boldsymbol{x}' - \boldsymbol{x}\|_2$ by solving Form. (1) with different losses ℓ (cross-entropy and margin), different distance metrics d (ℓ_1, ℓ_2 and ℓ_∞) and different methods (APGD and PWCF). *The larger the sparsity measure, the denser the pattern.* For fair comparisons, we use a non-adversarily trained model to generate the above samples so that each x' is a successful attack. With the same d and ℓ, the distribution difference of the sparsity measures between APGD and PWCF clearly shows the solution pattern differences; the variance of the sparsity is noticeably larger in PWCF than in APGD when the same d and ℓ is used.

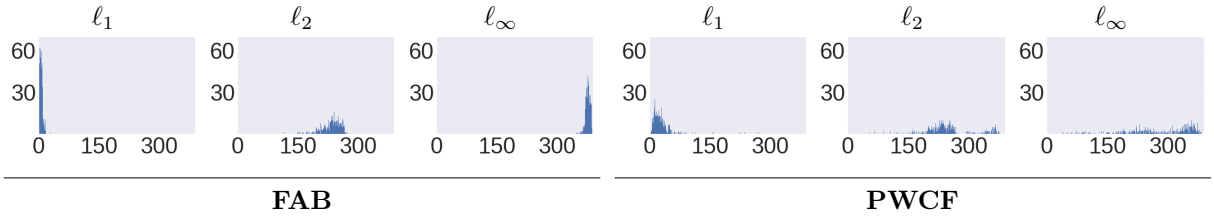


Fig. 18: Histograms of the sparsity measure ($\|\boldsymbol{x}' - \boldsymbol{x}\|_1 / \|\boldsymbol{x}' - \boldsymbol{x}\|_2$) by solving Form. (2) with different distance metrics d (ℓ_1, ℓ_2 and ℓ_∞). Under the same solver (FAB or PWCF), the shift of solution patterns from sparse to dense due to d is obvious. Given the same d, the solutions of PWCF have more variety in sparsity than the ones of FAB, showing the influence of the solver on the solution patterns.

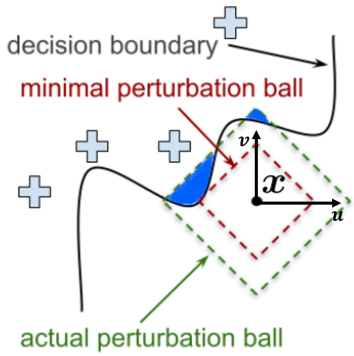


Fig. 19: Geometry of Form. (1) with multiple global maximizers. $\boldsymbol{u}$ and $\boldsymbol{v}$ are the basis vectors of the 2-dimensional coordinate. Here we consider the ℓ_1-norm ball around $\boldsymbol{x}$, and ignore the box constraint $\boldsymbol{x}' \in [0,1]^n$ that typically does not substantially change the geometry. Depending on the loss ℓ used, part or the whole of the blue regions becomes global or near-global maximizers.

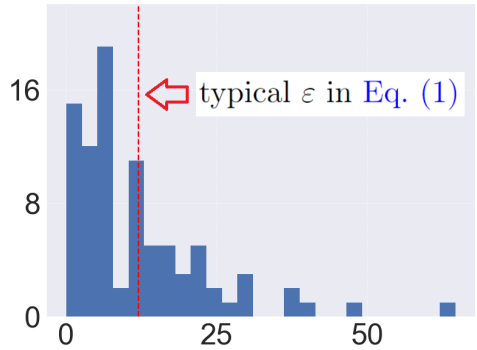


Fig. 20: Histogram of the ℓ_1 robustness radius estimated by solving Form. (2) for 100 CIFAR-10 images. $\varepsilon = 12$ is typically used in Form. (1) as is shown in dashed red line.

estimated by solving Form. (2)—see Fig. 20. Thus, there could be infinitely many global maximizers (the shaded blue regions in Fig. 19), depending on the shape of the decision boundary. As for the patterns, solutions in the left shaded blue region in Fig. 19 is denser in pattern than the upper ones. For other general losses, such as cross-entropy or margin loss, the set of global maximizers might be smaller, but the patterns can also be more complicated due to the typical complicated nonlinear decision boundaries associated with DNN models.

6 Direct implications from the patterns

Now that we have demonstrated the complex interplay of loss ℓ, distance metric d, and the numerical solver for the final solution patterns in Section 5, we will discuss what it potentially implies to the current practice of adversarial robustness research.

6.1 Robust evaluation by Form. (1) is hardly sufficient

The most popular practice of RE is by solving Form. (1) with a preset level of ε, using a fixed set of attack algorithms and report the robust accuracy [39, 68, 69]. As is shown in Section 5, the optimal perturbations found by different algorithms can have different sparsity patterns. We have also shown in Table 2 that combining multiple algorithms can lead to the lowest robust accuracy, even if a single solver can beat all others individually. These facts imply that in order to have a reliable and accurate measure of the model's robustness via Form. (1), instead of relying on any small set of "strong" attack algorithms, we should

include as many solvers as possible, which seems costly and impractical.

6.2 Form. (2) may be the better choice for RE

On the other hand, using Form. (2) for RE seems more advantageous, especially if our goal is to *understand the limitations of our models* instead of the attacker-defender setup: the solutions (minimal robustness radius) are automatically the smallest ε for every sample (v.s. the preset, fixed ε in Form. (1) for all samples). Such sample-wise radius ε not only can provide statistically meaningful comparisons among models, it can also help us identify hard samples $\boldsymbol{x}$ from a dataset, which may in turn help us better understand the training process, or provide clues to adjust the training scheme to improve the model's performance.

Table 5: Robust accuracy against various adversarial attacks reported in Table 3 from [12]. PAT-AlexNet is an adversarially trained model with the LPIPS distance. The ℓ_2 and ℓ_∞ attacks during evaluation are generated by the `AutoAttack` package, whose backbone is mainly APGD with cross-entropy loss. [12] draws the conclusion that PAT generalizes well to unforeseen perturbation types, whereas we think that PAT performs only similarly to a ℓ_2 trained model and does not have better generalization.

	Test Attack (Robust Acc.)				
Training	**Clean**	ℓ_∞	ℓ_2	**PPGD**	**LPA**
ℓ_∞	81.7	55.7	3.7	1.5	0.0
ℓ_2	75.3	46.1	41.0	22.0	0.5
PAT-AlexNet	75.7	46.8	41.0	31.1	1.6

7 Many other implications and discussions

7.1 Form. (1) may be Incapable of measuring robustness

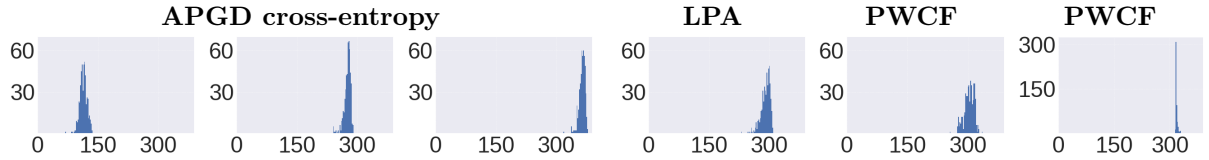
The motivation to perform RE with Form. (1) is usually associated with the attacker-defender setup, where RE is viewed as a test bench for all possible future attacks, and the network is required ideally to be robust against *all* of them. However, it is questionable whether the popular 'robust accuracy' is a good measure for such notion of robustness. E.g., the reason why the current attack budget ε used in practice is a reasonable choice, e.g., $\varepsilon = 0.03$ for ℓ_∞-attack in [39], needs to be justified. We did not find rigorous answers to this question in previous literatures and suspect that the choices are purely empirical. E.g, [39] states that their motivations to choose such ε

> ... the true label should stay the same for each in-distribution input within the perturbation set ...

but other values can also meet this standard. More importantly, having a higher robust accuracy at at level ε_1 does not imply a model being more robust at other levels—E.g., see Figure 1 in [70], where the most 'robust' model at each testing ε's are all different. The clean-robust accuracy trade-off [71, 72]—where a non-adversarially trained model has the best clean accuracy (at level $\varepsilon = 0$) and the worst robust accuracy (at the commonly used ε), while adversarially trained models have better robust accuracy, but worse clean accuracy, can be interpreted as such too. As a result, conclusions as 'the model is more robust due to higher robust accuracy achieved' using current adversarial robustness benchmark is misleading.

7.2 Achieving adversarial robustness via adversarial training may be difficult

Despite the effort of looking for ways to achieve generalizable adversarial robustness, it is widely known that robustness achieved by adversarial training (Form. (3)) does not even generalize across simple ℓ_p distance models [73, 74]. E.g., models trained by Form. (3) with the ℓ_∞ norm fail to achieve good robust accuracy in RE with ℓ_2 attack; the distance ℓ_1 seems to be a strong attack model for all other distances such as ℓ_2 and ℓ_∞, and even on itself. Now that [11] has observed the (approximate) global maximizers to be distinct and spatially scattered, the patterns we discussed in Section 5 provide a plausible explanation for why it is expected *not* to be generalizable—the model simply cannot generalize to a new distribution (patterns) which it has not seen during training. [12] claims that using the LPIPS distance (Eq. (16)) as d in Form. (1) can approximate the universal set of adversarial attacks, and perceptual adversarially trained (PAT)



(a) ℓ_1 **(b)** ℓ_2 **(c)** ℓ_∞ **(d)** LPIPS-ℓ_2 **(e)** LPIPS-ℓ_2 **(f)** LPIPS-ℓ_1

Fig. 21: Histograms of the sparsity measure ($\|\boldsymbol{x}' - \boldsymbol{x}\|_1 / \|\boldsymbol{x}' - \boldsymbol{x}\|_2$) on 500 ImageNet100 images, by solving Form. (1) with different distance metrics d (ℓ_p and LPIPS) and different solvers (APGD with cross-entropy loss, LPA and PWCF). LPA-LPIPS-ℓ_2 is the perceptual attack used in [12]. The adversaries are similar to APGD-ℓ_2 and APGD-ℓ_∞ in the sparsity patterns, which we think explains why PAT-AlexNet have good robust accuracy w.r.t ℓ_2 and ℓ_∞ attacks in Table 5. Also, using the LPIPS distance as d in Form. (1) will demonstrate different pattern preferences due to the solver and norm used (see the above three figures related to LPIPS), which makes it less convincing that LPIPS can approximate a universal adversarial model as is claimed in [12].

models (Form. (1) with LPIPS distance) can generalize to other unseen attacks, supported by result in Table 5. We disagree with their conclusion and suspect that PAT models proposed in [12] only performs similarly to the ℓ_2 trained one:

1. If we test the ℓ_2 and PAT-AlexNet models in Table 5 by APGD-ℓ_1 ($\varepsilon = 1200$) attack (on ImageNet-100), both will achieve 0% robust accuracy—PAT models not generalizing universally.
2. Performing the sparsity analysis as discussed in Section 5, the patterns show that the adversaries generated by perceptual attack are close to the APGD-CE-ℓ_2 generated ones, see (a)-(d) in Fig. 21. Note that the robust accuracy achieved by PAT-Alexnet is also similar to the ℓ_2 model in Table 5.
3. Substituting the ℓ_2 norm by ℓ_1 in the LPIPS definition Eq. (16):

$$d(\boldsymbol{x}, \boldsymbol{x}') \doteq \|\phi(\boldsymbol{x}) - \phi(\boldsymbol{x}')\|_1 \qquad (17)$$

the solution patterns will alter, see (e)-(f) in Fig. 21. Furthermore, (d)-(e) in Fig. 21 also shows that different solvers (LPA and PWCF) will also result into different patterns even using LPIPS—LPIPS will likely suffer in ways similar to ℓ_p norms in the robustness formulations and is not 'universal'.

8 Summary

In this paper, we introduce a new algorithmic framework, `PyGRANSO` **w**ith-**C**onstraint-**F**olding (PWCF), to solve the robustness evaluation (RE) problems in both max- form (Form. (1)) and min-form (Form. (2)). Our PWCF can handle any piecewise differentiable distance metrics that are beyond the reach of existing methods (such as ℓ_p where $p > 0$, and perceptual distance such as LPIPS), while achieving performance comparable to the existing SOTA methods when the distance metric d is ℓ_1, ℓ_2 or ℓ_∞ norm. We observe that using different combinations of loss ℓ, distance metric d and algorithm to solve Form. (1) and (2) will lead to different sparsity patterns in solution. We then provide an explanation on why it happens and discuss what this implies to adversarial robustness:

1. The current practice of RE based on solving Form. (1) can be insufficient and misleading.
2. The pattern difference may reveal a crucial limitation of the current adversarial training (AT) pipeline (Form. (3))—models trained in such pipeline may be intrinsically hard to generalize.

[75] has criticized the practicality of studying 'small ℓ_p perturbations' and urged researchers to be more explicit about their specific motivations when presenting results, we here re-iterate their thoughts with the following questions as our reflections on future researches:

1. Is the current adversarial robustness (attacker-defender) a goal too ambitious to achieve in practice? Do we need a more practical one?
2. What does the current RE pipeline (using Form. (1)) really tell us? Should we standardize the RE using the min- form Form. (2) instead?

3. What can AT (Form. (3)) actually achieve in practice? Are we expecting more than its capability?

As a result, the tool (PWCF) we presented in this paper is neither intended to compare the attack performance with the existing SOTA ℓ_p algorithms, nor to improve the adversarial training pipeline. The goal of PWCF is to provide a numerical framework that is practical, general and reliable to test Form. (1) and (2) which can be helpful to understand our networks and robustness.

Acknowledgments. Acknowledgments are not compulsory. Where included they should be brief. Grant or contribution numbers may be acknowledged.

Please refer to Journal-level guidance for any specific requirements.

Appendix A

Projection onto the intersection of norm ball and box constraints APGD for solving Form. (1) with ℓ_p distances entails solving Euclidean projection subproblems of the form:

$$\begin{aligned} &\min_{\boldsymbol{x}' \in \mathbb{R}^n} \ \|\boldsymbol{z} - \boldsymbol{x}'\|_2^2 \\ &\text{s. t. } \|\boldsymbol{x} - \boldsymbol{x}'\|_p \leq \varepsilon, \quad \boldsymbol{x}' \in [0,1]^n \end{aligned} \tag{A1}$$

where $\boldsymbol{z} = \boldsymbol{x} + \boldsymbol{w}$ is a one-step update of $\boldsymbol{x}$ towards direction $\boldsymbol{w}$. After a simple reparametrization, we have

$$\begin{aligned} &\min_{\boldsymbol{\delta} \in \mathbb{R}^n} \ \|\boldsymbol{w} - \boldsymbol{\delta}\|_2^2 \\ &\text{s. t. } \|\boldsymbol{\delta}\|_p \leq \varepsilon, \quad \boldsymbol{x} + \boldsymbol{\delta} \in [0,1]^n \end{aligned} \tag{A2}$$

We focus on $p = 1, 2, \infty$ which are popular in the AR literature. In early works, a "lazy" projection scheme—sequentially projecting onto the ℓ_p ball and then to the $[0,1]^n$ box, is used. [76] has recently identified the detrimental effect of lazy projection on the performance for $p = 1$, and has derived a closed form solution. Here, we prove the correctness of the sequential projection for $p = \infty$ (Lemma A.1), and discuss problems about the $p = 2$ case (Lemma A.3).

For $p = \infty$, obviously we only need to consider the individual coordinates.

Lemma A.1. *Assume $x \in [0,1]$. The unique Solution for*

$$\begin{aligned} &\min_{\delta \in \mathbb{R}} \ (w - \delta)^2 \\ &\text{s. t. } |\delta| \leq \varepsilon, \quad x + \delta \in [0,1] \end{aligned} \tag{A3}$$

is

$$\mathcal{P}_{\infty,\text{box}} =$$

$$\begin{cases} w, & w \in [\max(-x,-\varepsilon), \min(1-x,\varepsilon)] \\ \max(-x,-\varepsilon), & w \le \max(-x,-\varepsilon) \\ \min(1-x,\varepsilon), & w \ge \min(1-x,\varepsilon) \end{cases} \tag{A4}$$

which agrees with the sequential projectors $\mathcal{P}_\infty\mathcal{P}_{\text{box}}$ *and* $\mathcal{P}_{\text{box}}\mathcal{P}_\infty$.

One can derive the one-step projection formula Eq. (A4) easily once recognizing the two box constraints can be combined into one:

$$\max(-\varepsilon,-x) \le \delta \le \min(\varepsilon, 1-x) \tag{A5}$$

To show the equivalence to $\mathcal{P}_\infty\mathcal{P}_{\text{box}}$ and $\mathcal{P}_{\text{box}}\mathcal{P}_\infty$, we could write all projectors analytically and directly verify the claimed equivalence. But that tends to be cumbersome. Here, we invoke an elegant result due to [77]. For this, we need to quickly set up the notations. For any function $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$, its proximal mapping $\text{Prox}_f(\boldsymbol{y})$ is defined as

$$\text{Prox}_f(\boldsymbol{y}) \doteq \arg\min_{\boldsymbol{z}\in\mathbb{R}^n} \frac{1}{2}\|\boldsymbol{y}-\boldsymbol{z}\|_2^2 + f(\boldsymbol{z}) \tag{A6}$$

When f is the indicator function $\imath_C$ for a set C defined as

$$\imath_C(\boldsymbol{z}) = \begin{cases} 0 & \boldsymbol{z} \in C \\ \infty & \text{otherwise} \end{cases} \tag{A7}$$

$\text{Prox}_f(\boldsymbol{y})$ is the Euclidean projector $\mathcal{P}_C(\boldsymbol{y})$. For two closed proper convex functions f and g, [77] studies when $\text{Prox}_{f+g} = \text{Prox}_f \circ \text{Prox}_g$. If f and g are two set indicator functions, this exactly asks when the sequential projector is equivalent to the true projector.

Theorem A.2 (adapted from Theorem 2 of [77]). *If* $f = \imath_C$ *for a closed convex set* $C \subset \mathbb{R}$, $\text{Prox}_f \circ \text{Prox}_g = \text{Prox}_{f+g}$ *for all closed proper convex functions* $g : \mathbb{R} \to \mathbb{R} \cup \{\pm\infty\}$.

The equivalence of projectors we claim in Lemma A.1 follows by setting $f = \imath_\infty$ and $g = \imath_{\text{box}}$, and vise versa.

For $p = 2$, the sequential projectors are not equivalent to the true projector in general, although empirically we observe that $\mathcal{P}_2\mathcal{P}_{\text{box}}$ is a much better approximation than $\mathcal{P}_{\text{box}}\mathcal{P}_2$. The former is used in the APGD algorithm of current `AutoAttack`.

Lemma A.3. *Assume* $\boldsymbol{x} \in [0,1]^n$. *When* $p = 2$, *the projector for Form. (A2)* $\mathcal{P}_{2,\text{box}}$ *does not agree with the sequential projectors* $\mathcal{P}_2\mathcal{P}_{\text{box}}$ *and* $\mathcal{P}_{\text{box}}\mathcal{P}_2$ *in general. However, both* $\mathcal{P}_2\mathcal{P}_{\text{box}}$ *and* $\mathcal{P}_{\text{box}}\mathcal{P}_2$ *always find feasible points for the projection problem.*

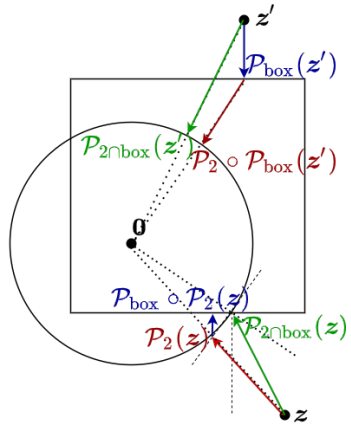


Fig. A1: Illustration of the problem with the sequential projectors when $p = 2$. In general, neither of the sequential projectors produces the right projection.

Proof For the nonequivalence, we present a couple of counter-examples in Fig. A1. Note that the point $\boldsymbol{z}$ is inside the normal cone of the bottom right corner point of the intersection: $\left\{\boldsymbol{\delta} \in \mathbb{R}^2 : \|\boldsymbol{\delta}\|_2 \le \varepsilon\right\} \cap \left\{\boldsymbol{\delta} \in \mathbb{R}^2 : \boldsymbol{x} + \boldsymbol{\delta} \in [0,1]^2\right\}$.

For the feasibility claim, note that for any $\boldsymbol{y} \in \mathbb{R}^n$

$$\mathcal{P}_2(\boldsymbol{y}) = \begin{cases} \varepsilon \frac{\boldsymbol{y}}{\|\boldsymbol{y}\|_2} & \|\boldsymbol{y}\|_2 \ge \varepsilon \\ \boldsymbol{y} & \text{otherwise} \end{cases} \tag{A8}$$

and for any $y \in \mathbb{R}$,

$$\mathcal{P}_{\text{box}}(y) = \begin{cases} 1-x & y \ge 1-x \\ y & -x < y < 1-x \\ -x & y \le -x \end{cases} \tag{A9}$$

and $\mathcal{P}_{\text{box}}(\boldsymbol{y})$ acts on any $\boldsymbol{y} \in \mathbb{R}^n$ elementwise. For any $\boldsymbol{y}$ inside the ℓ_2 ball,

$$\begin{aligned} \|\mathcal{P}_{\text{box}}(\boldsymbol{y})\|_2 &= \|\mathcal{P}_{\text{box}}(\boldsymbol{y}) - \mathcal{P}_{\text{box}}(\boldsymbol{0})\|_2 \\ &\le \|\boldsymbol{y} - \boldsymbol{0}\|_2 = \|\boldsymbol{y}\|_2 \le \varepsilon \end{aligned} \tag{A10}$$

due to the contraction property of projecting onto convex sets. Therefore, $\mathcal{P}_{\text{box}}\mathcal{P}_2(\boldsymbol{y})$ is feasible for any $\boldsymbol{y} \in \mathbb{R}^n$. Now for any $\boldsymbol{y}$ inside the box:

- if $\|\boldsymbol{y}\|_2 < \varepsilon$, $\mathcal{P}_2(\boldsymbol{y}) = \boldsymbol{y}$ and so $\mathcal{P}_2(\boldsymbol{y})$ remains in the box;
- if $\|\boldsymbol{y}\|_2 \geq \varepsilon$, $\mathcal{P}_2(\boldsymbol{y}) = \varepsilon \frac{\boldsymbol{y}}{\|\boldsymbol{y}\|_2}$. Since $\varepsilon/\|\boldsymbol{y}\|_2 \in [0, 1]$, $\boldsymbol{P}_2(\boldsymbol{y})$ shrinks each component of $\boldsymbol{y}$ but keeps their original signs. Thus $\boldsymbol{P}_2(\boldsymbol{y})$ remains in the box if $\boldsymbol{y}$ is in the box.

We conclude that $\mathcal{P}_2\mathcal{P}_{\text{box}}(\boldsymbol{y})$ is feasible for any $\boldsymbol{y}$, completing the proof. □

Appendix B

Sketch of the BFGS-SQP algorithm in GRANSO GRANSO is among the first optimization solvers targeting general nonsmooth, nonconvex problems with nonsmooth constraints [41].

The key algorithm of GRANSO package is a sequential quadratic programming (SQP) that employs a quasi-Newton algorithm, Broyden-Fletcher-Goldfarb-Shanno (BFGS) [46], and an exactly penalty function (Penalty-SQP).

The Penalty-SQP calculates the alternative search direction from the following quadratic programming (QP) problem :

$$\begin{aligned} \min_{\boldsymbol{d}\in\mathbb{R}^n, \boldsymbol{s}\in\mathbb{R}^p} \; & \mu\left(f\left(\boldsymbol{x}_k\right) + \nabla f\left(\boldsymbol{x}_k\right)^{\intercal} \boldsymbol{d}\right) \\ & + \boldsymbol{e}^{\intercal}\boldsymbol{s} + \frac{1}{2}\boldsymbol{d}^{\intercal}\boldsymbol{H}_k\boldsymbol{d} \\ \text{s.t. } & c\left(\boldsymbol{x}_k\right) + \nabla c\left(\boldsymbol{x}_k\right)^{\intercal}\boldsymbol{d} \leq \boldsymbol{s}, \quad \boldsymbol{s} \geq 0 \end{aligned} \tag{B11}$$

Here we abuse the notation $c(\cdot)$ to be the total constraints for simplicity (i.e., representing all c's and h's in Eq. (8)). The dual of problem (Eq. (B11)) is used in GRANSO package:

$$\begin{aligned} \max_{\boldsymbol{\lambda}\in\mathbb{R}^p} \; & \mu f(\boldsymbol{x}_k) + c(\boldsymbol{x}_k)^{\intercal}\boldsymbol{\lambda} \\ & - \frac{1}{2}\left(\mu\nabla f(\boldsymbol{x}_k) + \nabla c(\boldsymbol{x}_k)\boldsymbol{\lambda}\right)^{\intercal}\boldsymbol{H}_k^{-1} \\ & \cdot \left(\mu\nabla f(\boldsymbol{x}_k) + \nabla c(\boldsymbol{x}_k)\boldsymbol{\lambda}\right) \\ \text{s.t. } & 0 \leq \boldsymbol{\lambda} \leq \boldsymbol{e} \end{aligned} \tag{B12}$$

which has only simple box constraints that can be easily handled by many popular QP solvers such as OSQP (ADMM-based algorithm) [78]. Then the primal solution $\boldsymbol{d}$ can be recovered from the dual solution $\boldsymbol{\lambda}$ by solving Form. (B12):

$$\boldsymbol{d} = -\boldsymbol{H}_k^{-1}\left(\mu\nabla f(\boldsymbol{x}_k) + \nabla c(\boldsymbol{x}_k)\boldsymbol{\lambda}\right) \tag{B13}$$

The search direction calculated at each step controls the trade-off between minimizing the objective and moving towards the feasible region. To measure the how much violence the current searching direction will give, a linear model of constraint violation is used:

$$l(\boldsymbol{d};\ \boldsymbol{x}_k) := \left\|\max\left\{c(\boldsymbol{x}_k) + \nabla c(\boldsymbol{x}_k)^{\intercal}\boldsymbol{d}, \boldsymbol{0}\right\}\right\|_1 \tag{B14}$$

To dynamically set the penalty parameter, a steering strategy as Algorithm 3 is used:

Algorithm 3
$[\boldsymbol{d}_k, \mu_{\text{new}}] = \texttt{sqp\_steering}(\boldsymbol{x}_k, \boldsymbol{H}_k, \mu)$

Require: $\boldsymbol{x}_k, \boldsymbol{H}_k, \mu$ at current iteration
Require: constants $c_v \in (0, 1), c_\mu(0, 1)$
1: Calculate $\boldsymbol{d}_k$ from Eq. (B13) and Form. (B12) with $\mu_{\text{new}} = \mu$
2: **if** $l_\delta(\boldsymbol{d}_k;\ \boldsymbol{x}_k) < c_v v(\boldsymbol{x}_k)$ **then**
3: Calculate $\tilde{\boldsymbol{d}}_k$ from Eq. (B13) and Form. (B12) with $\mu = 0$
4: **while** $l_\delta(\boldsymbol{d}_k;\ \boldsymbol{x}_k) < c_v l_\delta(\tilde{\boldsymbol{d}}_k;\ \boldsymbol{x}_k)$ **do**
5: $\mu_{new} := c_\mu\mu_{\text{new}}$
6: Calculate $\boldsymbol{d}_k$ from Eq. (B13) and Form. (B12) with $\mu = \mu_{\text{new}}$
7: **end while**
8: **end if**
9: **return** $\boldsymbol{d}_k, \mu_{\text{new}}$

For non-smooth problems, it is usually hard to find a reliable stopping criterion as the norm of the gradient will not decrease when approaching the minimizer. GRANSO uses an alternative stopping strategy, which is based on the idea of gradient sampling [79] [55].

Define the neighboring gradient information (from the l most recent iterates) as:

$$\begin{aligned} \boldsymbol{G} &:= \left[\nabla f(\boldsymbol{x}_{x_{k+1-l}}) \ldots \nabla f(\boldsymbol{x}_k)\right], \\ \boldsymbol{J}_i &:= \left[\nabla c_i(\boldsymbol{x}_{x_{k+1-l}}) \ldots \nabla c_i(\boldsymbol{x}_k)\right], \\ & i \in \{1, \ldots, p\} \end{aligned} \tag{B15}$$

Augment Form. (B11) and its dual Form. (B12) in the steering strategy, we can obtain the augmented dual problem:

$$\begin{aligned}&\max_{\boldsymbol{\sigma}\in\mathbb{R}^l,\boldsymbol{\lambda}\in\mathbb{R}^{pl}} \sum_{i=1}^{p} c_i(\boldsymbol{x}_k)\boldsymbol{e}^\intercal\boldsymbol{\lambda}_i \\ &-\frac{1}{2}\begin{bmatrix}\boldsymbol{\sigma}\\ \boldsymbol{\lambda}\end{bmatrix}^\intercal [\boldsymbol{G},\boldsymbol{J}_1,\ldots,\boldsymbol{J}_p]^\intercal \boldsymbol{H}_k^{-1}[\boldsymbol{G},\boldsymbol{J}_1,\ldots,\boldsymbol{J}_p]\begin{bmatrix}\boldsymbol{\sigma}\\ \boldsymbol{\lambda}\end{bmatrix} \\ &\text{s.t.}\ \boldsymbol{0}\le\boldsymbol{\lambda}_i\le\boldsymbol{e},\quad \boldsymbol{e}^\intercal\boldsymbol{\sigma}=\mu,\quad \boldsymbol{\sigma}\ge\boldsymbol{0}\end{aligned} \tag{B16}$$

By solving Eq. (B16), we can obtain $\boldsymbol{d}_\diamond$:

$$\boldsymbol{d}_\diamond = \boldsymbol{H}_k^{-1}[\boldsymbol{G},\boldsymbol{J}_1,\ldots,\boldsymbol{J}_p]\begin{bmatrix}\boldsymbol{\sigma}\\ \boldsymbol{\lambda}\end{bmatrix} \tag{B17}$$

If the norm of $\boldsymbol{d}_\diamond$ is sufficiently small, the current iteration can be viewed as near a small neighborhood of a stationary point.

Algorithm 4

$[\boldsymbol{x}_*, f_*, \boldsymbol{v}_*] = \texttt{bfgs\_sqp}(f(\cdot), \boldsymbol{c}(\cdot), \boldsymbol{x}_0, \mu_0))$

Require: $f, \boldsymbol{c}, \boldsymbol{x}_0, \mu_0$
Require: constants $\tau_\diamond, \tau_v$
$\boldsymbol{H}_0 := \boldsymbol{I},\ \mu := \mu_0$
$\phi(\cdot) = \mu f(\cdot) + v(\cdot)$
$\nabla\phi(\cdot) = \mu\nabla f(\cdot) + \sum_{i\in\mathcal{P}}\nabla c_i(\cdot)$
$v(\cdot) = \|\max\{c(\cdot), 0\}\|_1$
$\phi_0 := \phi(\boldsymbol{x}_0;\ \mu), \nabla\phi_0 := \nabla\phi(\boldsymbol{x}_0;\ \mu), v_0 := v(\boldsymbol{x}_0)$
for $k = 0, 1, 2, \ldots$ **do**
  $[\boldsymbol{d}_k, \hat{\mu}] := \texttt{sqp\_steering}(\boldsymbol{x}_k, \boldsymbol{H}_k, \boldsymbol{\mu})$
  if $\hat{\mu} < \mu$ **then**
    $\mu := \hat{\mu}$
    $\phi_k := \phi(\boldsymbol{x}_k;\ \mu), \nabla\phi_k := \nabla\phi(\boldsymbol{x}_k;\ \mu), v_k := v(\boldsymbol{x}_k)$
  end if
  $[\boldsymbol{x}_{k+1}, \phi_{k+1}, \nabla\phi_{k+1}, v_{k+1}] := \texttt{Armijo\_Wolfe}(\boldsymbol{x}_k, \phi_k, \nabla\phi_k, \phi(\cdot), \nabla\phi(\cdot))$
  Get $\boldsymbol{d}_\diamond$ from Eq. (B17) and Form. (B16)
  if $\|\boldsymbol{d}_\diamond\|_2 < \tau_\diamond$ and $v_{k+1} < \tau_v$ **then**
    break
  end if
  BFGS update $\boldsymbol{H}_{k+1}$
end for
return $\boldsymbol{x}_*, f_*, \boldsymbol{v}_*$

Appendix C

Danskin's theorem and min-max optimization In this section, we discuss the importance of computing a good solution to the inner maximization problem when applying first-order methods for AT, i.e., solving Form. (3).

Consider the minimax problem

$$\min_{\boldsymbol{\theta}} g(\boldsymbol{\theta}) \doteq \left[\max_{\boldsymbol{x}'\in\Delta} h(\boldsymbol{\theta}, \boldsymbol{x}')\right], \tag{C18}$$

where we assume the function h is locally Lipschitz continuous. To apply first-order methods to solve Eq. (C18), one needs to evaluate a (sub)gradient of g at any given $\boldsymbol{\theta}$. If $h(\boldsymbol{\theta}, \boldsymbol{x}')$ is smooth in $\boldsymbol{\theta}$, one can invoke Danskin's theorem for such an evaluation (see, for example, [11, Appendix A]). However, in DL applications with nonsmooth activations or losses, $h(\boldsymbol{\theta}, \delta)$ is not differentiable in $\boldsymbol{\theta}$, and hence a general version of Danskin's theorem is needed.

To proceed, we first introduce a few basic concepts in nonsmooth analysis; general background can be found in [52–54]. For a locally Lipschitz continuous function $\varphi : \mathbb{R}^n \to \mathbb{R}$, define its Clarke directional derivative at $\bar{\boldsymbol{z}} \in \mathbb{R}^n$ in any direction $\boldsymbol{d} \in \mathbb{R}^n$ as

$$\varphi^\circ(\bar{\boldsymbol{z}};\ \boldsymbol{d}) \doteq \limsup_{t\downarrow 0, \boldsymbol{z}\to\bar{\boldsymbol{z}}} \frac{\varphi(\boldsymbol{z}+t\boldsymbol{d}) - \varphi(\boldsymbol{z})}{t}$$

We say φ is Clarke regular at $\bar{\boldsymbol{z}}$ if $\varphi^\circ(\bar{\boldsymbol{z}};\ \boldsymbol{d}) = \varphi'(\bar{\boldsymbol{z}};\ \boldsymbol{d})$ for any $\boldsymbol{d} \in \mathbb{R}^n$, where $\varphi'(\bar{\boldsymbol{z}};\ \boldsymbol{d}) \doteq \lim_{t\downarrow 0}\frac{1}{t}(\varphi(\bar{\boldsymbol{z}}+t\boldsymbol{d}) - \varphi(\bar{\boldsymbol{z}}))$ is the usual one-sided directional derivative. The Clarke subdifferential of φ at $\bar{\boldsymbol{z}}$ is defined as

$$\partial\varphi(\bar{\boldsymbol{z}}) \doteq \{\boldsymbol{v} \in \mathbb{R}^n : \varphi^\circ(\bar{\boldsymbol{z}};\ \boldsymbol{d}) \ge \boldsymbol{v}^\intercal\boldsymbol{d}\}$$

The following result has its source in [80, Theorem 2.1]; see also [54, Section 5.5].

Theorem C.1. *Assume that Δ in Eq. (C18) is a compact set, and the function h satisfies*

1. *h is jointly upper semicontinuous in $(\boldsymbol{\theta}, \boldsymbol{x}')$;*
2. *h is locally Lipschitz continuous in $\boldsymbol{\theta}$, and the Lipschitz constant is uniform in $\boldsymbol{x}' \in \Delta$;*
3. *h is directionally differentiable in $\boldsymbol{\theta}$ for all $\boldsymbol{x}' \in \Delta$;*

If h is Clarke regular in $\boldsymbol{\theta}$ for all $\boldsymbol{\theta}$, and $\partial_{\boldsymbol{\theta}} h$ is upper semicontinuous in $(\boldsymbol{\theta}, \boldsymbol{x}')$, we have that for any $\bar{\boldsymbol{\theta}}$

$$\partial g(\bar{\boldsymbol{\theta}}) = \text{conv}\{\partial h(\bar{\boldsymbol{\theta}}, \boldsymbol{x}') : \boldsymbol{x}' \in \Delta^*(\bar{\boldsymbol{\theta}})\} \quad \text{(C19)}$$

where $\text{conv}(\cdot)$ *denotes the convex hull of a set, and* $\Delta^*(\bar{\boldsymbol{\theta}})$ *is the set of all optimal solutions of the inner maximization problem at* $\bar{\boldsymbol{\theta}}$.

The above theorem indicates that in order to get an element from the subdifferential set $\partial g(\bar{\boldsymbol{\theta}})$, we need to get at least one **optimal** solution $\boldsymbol{x}' \in \Delta^*(\bar{\boldsymbol{\theta}})$. A suboptimal solution to the inner maximization problem may result in a useless direction for the algorithm to proceed. To illustrate this, let us consider a simple one-dimensional example

$$\min_{\theta} g(\theta) := \left[\max_{-1 \le x' \le 1} \max(\theta x', 0)^2 \right]$$

which corresponds to a one-layer neural network with one data point $(0, 0)$, the ReLU activation function and squared loss. Starting at $\theta_0 = 1$, we get the first inner maximization problem $\max_{-1 \le x' \le 1} \max(x', 0)^2$. Although its global optimal solution is $x'_* = 1$, the point $x' = 0$ is a stationary point satisfying the first-order optimality condition. If the latter point is mistakenly adopted to compute an element in $\partial g(\theta^0)$, it would result in a zero direction so that the overall gradient descent algorithm cannot proceed.

References

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[2] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6572

[3] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," in *International Conference on Learning Representations*, 2018.

[4] L. Engstrom, B. Tran, D. Tsipras, L. Schmidt, and A. Madry, "Exploring the landscape of spatial robustness," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 1802–1811. [Online]. Available: https://proceedings.mlr.press/v97/engstrom19a.html

[5] C. Xiao, J. Zhu, B. Li, W. He, M. Liu, and D. Song, "Spatially transformed adversarial examples," in *International Conference on Learning Representations*, 2018.

[6] E. Wong, F. R. Schmidt, and J. Z. Kolter, "Wasserstein adversarial examples via projected sinkhorn iterations," *arXiv:1902.07906*, Feb. 2019.

[7] C. Laidlaw and S. Feizi, "Functional adversarial attacks," *Advances in neural information processing systems*, vol. 32, 2019.

[8] H. Hosseini and R. Poovendran, "Semantic adversarial examples," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 1614–1619.

[9] A. Bhattad, M. J. Chong, K. Liang, B. Li, and D. A. Forsyth, "Big but imperceptible

adversarial perturbations via semantic manipulation," *arXiv preprint arXiv:1904.06347*, vol. 1, no. 3, 2019.

[10] R. Huang, B. Xu, D. Schuurmans, and C. Szepesvari, "Learning with a strong adversary," *arXiv:1511.03034*, Nov. 2015.

[11] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[12] C. Laidlaw, S. Singla, and S. Feizi, "Perceptual adversarial robustness: Defense against unseen threat models," in *ICLR*, 2021.

[13] Y. Zhang, G. Zhang, P. Khanduri, M. Hong, S. Chang, and S. Liu, "Revisiting and advancing fast adversarial training through the lens of bi-level optimization," *arXiv:2112.12376*, Dec. 2021.

[14] F. Croce and M. Hein, "Minimally distorted adversarial examples with a fast adaptive boundary attack," in *International Conference on Machine Learning*. PMLR, 2020, pp. 2196–2205.

[15] ——, "Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks," in *International conference on machine learning*. PMLR, 2020, pp. 2206–2216.

[16] M. Pintor, F. Roli, W. Brendel, and B. Biggio, "Fast minimum-norm adversarial attacks through adaptive norm constraints," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[17] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev, "Fast and effective robustness certification," *Advances in neural information processing systems*, vol. 31, 2018.

[18] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "Boosting robustness certification of neural networks," in *International conference on learning representations*, 2018.

[19] ——, "An abstract domain for certifying neural networks," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–30, 2019.

[20] H. Salman, G. Yang, H. Zhang, C.-J. Hsieh, and P. Zhang, "A convex relaxation barrier to tight robustness verification of neural networks," *arXiv:1902.08722*, Feb. 2019.

[21] S. Dathathri, K. Dvijotham, A. Kurakin, A. Raghunathan, J. Uesato, R. R. Bunel, S. Shankar, J. Steinhardt, I. Goodfellow, P. S. Liang *et al.*, "Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming," *Advances in Neural Information Processing Systems*, vol. 33, pp. 5318–5331, 2020.

[22] M. N. Müller, G. Makarchuk, G. Singh, M. Püschel, and M. Vechev, "PRIMA: general and precise neural network certification via scalable convex hull approximations," *Proceedings of the ACM on Programming Languages*, vol. 6, no. POPL, pp. 1–33, jan 2022.

[23] E. Wong and J. Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," *arXiv:1711.00851*, Nov. 2017.

[24] A. Raghunathan, J. Steinhardt, and P. Liang, "Certified defenses against adversarial examples," *arXiv:1801.09344*, Jan. 2018.

[25] E. Wong, F. Schmidt, J. H. Metzen, and J. Z. Kolter, "Scaling provable adversarial defenses," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[26] K. Dvijotham, S. Gowal, R. Stanforth, R. Arandjelovic, B. O'Donoghue, J. Uesato, and P. Kohli, "Training verified learners with learned verifiers," *arXiv:1805.10265*, May 2018.

[27] S. Lee, W. Lee, J. Park, and J. Lee, "Towards better understanding of training certifiably robust models against adversarial examples," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[28] V. Tjeng, K. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," *arXiv:1711.07356*, Nov. 2017.

[29] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," *arXiv:1702.01135*, Feb. 2017.

[30] R. Bunel, P. Mudigonda, I. Turkaslan, P. Torr, J. Lu, and P. Kohli, "Branch and bound for piecewise linear neural network verification," *Journal of Machine Learning Research*, vol. 21, no. 2020, 2020.

[31] L. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, L. Daniel, D. Boning, and I. Dhillon, "Towards fast computation of certified robustness for relu networks," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5276–5285.

[32] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," *Advances in neural information processing systems*, vol. 31, 2018.

[33] T. Weng, H. Zhang, P. Chen, J. Yi, D. Su, Y. Gao, C. Hsieh, and L. Daniel, "Evaluating the robustness of neural networks: An extreme value theory approach," *arXiv preprint arXiv:1801.10578*, 2018.

[34] Z. Lyu, C. Ko, Z. Kong, N. Wong, D. Lin, and L. Daniel, "Fastened crown: Tightened neural network robustness certificates," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 5037–5044.

[35] S. M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," *arXiv:1511.04599.*, Nov. 2015.

[36] M. Hein and M. Andriushchenko, "Formal guarantees on the robustness of a classifier against adversarial manipulation," *arXiv:1705.08475*, May 2017.

[37] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," *arXiv:1608.04644*, Aug. 2016.

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

[39] F. Croce, M. Andriushchenko, V. Sehwag, E. Debenedetti, N. Flammarion, M. Chiang, P. Mittal, and M. Hein, "Robustbench: a standardized adversarial robustness benchmark," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[40] T. Bai, J. Luo, J. Zhao, B. Wen, and Q. Wang, "Recent advances in adversarial training for adversarial robustness," *arXiv preprint arXiv:2102.01356*, 2021.

[41] F. E. Curtis, T. Mitchell, and M. L. Overton, "A bfgs-sqp method for nonsmooth, nonconvex, constrained optimization and its evaluation using relative minimization profiles," *Optimization Methods and Software*, vol. 32, no. 1, pp. 148–181, 2017.

[42] B. Liang, T. Mitchell, and J. Sun, "NCVX: A general-purpose optimization solver for constrained machine and deep learning," 2022.

[43] M. Mosbach, M. Andriushchenko, T. Trost, M. Hein, and D. Klakow, "Logit pairing methods can fool gradient-based attacks," *arXiv preprint arXiv:1810.12042*, 2018.

[44] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. Goodfellow, A. Madry, and A. Kurakin, "On evaluating adversarial robustness," *arXiv:1902.06705*, Feb. 2019.

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

[46] S. Wright, J. Nocedal *et al.*, "Numerical optimization," *Springer Science*, vol. 35, no. 67-68, p. 7, 1999.

[47] D. Bertsekas, *Nonlinear Programming 3rd Edition*. Athena Scientific, 2016.

[48] G. Pillo and M. Roma, *Large-scale nonlinear optimization*. Springer Science & Business Media, 2006, vol. 83.

[49] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, no. 1, pp. 25–57, 2006.

[50] S. Laue, M. Mitterreiter, and J. Giesen, "Geno–generic optimization for classical machine learning," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[51] S. Laue, M. Blacher, and J. Giesen, "Optimization for classical machine learning problems on the gpu," *arXiv:2203.16340*, Mar. 2022.

[52] F. H. Clarke, *Optimization and nonsmooth analysis*. SIAM, 1990.

[53] A. Bagirov, N. Karmitsa, and M. M. Mäkelä, *Introduction to Nonsmooth Optimization*. Springer International Publishing, 2014.

[54] Y. Cui and J. S. Pang, *Modern Nonconvex Nondifferentiable Optimization*. Society for Industrial and Applied Mathematics, Jan 2021.

[55] J. V. Burke, F. E. Curtis, A. S. Lewis, M. L. Overton, and L. E. Simões, "Gradient sampling methods for nonsmooth optimization," *Numerical Nonsmooth Optimization*, pp. 201–225, 2020.

[56] J. M. Danskin, *The Theory of Max-Min and its Application to Weapons Allocation Problems*. Springer Berlin Heidelberg, 1967.

[57] P. Bernhard and A. Rapaport, "On a theorem of danskin with an application to a theorem of von neumann-sion," *Nonlinear Analysis: Theory, Methods & Applications*, vol. 24, no. 8, pp. 1163–1181, apr 1995.

[58] M. Razaviyayn, T. Huang, S. Lu, M. Nouiehed, M. Sanjabi, and M. Hong, "Non-convex min-max optimization: Applications, challenges, and recent theoretical advances," *IEEE Signal Processing Magazine (Volume: 37, Issue: 5, Sept. 2020)*, Jun. 2020.

[59] J. Martins and N. M. Poon, "On structural optimization using constraint aggregation," in *VI World Congress on Structural and Multidisciplinary Optimization WCSMO6, Rio de Janeiro, Brasil*. Citeseer, 2005.

[60] K. Zhang, Z. Han, Z. Gao, and Y. Wang, "Constraint aggregation for large number of constraints in wing surrogate-based optimization," *Structural and Multidisciplinary Optimization*, vol. 59, no. 2, pp. 421–438, sep 2018.

[61] F. Domes and A. Neumaier, "Constraint aggregation for rigorous global optimization," *Mathematical Programming*, vol. 155, no. 1-2, pp. 375–401, dec 2014.

[62] Y. M. Ermoliev, A. V. Kryazhimskii, and A. Ruszczyński, "Constraint aggregation principle in convex optimization," *Mathematical Programming*, vol. 76, no. 3, pp. 353–372, mar 1997.

[63] A. C. Trapp and O. A. Prokopyev, "A note on constraint aggregation and value functions for two-stage stochastic integer programs," *Discrete Optimization*, vol. 15, pp. 37–45, feb 2015.

[64] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[65] C. Studer, W. Yin, and R. G. Baraniuk, "Signal representations with minimum ℓ_∞," in *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, oct 2012.

[66] T. Hastie, R. Tibshirani, and M. Wainwright, *Statistical Learning with Sparsity*. Chapman and Hall/CRC, may 2015.

[67] J. Wright and Y. Ma, *High-Dimensional Data Analysis with Low-Dimensional Models Principles, Computation, and Applications*. University of Cambridge ESOL Examinations, 2021.

[68] N. Papernot, F. Faghri, N. Carlini, I. Goodfellow, R. Feinman, A. Kurakin, C. Xie, Y. Sharma, T. Brown, A. Roy *et al.*, "Technical report on the cleverhans v2. 1.0 adversarial examples library," *arXiv preprint arXiv:1610.00768*, 2016.

[69] J. Rauber, W. Brendel, and M. Bethge, "Foolbox: A python toolbox to benchmark the robustness of machine learning models," *arXiv preprint arXiv:1707.04131*, 2017.

[70] K. Sridhar, S. Dutta, R. Kaur, J. Weimer, O. Sokolsky, and I. Lee, "Towards alternative techniques for improving adversarial robustness: Analysis of adversarial training at a spectrum of perturbations," *arXiv preprint arXiv:2206.06496*, 2022.

[71] A. Raghunathan*, S. M. Xie*, F. Yang, J. Duchi, and P. Liang, "Adversarial training can hurt generalization," in *ICML 2019 Workshop on Identifying and Understanding Deep Learning Phenomena*, 2019. [Online]. Available: https://openreview.net/forum?id=SyxM3J256E

[72] Y.-Y. Yang, C. Rashtchian, H. Zhang, R. R. Salakhutdinov, and K. Chaudhuri, "A closer look at accuracy vs. robustness," *Advances in Neural Information Processing Systems*, vol. 33, pp. 8588–8601, 2020.

[73] P. Maini, E. Wong, and Z. Kolter, "Adversarial robustness against the union of multiple perturbation models," in *International Conference on Machine Learning*. PMLR, 2020, pp. 6640–6650.

[74] F. Croce and M. Hein, "Provable robustness against all adversarial ℓ_p-perturbations for $p \geq 1$," *arXiv preprint arXiv:1905.11213*, 2019.

[75] J. Gilmer, R. P. Adams, I. Goodfellow, D. Andersen, and G. E. Dahl, "Motivating the rules of the game for adversarial example research," *arXiv preprint arXiv:1807.06732*, 2018.

[76] F. Croce and M. Hein, "Mind the box: ℓ_1-apgd for sparse adversarial attacks on image classifiers," in *International Conference on Machine Learning*. PMLR, 2021, pp. 2201–2211.

[77] Y.-L. Yu, "On decomposing the proximal map," *Advances in neural information processing systems*, vol. 26, 2013.

[78] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: an operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020. [Online]. Available: https://doi.org/10.1007/s12532-020-00179-2

[79] A. S. Lewis and M. L. Overton, "Nonsmooth optimization via quasi-newton methods," *Mathematical Programming*, vol. 141, no. 1, pp. 135–163, 2013.

[80] F. H. Clarke, "Generalized gradients and applications," *Transactions of the American Mathematical Society*, vol. 205, pp. 247–262, 1975.