6.883 Science of Deep Learning – Spring 2018 March 14, 2018

Lecture 11: Adversarial Examples and Misclassification Attacks

Lecturer: Aleksander Mądry *Scribe: Anastasiya Belyaeva, Ryan Chung, Samuel Finalyson*
(Revised by Andrew Ilyas and Dimitris Tsipras)

1 Introduction: Security in the ML Pipeline

The dramatic success of AlexNet in the 2012 ImageNet challenge [10] has ushered in – seemingly overnight – a veritable deep learning frenzy. In the years since, there has been impressive progress on challenging benchmarks, with deep neural networks often surpassing human performance on these tasks. This progress has lead many to believe that Deep Learning is currently at a state to be applied robustly and reliably on virtually any task.

Nevertheless, current deep learning systems exhibit considerable brittleness, and are vulnerable to manipulation at all stages of development and deployment. In this next phase of the course, we will discuss several key limitations to current deep learning systems, and attempt to provide principled formulations for why they exist and how we might address them.

- **Training**: data poisoning
- **Inference**: adversarial examples
- **Deployment**: model stealing and privacy

In this lecture and the next, we'll focus on adversarial examples, which have arguably received the most attention in the field of ML security. In the lectures following, we will delve into the other parts of the machine learning pipeline.

(Also, you might find this blogpost http://gradientscience.org/intro_adversarial/ (and the two following blogposts) as well as this tutorial https://adversarial-ml-tutorial.org/ to be interesting supplementary readings.)

2 Adversarial Attacks 101

2.1 History and examples

Adversarial examples are inputs to machine learning models that an attacker has specifically crafted to cause the model to misbehave in some pre-specified way. For example, in the case of image classifiers, an adversarial example might be a natural image that is imperceptibly perturbed by an attacker in order to induce misclassification. Adversarial examples come in a variety of forms, and demonstrate that ML models are not at a human level of robustness. Adversarial examples are also not unique to deep convolutional neural networks, and have been shown to exist for a wide range of ML systems [4].

In the context of deep neural networks, adversarial examples where first observed by Szegedy et al. [13]. They show that imperceptible changes to an image can cause a well-performing neural network to misclasssify it (see Figure 1 for an illustration). Adversarial attacks have also been successfully translated into the physical world. It was shown that they continue to be adversarial even when printed and photographed [11]. Recently, it was shown that one can 3D-print physical models that are misclassified from many different angles and lighting conditions [2].

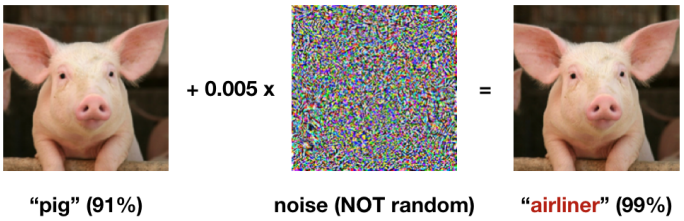


Figure 1: Image of a pig to which small and imperceptible to the human eye noise has been added – this caused misclassification of the image as an airliner.

2.2 Mathematical formulation

First, let's try to formulate adversarial examples in a more concrete, mathematically way, and in particular from the perspective of optimization. Recall that our fundamental goal in supervised learning is to provide an accurate mapping from an input X_i to an output y_i by optimizing over some set of model parameters θ. This can be formulated as the following minimization problem:

$$\min_{\theta} \text{loss}(\theta, X_i, y_i),$$

which is typically solved by performing stochastic gradient descent on the model parameters (i.e. model training).

The above approach has been the workhorse of supervised learning. A very similar approach can be taken in order to cause the model to misclassify a specific input. To execute an adversarial attack, we consider our model parameters θ as fixed and instead optimize over the *input space*.

Specifically, we search for a perturbation δ that can be added to X_i to *maximize* the resulting model loss:

$$\max_{\delta \in \Delta} \text{loss}(\theta, X_i + \delta, y_i).$$

In order to ensure that the adversarial perturbation is not completely changing the image, we restrict it to be "small". In particular, we constrain the adversarial perturbation δ to be from a class of appropriate perturbations Δ. The choice of how to define Δ is domain-specific and should thus be chosen after carefully considering the problem at hand. Common approaches include:

- **ℓ_p-norm bound**: Δ is a small-norm ℓ_p ball, and so δ is constrained to be small with respect to some norm ($\|\delta\|_p \leq \epsilon$ for some ϵ and p). Common choices are $p = \infty$ which prevents any pixel from being changed by more that ϵ, and $p = 2$ which constrains the adversarial example to be close in Euclidean norm to the original input).
- **VGG feature similarity**: Δ is the set of images that trigger similar activations in one of the hidden layers of a VGG network. Thus, the perturbation δ cannot change the VGG features (a specific embedding to feature space, obtained through a DNN) of the image by too much.
- **Other pertubation models**: There are also many more "natural perturbations" that are not captured above, including small rotations and translations [7, 8], as well as pixel displacements [15].

For the rest of this lecture, we will assume that $\Delta = \{\delta \mid ||\delta||_\infty \leq \epsilon\}$, and so the adversary is allowed to perturb each pixel in the input image by a certain budget ϵ.

3 Properties of adversarial examples

In this section, we'll review some key properties of adversarial examples that will prove important in our investigation of adversarial robustness.

3.1 Adversarial examples exist for linear models

Adversarial examples are not unique to sophisticated DNN models. Let's consider a simple linear regression model

$$f(x) = w^T X.$$

If we apply a perturbation δ to the input of this model, we have

$$f(X + \delta) = w^T(x + \delta) = w^T X + w^T \delta.$$

To maximize the effect of this perturbation subject to having $\|\delta\|_\infty \leq \epsilon$, we set $\delta_i = \text{sign}(w_i)\epsilon$, then

$$f(X + \delta) = w^T X + \epsilon \|w\|_1.$$

In the case that w has high dimensionality (and thus usually large ℓ_1-norm), the small perturbation can be magnified by applying a bias in the direction away from this image's true class.

Also, while deep neural networks possess many nonlinearities, these models are sometimes thought to be *locally linear*. In fact, in the case of the most popular nonlinearity (the ReLU), deep neural networks are actualls *piecewise linear* functions of their input. Thus, one hypothesis concerning the existence of adversarial examples states that it is actually linear perturbations just like the above that drive adversarial examples in deep networks [9].

3.2 Adversarial examples are not random

While it may be tempting to assume that adversarial examples are a product of the models poor robustness to random noise, empirical studies have demonstrated that this is not the case. In fact [9], adding random noise to the inputs produces adversarial examples a small fraction of the time (Figure 2), whereas adding a perturbation in the direction of a calculated attack (the FGSM attack in this case, see below for details), adversarial examples abound (Figure 3).

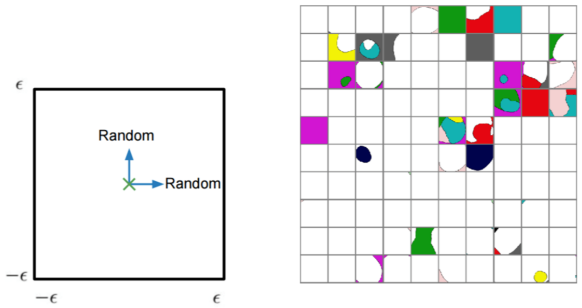


Figure 2: Adversarial examples are not noise.

Precisely how large is the space of adversarial examples? Tramer et al. [14] sought to quantify the dimensionality of the adversarial subspaces. They found that only approximately 50 directions were adversarial (Figure 4).

Despite the comparatively small dimensionality of the adversarial subspaces, however, other studies have shown that models can be "wrong almost everywhere." [9]. In other words, if you choose *random* inputs to the neural network, you can still perturb the images into any any class you desire in most cases (Figure 5).

3.3 A word of caution

A cottage industry has sprung up generating various ad-hoc defenses to specific adversarial attacks. Time and time again, these defenses have been repeatedly broken, often extremely quickly [6, 1]. This emphasizes the need to evaluate proposed defenses in a rigorous manner, considering adaptive adversaries. Relying on "security through obscurity" only adds noise to the field. (See the recent survey by Carlini et al [5] that discusses the best practices in this context.)

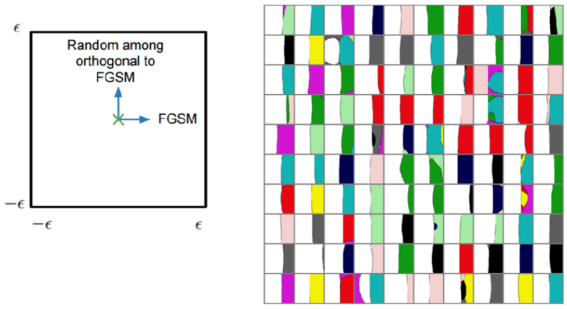


Figure 3: Adversarial examples are plenty in certain directions.

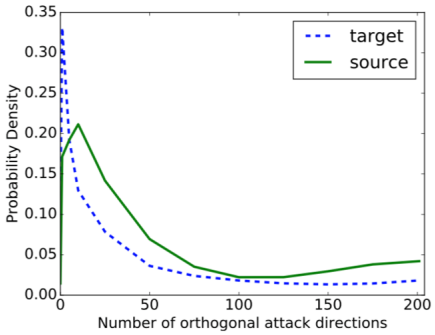


Figure 4: Number of successful orthogonal adversarial perturbations on the source DNN model and of the number of perturbations that transfer to the target DNN model.

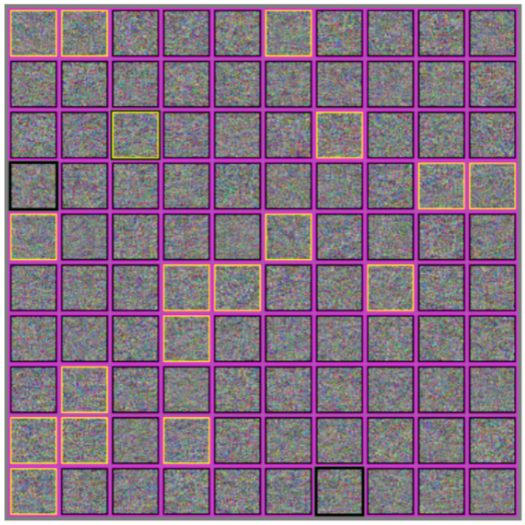


Figure 5: Randomly generated fooling images for a convolutional network trained on CIFAR-10. These examples were generated by drawing a sample from an isotropic Gaussian, then taking a gradient sign step in the direction that increases the probability of the airplane class. Yellow boxes indicate samples that successfully fool the model into believing an airplane is present with at least 50% confidence.

4 A principled approach for avoiding adversarial examples

In this section, we'll build up to proposing a principled defense against adversarial examples, based on the well-established field of robust optimization. First, we discuss why the existence of adversarial examples may actually be expected based on our current methods for training machine learning systems.

4.1 Should the existence of adversarial examples surprise us?

As we attempt to avoid adversarial examples, it is first important to note that the phenomenon as it currently exists does not necessarily demonstrate a *failure* of machine learning models per se. Rather, adversarial examples reflect a disconnect between what we explicitly optimize our models to do and the behavior that we (often mistakenly) expect our models to implicitly learn along the way.

To make this statement more precise, let's consider the problem we generally hope to solve in supervised learning from a statistical perspective. In learning our model, we generally seek to minimize our expected model loss over some true data distribution (X, y) $\mathcal{D}$:

$$\min_{\theta} \mathbb{E}_{(X,y)\sim\mathcal{D}} \left[\text{loss}(\theta, X, y)\right].$$

Likewise, our training procedure can be framed as seeking to approximately minimize this expected loss by finding parameters θ which minimize the empirical (mean) loss over a finite training set drawn from the distribution:

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \text{loss}(\theta, X_i, y_i) \{(X_i, y_i) \sim \mathcal{D}\}.$$

Both of the formulations above are designed to solve *exactly* the task of expected loss minimization on a given distribution, and neither account in any way for the presence of an adversary in making predictions. As such, it is not reasonable to expect that the resulting model will be robust in the presence of an adversary, and particularly one whose actions take place in a rather low-dimensional space. Making correct predictions in the face of the non-random, adversarial perturbations that we have seen so far is thus completely orthogonal to the empirical risk minimization problem that our training procedure tries to solve. Thus, we should not really be surprised to see that ML models obtained through standard training are vulnerable to these sorts of adversarial examples.

4.2 Towards a paradigm for adversarial robustness

Once an understanding has been established of the task that our machine learning algorithm is solving and not solving, we can redefine our machine learning optimization task to suit our particular goals.

Now that we've identified (at a high level) why these adversarial examples may not be surprising given current training methods, we wish to find a method of training that *will* lead to adversarially robust models. Specifically, by appropriately modifying our training objective and accounting for the presence of an adversary, we can build supervised learning models that are far more robust to adversarial examples.

Rather than simply minimizing expected empirical risk, we now ask our model parameters to minimize *adversarial risk*. In particular, we ask the model to minimize the loss of the "worst-case" perturbations of the data distribution (where we once again use the set Δ to constrain the perturbations):

$$\min_{\theta} \mathbb{E}_{(X,y)\sim\mathcal{D}} \left[\max_{\delta\in\Delta} \text{loss}(\theta, X+\delta, y)\right]. \tag{1}$$

Formulating and solving such min-max problems has a rich history in the field of *robust optimization* [3] (in the context of deep learning, robust optimization is often referred to as *adversarial training*).

How should we solve this adversarial risk minimization problem? To answer this question, we break our training objective (1) into two parts: the *outer minimization*, and the *inner maximization*.

4.2.1 Outer minimization

We can abstract the inner problem by defining

$$\text{loss}_{\text{adv}}(\theta, X_i, y_i) = \max_{\delta \in \Delta} \text{loss}(\theta, X_i + \delta, y_i).$$

This allows us to think of the outer minimization problem as a more familiar loss minimization problem, where we minimize loss_{adv} with respect to θ. A natural approach we might use to solve such a minimization problem, then, is gradient descent. In particular, we want to update θ according to:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \text{loss}_{\text{adv}}(\theta, X, y)$$

Our question now becomes: *how do we actually attain this gradient?* In practice, we take inspiration from a fundamental theorem of min-max optimization, known as Danskin's Theorem:

Theorem 1 (Danskin's Theorem) *Suppose we have a function $f(x, y)$, where x is a vector in $\mathbb{R}^d$, and y is a member of some compact set $\mathcal{Y}$. Furthermore, suppose that f is convex in x for every y. We define*

$$g(x) = \max_{y \in \mathcal{Y}} f(x, y).$$

Then, so long as $f(x, y)$ is continuous in both its arguments, we have that:

$$\frac{d}{dx} g(x) = \frac{d}{dx} f(x, y^*) \qquad \text{where } y^* = \arg\max_{y \in \mathcal{Y}} f(x, y).$$

Intuitively, Danskin's Theorem tells us that the gradient of the outer minimization problem is actually simply the gradient of the inner maximization problem, *evaluated at the maximizing point.* In our setting (modulo the non-convexity of deep networks), Danskin's Theorem tells us that the gradient $\nabla_\theta \text{loss}_{\text{adv}}(\theta, X, y)$ is actually equal to $\nabla_\theta \text{loss}(\theta, X^*, y)$, where X^* is just the loss-maximizing input for the current value of θ (i.e. an *adversarial example*!). We end up with the following robust optimization algorithm for deep networks:

1. Sample a batch of inputs X_i and their corresponding labels y_i.
2. Calculate the "worst-case perturbation" of X_i with respect to the current parameters θ_t, i.e. solve:

$$\delta^* = \max_{\delta \in \Delta} \text{loss}(\theta_t, X_i + \delta, y_i)$$

3. Update the parameters θ_t according to the minimization problem at $X_i + \delta^*$:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \text{loss}(\theta_t, X_i + \delta^*, y_i)$$

This solution to the adversarial learning problem should not be confused with GANs. Instead, this approach is akin to data augmentation, where the model is trained not on the data itself but on modified data and in this particular case, adversarially modified data. More generally, data augmentation can be thought of as replacing the maximization in the training equation above with an expectation over random transformations.

Although the formulation above is non-convex (and thus Danskin's Theorem is not directly applicable), it turns out that our deep learning methods can be made remarkably robust by applying this methodology.

4.2.2 Inner maximization

We have just presented a framework for training adversarially robust models inspired by Danskin's Theorem from convex optimization. Note, however, that solving the outer minimization requires access to δ^*, the loss-maximizing perturbation of the input X. Finding δ^* thus corresponds to solving the inner maximization problem.

This maximization problem boils down to finding the direction to move within a constraint set $\Delta_X = \{X + \delta | \delta \in \Delta\}$ around each input X_i which maximizes the probability the model assigns to an incorrect class. Several techniques for solving this problem have been proposed, including both single-step and iterative methods.

Fast gradient sign method (FGSM). The fast gradient sign method (FGSM) (Figure 6a) was the first simple adversarial attack proposed and consists of taking a single step in a direction maximizing correlation with the gradient:

$$\delta_{\text{FGSM}} = \max_{\delta \in \Delta} \langle \nabla_X \text{loss}(\theta, X, y), \delta \rangle$$

FGSM was developed for the specific setting in which the set of admissible pertubations Δ is the ℓ_∞ ball, i.e. $\{\delta | \max_i \delta_i \leq \epsilon\}$. In this case, FGSM induces the following update step:

$$\delta = \epsilon \cdot \text{sign}\left(\nabla_X \text{loss}(\theta, X_i, y_i)\right)$$

While FGSM benefits from being the simplest adversarial attack, it is often relatively ineffective at solving the inner maximization problem. In particular, the attack requires that the local information at any point X_i be predictive of the model's behaviour in the neighbourhood of X_i. As such, the use of FGSM in adversarial attack and defense are heavily discouraged.

Projected gradient descent methods (PGD). The canonical method for generating adversarial examples (and thus for solving the inner maximization) is via *projected gradient descent* (PGD). Projected gradient descent methods have a long history in constrained optimization, and are an established method for minimizing (or equivalently, maximizing) a convex function within a given constraint set (again, deep neural networks are certainly non-convex, but just as in the case of training, first-order methods tend to be quite effective).

PGD is an iterative method; starting from the original input $X^{(0)} := X$, the input is updated as follows:

$$X^{(t+1)} = \Pi_\Delta \left[X^{(t)} + \alpha \cdot g_t \right]$$
$$\text{where } g_t = \max_{\delta \in \Delta} \langle \nabla_X \text{loss}(\theta, X_i^{adv}, y_i), \delta \rangle$$

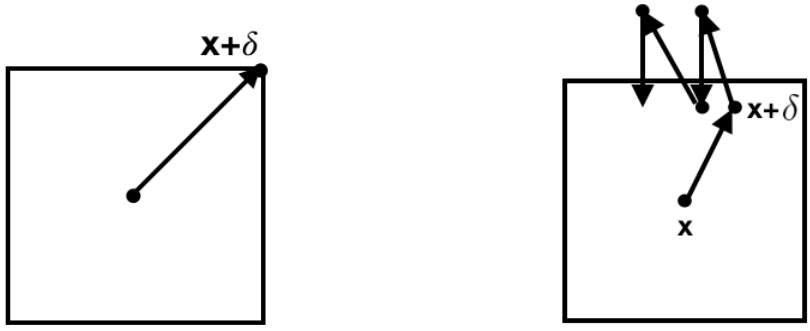
where Π_Δ is the projection operator onto the set Δ, i.e:

$$\Pi_\Delta[x] = \min_{\delta \in \Delta} \|x - \delta\|^2.$$

In the case where Δ is the ℓ_∞ ball, we get the update step:

$$X^{(t+1)} = \text{clip}\left(X^{(t)} + \alpha \cdot \text{sign}\left(\nabla_X \text{loss}(\theta, X^{(t)}, y) \right),\ X_0 - \epsilon,\ X_0 + \epsilon \right)$$

Intuitively this method can be thought of as iteratively applying FGSM, but forcing the perturbed image to stay within the admissible set Δ at every step. To ensure this, each time PGD takes a step, it checks if it has moved out of the box, and applies a projection back into the box if necessary (Figure 6b). PGD is naturally more expensive than FGSM, but allows for more effective attacks.



(a) Fast Gradient Sign Method (FGSM) (b) Projected Gradient Descent (PGD)

Figure 6: Gradient updates for FGSM (a) vs. PGD (b) where x is the training example, the box is the space of perturbations and δ is a particular perturbation.

5 Properties of adversarial attacks

Through experiments using MNIST and CIFAR10, the following properties of adversarial training have been observed [12]:

- The choice of the attack method is important for the final robustness of deep learning models.
- Even though the inner maximization problem is highly non-convex and can have many local maxima, the loss of the inner maximization problem over many random initializations of PGD seems to be a well-concentrated distribution
- Adversarial robustness may require larger model capacity/expressivity (wider models tend to be needed).

The choice of attack matters. A comparison of FGSM vs. PGD reveals that PGD results in much better performance than FGSM (Figure 7), which linearizes the inner maximization problem.

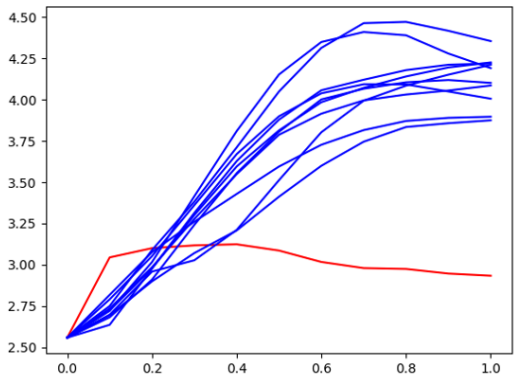


Figure 7: The choice of attack method - FGSM (red) vs. PGD (blue) matters.

Losses are well-concentrated. Since PGD solves a non-convex problem, the solution given by PGD is one of possible local maxima. However, it is possible that much larger local maxima exist that PGD is unable to find. Madry et al. have investigated this behavior by performing many restarts. As shown in Figure 8, the final loss follows a well-concentrated distribution.

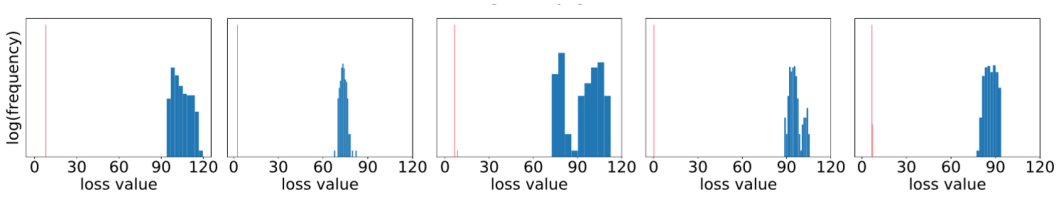


Figure 8: Values of the local maxima given by the cross-entropy loss for five examples from the MNIST and CIFAR10 evaluation datasets. For each example, PGD is started uniformly at random around the example and iterated until the loss plateaus.The blue histogram corresponds to the loss on a naturally trained network, while the red histogram corresponds to the adversarially trained counterpart. The loss is significantly smaller for the adversarially trained networks, and the final loss values are very concentrated without any outliers.

Model capacity matters. Training against strong adversarial examples requires a more expressive classifier with more capacity to distinguish one class from another. Figure 9 shows that a simple linear decision boundary is unable to separate all adversarial examples within ℓ_∞ of the training point because it is not complex enough. However, when a more complicated decision boundary can be learned, smaller number of adversarial examples will be misclassified.

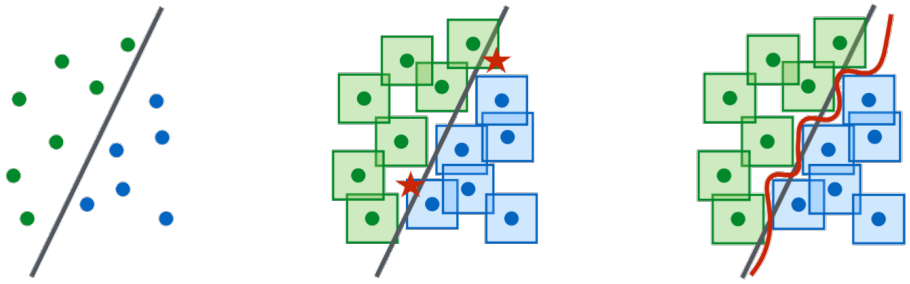


Figure 9: Natural classification (left) vs. adversarial boundaries (right) corresponding to ℓ_∞ ball around training points.

This intuition has been captured via experimental results that trained convolutional networks on natural examples, FGSM examples and PGD examples while doubling the size of network. As shown in Figure 10, small capacity networks have larger loss even on natural examples, indicating that capacity alone increases accuracy. When adversaries like PGD are added, for small capacity networks PGD fails to learn a meaningful decision boundary and performance is sacrificed for robustness. On the other hand, for large capacity networks a robust and accurate solution can be achieved with PGD adversary.

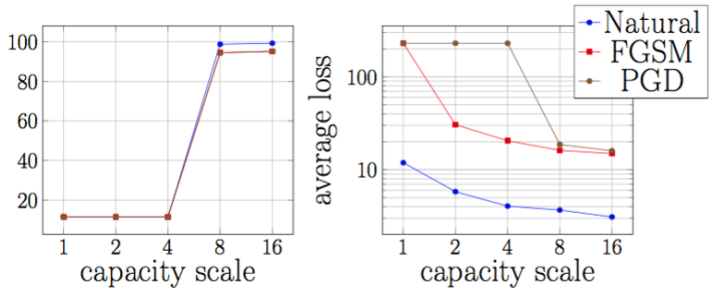
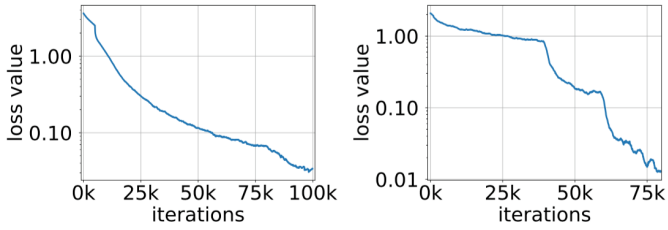


Figure 10: Average accuracy and average loss vs. capacity of convolutional neural networks trained with natural examples, FGSM adversary and PGD adversary.

The PGD adversary was trained for both MNIST and CIFAR10 and it has been shown that there is a steady decrease in the training loss of adversarial examples (Figure 11) showing an indication that the original adversarial training optimization problem is indeed being solved during training.

6 Conclusions

Deep learning models are susceptible to adversarial examples where the differences between training and adversarial images are indistinguishable to the human eye. It has been shown that adversarial examples are not random noise, but are a subspace of specifically chosen perturbations. A variety of approaches that come up with new attack and defense methods have been coming out, but many of these are ad-hoc and rely on obscurity rather than model robustness. Recently a new paradigm that explicitly optimizes for robustness to attacks has been developed, based on the classical theoretical framework of robust optimization. Taking inspiration from the convex case, we can successfully train deep learning models



(a) MNIST (b) CIFAR10

Figure 11: Value of adversarial loss function on MNIST and CIFAR10 datasets during training.

that are far more resistant to adversarial attack. These networks also let us study interesting properties of adversarial robustness in the context of deep learning.

References

[1] A. Athalye, N. Carlini, and D. Wagner. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. *arXiv*, 2018.

[2] A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok. Synthesizing Robust Adversarial Examples. *ICLR*, 2018.

[3] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust optimization*. Princeton University Press, 2009.

[4] B. Biggio and F. Roli. Wild Patterns: Ten Years After the Rise of Adversarial Machine Learning. *arXiv*, pages 32–37, 2017.

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

[6] N. Carlini and D. Wagner. Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods. *arXiv*, 2017.

[7] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh. Are classifiers really invariant? In *British Machine Vision Conference (BMVC)*, 2015.

[8] L. Engstrom, B. Tran, D. Tsipras, L. Schmidt, and A. Madry. A rotation and a translation suffice: Fooling CNNs with simple transformations. In *ArXiv preprint arXiv:1712.02779*, 2018.

[9] I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. 12 2014.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. 25, 01 2012.

[11] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial examples in the physical world. *ICLR*, pages 1–14, 2017.

[12] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. *arXiv*, pages 1–27, 2017.

[13] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv*, pages 1–10, 2014.

[14] F. Tramèr, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel. The Space of Transferable Adversarial Examples. *arXiv*, pages 1–15, 2017.

[15] C. Xiao, J. Zhu, B. Li, W. He, M. Liu, and D. Song. Spatially transformed adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2018.