CHAPTER 2

White-box adversarial attacks

Prediction-evasive adversarial attacks are procedures to generate adversarial examples on a given machine learning model. For a target model F_θ (also called the victim model from the security point of view), the most general definition of adversarial example is an input $\boldsymbol{x}$ such that the model prediction $F_\theta(\boldsymbol{x})$ is clearly inconsistent with human perception, where θ is the set of model parameters. For brevity, we will use F to denote the target model in the remaining chapter. To construct an adversarial example, a typical procedure is to start with a natural data input (e.g., an image) that is correctly classified by the machine learning model and then add a human imperceptible perturbation to change the predicted label. This is also called evasion attack or test-time attack, since the attacker is trying to add a small perturbation in the test time to make the machine learning model misclassify without interfering the training procedure. Fig. 2.1 gives an example of an adversarial attack, where the original bagel image is correctly classified as a *bagel*, but the perturbed example, after adding a human imperceptible noise, will be classified as a *piano*. In practice, an adversarial perturbation could lead to catastrophic damage to a real-time system. For instance, researchers have demonstrated that it is possible to slightly perturb a stop sign to make it being recognized as a speed limit sign by a machine learning-based image recognition system (Eykholt et al., 2018).

In this chapter, we discuss how to conduct adversarial attacks in the *white-box setting*. In this setting the attacker is assumed to have full knowledge of the victim machine learning model. For instance, if the victim

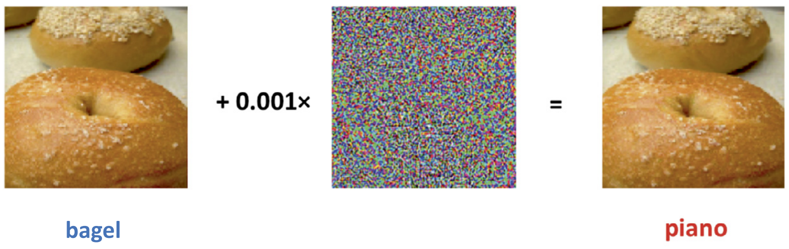

Figure 2.1 An illustration of adversarial example from (Chen et al., 2017a).

model F is a neural network, we will assume that the attacker knows both the architecture and weights of this network in the white-box setting. Since the attacker can leverage full knowledge to attack the victim model, white-box attacks are the easiest way to construct adversarial examples and also lead to the highest successful rate and smallest perturbations. In practice, the white-box assumption is often too strong since the underlying model used in production is usually hidden to the attacker, and we will discuss how to conduct the attacks under those more realistic settings in the next chapter. Despite being unrealistic, white-box attacks are more reliable ways to evaluate the robustness of a model, since they usually lead to strongest attacks that can be used to evaluate the worst-case performance of a system. Therefore white-box attacks can be used by model developers to provide in-house performance evaluation.

2.1 Attack procedure and notations

Now we formally define the procedure of an adversarial attack. For simplicity, we will mainly focus on the multiclass classification problem in this chapter. For a K-way multiclass classification problem, we define the model as $F : \mathbb{R}^d \rightarrow \{1, \ldots, K\}$ that maps an input $\boldsymbol{x}$ to a predicted class label. We assume that the model makes the final decision by first computing a score for each class and then choosing the class with the maximum score. Mathematically, we use $f_j(\boldsymbol{x})$ to denote the predicted score for the jth class, and the label with highest score will be the final decision of the model: $F(\boldsymbol{x}) = \text{argmax}_{j\in\{1,\ldots,K\}} f_j(\boldsymbol{x})$.

Assuming that $\boldsymbol{x}_0 \in \mathbb{R}^d$ is a natural example with the ground truth label $y_0 \in \{1, \ldots, K\}$, a prediction-evasion attack will try to find a small perturbation $\boldsymbol{\Delta}$ such that $F(\boldsymbol{x}_0 + \boldsymbol{\Delta}) \neq y_0$. The perturbed example $\boldsymbol{x}^* = \boldsymbol{x}_0 + \boldsymbol{\Delta}$ is known as an *adversarial example*; this perturbed example is supposed to belong to class y_0 since $\boldsymbol{x}^*$ is only slightly perturbed from the original example $\boldsymbol{x}_0$, but the model will misclassify $\boldsymbol{x}^*$ as another label. The overall procedure is illustrated in Fig. 2.2. The notion of "perturbation" can be generalized to other meaningful data manipulations beyond additive noises.

Untargeted v.s. targeted attack. Multiple attack methods have been introduced for crafting adversarial examples to attack a machine learning model. In general, there are two kinds of "attack goals", targeted and untargeted. For untargeted attack, the attack is successful if the input image is predicted with the wrong label. Take Fig. 2.2 as an example: as long as the bagel image is not classified as a bagel, the attack is considered successful. For

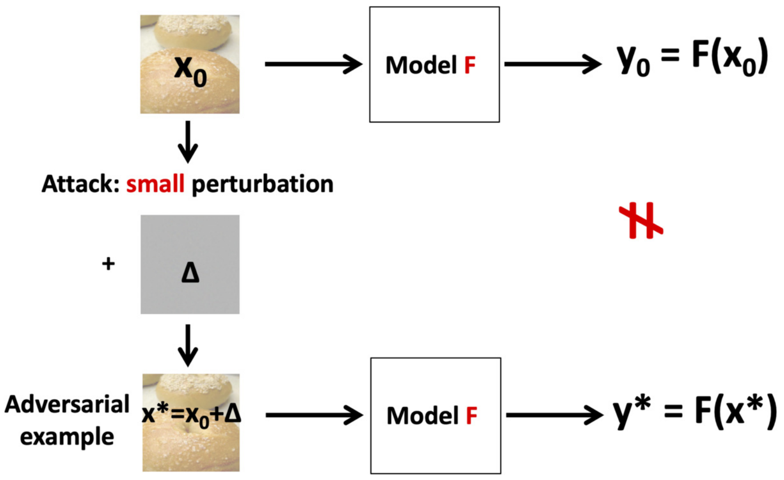

Figure 2.2 The procedure of adversarial attacks.

targeted attack, the attack is successful only when the adversarial example is classified as the target class. In this example, if the target class is piano, then the attack is considered successful only when the perturbed image on the right-hand side is labeled as piano by the victim classifier.

2.2 Formulating attack as constrained optimization

Given the original example $\boldsymbol{x}_0$, an adversarial attack aims to find an adversarial example $\boldsymbol{x}^* = \boldsymbol{x}_0 + \boldsymbol{\Delta}$ that can successfully achieve the attack goal (either targeted or untargeted attack) under the constraint that $\boldsymbol{\Delta}$ should be small. This can be formulated as a constrained optimization problem: we aim to find an adversarial example $\boldsymbol{x}$ that maximizes the "attack successfulness" (or minimizes the attacker's loss function), whereas $\boldsymbol{x}$ is constrained to be close to the original example $\boldsymbol{x}_0$. This can be mathematically written as the following problem:

$$\underset{\boldsymbol{x}}{\operatorname{argmax}}\, g(\boldsymbol{x}) \text{ s.t. } \boldsymbol{x} \in B(\boldsymbol{x}_0), \tag{2.1}$$

where $g(\boldsymbol{x})$ measures the successfulness of the attack, and $B(\boldsymbol{x}_0)$ is a small region around $\boldsymbol{x}_0$ indicating the feasible perturbation range.

The definition of $g(\boldsymbol{x})$ can be defined differently for different attacks. As described in Chapter 1, neural network training usually involves minimizing the cross-entropy (CE) loss, defined as the negative log-likelihood of predicting label y_0:

$$\ell_{\text{CE}}(f(\boldsymbol{x}_0), y_0) := -\log P(y_0 \mid \boldsymbol{x}_0), \quad \text{where } P(y_0 \mid \boldsymbol{x}_0) := \frac{e^{f_{y_0}(\boldsymbol{x}_0)}}{\sum_{j=1}^{K} e^{f_j(\boldsymbol{x}_0)}}. \quad (2.2)$$

Here $f_j(\boldsymbol{x}_0)$ is the class prediction score of class j, and the prediction probability of class j is computed using a softmax operation for normalization. We can define the $g(\boldsymbol{x})$ for adversarial attacks by slightly changing this loss function. For untargeted attack, the goal of "making model predicts another label" can be formulated as "minimizing the probability of predicting class y_0". This can be naturally achieved by setting

$$\text{Untargeted CE loss:} \quad g(\boldsymbol{x}) := \ell_{\text{CE}}(f(\boldsymbol{x}), y_0). \quad (2.3)$$

Since here we want to maximize $g(\boldsymbol{x})$ instead of minimizing it, the perturbation will try to maximally increase the cross-entropy loss to achieve the worst prediction.

For targeted attack, we aim to maximize the predicted probability of the target class t. The attack successfulness can thus be defined as

$$\text{Targeted CE loss:} \quad g(\boldsymbol{x}) := -\ell_{\text{CE}}(f(\boldsymbol{x}), t), \quad (2.4)$$

which is the log-likelihood of the target class probability.

The constraint set $B(\boldsymbol{x}_0)$ is usually defined as a small region around $\boldsymbol{x}_0$. If the ℓ_p norm is used for measuring distance, then we can define

$$B(\boldsymbol{x}_0) = \{\boldsymbol{x} : \|\boldsymbol{x} - \boldsymbol{x}_0\|_p \leq \epsilon\},$$

where ϵ is a predefined small constant, known as the perturbation threshold or budget. For example, on the CIFAR-10 image classification dataset (which is a standard benchmarking dataset for evaluating adversarial attacks), the ϵ is often set as $8/255$ (for ℓ_∞ norm) when each pixel is in the range of $[0, 255]$. Note that for computer vision applications, we usually need to enforce an additional elementwise lower and upper bounds to ensure that the adversarial example $\boldsymbol{x}$ is still a valid image, so the constraint set will become the intersection of ℓ_p ball and $\boldsymbol{x} \in [0, 255]^d$. For simplicity, we will omit this constraint in our derivations since they can often easily done by a simple elementwise projection to the feasible data space.

Once we formulate an adversarial attack as a constraint optimization problem, any existing optimization algorithm can be used for solving this problem to generate an adversarial example. For example, a projected gradient descent method can be applied: at each iteration, we update the current solution $\boldsymbol{x}_t$ by

$$\boldsymbol{x}_{t+1} \leftarrow \Pi(\boldsymbol{x}_t + \alpha \nabla g(\boldsymbol{x})), \tag{2.5}$$

where α is the learning rate of gradient descent, and $\Pi(\cdot)$ projects the vector to $B(\boldsymbol{x}_0)$. However, in practice, we may find that this plain projected gradient descent update is not very effective in solving the attack problem, especially when we use ℓ_∞ norm constraint, which is the most common choice in the computer vision community. In the ℓ_∞ norm case the constraint set $B(\boldsymbol{x})$ is a hyperrectangle (high-dimensional rectangle), and the successful attacks are often (but not always) lying on the corner of this rectangle, i.e., most of the elements of a successful adversarial example $\boldsymbol{x}_i^*$ should be close to $(\boldsymbol{x}_0)_i \pm \epsilon$. If the plain projected gradient descent update (2.5) is used, then some coordinates will require many iterations to reach $\pm\epsilon$, leading to slow convergence. Therefore it is a common practice to use the *steepest descent algorithm* for conducting an attack, as detailed below.

2.3 Steepest descent, FGSM and PGD attack

In this section, we will first consider the ℓ_∞ norm attack (i.e., $B(\boldsymbol{x}) = \{\boldsymbol{x} : \|\boldsymbol{x} - \boldsymbol{x}_0\|_\infty \leq \epsilon\}$). Let us consider the problem of conducting only *one update* to approximately solve (2.1). The idea of steepest descent update is forming a linear approximation of the objective function and finding the best update to maximize this linear approximation. Concretely, we can conduct the first-order Taylor expansion of $g(\boldsymbol{x})$ on the initial solution $\boldsymbol{x}_0$ (assuming that we start from the original natural example):

$$g(\boldsymbol{x}) \approx g(\boldsymbol{x}_0) + \nabla g(\boldsymbol{x}_0)^T(\boldsymbol{x} - \boldsymbol{x}_0).$$

We can then solve the following constraint optimization problem, which is an approximation of (2.1):

$$\underset{\boldsymbol{x}}{\operatorname{argmax}} \nabla g(\boldsymbol{x}_0)^T(\boldsymbol{x} - \boldsymbol{x}_0) \quad \text{s.t.} \quad \boldsymbol{x} \in \|\boldsymbol{x} - \boldsymbol{x}_0\|_\infty \leq \epsilon. \tag{2.6}$$

Note that we drop the $g(\boldsymbol{x}_0)$ term in the Taylor expansion since it is a constant irrelevant to $\boldsymbol{x}$. Now we can easily see that (2.6) has a closed-form solution

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 + \epsilon \operatorname{sign}(\nabla g(\boldsymbol{x}_0)). \tag{2.7}$$

Here $\text{sign}(\cdot) \in \{+1, -1\}$ denotes elementwise sign values. This is the one-step steepest descent update, and surprisingly, it has been found that even this simple step is very effective for constructing adversarial examples. This rule in (2.7) is known as the fast gradient sign method (FGSM), first proposed by Goodfellow et al. (2015).

A straightforward extension of FGSM is to run steepest descent for multiple iterations. In this case, we need to choose a step size α for each iteration, that is, from the current iteration $\boldsymbol{x}_t$ we allow the update in $\{\boldsymbol{x}_{t+1} : \|\boldsymbol{x}_{t+1} - \boldsymbol{x}_t\|_\infty \leq \alpha\}$ and minimize the linear approximation within this region. This will lead to the same subproblem with (2.6) with ϵ being replaced by α, so the optimal update is $\boldsymbol{x}_t + \alpha \text{sign}(\nabla g(\boldsymbol{x}_t))$. Another issue when running multiple iterations is that the variables may go outside the constraint set $B(\boldsymbol{x}_0)$ after few iterations. Therefore we have to further project the update back to the constraint set, leading to the following iterative steepest descent update:

$$\boldsymbol{x}_{t+1} = \Pi(\boldsymbol{x}_t + \alpha \text{sign}(\nabla g(\boldsymbol{x}_t))) \quad \forall t = 0, \ldots, T-1. \tag{2.8}$$

If we run this update for T iterations, then the final solution $\boldsymbol{x}_T$ is supposed to be better than FGSM. This is called the iterative FGSM (I-FGSM) attack by Kurakin et al. (2016), also called the (basic) PGD attack by Madry et al. (2018). This PGD attack is one of the most effective first-order (gradient-based) algorithms for constructing adversarial examples. Furthermore, it is very easy to incorporate in the adversarial training algorithm to train a robust model, which will be discussed in Chapter 11.

One issue for running PGD attack is to choose a suitable step size α. If $\alpha < \frac{\epsilon}{T}$, then the solution $\boldsymbol{x}_T$ will not reach the boundary of $B(\boldsymbol{x}_0)$, and so the attack will not be very effective. In practice, α is usually set as a number slightly larger than $\frac{\epsilon}{T}$. Its value can also vary per attack iteration. Also, one can randomly initialize $\boldsymbol{x}_0$ inside $B(\boldsymbol{x}_0)$ and possibly repeat this process multiple times for finding adversarial examples.

2.4 Transforming to an unconstrained optimization problem

Instead of solving the constrained optimization problem in (2.1), we can actually transform it into an unconstrained optimization problem and apply standard gradient descent or other first-order methods (without projection) to solve the problem. This is done by solving the following "soft-constrained" form, where we remove the constraint and add a penalty in the objective function to enforce small perturbation:

$$\underset{\boldsymbol{x}}{\operatorname{argmin}} -g(\boldsymbol{x}) + \lambda \cdot \|\boldsymbol{x} - \boldsymbol{x}_0\|_p, \tag{2.9}$$

where $\lambda > 0$ is a constant to control the balance between the two terms. A larger λ will lead to a smaller perturbation in ℓ_p norm but lower attack successfulness, whereas a smaller λ will find larger perturbations with higher attack success rate.

Note that we can find adversarial examples within a certain ϵ perturbation constraint by solving a series of unconstrained problems (2.9). Let $\boldsymbol{x}^*(\lambda)$ be the solution of (2.9) with a constant λ; then the perturbation $\|\boldsymbol{x}^*(\lambda) - \boldsymbol{x}_0\|_p$ will be a decreasing function of λ. Therefore we can find the ideal λ that produces $\|\boldsymbol{x}^*(\lambda) - \boldsymbol{x}_0\|_p = \epsilon$ by binary search. This approach has been used in the Carlini and Wagner (C&W) attack (Carlini and Wagner, 2017b), which solves the unconstrained form with a different loss function defined below.

2.5 Another way to define attack objective

Defining the attack successfulness $g(\boldsymbol{x})$ by cross-entropy loss is intuitive, but it has been found that sometimes the nonconvexity introduced by cross-entropy loss makes PGD attack converge slowly and sometimes result in a bad local minimum. Therefore several alternative definitions of $g(\boldsymbol{x})$ have been studied in the literature. Among them, Carlini and Wagner (2017b) showed that defining $g(\boldsymbol{x})$ with a so-called "hinge loss" is most effective. Attacks using this approach are called C&W attack. For untargeted attack, the C&W loss can be written as

$$\text{Untargeted C\&W loss: } g(\boldsymbol{x}) = -\max\{f_{y_0}(\boldsymbol{x}) - \max_{j \neq y_0} f_j(\boldsymbol{x}), -\kappa\}, \tag{2.10}$$

where y_0 is the ground truth class, and $\kappa \geq 0$ is a constant. For $\kappa = 0$, this function will be negative if the model predicts the original class ($f_{y_0}(\boldsymbol{x}) \geq \max_{j \neq y_0} f_j(\boldsymbol{x})$) and zero when the model gives any top-1 prediction other than y_0 (i.e., the "wrong" prediction). Therefore maximizing this function will obtain a successful attack. Setting κ to be some positive value can enforce a margin between $f_{y_0}(\boldsymbol{x})$ and $\max_{j \neq y_0} f_j(\boldsymbol{x})$, but this is usually only used in special circumstances when we want to have stronger adversarial examples that are further away from the decision boundary. With a similar intuition, the loss for C&W targeted attack can be written as

$$\text{Targeted C\&W loss: } g(\boldsymbol{x}) = -\max\{\max_{j \neq t} f_j(\boldsymbol{x}) - f_t(\boldsymbol{x}), -\kappa\}, \tag{2.11}$$

where t is the target class. It has been observed that this loss function leads to better attack, especially for the ℓ_2 perturbation case.

2.6 Attacks with different ℓ_p norms

For computer vision, the commonly used ℓ_p norms for adversarial perturbations include ℓ_∞, ℓ_2, ℓ_1, and ℓ_0 norms.[1] The family of ℓ_p norms carry physical meanings when crafting adversarial examples. The ℓ_∞ norm confines the maximal change in all dimensions, the ℓ_2 norm measures the Euclidean distance between $\boldsymbol{x}$ and $\boldsymbol{x}_0$, the ℓ_1 norm measures the sum of total changes (in absolute value) between $\boldsymbol{x}$ and $\boldsymbol{x}_0$, and the ℓ_0 norm measures the number of modified elements (e.g., modified pixels for images). Depending on the context, we can use mixed norms for adversarial attacks, either in the form of constraints or penalty functions in the optimization formulation. For instance, Chen et al. (2018b) proposed the elastic-net attack to deep neural networks (EAD attack), which uses a mixture of ℓ_1 and ℓ_2 norms to find sparse adversarial examples and is shown to improve the attack performance of C&W attack. Su et al. (2019) proposed an ℓ_0-norm based attack that only modifies few pixels to cause prediction changes. We can also leverage the convolution filters and use group norms to design structured adversarial attacks as proposed by Xu et al. (2019b).

2.7 Universal attack

Up to now, all the attacks we introduced are generating a perturbation $\boldsymbol{\Delta}$ for each image, and the perturbations for each image are different. An interesting question studied in the literature is "Can we use the same adversarial perturbation on all the images to fool the victim model?" Moosavi-Dezfooli et al. (2017) first introduced this concept and call it *universal adversarial perturbation*. Note that the perturbation, although universal to all the images, is designed for each particular machine learning model. Therefore the universal perturbation designed for one model may not generalize

[1] For any $p \geq 1$, the ℓ_p norm of x is defined as $\|\boldsymbol{x}\|_p = (\sum_{i=1}^{d} |\boldsymbol{x}_i|^p)^{1/p}$. It can be shown that $\|\boldsymbol{x}\|_\infty = \max_i |\boldsymbol{x}_i|$. The ℓ_0 norm of $\|\boldsymbol{x}\|_0$ is defined as the number of nonzero entries in $\boldsymbol{x}$. Rigorously speaking, ℓ_0 is a seminorm.

to another model. Finding the universal perturbation can make attack more efficient in real time and will reveal the weakness of a particular model across all the images.

The problem of generating the universal adversarial perturbation for a model F can also be formulated as an optimization problem. Instead of considering only one image, we have to consider a set of images, denoted by D. Both constrained and unconstrained optimization formulations can then be generalized by considering the overall loss on multiple examples. For instance, the constrained form in (2.1) can be generalized to the universal attack setting as

$$\operatorname*{argmax}_{\boldsymbol{\Delta}} \sum_{\boldsymbol{x} \in D} g(\boldsymbol{x} + \boldsymbol{\Delta}) \quad \text{s.t.} \quad \|\boldsymbol{\Delta}\|_p \leq \epsilon. \tag{2.12}$$

The optimizer of this problem will then be a small perturbation that can successfully attack as many images as possible. Similarly, FGSM or PGD attack can be derived to solve this problem. Moreover, we can also generalize the unconstrained form (2.9) to the universal attack setting. Wang et al. (2021a) improve the performance of universal attack over (2.12) by using a min-max optimization solver.

2.8 Adaptive white-box attack

In early days, in the research community, there are active and rolling discussions on how claiming a defense is effective against an attack, because the conclusion will differ significantly based on the action order of the attacker and defender, which is an important specification of the threat model. If the attacker makes the first move by creating adversarial examples, and later the defender comes into discern those generated adversarial examples, such as using techniques based on input filtering or data reconstruction prior to data inference, then many attacks can be alleviated or mitigated. On the other hand, if the defender makes the first move, and the attacker knows the defense in place, then many defenses were shown to be ineffective, or "broken", in the latter setting (Carlini and Wagner, 2017a; Athalye et al., 2018; Carlini et al., 2019a).

The latter setting is known as the *adaptive white-box attack* setting, which essentially assumes that the defense is totally transparent to the attacker (other than some uncontrollable randomness) when crafting adversarial attacks. This setting may be unrealistic for implementing practical attacks, but it offers honest (worst-case) assessment about the true robustness of the protected model against adversarial attacks beyond the advantage from the information asymmetry between the attacker and defender. Otherwise, it is argued that models that are robust in the first setting may give a false sense of security or robustness if they are not robust against adaptive attacks (Carlini and Wagner, 2017a; Athalye et al., 2018; Carlini et al., 2019a). One typical example is the so-called "gradient obfuscation" effect brought by defenses that confuse gradient-based attacks due to information obfuscation (e.g., overly smooth or noisy gradients on the loss landscape in the input space) as discussed by Athalye et al. (2018). However, an attacker that is aware of the defense mechanism can propose advanced methods or adopt nongradient-based attacks to render these defenses ineffective. Carlini et al. (2019a) provide many principles on evaluating adversarial robustness based on adaptive attacks.

2.9 Empirical comparison

Following Chen et al. (2018b); Xu et al. (2019b), Table 2.1 compares white-box attacks of targeted attacks with randomly selected classes and with different ℓ_p norms on three image classification tasks: MNIST, CIFAR-10, and ImageNet. A set of images from the test set with correct classification are selected to perform adversarial attacks. For each image, we report the smallest distortion leading to successful attacks. If all tested ϵ thresholds of a given attack fail, then we mark it as an unsuccessful attack. The attack success rate (ASR) reports the fraction of successful attacks for all tested images. The FGSM/I-FGSM attacks use norm projections to a specified perturbation threshold as described by Kurakin et al. (2016).[2] We provide a fine grid of ϵ thresholds for FGSM/I-FGSM attacks. For norm-penalty-based attacks such as C&W and EAD, we use the binary search strategy for tuning the coefficient λ as proposed by Carlini and Wagner (2017b), which balances attack successfulness and distortion. For C&W attack, we use the ℓ_2 norm penalty. For EAD attack, we use the mixed $\ell_1 + \ell_2$ norm penalty (i.e., the elastic net norm $\|\boldsymbol{x} - \boldsymbol{x}_0\|_2 + \beta \cdot \|\boldsymbol{x} - \boldsymbol{x}_0\|_1$ with $\beta = 10^{-3}$) and select the best distortion according to either the elastic net norm (EN rule) or the ℓ_1 norm (ℓ_1 rule). As seen from Table 2.1, all

[2] Note that the ℓ_1 norm projection in (Kurakin et al., 2016) is only an approximation of the true projection function. The exact projection requires solving another optimization subproblem as described by Xu et al. (2019b). We include the empirical results here just for completeness.

Table 2.1 Comparison of different attacks on MNIST, CIFAR10, and ImageNet (average case). ASR means attack success rate (%) of random targeted attacks. The distortion metrics are averaged over successful examples. EAD, the C&W attack, and I-FGSM-ℓ_∞ attain the least ℓ_1, ℓ_2, and ℓ_∞ distorted adversarial examples, respectively.

	MNIST				**CIFAR-10**				**ImageNet**			
Attack method	**ASR**	ℓ_1	ℓ_2	ℓ_∞	**ASR**	ℓ_1	ℓ_2	ℓ_∞	**ASR**	ℓ_1	ℓ_2	ℓ_∞
C&W (ℓ_2)	**100**	22.46	**1.972**	0.514	**100**	13.62	**0.392**	0.044	**100**	232.2	**0.705**	0.03
FGSM-ℓ_1	39	53.5	4.186	0.782	48.8	51.97	1.48	0.152	1	61	0.187	0.007
FGSM-ℓ_2	34.6	39.15	3.284	0.747	42.8	39.5	1.157	0.136	1	2338	6.823	0.25
FGSM-ℓ_∞	42.5	127.2	6.09	0.296	52.3	127.81	2.373	0.047	3	3655	7.102	0.014
I-FGSM-ℓ_1	**100**	32.94	2.606	0.591	**100**	17.53	0.502	0.055	77	526.4	1.609	0.054
I-FGSM-ℓ_2	**100**	30.32	2.41	0.561	**100**	17.12	0.489	0.054	**100**	774.1	2.358	0.086
I-FGSM-ℓ_∞	**100**	71.39	3.472	**0.227**	**100**	33.3	0.68	**0.018**	**100**	864.2	2.079	**0.01**
EAD (EN rule)	**100**	**17.4**	2.001	0.594	**100**	**8.18**	0.502	0.097	**100**	**69.47**	1.563	0.238
EAD (ℓ_1 rule)	**100**	**14.11**	2.211	0.768	**100**	**6.066**	0.613	0.17	**100**	**40.9**	1.598	0.293

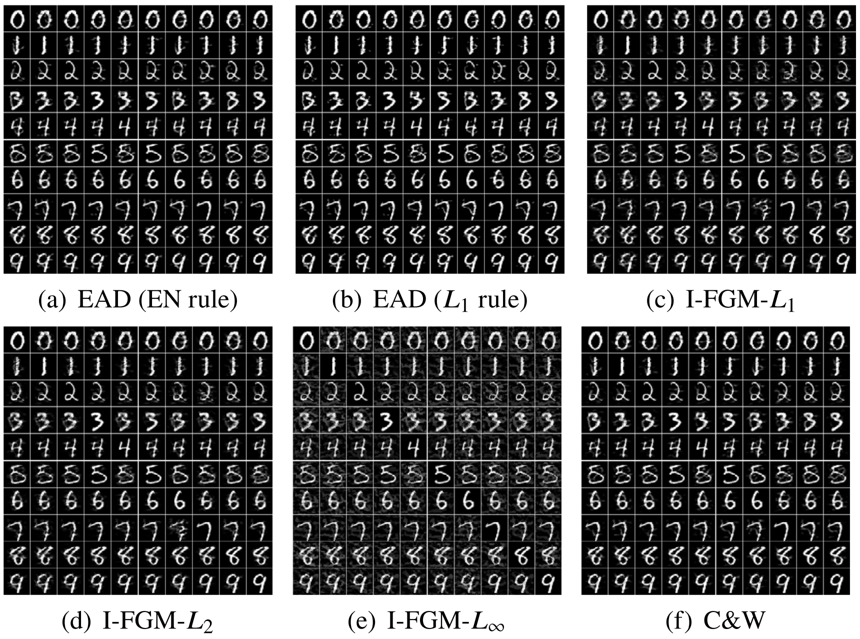
(a) EAD (EN rule) (b) EAD (L_1 rule) (c) I-FGM-L_1 (d) I-FGM-L_2 (e) I-FGM-L_∞ (f) C&W

Figure 2.3 Visual illustration of adversarial examples crafted by different attack methods on MNIST. For each method, the images displayed on the diagonal are the original examples. In each row the off-diagonal images are the corresponding adversarial examples with columns indexing target labels (from left to right: digits 0 to 9).

attacks other than FGSM can reach 100% ASR due to the use of iterative gradient optimization. Attacks tailored with different norm constraints or penalty functions can indeed find corresponding adversarial examples with small perturbations.

For visual illustration, the adversarial examples of selected benign images from the test sets are displayed in Figs. 2.3, 2.4, and 2.5. On CIFAR10 and ImageNet the adversarial examples are visually indistinguishable. On MNIST the I-FGM examples are blurrier than EAD and the C&W attack.

2.10 Extended reading

- Adversarial attacks beyond ℓ_p norms include semantically similar examples such as color space shifting (Hosseini and Poovendran, 2018) and spatial transformations (e.g., rotation) (Xiao et al., 2018; Engstrom et al., 2019b). Tsai et al. (2022) proposed composite adversarial attacks consdering multiple attack types and their attack orders.

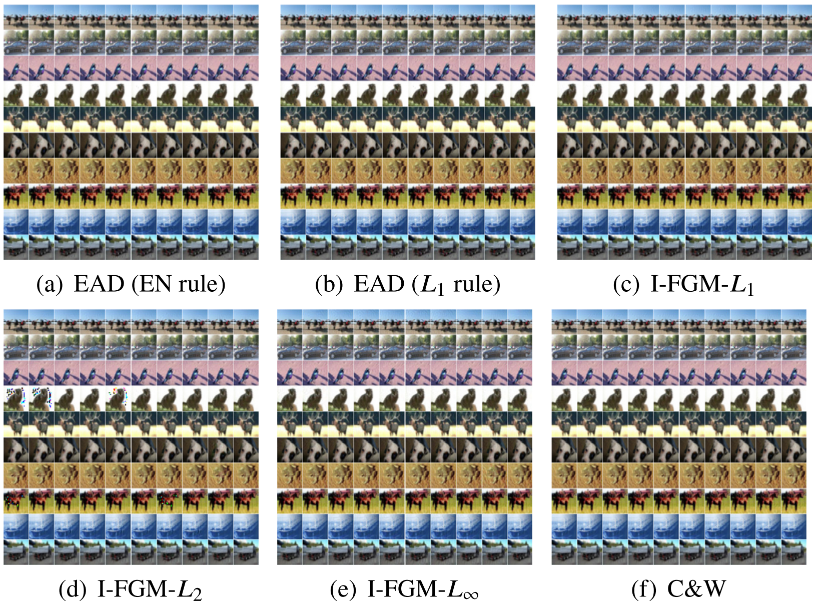

Figure 2.4 Visual illustration of adversarial examples crafted by different attack methods on CIFAR10. For each method, the images displayed on the diagonal are the original examples. In each row the off-diagonal images are the corresponding adversarial examples with columns indexing target labels.

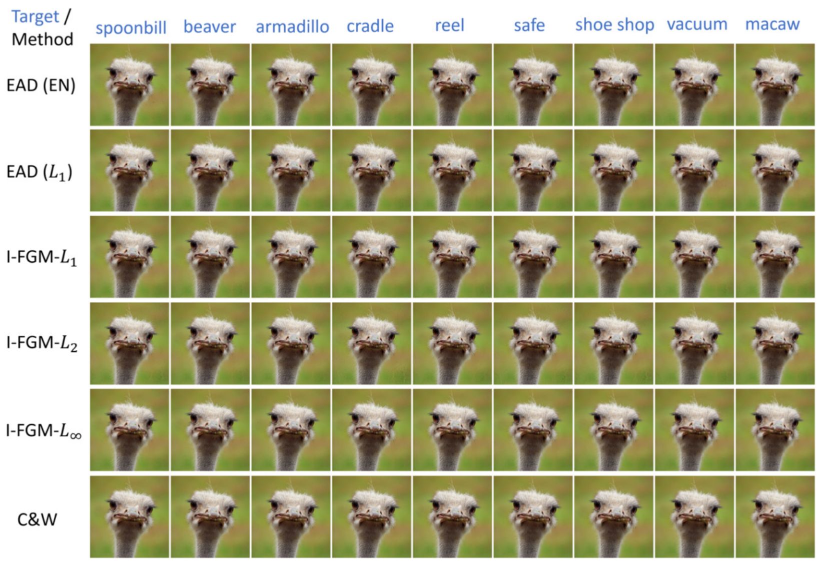

Figure 2.5 Visual illustration of adversarial examples crafted by different attack methods on ImageNet. Each column represents a targeted class to attack, and each row represents an attack method.

- Adversarial attacks considering multiple ℓ_p norms simultaneously (Xu et al., 2019b; Tramer and Boneh, 2019).
- Auto-Attack: An ensemble of parameter-free adversarial attacks (Croce and Hein, 2020).
- Unrestricted adversarial examples (Brown et al., 2018).
- On and off manifold adversarial examples (Stutz et al., 2019).
- Natural adversarial examples (Hendrycks et al., 2021).