Lecture 10: Empirical defenses for adversarial examples

October 28, 2019

In this lecture, we'll go over a number of empirically proposed defenses for adversarial attacks. We'll see how some of them have been broken, but some appear to work quite well, at least in practice. Notably, none of these defenses have provable guarantees. This distinguishes these approaches from later defenses that have provable guarantees, which are often known as *certified* defenses.

1 Approaches that don't seem to work

Here we'll first go over a number of general, fairly natural, strategies, for defending against adversarial examples, that as a rule of thumb, do not appear to work very well. In our presentation we will mostly follow the framework of [1].

Shattering gradients One approach that was tried fairly extensively early on in the study of adversarial examples was to somehow "obfuscate" the gradient of the model. The idea was that if the gradients of the model were made to be very non-linear, then PGD-based attacks would be difficult to carry out. There were a number of attempts at doing so. For instance, some took a pretrained model $f(\cdot)$, constructed a non-smooth preprocessing step $g(\cdot)$ so that $g(x)$, then defined $F = f \circ g$ to be the robust classifier. Then naively, if g is neither smooth nor differentiable, one cannot apply directly PGD attacks to construct adversarial examples for the full classifier F.

However, [1] demonstrates a general way to to attack these models, which they call Backwards Pass Differentiable Approximation (BPDA). The idea is relatively straightforward. Consider the special case where g is a non-smooth, non-differentiable approximation to the identity. In this case the idea is typically that g acts as a sort of denoiser, but which is hard to backpropagate through. But if $g(x) \approx x$, then there is a simple "hack" to approximate the derivative of F at a point $\widehat{x}$:

$$\nabla_x f(g(x)) \mid_{x=\widehat{x}} \approx \nabla_x f(x) \mid_{x=g(\widehat{x})} \ . \tag{1}$$

That is, we simply take the gradient of f at $g(\widehat{x})$. This can be easily implemented by changing the back-propagation step by replacing $g(x)$ with the identity map during the backwards pass. More generally, for any g, we can do something similar by using a smooth differentiable function during the backwards pass. The paper [1] found that by doing so, they were able to decrease the robust accuracy of defenses based on such shattered gradients to essentially 0%.

Stochastic gradients Another approach was to make the gradients of the network random, by e.g. dropping random pixels, or by randomly cropping images. Even JPEG compression was considered as a defense. The idea is that if we make the gradient random, then there is no deterministic direction for PGD to make progress in. However, we've already seen the way to circumvent these defenses: the EOT attacks (or RP2) attacks are already designed to account for random transformations in the data. Unsurprisingly, [1] demonstrates that EOT-based attacks are able to circumvent these defenses.

Detecting out-of-distribution shift Another approach is to try to devise some method of checking whether or not a sample is in distribution. The hope is that since adversarial examples are clearly out-of-distribution, these methods can realize, given an adversarial example, that the sample is tampered with. While this may not directly allow for better robust loss, one could hope that it could be useful as a first step to getting something robust. Many different approaches have been proposed to do this: some have been based on GANs, some are based on statistics of the layers of the neural network, or on methods derived from interpretable ML. While these approaches are indeed often able to detect adversarial examples derived from vanilla PGD, they have pretty universally failed against targeted attacks. One successful attack paradigm is the *high confidence adversarial examples* of [2], where they change the objective of the PGD attack to force that the resulting adversarial example is classified incorrectly with high probability. The resulting adversarial example then appears to circumvent these detection methods.

I have a defense, how do I make people believe me? Of course, we do not wish to make the claim that no method of the types considered above will ever succeed to be truly robust. It is possible that we just aren't there yet algorithmically. For instance, if we had a perfect out-of-distribution detector for natural images, then certainly adversarial examples should detected. However, the community is (understandably) suspicious of new techniques that try to use these methods to defend against adversarial examples. More generally, since adversarial examples have turned out to be slippery little buggers, it is advised to take a more security-driven approach to empirical defenses. To gain traction as an empirical defense, it is a good rule of thumb to follow these guidelines:

1. **Make concrete, reproducible claims.** Fully specify a formal threat model (i.e. what the adversary is given access to, what the defender is trying to do). Then, make concrete claims about what your defense is able to do in this threat model. Make sure this claim is reproducible: it is highly recommended that you provide source code for your defense, so that interested parties can try to attack it.

2. **Evaluate against targeted attacks.** So many defenses work against vanilla PGD, if the adversarial attack has not been designed to defeat the specific defense. Of course, this does not mean that the defense is secure. To ensure this, one should conduct comprehensive evaluations against attacks that are truly targeted against your own defense, until you are really satisfied that nothing can break it. This will save you a lot of headache in the future: it is substantially better to break your own defense privately than to have someone else break it publicly.

If you don't follow these guidelines and publish your defense, then Nicholas Carlini will come to your office and beat you up.[1]

2 Adversarial training

In a slight oversimplification, the one class of defenses that appears to have stood the test-of-time against attacks so far are the defenses based on *adversarial training*. The idea is, at a high level, quite straightforward. We will try to run SGD on the robust loss of the classifier directly. To do so, we will need to try to take gradients of the robust loss.

Recall that the robust loss of a soft classifier f with respect to a loss ℓ is defined to be

$$R_{\text{rob}}(f) = \mathop{\mathbb{E}}_{(X,y)\sim\mathcal{D}} \left[\sup_{X' \in \mathcal{P}(X)} \ell(f(X'), y) \right] .$$

Thus if $f = f_\theta$ is parametrized by weights θ, under mild assumptions, the gradient of the robust loss of f can be written as

$$\nabla_\theta R_{\text{rob}}(f_\theta) = \nabla_\theta \left(\mathop{\mathbb{E}}_{(X,y)\sim\mathcal{D}} \left[\sup_{X' \in \mathcal{P}(X)} \ell(f(X'), y) \right] \right) = \mathop{\mathbb{E}}_{(X,y)\sim\mathcal{D}} \left[\nabla_\theta \left(\sup_{X' \in \mathcal{P}(X)} \ell(f(X'), y) \right) \right] ,$$

[1]this is probably false

where we've just pulled the derivative inside the expectation. In the special case of $\mathcal{P} = \mathcal{P}_{p,\varepsilon}$, this has the additional nice form as

$$\mathop{\mathbb{E}}_{(X,y)\sim\mathcal{D}}\left[\nabla_\theta\left(\sup_{\delta\in\mathcal{P}_{p,\varepsilon}(0)}\ell(f(X+\delta),y)\right)\right] .$$

The expectation over the outside shouldn't really bother us: that's handled for us since SGD only requires an unbiased estimate of the gradient. The main question is how we can possibly hope to take the gradient of the inner expression, at a given (X, y). We don't know how to do so exactly (as otherwise we could actually take stochastic gradient steps for the robust loss), however, we can apply a heuristic based on a classical theorem in optimization known as *Danskin's theorem*:

Theorem 2.1 (Danskin's theorem). *Suppose that for every θ, the map $\ell(f_\theta(X'), y)$ is differentiable as a function of θ for every fixed $X' \in \mathcal{P}_{p,\varepsilon}(X)$, and $\nabla_\theta \ell(f_\theta(X'), y)$ is continuous as a function on (θ, X'). Then, the function*

$$L(\theta) = \sup_{X'\in\mathcal{P}_{p,\varepsilon}(0)(X}\ell(f(X'),y)$$

is locally Lipschitz continuous, directionally differentiable, and for every direction v, its directional derivatives satisfy

$$\frac{\partial L}{\partial v}(\theta) = \sup_{X'\in\Delta(\theta)} v^\top \nabla_\theta \ell(f_\theta(X'),y) ,$$

where $\Delta(\theta)$ is the set of X which achieve the supremum at θ. In particular, if $\Delta(\theta) = \{X^*(\theta)\}$ is a singleton, then L is differentiable at θ, and

$$\nabla_\theta L(\theta) = \nabla_\theta \ell(f_\theta(X^*(\theta)),y) .$$

Roughly speaking, this says that, under suitable conditions, to evaluate the gradient of the supremum of a class of functions, one should simply evaluate the gradient of the function in the class that actually obtains the maximum (if one exists). It's not hard to check that for $\mathcal{P} = \mathcal{P}_{p,\varepsilon}$ for $p > 0$, for f being a neural network with smooth activations, and a smooth loss function such as cross-entropy, the conditions of this theorem are satisfied. So that's great: that says that if we want to a stochastic gradient step for the robust loss, we just need to find a perturbation that maximizes the loss, and evaluate the gradient of the model at this point.

Of course, the problem is that we don't know how to do this maximization problem exactly. Doing so would entail finding the "optimal" adversarial perturbation within the allowed perturbation set, which would be a very non-concave optimization problem. However, this does suggest a natural heuristic: since we do have pretty good attacks, why not simply use them as a proxy for this gradient? That is, pretend that PGD finds the truly optimal adversarial perturbation, and evaluate the gradient at the point that PGD finds. Thus, an iteration of the overall training algorithm would proceed as follows. Given a current model θ:

1. Sample $(X, y) \sim D$.
2. Using whatever attack you want (e.g. PGD), find a point $X' \in \mathcal{P}(X)$ with large $\ell(f_\theta(X'), y)$.
3. Update θ as

   $$\theta \leftarrow \theta - \eta\nabla_\theta(f_\theta(X'),y) ,$$

   where η is some step-size parameter.

Iteratively doing this process is known as *adversarial training* [3]. In practice, step (2) is typically done with as strong of an attack as you can afford. Also, as is typical in deep learning, rather than only doing one pass through your data, one does many epochs through the data, repeating this process. These updates are also typically applied in a mini-batch manner: given a mini-batch of samples $(X_1, y_1), \ldots, (X_m, y_m)$, we find an adversarial example for each (X_i, y_i), then apply the mini-batch gradient by averaging the gradients evaluated at each adversarial example.

How well does this perform? Getting the state-of-the-art numbers for empirical robustness is a rather murky task, and the numbers are always changing. As of writing, adversarial training is able to get non-trivial accuracy against all known attacks, however, these numbers still lag behind non-robust accuracy quite substantially. For instance, on CIFAR-10, ResNet50 models are able to achieve non-robust accuracy $94-95\%$, but the state-of-the-art robust models are able to achieve robust accuracy around $50-55\%$ against normalized ℓ_∞ perturbations of size $\varepsilon = 8/255$, after normalizing the pixels to be within the range $[0, 1]$, see e.g. [4].

2.1 Runtime of adversarial training

One of the main bottlenecks of adversarial training is that it is quite slow. In particular, step (2) of the loop above usually requires a number of iterations of PGD, say 10 or so, to obtain good adversarial examples. But this means that each mini-batch now takes 10 times as many forward-backwards steps as it did previously to process, and since the necessary number of iterations and epochs is generally unchanged from clean training, this translates to a roughly $10\times$ slowdown of training. For instance, adversarial training on CIFAR-10 often takes on the order of days, and adversarial training on ImageNet often takes on the order of weeks, unless you have access to a large amount of compute. A couple of groups [5, 6] have suggested ways of decreasing the runtime of adversarial training, however these remain relatively unvetted.

2.2 Tradeoffs between robustness and accuracy

One downside of adversarial training is that the resulting classifiers typically get lower clean accuracy than non-robust models. That is, when you evaluate the adversarially trained model on clean (non-perturbed) data points, the accuracy can drop fairly substantially, on the order of 10% or so. It has been suggested that this might be inherent [7, 8].

For instance, [7] considers the following toy model: consider the following classification task. We have (X, y) pairs generated as follows: first, we sample $y \sim \{\pm 1\}$ uniformly at random, then we set $X \in \mathbb{R}^d$ so that $X_1 = y$ w.p. p and $-y$ otherwise, and $X_j = \mathcal{N}(\eta y, 1)$ for $j = 2, \ldots, d$. When d is sufficiently large, there is a very good non-robust classifier: namely, $\widehat{y} = \operatorname{sgn}\left(\sum_{j=2}^d X_i\right)$. Then, for any $\delta > 0$, so long as $d \gtrsim \frac{\log 1/\delta}{\eta^2}$, it is not hard to show that this classifier is correct with probability $1 - \delta$. However, notice that all the features $X_2, \ldots, X_d$ can be completely fooled by an ℓ_∞ perturbation to X of size η. Thus, the best robust estimator for y is simply the estimator $\widehat{y} = X_1$. However, this can never do better than error rate $1 - p$. At a high level, this phenomena occurs because for this problem, there are a number of "non-robust features", namely $X_2, \ldots, X_d$, which are fundamentally very useful for non-robust classification, that we simply cannot use if we wish to have a robust classifier. Thus, for this problem, the accuracy of the problem inherently drops if we insist that the classifier be robust. More generally, [8] demonstrates that this phenomena might occur more generally as well.

2.3 Pretraining and semi-supervision

Recently, several groups have demonstrated that relatively naive ways of incorporating more data into the pipeline can improve robust accuracy. One method is known as *pretraining*: we take a large dataset, such as ImageNet, and pretrain a robust model on ImageNet. Then, we take the learned representation (and after appropriately resizing the last layer), fine tune the representation by adversarially training for fewer epochs on CIFAR-10. This essentially corresponds to using ImageNet to get a good initialization for the representation. The paper [9] reports that using this technique they are able to increase the robust accuracy on CIFAR-10 by up to 10%. Notably, pretraining is known to not help with clean accuracy, but here appears to help quite significantly for robust accuracy. We'll discuss why this might be the case in the next lecture.

Another technique is known as *semi-supervision.* This assumes access to a large unlabeled dataset (e.g. pictures on the internet). Then, to train using this this dataset, they assign "pseudo-labels" to them by using some non-robust classifier, then alternating iterations of adversarial training on the real dataset and adversarial training on these pseudo-labelled data points, but with typically a smaller learning rate. The paper [10] reports that this technique is also able to achieve better robust accuracy, around $5 - 10\%$ empirically.

References

[1] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International Conference on Machine Learning*, pages 274–283, 2018.

[2] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 3–14. ACM, 2017.

[3] 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.

[4] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, and Dimitris Tsipras. Robustness (python library), 2019.

[5] Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. You only propagate once: Painless adversarial training using maximal principle. *arXiv preprint arXiv:1905.00877*, 2019.

[6] Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! *arXiv preprint arXiv:1904.12843*, 2019.

[7] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. In *International Conference on Learning Representations*, number 2019, 2019.

[8] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric Xing, Laurent El Ghaoui, and Michael Jordan. Theoretically principled trade-off between robustness and accuracy. In *International Conference on Machine Learning*, pages 7472–7482, 2019.

[9] Dan Hendrycks, Kimin Lee, and Mantas Mazeika. Using pre-training can improve model robustness and uncertainty. *arXiv preprint arXiv:1901.09960*, 2019.

[10] Yair Carmon, Aditi Raghunathan, Ludwig Schmidt, Percy Liang, and John C Duchi. Unlabeled data improves adversarial robustness. *arXiv preprint arXiv:1905.13736*, 2019.