Robustness

Lecturer: Kamalika Chaudhuri

April 24, 2020

In the next few lectures, we will talk about machine learning in the presence of adversaries. Adversaries arise in many practical applications of machine learning – essentially in any application where there is a party who has an incentive to make a classifier predict in a particular manner. For example, a malware developed may be incentivized to ensure that his malware is classified as benign, and an adversary might want to play audio clips that sends hidden commands to a home assistant device such an Alexa. As we have seen before, the standard statistical learning framework does not take adversaries into account, which we will address next.

1 Types of Adversaries

Adversaries can occur at multiple points in the machine learning pipeline. Two major kinds of attacks are training time and test time. In a training time attack, the adversary has the power to modify or add a small fraction of training data, and their goal is to either cause the trained classifier to perform poorly, or to misclassify specific test inputs. The former is called a data poisoning attack, and the latter a backdoor attack. An example is a malware developer who might introduce benign looking malware into a repository. We will not go into details of training time adversaries in this class.

A test time adversary has no control over the training process in any way, and has access to the classifier only at test time. Its goal is to modify legitimate test inputs – slightly enough that the modification is undetectable to humans – but in a way that causes classifiers to misclassify. An example is an adversary who puts strategically placed stickers on a stop sign; humans still know its a stop sign but a neural network used by a self-driving car might misclassify it as an yield sign causing an accident. For the rest of the lectures, we will focus on test-time adversaries.

1.1 Test Time Adversaries: The Threat Model

Let us look more closely at the threat model of the test time adversary. The test time adversary has access to a classifier f, and a test input x. Their goal is to find an input $\tilde{x}$ close to x such that $f(x) \neq f(\tilde{x})$. $\tilde{x}$ is called an *adversarial example*.

Literature has looked at multiple threat models on how the adversary can access f. In the *white box* access model, the adversary has complete access to f – it knows the type of classifier (eg, linear, neural networks) and all the parameters. The *black box* access model is more restrictive – the adversary may know some basic facts about f (eg, what kind of classifier it is) and can make a limited number of queries $f(x)$ – but does not know its parameters. Interestingly, for many classifiers, this limited level of access is sufficient for generating adversarial examples.

More formally, given a classifier f, a test input x and an attack radius r, the goal of the test time adversary is to find an adversarial example $\tilde{x}$ such that $f(x) \neq f(\tilde{x})$, but $\|x - \tilde{x}\| \leq r$. Usually the distance $\|x - \tilde{x}\|$ is measured in the L_p-norm, and usually $p = 2$ or ∞. While more complex perceptual distances make sense for image datasets and have been studied, they make attacking and defending significantly more expensive – and hence are not more widely studied.

2 Attacks

There is a growing body of literature on adversarial attacks under a number of different threat models; we next discuss a few simple, canonical ones, and leave the more complicated and powerful attacks for follow-up reading.

2.1 White Box Attacks

2.1.1 Fast Gradient Sign Method

The simplest white box attack is the Fast Gradient Sign Method (FGSM). In FGSM, the adversary is given a classifier f, a text input (x, y) and an adversarial radius r. In addition, they have access to the loss function $L(f, x, y)$ that was minimized to train the model f; usually this is not a great deal of extra knowledge as most neural networks are trained using the same cross-entropy loss. The adversary's goal is to find an adversarial example $\tilde{x}$ within distance r of x.

In the L_∞ version of the attack, the adversary outputs the following example $\tilde{x}$:

$$\tilde{x} = x + r \cdot \text{sign}(\nabla_x L(f, x, y)),$$

where sign is a coordinatewise sign operation.

A couple of observations are in order. First, observe that the gradient of L is taken with respect to x (and not parameters of f as is usually done in empirical risk minimization). So essentially what the attack is doing is taking a gradient ascent step to get to an $\tilde{x}$ so that the loss function $L(f, \tilde{x}, y)$ is high; the hope is that since f is obtained by minimizing L over the training data, if this loss $L(f, \tilde{x}, y)$ is high enough, then $f(\tilde{x})$ will be a different label than y. A second observation is that $\|\tilde{x} - x\|_\infty = r$, so $\tilde{x}$ is exactly at L_∞ distance r. Finally, observe that the attack may not succeed and offers no guarantees – it may be the case that $f(\tilde{x}) = f(x)$, even though there is another adversarial example x' that is much closer.

The biggest advantage of the FGSM is its simplicity and speed; the entire attack requires a single gradient computation! The disadvantage of course is that it is not powerful enough, and fails quite often. The loss function L is a complex non-linear and non-convex function, and a single gradient step is often not enough to find a good solution.

2.1.2 Projected Gradient Descent

This leads us to our next white box attack – the Projected Gradient Descent (PGD). PGD is essentially an iterative version of FGSM. The idea is that instead of taking a single gradient step of length r, we could take multiple smaller gradient steps, in the hope of converging to an adversarial example that is closer. More precisely, this leads to the following iterative algorithm:

$$\begin{aligned} x_0 &= x \\ x_{t+1} &= \text{clip}_{r,x}(x_t + \alpha \cdot \text{sign}(\nabla_x L(f, x_t, y))) \end{aligned}$$

Here α is a step size, and the operation $\text{clip}_{r,x}(u)$ projects a vector u back into an L_∞ ball of radius r around x. An L_2-version of the attack can be similarly designed. Typically, the PGD-k attack uses k iterations of this process with α set to r/k. There are also versions where the attack is started from multiple random initial points, and the closest adversarial example is returned.

PGD turns out to be a much more powerful attack than FGSM even for relatively small k, such as 5 or 10. While there are other, even more powerful attacks, the advantage of PGD is that it is powerful enough, while not being too computation intensive. As we will see later, many defense methods need to generate adversarial examples during the training process; to train these methods within a reasonable amount of time, we therefore need attacks that are powerful yet not too computationally demanding. PGD suffices for this purpose.

2.2 Black Box Attacks

Recall that in the black box access model, the adversary does not directly have access to the parameters of the classifier f. We will touch upon two major kinds of black box attacks – Substitution Attacks, and attacks that are truly Black Box.

2.2.1 Substitution Attacks

In a substitution attack, the adversary does not know the parameters of the classifier f, but can make queries of the form $f(x)$. What sets these attacks apart from a truly black box attack is that the adversary has in addition either a subset of the training data, or another dataset that is drawn from the same distribution.

In the simplest kind of substitution attack, the adversary uses this additional data to directly train a classifier f' and then generates an adversarial example for f' around x. f' may be a different kind of classifier than f – eg, a linear classifier instead of a neural network, or a smaller neural network. The interesting thing however is that in many cases, an adversarial example for f' around x also turns out to be an adversarial example for f. More complex substitution attacks only use a small amount of training data to bootstrap the training of f', and then use some form of active learning to synthesize more unlabeled examples; these unlabeled examples are then labeled by f, and incorporated into training. Iterating this process a few times leads to a f' that can nicely mirror f.

Substitution attacks have an interesting history. In the early days of work on adversarial examples, a number of proposed defenses worked through gradient-masking – which essentially ensured that the gradient $\nabla_x L(f, x, y)$ was close to 0 on legitimate test inputs. This ensured that attacks like FGSM and PGD failed – since the gradients around (x, y) did not give any useful information – but did nothing further to defend against adversarial examples. Substitution attacks worked very well against these methods, and clearly showed their limitations. This may be a bit surprising because in theory black box attacks are weaker than white box attacks; however, in this case, the white box attacks were only using a certain kind of information.

2.2.2 Black Box Attacks

We now discuss a canonical true black box attack, where we assume that the adversary only has query access to f. Here, by query access, we mean that the adversary can supply an input x, and get the label $f(x)$ back. Given an input x, an attack radius r and query access to f, the adversary's goal is to find an adversarial example for f within distance r of x.

Why does some form of FGSM not work here? The reason why these analogues do not apply is that there is no gradient $\nabla_x L(f, x, y)$. Even if we knew that the loss function L is cross-entropy, we would not be able to compute the gradient since $f(x)$ itself is a discontinuous 0/1 function. This is the main difficulty of this problem.

Suppose, for a minute, that instead of just a label, we had query access to a continuous function $g(u)$. Then we could estimate the gradient of $g(u)$, and use it for gradient descent. If u is a scalar, and if δ is small, then the estimate follows from simple calculus:

$$\hat{g}'(u) \approx \frac{g(u+\delta) - g(u)}{\delta}$$

When g is a function of a vector, we use the following estimate which is quite standard in Zero-th Order Optimization:

$$\nabla_u \hat{g}(u) \approx \frac{g(u+\beta v) - g(u)}{\beta} \cdot v, \quad v \sim \mathcal{N}(0, I_d)$$

It can be shown that for small enough β the expected value of this estimate is the gradient $\nabla_u \hat{g}(u)$, and that its variance is bounded. Thus, all it remains to do is to use f to find a suitable continuous function g that we can minimize to get an adversarial example around x

The g that we choose is as follows. Given a vector $u \in \mathbb{R}^d$,

$$g(u) = \min_{\lambda > 0} \lambda, \text{s. t.} \quad f\left(x + \lambda \frac{u}{\|u\|}\right) \neq y$$

Thus $g(u)$ measures the minimum distance we have to go from x along u so as to cause the label to change from y. Observe that if the decision boundary induced by f is nice and smooth, then this g will be continuous as well, and hence we can use the trick we just discussed to compute the gradient of g. Moreover, minimizing g over u will give an adversarial example around x – in fact, if g could be minimized exactly, then we could find x's closest adversarial example.

It remains to calculate g. This can be done by standard algorithms – a grid search followed by a binary search. We fix an α and look at

$$x + \alpha u, x + 2\alpha u, \ldots, x + k\alpha u$$

until we find an i such that $f\left(x + i\frac{u}{\|u\|}\right) = y$ but $f\left(x + (i+1)\frac{u}{\|u\|}\right) \neq y$. Then we can do binary search in the interval $\left[x + i\frac{u}{\|u\|}, x + (i+1)\frac{u}{\|u\|}\right]$ to find $g(u)$.

3 Defenses

Early defenses against adversarial examples were based on gradient-masking – where the idea is to ensure that the gradient of the loss function $\nabla_x L(f, x, y)$ is close to zero for legitimate inputs (x, y). These defenses were successful against gradient-based attacks such as FGSM and milder forms of PGD; however, they completely failed against more complicated white box attacks as well as substitution attacks.

Before we get to specific defenses, let us take a quick look at a couple of formal definitions, and what might be expected of any defense.

Definition 1 (Robustness Radius)**.** *The robustness radius of a classifier f at x is the distance to the closest $\tilde{x}$ such that $f(x) \neq f(\tilde{x})$; formally:*

$$\rho(f, x) = \inf\{d(x, \tilde{x}) : f(x) \neq f(\tilde{x})\}$$

In other words, the robustness radius is the distance to the closest adversarial example.

While building a defense we would like to build an f that has a certain robustness radius – say at least r – over a large region. But which region? Observe that an f which is robust everywhere can only be a constant classifier. Thus, we typically aim to build an f that is robust on the underlying data distribution, and as accurate as possible. Thus the goal of robust classification is to maximize, instead of accuracy, what we call *astuteness*, defined as follows.

Definition 2 (Astuteness)**.** *The astuteness of a classifier f at radius r with respect to a data distribution D is the fraction of points according to D on which f is robust with radius r, as well as accurate. Formally,*

$$A_D(f) = \Pr_{(X,Y)\sim D}[f(X) = Y, f(X) = f(X'), \forall X' \in B(X, r)]$$

Observe that unlike accuracy, for many classifiers, astuteness cannot be directly measured or even estimated. Even if we used test samples in lieu of D, it is computationally challenging to find the closest adversarial example to a particular x, and hence to determine the robustness radius. In practice, we use robust accuracy – which measures how accurate a classifier is to adversarial examples – as a proxy. In practice, we also attempt to maximize astuteness or some proxy of it on the training data – since D is usually unknown.

We next address a few specific defenses that attempt to achieve high astuteness.

3.1 Adversarial Training

The first defense that led to reasonable accuracy in the presence of a wide variety of attacks was Adversarial Training (AT). The main idea in adversarial training is to find a classifier f^* that minimizes, instead of the expected loss as we do in the statistical learning framework, the following objective:

$$f^* = \text{argmin}_f \mathbb{E}_{(x,y)\sim D}[\sup_{x':\|x-x'\|\leq r} L(f, x', y)]$$

Here the distance $\|x - x'\|$ is the metric in which we measure the adversarial radius. Given a training data set $(x_1, y_1), \ldots, (x_n, y_n)$, the empirical version of the objective is:

$$f^* = \text{argmin}_f \frac{1}{n} \sum_{i=1}^{n} \max_{x_i':\|x_i-x_i'\|\leq r} L(f, x_i', y_i) \tag{1}$$

How do we implement adversarial training? The main difference between the objective in (1) and natural training is the inner maximization. A close look at the equation reveals that for a particular i, x_i' is just an adversarial example for f around (x_i, y_i) within a radius r. So we can implement adversarial training through a modified stochastic gradient update – given training example (x_i, y_i), find an adversarial example x_i' in a ball of radius r around x_i, and then do the usual update on (x_i', y_i). Observe that this does increase training time over natural training – because for each stochastic gradient update, we need to find an adversarial example.

The main advantage of adversarial training is that it gives a robust classifier that is resistant to a variety of attacks – although it does not guarantee robustness. Defenses prior to adversarial training did not do this. A disadvantage is that the type of adversarial attack used while training matters; if a weaker attack is used during training, then success against a more powerful attack is more limited. Coupled with the computational challenge, this does present a problem. We cannot use very powerful attacks during training since they are expensive; yet if we train on weaker attacks, then the defense is weak. A second disadvantage of adversarial training is that the clean accuracy – that is, accuracy on legitimate test inputs – sometimes decreases as a result of adversarial training. One way to address this is to train on a linear combination of the natural and the adversarial loss, but it does not completely solve the problem. We will get back to this in more detail later on.

3.2 Local Lipschitzness and Large Margin

We now look at an alternative take on adversarial robustness as a large margin problem. To better understand this, let us start with a simple case – linear classifiers and L_2-robustness.

Consider the simplest case of linear classification – binary linear classification when the data is linearly separable. In this case, the linear classifier that maximizes the L_2-margin on the training data is the classical Hard Margin Support Vector Machines (SVM) classifier. Recall that for binary linear classifiers, the decision boundary is a hyperplane; if the normal vector to the hyperplane is denoted by w, then the SVM classifier is obtained by solving the following optimization problem:

$$\begin{aligned} w^* \quad = \quad & \min_w \frac{1}{2}\|w\|^2 \\ & \text{subject to:} \\ & y_i w^\top x_i \geq 1, \quad \forall i \end{aligned}$$

What can be shown is that the solution w^* to the SVM maximizes the minimum robustness radius on the training data, while ensuring that all training points are correctly classified. In fact, more specifically, what we will show is this.

Lemma 1. *Suppose there exists a linear classifier that accurately classifies the training data with an L_2-robustness radius r. Then, any feasible solution to* (2) *accurately classifies the training data, and has an L_2-robustness radius $\geq r$.*

$$\begin{aligned} \|w\|^2 \quad & \leq \quad 1/r^2 \\ y_i w^\top x_i \quad & \geq \quad 1, \forall i \end{aligned} \tag{2}$$

Proof. Let w be a feasible solution to (2) and without loss of generality, let $(x_i, 1)$ be a training data point. Then, by the second constraint, we have that $w^\top x_i \geq 1$. We will show that any x' in a L_2-ball of radius r around x continues to be labeled as 1 by w.

Let x' be such that $\|x_i - x'\| \leq r$. Then,

$$w^\top x' = w^\top x_i - w^\top (x_i - x') \geq w^\top x_i - \|w\| \cdot \|x_i - x'\| \geq 1 - \frac{1}{r} \cdot r \geq 0,$$

where the second step follows from Cauchy Schwartz, and the third step from the fact that $\|w\| \leq 1/r$ and $\|x_i - x'\| \leq r$. Thus we see that x' is also labeled as 1 by w, thus establishing robustness. The case for the negatively labeled examples is analogous. □

Thus, Lemma 1 shows that for linear classifiers, a solution with robustness radius r, if one exists, can be obtained by solving the SVM. One can also show a soft-margin version of the lemma for the case when there is no such perfectly accurate solution; in this case, it will not be possible to classify the training data with robustness radius r, and there will be a robustness-accuracy tradeoff. We leave it as an exercise.

What about more complex non-linear classifiers? For simplicity, consider binary classification. Suppose we have a non-linear classifier f that is obtained from taking the sign of a function g, and we are trying to build an analogoue of (2). The second constraint can be written as:

$$y_i g(x_i) \geq 1, \quad \forall i \tag{3}$$

and what it ensures is that the training data has large absolute values of g – that is, $g(x)$ for the training data x is bounded away from zero. But that does not automatically mean that training data lies far away from the decision boundary in space. Additionally, emulating the first constraint for any general g is not easy – unlike the linear case, bounding the L_2-norm of g does not put a limit on the spatial margin.

However, what is true is that if g is locally Lipschitz in the neighborhood of the training data, then ensuring (3) will also ensure spatial robustness. Thus an analogoue of the first constraint in (2) is local Lipschitzness. We begin by defining local Lipschitzness, and then we show this fact in Lemma 2.

Definition 3. *A function g is said to be L locally Lipschitz in a ball of radius r around x if for all x' such that $\|x - x'\| \leq r$, we have:*

$$|g(x) - g(x')| \leq L\|x - x'\|$$

Observe that Definition 3 can be applied with any metric of choice. With this definition in place, we can now show the following lemma.

Lemma 2. *If a function f is $\frac{1}{r}$-Locally Lipschitz in a ball of radius r around x, and if $yf(x) \geq 1$, then the classifier $g = sign(f)$ is astute at (x, y) with radius r, where astuteness is measured in the same metric that we use to define local Lipschitzness.*

Proof. Without loss of generality, let $y = 1$. If x' is any point in a ball of radius r around x, then from the local Lipschitzness condition, we can show that $f(x') \geq f(x) - 1 \geq 0$. Thus $g(x') = 1 = g(x)$. This implies that g assigns a label 1 to the entirety of $B(x, r)$ from which the lemma follows. □

Thus, the analogue of the first constraint in (2) is that f is locally Lipschitz in a ball of radius r around each training point x_i. How do we impose this constraint in practice? While training a neural network, it is challenging to impose hard constraints, and instead we must use soft ones. A method that does so is called TRADES, and it minimizes the following objective over continuous functions f:

$$\min_f \mathbb{E}_{(x,y)\sim D}[L(f, x, y) + \beta \cdot \max_{x':\|x'-x\|\leq r} \ell(f(x), f(x'))], \tag{4}$$

where ℓ is a continuous loss function that attempts to enforce $f(x)$ and $f(x')$ to be close. The empirical version of the objective is:

$$\min_f \frac{1}{n} \sum_{i=1}^{n} L(f, x_i, y_i) + \beta \cdot \max_{x_i':\|x_i-x_i'|\leq r} \ell(f(x_i), f(x_i')) \tag{5}$$

To optimize this objective, observe that the x'_i in the inner minimization is an adversarial example – and hence any technique that finds adversarial examples works.

Observe also that in spite of the apparent similarity, there is a major difference between TRADES and adversarial training. Adversarial training tries to enforce that $\text{sign}(f(x_i)) = y_i$ – that is, the adversarial examples have the same labels as the original examples they are close to. TRADES instead tries to enforce that $f(x_i) \approx f(x'_i)$ – that is, the function f itself is close on x_i and x'_i. The objectives coincide in the linear case when the function f is highly constrained, but for more complex and less constrained non-linear functions, they are different – especially when the inner maximum cannot be solved exactly. For example, it may be possible to ensure that $f(x_i) = y_i$ through a very wiggly function that crosses the decision boundary at some other point between x_i and x'_i.

The advantage of TRADES over adversarial training is that because it directly enforces local Lipschitzness, it provides better accuracy-robustness tradeoffs. The disadvantage continues to be the computation cost – since each step of training still involves finding an adversarial example, the computational complexity remains high.

4 Large Sample Limits under Adversarial Robustness

Suppose we had all the resources we wanted – data, computation, time. If we cared about robustness up to a certain radius r, what is the ideal classifier that we should aim for? In other words, what is the robustness analogoue to the Bayes Optimal classifier? Understanding this is useful for designing robust classification algorithms that behave in the rightmanner as we get more and more data.

Recall that if we had k class labels, then the Bayes optimal classifier f^* is defined as follows:

$$f^*(x) = \text{argmin}_i p(y = i|x)$$

Is the Bayes Optimal still our goal? Unfortunately, the Bayes Optimal itself can be highly non-robust – imagine a data distribution with two labels where $p(y = 1|x)$ fluctuates rapidly between 0 and 1. Here the Bayes Optimal may have accuracy 1 but will be highly non-robust even for relatively small radii r.

4.1 The r-Optimal Classifier

We therefore need a different limit. We begin with some notation. Given an input $x \in \mathcal{X}$, and a set $S \subseteq \mathcal{X}$, we use the notation $d(x, S)$ to denote the distance between x and the closest point in S. Given two subsets S and S' of the instance space, we use $d(S, S')$ to denote the distance between the closest points in them.

Suppose we cared about robustness within radius r; the corresponding limit is called the r-optimal classifier, and is described as follows. Suppose we have k classes, and suppose $S^*_1, \ldots, S^*_k$ be subsets of the support of the data distribution D; we obtain them by solving the following optimization problem:

$$(S^*_1, \ldots, S^*_k) = \text{argmax}_{S_1,\ldots,S_k} \quad \sum_{i=1}^{k} \int_{x \in S_i} p(y = i|x) d\mu(x) \tag{6}$$

$$\text{subject to:} \qquad d(S_i, S_j) \geq 2r \quad \forall i \neq j, \tag{7}$$

The r-optimal classifier f_r^* is defined as:

$$f_r^*(x) = \text{argmin}_i d(x, S_i) \tag{8}$$

In other words, we output label i for an input x if S_i is the closest set to x (out of $S_1, \ldots, S_k$). We begin by showing that f_r^* indeed has robustness radius at least r in $S_1^* \cup \ldots \cup S_k^*$.

Lemma 3. *Let x be any point in $S_1^* \cup \ldots \cup S_k^*$. Then f_r^* has robustness radius at least r at x.*

Proof. Let $x \in S_i$ and let x' be any point in the open ball $B_o(x, r)$. For any $j \neq i$, we have:

$$d(x', S_j) \geq d(S_i, S_j) - d(x, S_i) > 2r - r > r,$$

where the first step follows from the Triangle Inequality, and the second step from constraint (7) and that $x' \in B_o(x, r)$. Thus, $d(x', S_i) < d(x', S_j)$, and hence $f_r^*(x') = i$ as well. The lemma follows. □

We next show that f_r^* maximizes the astuteness on D.

Lemma 4. *Let g be any classifier on D. Then the astuteness of f_r^* on D is greater than or equal to the astuteness of g.*

Proof. Let T_i be subsets of the support of D on which g predicts i and has robustness radius r. We claim that for any $i \neq j$, $d(T_i, T_j) \geq 2r$. Suppose this is not the case – that is, there exists some i and j such that $d(T_i, T_j) < 2r$. Let $x_i \in T_i$ and $x_j \in T_j$ be the two points in T_i and T_j respectively that minimize $d(x_i, x_j)$. Since $d(x_i, x_j) < 2r$, there exists some $x \in B_o(x_i, r) \cap B_o(x_j, r)$. Either $g(x) = i$, $g(x) = j$ or $g(x)$ could be a third label altogether. In either case, g cannot have robustness radius r – a contradiction.

Now, since $d(T_i, T_j) \geq 2r$, $T_1, \ldots, T_k$ are feasible solutions to the optimization problem (6). Additionally, the accuracy obtained by predicting i on T_i is $\int_{x \in T_i} p(y = i|x) d\mu(x)$; putting all labels together this means that the astuteness of g is $\sum_{i=1}^k \int_{x \in T_i} p(y = i|x) d\mu(x)$. Since the solution $S_1^*, \ldots, S_k^*$ maximizes this astuteness, the astuteness of f_r^* is at least that of g. The lemma follows. □

It is easy to see that the r-optimal is different from the Bayes optimal when different classes overlap; but it may even be different when data is well-separated. Consider for example a data distribution with two clusters C^+ and C^- where $d(C^+, C^-) \geq 2r$, and any point in C^+ (respectively C^-) is labeled + (respectively −) with probability 1. In this case, the Bayes optimal outputs + on C^+, − on C^- but is undefined anywhere else; in fact there are an infinite number of Bayes optimal classifiers that differ by what they predict outside $C^+ \cup C^-$. The r-optimal's decision boundary in contrast is at least a distance r from any point in $C^+ \cup C^-$. Thus, for this distribution, the r-optimal is not just any Bayes optimal – but a Bayes optimal with some specific properties.

Just as there are non-parametric classifiers that converge to the Bayes optimal in the large sample limit, there are non-parametric classifiers that converge to the r-optimal in the large sample limit. When the data distribution is well-separated in the sense that inputs from differently labeled classes are at least $\geq 2r$ apart, the nearest neighbor and the kernel classifiers will converge to the r-optimal in the large sample limit. However, there are other classifiers – such as histograms – that will converge to the Bayes optimal in the large sample limit but not the r-optimal even when data is separated. These classifiers thus are inherently non-robust.

When different classes overlap, things are more complicated. In this case, one can use a preprocessing method such as Adversarial Pruning on the training data to make it well-separated prior to training a non-parametric classifier. It can be shown that after such preprocessing, nearest neighbors and kernel classifiers will converge to the r-optimal in the large sample limit.

5 Bibliographic Notes

Adversarial robustness to test time attacks has been studied in multiple communities for a while. The earliest work that I know of is [LM05]. FGSM is due to [GSS14], and PGD is due to [KGB16]. The earliest substitution attack that I know of is due to [BCM+13], and a more efficient one is by [PMG+17]. The hard margin black box attack described is by [CLC+18].

Adversarial training was introduced by [MMS+17] and Trades by [ZYJ+19]. The equivalence of robustness and margin in support vector machines was shown by [XCM09], and the interpretation of Trades as a locally Lipschitz classifier is taken from [YRZ+20]. Finally, the r-optimal classifier was derived by [YRWC19], and guaranteed convergence of non-parametric methods to the r-optimal was shown by [BC20].

References

[BC20] Robi Bhattacharjee and Kamalika Chaudhuri. When are non-parametric methods robust? *arXiv preprint arXiv:2003.06121*, 2020.

[BCM+13] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer, 2013.

[CLC+18] Minhao Cheng, Thong Le, Pin-Yu Chen, Jinfeng Yi, Huan Zhang, and Cho-Jui Hsieh. Query-efficient hard-label black-box attack: An optimization-based approach. *arXiv preprint arXiv:1807.04457*, 2018.

[GSS14] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[KGB16] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[LM05] Daniel Lowd and Christopher Meek. Adversarial learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 641–647, 2005.

[MMS+17] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[PMG+17] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In

Proceedings of the 2017 ACM on Asia conference on computer and communications security, pages 506–519, 2017.

[XCM09] Huan Xu, Constantine Caramanis, and Shie Mannor. Robustness and regularization of support vector machines. *Journal of machine learning research*, 10(Jul):1485–1510, 2009.

[YRWC19] Yao-Yuan Yang, Cyrus Rashtchian, Yizhen Wang, and Kamalika Chaudhuri. Adversarial examples for non-parametric methods: Attacks, defenses and large sample limits. *arXiv preprint arXiv:1906.03310*, 2019.

[YRZ+20] Yao-Yuan Yang, Cyrus Rashtchian, Hongyang Zhang, Ruslan Salakhutdinov, and Kamalika Chaudhuri. Adversarial robustness through local lipschitzness. *arXiv preprint arXiv:2003.02460*, 2020.

[ZYJ+19] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P Xing, Laurent El Ghaoui, and Michael I Jordan. Theoretically principled trade-off between robustness and accuracy. *arXiv preprint arXiv:1901.08573*, 2019.