

Towards Efficient Methods for Training Robust Deep Neural Networks

Sanjit Bhat*
Acton-Boxborough Regional High School
sanjit.bhat@gmail.com

Dimitris Tsipras
MIT
tsipras@mit.edu

Aleksander Mądry
MIT
madry@mit.edu

February 13, 2019

Abstract

In recent years, it has been shown that neural networks are vulnerable to adversarial examples, i.e., specially crafted inputs that look visually similar to humans yet cause machine learning models to make incorrect predictions. A lot of research has been focused on training robust models—models immune to adversarial examples. One such method is Adversarial Training, in which the model continuously trains on adversarially perturbed inputs. However, since these inputs require significant computation time to create, Adversarial Training is often much slower than vanilla training. In this work, we explore two approaches to increase the efficiency of Adversarial Training. First, we study whether faster yet less accurate methods for generating adversarially perturbed inputs suffice to train a robust model. Second, we devise a method for asynchronous parallel Adversarial Training and analyze a phenomenon of independent interest that arises—staleness. Taken together, these two techniques enable comparable robustness on the MNIST dataset to prior art with a $26\times$ reduction in training time from 4 hours to just 9 minutes.

1 Introduction

In recent years, Machine Learning (ML) systems have achieved surprising success in several domains such as Computer Vision and Natural Language Processing [KSH12, LBH15]. In contrast to traditional rule-based or symbolic AI systems, ML systems learn input and output pairings automatically through labeled data. This provides them independence from human programming, allowing them to act in complex scenarios without being explicitly programmed to. In addition, ML models have the potential to scale with more powerful hardware and larger datasets, which is not necessarily true for traditional AI systems.

Combined, these two properties have allowed ML systems to achieve super-human performance in several settings such as Computer Vision and complex game playing. In Computer Vision, large-scale datasets such as ImageNet [RDS⁺15] led to a race that resulted in more accurate Computer Vision models such as VGG [SZ15] and ResNet [HZRS16]. These ML models eventually surpassed human level performance in image classification tasks [HZRS15]. In game playing, Google DeepMind’s AlphaGo system used ML to act and perceive in a longstanding open problem, Go [SHM⁺16]. With

*Work done while in the MIT PRIMES program.

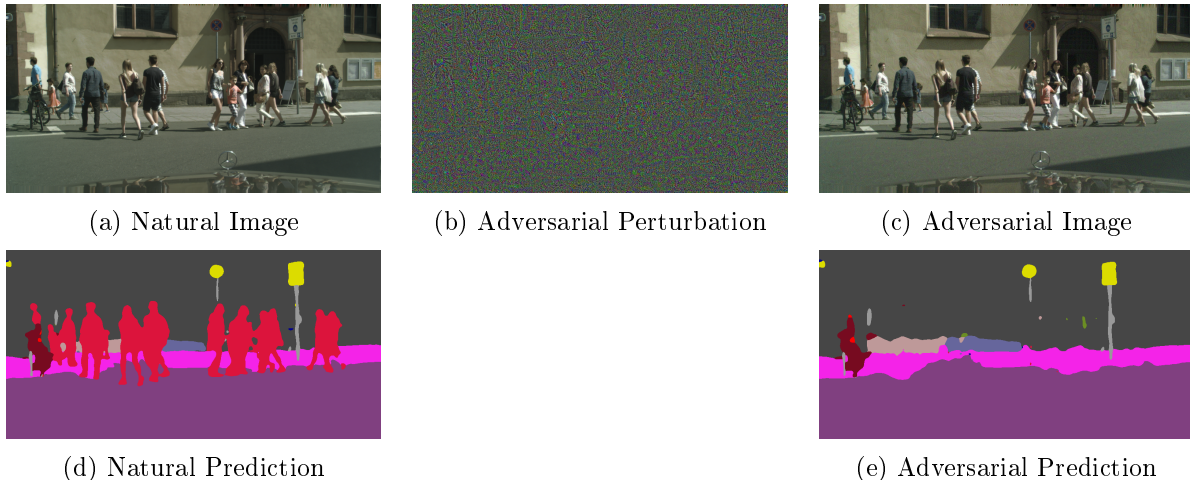


Figure 1: Adversarial examples for self-driving cars (images replicated from Fischer et al. [FCMB17]). Figure 1a shows a natural image taken from a car camera that is later fed into an image segmentation model. The output of the model’s predictions are shown in Figure 1d, with the car recognizing pedestrians crossing the street and most likely taking the necessary precautions. In the adversarial case, however, a small amount of seemingly random noise shown in Figure 1b is added to the natural image. Even though the resulting image in Figure 1c looks visually indistinguishable to humans, it causes the segmentation model to fail to recognize the humans, as shown in Figure 1e. Thus, the car would act as if no pedestrians were crossing and fail to apply the brakes, resulting in catastrophic consequences.

a combination of knowledge from prior games and self-play, AlphaGo defeated the foremost human Go player, Lee Sedol, in 2016 [Dee16].

Owing to its success in several fields, ML is closer to being adopted in security-critical applications. For example, Tesla [Lam18] and Waymo [Haw18] use Computer Vision systems to enable their self-driving cars to perceive their surroundings. Instead of classifying objects, these systems perform segmentation, distilling an image into its various classes (e.g., road signs, people, trees, and buildings) to aid the driving system in making critical decisions. Similarly, Apple’s new Iphone X uses Computer Vision to power its facial unlocking system [Tea17]. The phone extracts facial features, measures similarity to its owner’s face, and grants entrance to contacts, passwords, emails, and bank accounts if the two faces match.

In these security-critical applications, there are concerns about whether ML systems will properly function to protect one’s life and privacy. One prominent concern is the existence of adversarial examples [GSS15, MMS⁺18, WK18], inputs that look visually similar to their natural counterparts yet cause ML models to make wrong predictions. For example, with self-driving cars, the adversary crafts a perturbation such that to humans both the original and adversarial images look the same (see Figure 1). However, when fed into the image segmentation model, the adversarial example causes the car to not recognize the humans. These attacks can be constructed without full access to the ML model [IEAL18] and can be implemented in the physical world [AEIK18, KGB17].

Given the above discussion, adversarial examples bring into question the reliability of our state-of-the-art ML systems. Apart from the security implications of malicious actors being able to wreck

havoc on large-scale, security-critical systems, these flaws raise two other fundamental issues:

1. **Robustness to real-world perturbations.** Self-driving cars not only malfunction with random-noise-like input perturbations, but also on naturally occurring perturbations such as rain, sleet, and snow [Sto18]. In order to have full trust that ML models will ensure privacy and safety in all scenarios, including those with real-world perturbations, we need to make them robust to adversarial examples.
2. **Alignment with human intelligence.** One of the goals of AI research is to create a general-purpose system that can solve a wide variety of problems. Oftentimes, we use the human brain as an example of such a system. However, given that most adversarial examples use small, quasi-imperceptible perturbations, a human brain would likely not be tricked [Bro14]. Thus, the existence of adversarial examples points to possible flaw in current ML systems: these systems do not learn the same way that we humans do. Such human-like intelligence might be useful in creating general-purpose AI.

Recently, several works have proposed methods for training *robust* models, models that are not susceptible to adversarial inputs (see [ACW18] and references therein). While these “defenses” against adversarial attacks were shown to be robust in their original papers, many were soon circumvented by more sophisticated attacks [ACW18]. One defense that has withstood counter-attack is Adversarial Training—continuously augmenting training with adversarially perturbed inputs. However, while Adversarial Training provides resistance to adversarial attacks, it adds significant amounts of computational overhead compared to regular training. This reduces its practicality in the real-world, especially in applications that require frequent model re-training. Consequently, in this work, we ask the following question:

Can robust ML models be trained efficiently?

Our contributions. In this work, we study the above question via two complementary approaches. First, we investigate whether cruder and faster approximations can be used for adversarially perturbed inputs. Our results indicate that a balance point exists between the computational overhead of the approximation and the model’s final robustness. Second, we develop an asynchronous parallel implementation of Adversarial Training that provides a nearly-linear speedup with multiple GPUs. Of independent interest, we analyze a phenomenon called staleness that arises from using this parallelization technique. Finally, we bring our findings together to improve the efficiency of Adversarial Training, reducing the state-of-the-art robust training time on the MNIST dataset by 26×, from 4 hours to 9 minutes.

2 Background

Regular machine learning training. A standard neural network consists of several weight layers, parametrized by θ , that define an input-output pairing for a certain data distribution $\widehat{\mathcal{D}}$. By optimizing a loss function, \mathcal{L} , the network finds better values for θ . Since lower loss refers to a better prediction, the training procedure for a regular neural network corresponds to solving the

following optimization problem:

$$\min_{\theta} \left[\mathbb{E}_{(x,y) \sim \widehat{\mathcal{D}}} \mathcal{L}(x, y, \theta) \right]. \quad (1)$$

This optimization is typically done using Stochastic Gradient Descent.

Adversarial examples. Recall that adversarial examples are small input perturbations¹ that cause ML models to make wrong predictions. Fooling the model corresponds to maximizing the loss \mathcal{L} . In addition, to make “small input perturbations” concrete, we define the set of possible perturbations to be \mathcal{S} . Thus, the adversary tries to achieving the following:

$$\max_{\delta \in \mathcal{S}} \mathcal{L}(x + \delta, y, \theta). \quad (2)$$

Set of allowed perturbations. The set of small perturbations is broad, including natural semantically similar perturbations such as rain and snow and other distortions such as rotations and translations [ETT⁺17]. While robustness to all perturbations is an important topic of research, it is difficult to mathematically define perturbations such as rain and snow. In the absence of a concrete definition of small perturbations, we use the ℓ_{∞} metric, consistent with prior work [GSS15, MMS⁺18]. Specifically, every perturbation δ with components x_1, x_2, \dots, x_n must adhere to the following constraint:

$$\max_i |x_i| \leq \varepsilon. \quad (3)$$

Using this constraint, we precisely define the set \mathcal{S} of allowable perturbations, which forms an ε -ball around the natural point.

Computing adversarial examples. In prior work, researchers proposed algorithms such as the Fast Gradient Sign Method (FGSM) [GSS15] to optimize Equation 2. FGSM uses the gradient of the loss with respect to the input to form a locally linear approximation of the loss for any change in input. It then takes a step in the direction of the gradient to maximize the predicted loss. The “fast” in FGSM refers to the fact that the algorithm only takes one step of size ε , computing gradients once and hitting the edge of the ℓ_{∞} ε -ball immediately:

$$x + \varepsilon \operatorname{sgn}(\nabla_x \mathcal{L}(x, y, \theta)). \quad (4)$$

While FGSM provides a fast method for finding adversarial examples, it relies too heavily on the local linearity assumption. In reality, neural networks have complex, non-linear loss landscapes (see Figure 2), which makes it hard to predict the loss of a point far out in the ε -ball [MMS⁺18]. As such, a natural extension to FGSM known as Projected Gradient Descent (PGD) [KGB17, MMS⁺18] uses κ smaller steps of size α . At each step, PGD re-computes gradients relative to its current solution and projects back into the ε -ball. Intuitively, by re-measuring the local linearity at several checkpoints along its optimization trajectory, PGD relies less on any one approximation. This leads to better adversarial examples than those produced by FGSM [MMS⁺18]:

$$x^{t+1} = \Pi_{x+\mathcal{S}}(x^t + \alpha \operatorname{sgn}(\nabla_{x^t} \mathcal{L}(x^t, y, \theta))). \quad (5)$$

¹After all, if the adversary was unconstrained, he could simply use an image from a different class.

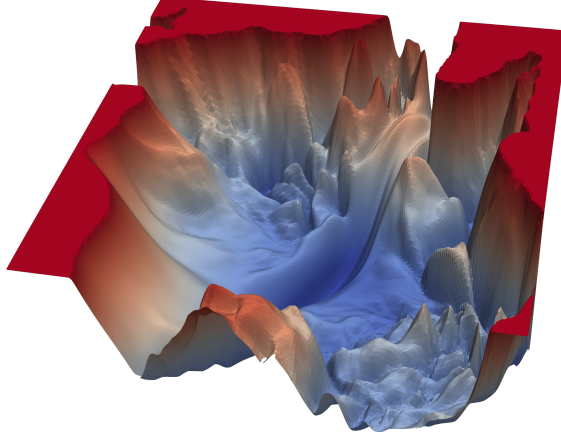


Figure 2: An actual neural network loss landscape from Li et al. [LXT⁺17]. Although the x and y axes represent changes in the network parameters rather than the network inputs, the visualization still leads to an important observation: neural networks have highly complex, non-concave loss landscapes. If a single-step adversary such as FGSM [GSS15] tries to approximate a far-away local maxima with locally linear information, it might not arrive at a reasonable solution.

In Equation 5, $\Pi_{x+\mathcal{S}}$ refers to projection back into the ε -ball surrounding the natural point. By projecting at each timestep, we ensure we always stay within the constraints. Specifically, for an adversarial example v with components v_1, v_2, \dots, v_n , a natural example x with components x_1, x_2, \dots, x_n , and an ℓ_∞ constraint, projection ensures that v_i lies between $x_i - \varepsilon$ and $x_i + \varepsilon$.

Training a robust deep neural network (DNN). One natural way of achieving robustness is Adversarial Training. Instead of training on natural examples, the DNN simulates the adversary, computes adversarial examples, and trains on them. It then updates its weight parameters, θ , to reflect its new knowledge of the adversary and its own prior flaws. Since the adversary continually points out new flaws to the network, the DNN needs several timesteps of Adversarial Training to achieve robustness.

Combining Equation 1 for natural DNN training with Equation 2 for the adversary’s objective, we arrive at the following min-max saddle-point formulation for Adversarial Training:

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y) \sim \hat{\mathcal{D}}} \left[\max_{\delta \in \mathcal{S}} \mathcal{L}(x + \delta, y; \theta) \right]. \quad (6)$$

This equation can be solved in practice using Danskin’s Theorem [Dan67, Ber99], which says that the gradients of the loss, $\nabla_{\theta} \mathcal{L}(x + \delta; y; \theta)$, are equivalent to the gradients of the max loss, $\nabla_{\theta} \max_{\delta \in \mathcal{S}} \mathcal{L}(x + \delta; y; \theta)$, when δ is a global maximum. However, as the loss function is non-concave (see Figure 2), the solutions yielded by PGD are actually local maxima, not global maximum. Nonetheless, using a sufficiently large number of steps usually leads to strong-enough local maxima to solve the min-max problem.

Thus, we use the descent direction provided by $\nabla_{\theta} \mathcal{L}(x + \delta; y; \theta)$ to minimize the max loss and solve the saddle point problem. For a strong-enough inner adversary, this procedure yields universally robust DNNs (i.e., DNNs robust to *any* first-order adversary constrained by the ℓ_∞ norm) [MMS⁺18, ACW18].

Intuitively, Adversarial Training corresponds to a two-player game. When competing against an “opponent”, it is necessary to understand how that opponent “plays”. Similarly, training a robust DNN can be viewed within this context. Throughout the training procedure, the adversary tries to exploit the DNN’s flawed feature mappings. To defeat the adversary, the DNN must then “understand” how it is being exploited by training on adversarial examples.

3 Towards Efficient Robust Training

As the universal robustness described in Section 2 is highly desirable, we improve the practicality of Adversarial Training in the following sections by addressing its efficiency.

While PGD builds upon FGSM by using a multi-step optimization procedure, it also introduces significant computation overhead. For each added step, the algorithm must compute one additional set of gradients using the Backpropagation algorithm [RHW86], a slow process. For example, with the current implementation of PGD in the Mađry Lab MNIST Challenge [Lab17], the added benefit of increased strength maximizers comes at the cost of a roughly $40\times$ longer training procedure. In Section 4, we study the broad impact of using a fewer number of steps and use our findings to drastically reduce the computational overhead of Adversarial Training.

Another downside of Adversarial Training is its synchronous nature. With each iteration of the min-max problem in Equation 6, the outer minimization process waits for the inner maximization to finish, and vice-versa. This increases the overall time to execute Adversarial Training. In addition, we cannot use multiple GPUs to speed up computation of a single adversarial batch since PGD is an inherently sequential process.

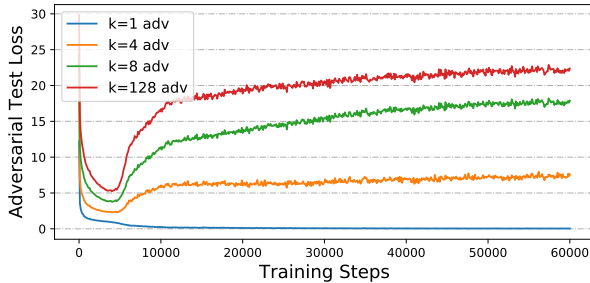
Instead, in Section 5 we develop an asynchronous implementation of PGD wherein the adversary and trainer work independently. This allows for multiple GPUs to drastically speed up adversarial example computation. However, it also results in *staleness*, or the adversarial examples pointing out flaws from previous network timesteps. We study staleness in-depth through extensive experiments, and we arrive at a final asynchronous implementation that gains a roughly-linear speedup as more GPUs become available.

Finally, combining our studies of attacker power and asynchrony, we present our results for efficient Adversarial Training on the MNIST dataset [LBBH98]. Using both techniques, we achieve a $26\times$ reduction in training time from 4 hours to 9 minutes.

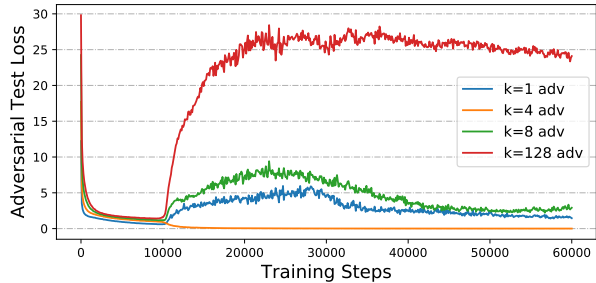
4 Understanding Attacker Power

To solve the min-max optimization problem in Equation 6 and train a robust model, Danskin’s Theorem proves that we should use a global maximum adversarial example to calculate gradients for descent directions. However, finding the global maximum in a non-concave, highly complex loss landscape is nearly impossible and would take far too long to compute. As such, researchers have used first-order methods such as FGSM and PGD to compute approximate global maximum, sacrificing quality for speed of computation [GSS15, MMS⁺18]. While it has been shown that FGSM Adversarial Training is not robust (see [MMS⁺18] and references therein), a less powerful PGD attack might still provide robustness. Thus, in this section, we investigate the trade-off between the quality of adversarial examples and their computational overheads.

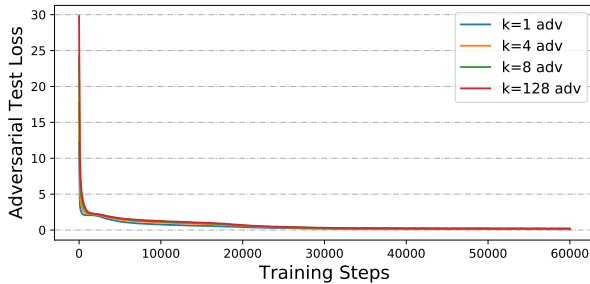
The quality of adversarial examples (e.g., their ability to maximize loss) is a function of the number of steps used to create them. Given a constant epsilon, ϵ , we split the overall trajectory



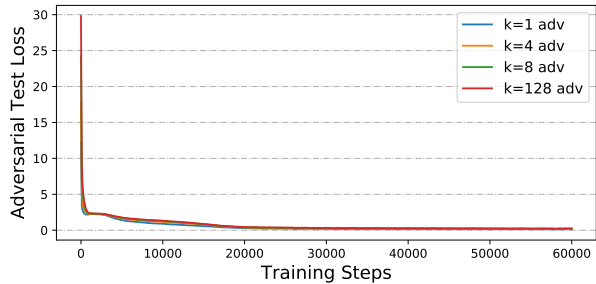
(a) Trained with FGSM



(b) Trained with $\kappa = 4$ PGD



(c) Trained with $\kappa = 8$ PGD



(d) Trained with $\kappa = 128$ PGD

Figure 3: Figures 3a, 3b, 3c, and 3d show adversarial loss on the testing set for networks trained with a $\kappa = 1, 4, 8,$ and 128 PGD adversary, respectively. The four lines within each figure display the loss graph for a $\kappa = 1, 4, 8,$ and 128 PGD adversary attacking that respective network. While the networks trained with $\kappa = 1$ and 4 PGD adversaries are only robust to the attacks they trained on, the $\kappa = 8$ and 128 networks exhibit robustness against a wide range of attacks.

into κ equally sized steps, recalculating gradients and a descent direction at each step. Since PGD uses first-order gradients to form a locally linear approximation at each step, a larger number of steps results in a smaller step size and less reliance on the approximation. Given how actual neural network landscapes are complex and non-linear, using more steps should therefore yield better quality adversarial examples, helping both the inner maximization, outer minimization, and overall robustness.

While using more steps results in better maxima, it also comes with significant computational costs. With normal training, each iteration of the outer minimization problem would only take one pass of the relatively slow Backpropagation algorithm. Adversarial Training, however, requires additional passes for each additional step, a linear increase in training time. With the trade-off between strength of adversarial examples and training time, the following question arises:

Is there a balance between attacker strength and training time?

Adversarial Training with a weak attack. Upon first glance, low-power and high-power PGD adversaries all perform the same for a network trained with a powerful adversary such as PGD with $\kappa = 8$ or 128 (see Figures 3c and 3d). However, their similar loss values result from the network’s strong robustness: *every* first-order attack will have low loss because the network has become robust to all of them. Conversely, Figures 3a and 3b show the intriguing difference between adversarial

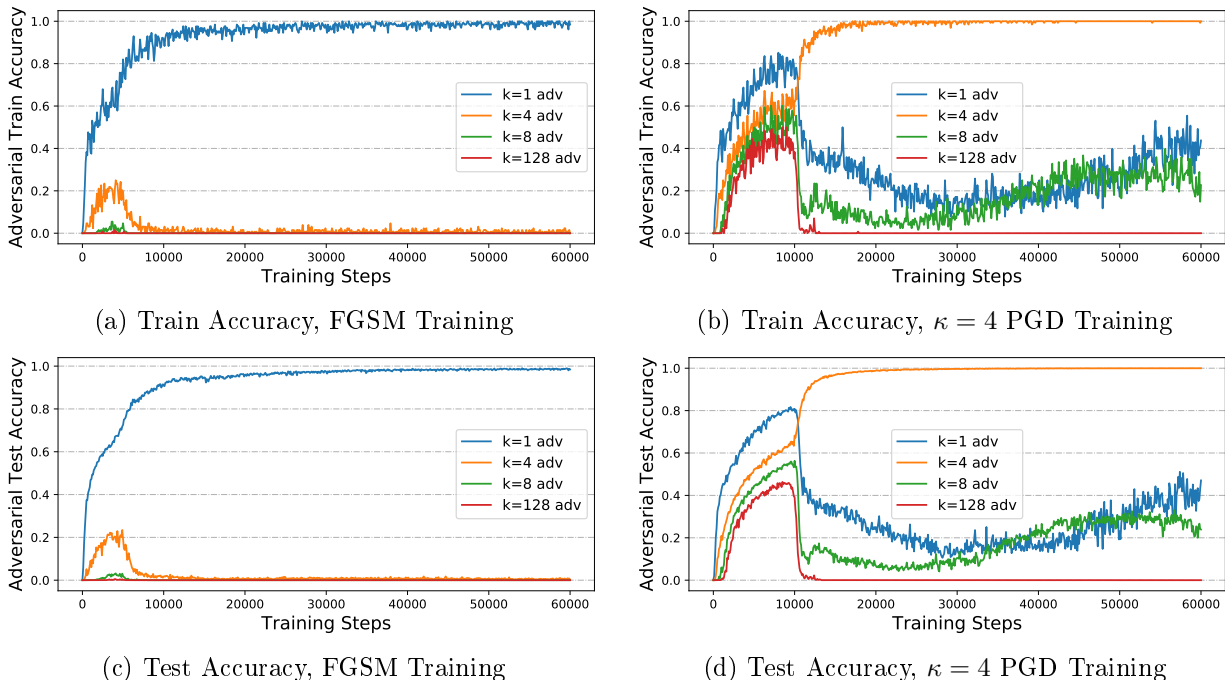


Figure 4: Figures 4a and 4b show adversarial accuracy on the training set for networks trained with a $\kappa = 1$ and 4 PGD adversary, respectively. Figures 4c and 4d show adversarial accuracy on the testing set for networks trained with a $\kappa = 1$ and 4 PGD adversary, respectively. The four lines within each figure display the accuracies for a $\kappa = 1, 4, 8,$ and 128 PGD adversary attacking that respective network. The 100% train accuracy against the $\kappa = 1$ adversary in Figure 4a and the $\kappa = 4$ adversary in Figure 4b point towards these networks memorizing their attacker training distribution. The corresponding near-100% test accuracy in Figures 4c and 4d points to a much more intriguing phenomenon: gradient shattering [ACW18].

power when weak attacks are used to train the model. Not only do higher power adversaries yield higher loss, but also the adversary used to train the network has a loss near 0 for a significant portion of training. Is this because the network has simply gained robustness against the weak adversary it was trained on, or is something more interesting going on?

Our evidence points to the latter. First, observe that for the networks trained with $\kappa = 1$ and 4 PGD, adversarial accuracy on the training set for that specific attacker goes to nearly 100% after a small number of training steps (see Figures 4a and 4b). Since natural accuracy for these MNIST models hovers around 98% and the adversarial setting is harder than its natural counterpart, these networks likely memorized the distribution of possible weak adversarial examples in the training set to achieve 100% adversarial accuracy. Indeed, recent work has shown that even state-of-the-art CNN’s are capable of memorizing completely random input data [ZBH⁺17]. Since the adversarial examples are not fully random and instead rely on the underlying distribution of natural training images, the network could certainly have memorized them.

However, more intriguing is the near 100% adversarial accuracy on the *testing* set (see Figures 4c and 4d). Firstly, adversarial accuracy for a higher κ , stronger adversary should never be higher than that of a lower κ , but it is in Figure 4d. In addition, the network never had the opportunity to

Table 1: Adversarial accuracy on the testing set for various attacks. Each value represents a testing set created with a certain κ PGD adversary on a certain κ source network and evaluated on a certain κ target network. The diagonal represents white-box attacks whereas the off-diagonal represents black-box attacks. The highlighted values show evidence of gradient shattering [ACW18]: weak white-box attacks on a weak network are not effective, but the same strength attack transferred over from a different network is effective.

Target κ	Source $\kappa = 1$	Source $\kappa = 4$	Source $\kappa = 8$	Source $\kappa = 128$	Adv κ
1	98.55%	48.63%	27.16%	28.36%	1
4	57.34%	99.98%	54.18%	60.31%	4
8	96.40%	97.27%	93.88%	94.26%	8
128	96.28%	96.69%	93.92%	91.70%	128

memorize images in the testing set, yet it still managed near-perfect accuracy against an adversarial attack of the same strength it was trained on. Taken together, both observations point to a phenomena called *gradient shattering* introduced by Athalye et al. [ACW18]. When one trains a network with a weak adversary, the model learns to augment its loss landscape such that any adversary with the same strength cannot generate meaningful gradients. Thus, even though the testing set consists of new natural images, the adversary cannot generate strong adversarial examples due to the network’s corrupted loss landscape.

The gradient shattering phenomena can be further evidenced in Table 1. All of the above experiments have been in the white-box setting, where the adversary has access to the model’s parameters and directly uses them to create examples. In this black-box setting, the adversary is not provided access to the model, needing to use a surrogate “source” network to transfer adversarial examples to a “target” network. For a black-box attack on the networks trained with $\kappa = 1$ and 4, notice how an adversary using the exact same power PGD manages to break the networks even though that same adversary cannot break the networks in a white-box setting. This indicates that networks trained with weak adversaries shatter their gradients to achieve robustness. Unfortunately, this method does not provide robustness to higher strength adversarial examples or even same-strength examples transferred from a surrogate network.

Towards a balance point. In our experiments, we search from $\kappa = 1$ to $\kappa = 128$ in powers of 2 to understand whether certain strength attackers yield favorable balance points between the quality of adversarial examples and their computational overhead. In the white-box setting, observe that networks trained with a $\kappa = 8$ and a $\kappa = 128$ adversary both yield strong robustness against the full spectrum of first-order attackers (see Figures 3c and 3d). This is unexpected since the network trained with a $\kappa = 8$ adversary never saw the more powerful $\kappa = 128$ adversary during its training procedure. One likely explanation is that with $\kappa \geq 8$, PGD produces sufficient approximates to the global maximum adversarial example, allowing the network to gain full, universal robustness on the MNIST dataset. Thus, even attackers using a greater numbers of steps cannot fool these networks.

Furthermore, moving from $\kappa = 4$ to $\kappa = 8$ and $\kappa = 128$, we see no evidence of gradient shattering in either of the latter two networks (see Table 1). For example, for the target network trained with $\kappa = 8$ PGD, computing a $\kappa = 8$ example on the $\kappa = 128$ source network yields marginal differences

in adversarial accuracy (93.88% and 94.26%, respectively).

In summary, while networks trained with $\kappa \leq 4$ lead to faster training times, they lack universal robustness and exhibit gradient shattering. Conversely, networks trained with $\kappa \geq 8$ offer these desirable traits while remaining otherwise indistinguishable (e.g., all such networks achieved around 98% natural accuracy and 89–91% adversarial accuracy on the test set). As such, we train our final network with $\kappa = 8$ compared to the $\kappa = 40$ in the Mądry Lab MNIST Challenge to balance robustness with training time. This technique provides a roughly $5\times$ improvement in training time.

5 Asynchronous Parallelization

In Section 4, we studied trade-offs with attacker strength to arrive at a $5\times$ reduction in training time. Here, we look at an orthogonal idea involving asynchronous parallelization to arrive at even greater speedups.

Traditionally, researchers have solved Equation 6 in a synchronous manner, simulating the adversary in the inner loop, feeding the adversarial examples to the model, and using the model to create adversarial examples. While this makes the adversarial examples used by the network fresh—they are computed for the most recent timestep—it also introduces significant delays; the minimization process must wait for the much slower maximization process before continuing. This raises the following question:

Can we parallelize the inner maximization to make it faster?

Unfortunately, due to the inherently serial nature of PGD wherein each step builds off of the previous step (see Equation 5), we cannot achieve parallelism by applying multiple GPUs to a single, small batch. Instead, we achieve parallelism by using multiple worker GPUs to simultaneously create adversarial batches and store them in a queue once completed. Later, a separate trainer GPU picks up batches from the queue, calculates gradients from them, and uses the gradients to minimize loss.

To demonstrate asynchronous parallelization, consider a scenario with one trainer GPU and two worker GPUs. The workers start computing an adversarial batch using parameters θ_t from the network at time t . Since both workers perform the same optimization, they finish at around the same time and add their batches to the queue. Assuming the trainer GPU was previously unoccupied, it takes the first batch and uses it to calculate new parameters, θ_{t+1} , that minimize the adversarial loss. Now, when the trainer takes the second batch from the queue, there exists a discrepancy between timesteps. Since the second adversarial batch was computed for θ_t while the network now has parameters θ_{t+1} , it points out slightly older network flaws, reducing its effectiveness. We call this phenomenon *staleness* and study its effects in greater depth to arrive at a final asynchronous parallelization implementation.

Simulating staleness. Real-world multi-GPU parallelization has several tunable parameters, making it hard to evaluate staleness. Instead, we use a single GPU with a queue of size s to simulate an exact staleness s that persists throughout network training. Specifically, at the start of training, we fill the queue to its maximum capacity and iterate through the normal, synchronous training procedure. Since each example is processed in one timestep, if the example at the back of the queue enters at timestep t , it will exit at timestep $t + s$. The difference between entrance and exit timesteps, s , matches the difference in the network parameters used to create the adversarial

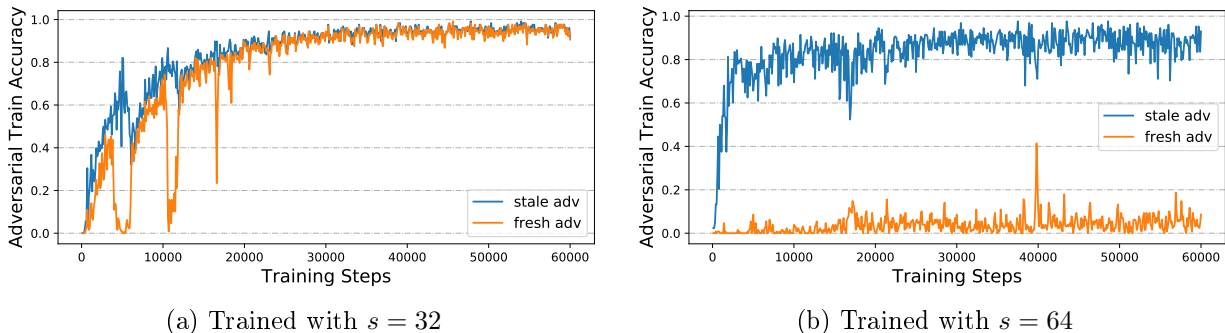


Figure 5: Figures 5a and 5b show adversarial accuracy on the training set for networks trained with $s = 32$ and 64 staleness, respectively. The two lines within each figure represent the accuracy on fresh and stale adversarial examples. Even though each network was trained on stale examples, we evaluate fresh accuracy by generating fresh examples for the current network timestep. The difference between fresh and stale training accuracy in Figure 5b indicates that $s = 64$ networks memorize the stale data distribution, lacking robustness to fresh adversarial examples. In contrast, the $s = 32$ network achieves high accuracy on both fresh and stale examples.

example. Since every example has this exact timestep difference s , we have a constant staleness s for every adversarial batch.

The downsides of high staleness. Just as the models trained with weak attacks in Section 4 overfit to their specific attack, high staleness networks overfit to stale data. Within the context of staleness, we use the terms “fresh” and “stale” to refer to accuracy on adversarial examples generated at the current timestep and at a previous timestep, respectively. Consider Figures 5a and 5b, which show fresh and stale adversarial accuracy for networks trained with $s = 32$ and $s = 64$. While fresh accuracy tracks stale accuracy for $s = 32$, it is much lower than stale accuracy for $s = 64$. This indicates that high staleness networks memorize the stale data distribution, failing to become robust to fresh adversarial examples. In the following discussion, we hypothesize on what happens in the network training procedure to cause these issues.

The cyclic behavior caused by high staleness. Consider taking a snapshot of the stale adversarial examples created throughout the training procedure, the same examples as those used for network training. For regular robust training with an ℓ_∞ constraint, the adversary does not care how many individual pixels it perturbs since the constraint restricts the maximum perturbation (see Equation 3). Figures 6a through 6d show the adversary’s choice to perturb what looks like a large number of random pixels. In contrast, we notice that with $s = 64$ in Figures 6e through 6l, the adversarial examples look a lot less random. Specifically, the set of stale examples shadows a line segment that moves through the image and back again.

A game-playing analogy. Why do the adversarial examples cycle for high staleness? To understand further, consider a thought experiment with a game of rock, paper, and scissors. For each move made by a player, there is exactly one other move that wins against it, losses against it, and ties against it. Thus, if the adversary played randomly, the optimal move is to randomly choose

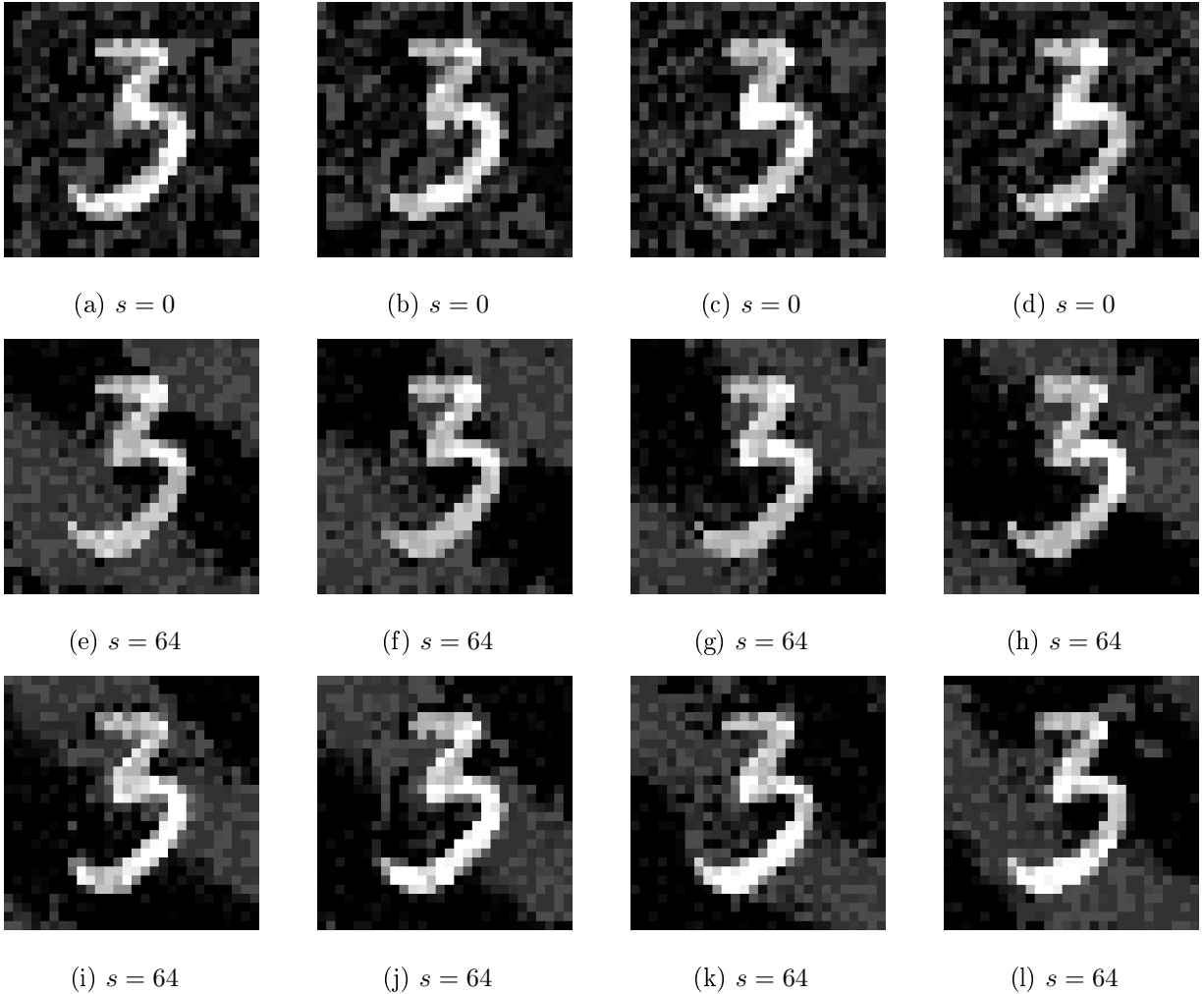


Figure 6: Figures 6a through 6d show adversarial examples at sequential timesteps for a network trained with no staleness. Figures 6e through 6l show adversarial examples at sequential timesteps for a network trained with $s = 64$. Note how the former looks nearly random while the latter shows a line segment cycling through the image. This cyclic behavior is one the drawbacks of high staleness: the network and adversary become entranced in a cat-and-mouse game trying to catch up with each other.

an action, yielding a $1/3$ chance of winning, a $1/3$ chance of losing, and a $1/3$ chance of tying. Now consider a large lag between the adversary’s actions and when they are played out (i.e., add staleness). If by random chance the player has several rocks that win, he might put more weight into that action, playing it more often. The adversary would certainly punish this decision, but his paper moves would not appear for several more turns.

Finally, once the model gets flooded by paper, it learns to switch its weights, constantly playing scissors instead. In response, the adversary changes to rock, but its actions still show paper due to the lag. This cat and mouse game would continue with the model switching to paper and the adversary switching to scissors until the model finally ends back where it started—rock.

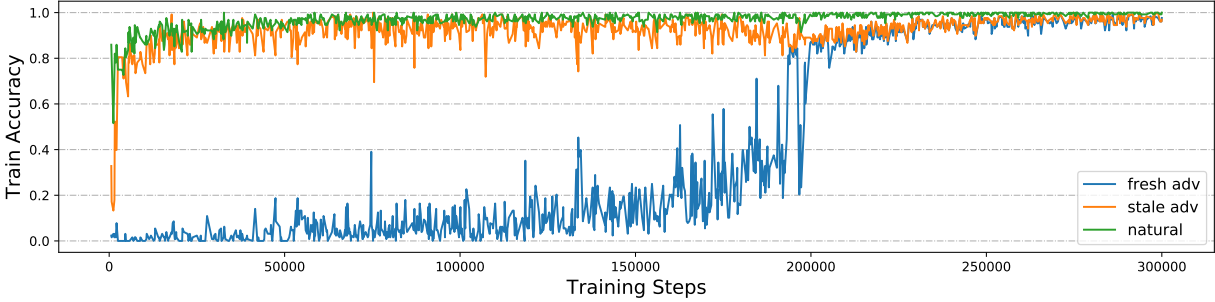


Figure 7: The broken $s = 64$ network training for 300,000 steps instead of our usual 60,000.

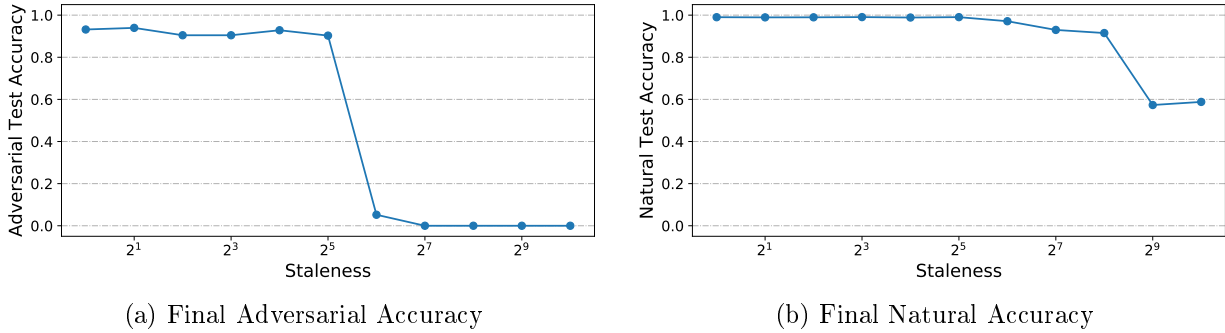


Figure 8: Figures 8a and 8b show final adversary and natural accuracies, respectively, on the testing set for varying values of staleness. We increment staleness on the x-axis in powers of 2.

This rock, paper, scissors analogy intuitively shows that cyclic behavior is caused by staleness between the player and the adversary. Both parties play the same move for a considerable amount of time due to the lag in their actions, resulting in a seemingly never-ending cycle. Drawing parallels between the rock, paper, scissors thought experiment and Adversarial Training, we believe high staleness also causes cyclic behavior, as evidenced by the line segments in Figure 6.

Breaking the cycles. Interestingly enough, we found that in some cases the cycles can be stopped by training the network with more timesteps. We took a network trained with $s = 64$ that previously exhibited memorization issues and let it run for 300,000 training steps instead of 60,000 training steps.

Figure 7 shows the results. About halfway through training, fresh adversarial accuracy suddenly climbs up to match stale adversarial accuracy and closely tracks it for the rest of training. While it is unclear what exactly happened here, one theory is that the cyclic staleness behavior has a certain entropy over time. For higher staleness, it takes a longer build-up of this entropy to break the network’s paralysis, but eventually it could happen.

Unfortunately, in the real-world it is impractical to train networks for 300,000 timesteps due to the significantly increased computation time. Thus, we pursue a balance between high staleness, which allows for greater parallelization, and its drawbacks, which include memorization, lack of robustness, and cyclic behavior.

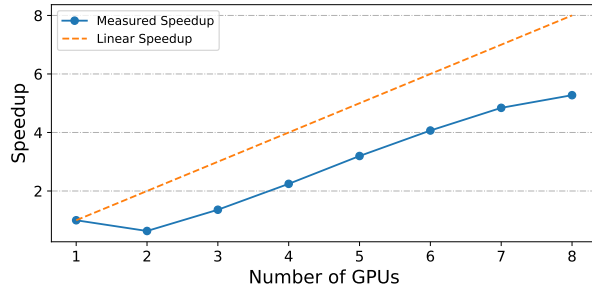


Figure 9: The speedup we achieve from using more GPUs in our asynchronous implementation of Adversarial Training. The baseline here (i.e., number of GPUs = 1) is regular PGD Adversarial Training with $\kappa = 8$. All subsequent runs use that same number of steps. Using asynchronous parallelization, we achieve a roughly-linear speedup as more GPUs are added.

Towards a suitable staleness value. In our experiments, we search between staleness values from 1 to 1024 in powers of 2 to determine if there is a clear breaking point where the effects of staleness hinder robustness. In Figure 8a, we plot the final adversarial accuracy on the *test* set as a function of a greater variety of staleness values.

We note a similar phenomena as before (see Figure 5) regarding the difference between $s = 32$ and 64. For $s \leq 32$, the final adversarial accuracy on the testing set remains about the same at 90%. However, after $s = 32$, adversarial accuracy drops down from 90% to around 5%, a major decrease that indicates a lack of robustness. Thus, for MNIST it appears that a network trained with $s = 32$ manages to achieve sufficient robustness, all while keeping a natural accuracy of around 98% (see Figure 8b).

Implementing Asynchronous Parallelization Finally, we bring staleness into the real world by implementing asynchronous parallelization. To achieve parallelism, we use between 0–7 worker GPUs and 1 trainer GPU. Between the workers and trainer sits a shared memory queue to store completed adversarial batches, and we restrict the queue’s maximum size to mitigate staleness. Finally, we curb staleness by also having the trainer GPU re-save the model’s parameters every 100 training steps, a fairly rapid pace.

In Figure 9, we plot the speedup as a function of the number of GPUs. Speedup is the time taken to train the baseline divided by the time taken to run the actual trial. Compared to the ideal linear-speedup shown in orange, we achieve slightly worse speedup for 2 GPUs since the trainer spends most of its time waiting for adversarial batches on the queue. We note that this can be fixed if the trainer also computes adversarial examples in its spare time waiting.

However, after 2 GPUs, we see an almost-linear speedup as the number of GPUs increases. By the time we get to 7 GPUs, we reach a nearly $5\times$ speedup and start to plateau. By this time, the worker GPUs combined begin to produce adversarial batches at a faster rate than the network can consume them. This causes the queue to fill up, blocking the workers from being utilized until the queue has more space. It makes sense that this saturation point should occur with 8 GPUs. We used $\kappa = 8$ for these experiments, so it takes approximately $8\times$ as long to make an adversarial batch as it does to train on it.

Comparison with the baseline. With our final 8 GPU configuration, we reach over a $5\times$ speedup compared to the $\kappa = 8$ baseline with one GPU. This configuration took a total of 9 minutes to train while achieving an adversarial accuracy of around 90% and a natural accuracy of 98%. Compared to the 4 hours for the Mađry Lab MNIST challenge code with $k = 40$ and synchronous training, we achieve an overall $26\times$ reduction in training time for the same level of robustness and same natural accuracy.

6 Experimental Setup

We perform all our experiments on the MNIST Computer Vision dataset [LBBH98], which consists of 65,000 labeled, grayscale, 28×28 images of handwritten digits. After feeding an MNIST image into a classifier, it must output the correct label, an integer from 0–9 that the image refers to. We use MNIST here since it has been widely studied not only in regular Machine Learning community, but also in the Adversarial Machine Learning community [GSS15, ETT⁺17, ACW18, WK18, MMS⁺18].

For all our tests, we use the same Convolutional Neural Network (CNN) used by Mađry et al. [MMS⁺18] and the TensorFlow MNIST tutorial [Dev18]. This network consists of two convolutional and max pooling layers with 32 and 64 filters, respectively, and a receptive field of size 5. After being sent through the convolutional layers, data is passed through a fully-connected layer with 1024 neurons before finally reaching a 10 neuron fully-connected output layer. We use the ReLU activation function for all layers except the final layer, which uses the softmax function to create a discrete probability distribution. The CNN is trained with the Adam optimizer [KB15], a variant of Stochastic Gradient Descent with fast convergence. We use categorical crossentropy for our loss function, a common choice for multinomial classification problems. The same random seed is used throughout all experiments, and we randomly shuffle the entire MNIST dataset and feed it in batches of 128 samples to the model.

To evaluate model performance, we use accuracy and loss across the train and test sets. The train set consists of 55,000 examples, while the test set contains 10,000 different examples. Since they are mutually exclusive, accuracy on the test set measures the generalization capability of the model to unseen data. In addition, we measure both accuracy and loss in the adversarial and natural setting. In the former, we replace the natural examples with adversarial examples to measure the model’s robustness.

Unless otherwise noted, we use 60,000 training steps for each experiment and fix $\varepsilon = 0.3$, one third of the 0–1 grayscale range for each pixel. For a given number of steps, the step size is calculated as follows:

$$\alpha = \frac{\varepsilon}{\kappa} \times 2.5.$$

This ensures that the full optimization trajectory is divided into κ equally-sized steps. In addition, we scale step size by 2.5 since all of our experiments use random restarts (i.e., before starting PGD, we move to a random starting point within the ε -ball). Random restarts allow PGD to explore new areas of the ε -ball. However, just as it takes 2 radii to travel from one end of a diameter to another, it takes at least 2ε to travel from one end of the ε -ball to another. The 0.5 adds a little extra tolerance for PGD to explore far-out adversarial examples.

Our baseline experiment is the randomly initialized model from Mađry et al.’s MNIST challenge, which achieves 89%–91% adversarial accuracy and 98% natural accuracy on the test set and takes roughly 4 hours to train.

7 Discussion

Other parallelization approaches. In this work, we approached parallelization through multiple GPUs asynchronously computing small batches of adversarial examples. Another possible approach is to use a large batch size, split the batch into several small chunks that fit into an individual GPU’s memory, and give the large batch to the trainer GPU. In recent work using this parallelization technique, researchers trained a Computer Vision model to achieve high natural accuracy on the ImageNet dataset [GDG⁺17] in just one hour.

Instead of viewing this technique as competing, we view it as an orthogonal idea. To further improve practicality, it is likely that techniques such as asynchronous parallelization must join with other approaches such as large batch size training.

8 Conclusion

In this work, we present two approaches for reducing the computational overhead of Adversarial Training: cruder approximations of adversarial perturbations and asynchronous parallelization. We present in-depth studies of both techniques, push their limits, and search for balance points wherein we do not sacrifice crucial traits such as adversarial robustness. Combining these two techniques, we achieve a 26× reduction in robust training time on the MNIST dataset, going from 4 hours to just 9 minutes. Overall, our work moves an important step towards developing efficient methods for training robust DNNs.

References

- [ACW18] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. In *Proceedings of the International Conference on Machine Learning*, 2018.
- The authors, researchers at MIT and UC Berkeley, find that most prior Adversarial ML defenses (except Adversarial Training) use a form of obfuscated gradients to provide security. Obfuscated gradients, they argue, provide a false sense of security and can be circumvented through their techniques. Among all defenses tested, Adversarial Training remains unbroken.
- [AEIK18] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing Robust Adversarial Examples. In *Proceedings of the International Conference on Machine Learning*, 2018.
- In this paper, the authors, researchers at MIT, 3D print robust physical adversarial examples that trick a state-of-the-art ML model over a variety of natural transformations (e.g., viewpoint shifts and camera noise).
- [Ber99] Dimitri P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1999.
- Bertsekas, an MIT professor, is one of the leading professors working in the field of mathematical optimization. He has written several books, including this one, where he provides a proof of a more general version of Danskin’s theorem [Dan67].
- [Bro14] Bethany Brookshire. How brains filter the signal from the noise. <https://www.sciencenews.org/blog/scicurious/how-brains-filter-signal-noise>, 2014.

In this article from ScienceNews, the author describes how humans have a key structure in their minds that allows them to filter out unnecessary random-noise from auditory stimuli. In contrast to humans, the article points out how ML systems might have a tougher time doing this.

- [Dan67] John M. Danskin. *The Theory of Max-Min and its Application to Weapons Allocation Problems*. Springer-Verlag Berlin Heidelberg, 1967.

In this classic monograph, Danskin provides a way of calculating the directional derivative of the maximum of an arbitrary directionally differentiable function. His theorem relies on finding the global maximum of the function, which we are unable to do since PGD converges to local maxima. In practice, we use iterative optimization methods such as PGD to reliably find close approximates to the global maximum perturbation.

- [Dee16] Google DeepMind. The Google DeepMind Challenge Match, March 2016. <https://deepmind.com/research/alphago/alphago-korea/>, 2016.

In this official blog post from Google DeepMind, their team announces a 4–1 win over Lee Sedol, a top Go player.

- [Dev18] TensorFlow Developers. Build a Convolutional Neural Network using Estimators. <https://www.tensorflow.org/tutorials/estimators/cnn>, 2018.

This is an official TensorFlow tutorial made by Google. The authors provide code for creating a basic Convolutional Neural Network for running on the MNIST dataset. We, along with Mađry et al. [MMS⁺18], use this code for our DNNs.

- [ETT⁺17] Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Mađry. A Rotation and a Translation Suffice: Fooling CNNs with Simple Transformations. *arXiv preprint arXiv:1712.02779*, 2017.

This paper from authors at MIT finds that neural networks are not only fooled by complicated optimizations, but also by rotations and translations, perturbations more likely to be found in the real world. This sheds light on the numerous invariances that could possibly be addressed in Adversarial ML.

- [FCMB17] Volker Fischer, Kumar Mummadi Chaithanya, Jan Hendrik Metzen, and Thomas Brox. Adversarial Examples for Semantic Image Segmentation. In *Proceedings of the International Conference on Learning Representations—Workshop Track*, 2017.

The authors show the existence of imperceptible adversarial perturbations for image segmentation, a technique used by self-driving cars to parse their surroundings. This may show that self-driving cars can be tricked to misclassify large numbers of pixels belonging to a particular class.

- [GDG⁺17] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv preprint arXiv:1706.02677*, 2017.

The authors, researchers at Facebook, develop analysis and techniques for large minibatch training in a distributed synchronous environment. Their techniques allow natural training on the ImageNet dataset in just 1 hour with comparable accuracy to small minibatch training. Although they do not study the adversarial setting, their techniques could potentially be transferred to this domain. However, further work must first be done on the possible side-effects of large minibatch Adversarial Training.

- [GSS15] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. In *Proceedings of the International Conference on Learning Representations*, 2015.
- This paper by Goodfellow et al. at Google is one of the earliest and most cited works in Adversarial Deep Learning. It relies on the linear properties of neural networks to create a fast, one-step optimization using first-order information—FGSM. One criticism of this work is the fact that neural networks often have complex, non-concave landscapes, which makes finding adversarial examples difficult. Mađry et al. point out this flaw and propose PGD, a multi-step attack, and we extend their work here by further studying the effects and tradeoffs of single vs. multi-step attacks.
- [Haw18] Andrew J. Hawkins. Inside Waymo’s Strategy to Grow the Best Brains for Self-Driving Cars. <https://www.theverge.com/2018/5/9/17307156/google-waymo-driverless-cars-deep-learning-neural-net-interview>, 2018.
- This article from popular science and technology magazine The Verge breaks down some of the technological background behind Waymo. While it does not go deep into technical details, it does outline how Waymo primarily uses AI and Deep Learning to power its self-driving car system.
- [HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Proceedings of the International Conference on Computer Vision*, pages 1026–1034, 2015.
- In this paper, the authors develop two techniques that extend the traditional Rectified Linear Unit (ReLU)—Parametric Rectified Linear Unit and a robust initialization method for ReLUs. Together, these enable the authors to build a deep rectified network that surpasses human-level performance, 5.1% top-5 test error, on the ImageNet 2012 classification dataset.
- [HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. 2016.
- In this seminal paper from Microsoft Research, the authors propose the Residual Network (ResNet) architecture for training deeper networks. This enables them to achieve high accuracy on datasets such as Imagenet.
- [IEAL18] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box Adversarial Attacks with Limited Queries and Information. In *Proceedings of the International Conference on Machine Learning*, 2018.
- This paper is one of many demonstrating the power of black-box adversarial attacks—attacks in which the attacker only has query access to the model. Specifically, this paper defines three different sub-settings of black-box attacks and develops novel methods to fool models in each setting.
- [KB15] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *Proceedings of the International Conference on Learning Representations*, 2015.
- This paper proposes Adam, a first-order method for performing stochastic gradient descent. Consistent with Mađry et al., we use Adam to train our DNN.
- [KGB17] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial Examples in the Physical World. In *Proceedings of the International Conference on Learning Representations—Workshop Track*, 2017.

This paper introduces the Basic Iterative Method (BIM), an extension to Goodfellow et al.’s FGSM that uses multi-step optimization. The authors apply BIM to synthesize real-world adversarial examples that fool a state-of-the-art ML model taking in cellphone images. Mađry et al. [MMS⁺18] will later recognize that BIM is equivalent to Projected Gradient Descent, (PGD) a multi-step optimization method proposed in prior mathematical literature.

- [KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 1097–1105, 2012.

In this seminal paper from researchers at the University of Toronto, the authors show the efficacy of deep convolutional neural networks on the Imagenet Challenge. This paper provided the impetus for the resurgence of deep learning over the past several years.

- [Lab17] Mađry Lab. Mađry Lab MNIST Challenge. https://github.com/MadryLab/mnist_challenge, 2017.

This is code from Mađry et al.’s paper [MMS⁺18]. We use it as the basis for our code, including our DNN architecture and our implementation of PGD and Adversarial Training.

- [Lam18] Fred Lambert. Tesla deploys massive new Autopilot neural net in v9, impressive new capabilities, report says. <https://electrek.co/2018/10/15/tesla-new-autopilot-neural-net-v9/>, 2018.

This article was written by Fred Lambert, Editor in Chief at Electrek, a news site that tracks progress related to electric transportation. In his article, Fred discusses the new deep neural network in the latest Tesla autonomous driving system, quoting official Tweets from Elon Musk, CEO of Tesla.

- [LBBH98] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

In this seminal paper, the authors show the power of convolutional neural networks in handwritten character recognition tasks. It introduces the MNIST dataset that is still used today, especially in Adversarial ML research, as a benchmark for model performance.

- [LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep Learning. *Nature*, 521:436–444, 2015.

This article, written by three of the leading Deep Learning researchers in the world and published in Nature, provides an overview of the current state of Deep Learning circa 2015. It also touches upon the numerous applications of unsupervised learning, CNNs, and RNNs.

- [LXT⁺17] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the Loss Landscape of Neural Nets. *arXiv preprint arXiv:1712.09913*, 2017.

In this paper, the authors develop techniques for visualizing the loss landscape of neural networks. They use these techniques to arrive at high-resolution 3D models that show the complex nature of deep neural networks. We note that while they generated loss landscapes with respect to model training parameters, here we refer to loss landscape with respect to the model inputs.

- [MMS⁺18] Aleksander Mađry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. In *Proceedings of the International Conference on Learning Representations*, 2018.

This paper, written by researchers at MIT, is the central paper on which we base our research. They show that robust deep neural networks can be trained through an intuitive process called Adversarial Training, in which the trainer simulates an adversary and augments the dataset with adversarially perturbed inputs. Adversarial Training has withstood attacks from Athalye and Carlini [ACW18] and has shown to be robust to all first-order attacks bounded by a particular ℓ_p norm.

- [RDS⁺15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

This paper introduces the ImageNet Challenge, a benchmark for large-scale object category classification and detection. Since 2010, ImageNet has been cited thousands of times and has spurred several new neural network architectures and training techniques.

- [RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning Representations by Back-Propagating Errors. *Nature*, 323:533–536, 1986.

In this classic paper co-authored by one of the godfathers of deep learning, Geoffrey Hinton, the authors introduce the Backpropagation Algorithm for efficient gradient computations in neural networks.

- [SHM⁺16] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

In this paper, the authors, researchers at Google DeepMind, explain the intricacies of their system AlphaGo. AlphaGo uses deep reinforcement learning and deep neural networks to synthesize highly competitive Go-playing. This led to its 4–1 victory against Lee Sedol in 2016 [Dee16].

- [Sto18] Kyle Stock. Self-Driving Cars Can Handle Neither Rain nor Sleet nor Snow. <https://www.bloomberg.com/news/articles/2018-09-17/self-driving-cars-still-can-t-handle-bad-weather>, 2018.

In this article from Bloomberg Businessweek, the author points out inclement weather (e.g., rain, sleet, and snow) as a major barrier to current self-driving car systems. This goes to show how some natural invariances that humans can sometimes deal with pose a major problem to current deep neural networks. One drawback of this article is its release time. Even though it was released on September 17, 2018, AI often progresses at a fast pace. It is possible that companies such as Waymo or Tesla have found ways to make robust self-driving systems in the rain, although to our best efforts we have found no reliable press releases indicating such feats.

- [SZ15] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *Proceedings of the International Conference on Learning Representations*, 2015.

This paper introduces the VGG network architecture, a relatively simple architecture consisting of several layers of 3×3 convolutions. Using the VGG architecture, the authors achieve first and second places in the localization and classification tasks, respectively, at the 2014 ImageNet Challenge.

- [Tea17] Apple Computer Vision Machine Learning Team. An On-device Deep Neural Network for Face Detection. <https://machinelearning.apple.com/2017/11/16/face-detection.html>, 2017.

In this official technical report by the Apple Computer Vision Machine Learning Team, the authors describe Apple’s use of deep learning Computer Vision algorithms in their face detection system. Furthermore, they discuss the efficiency challenges they incurred as well as their efforts to preserve user privacy.

- [WK18] Eric Wong and J. Zico Kolter. Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope. In *Proceedings of the International Conference on Machine Learning*, 2018.

The authors, researchers from Carnegie Mellon University, use a convex outer approximation to build a deep ReLU-based classifier that is provably robust to any norm-bounded adversarial attack. The authors apply their method to the MNIST dataset for digit recognition.

- [ZBH⁺17] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding Deep Learning Requires Rethinking Generalization. In *Proceedings of the International Conference on Learning Representations*, 2017.

The authors provide evidence contrary to conventional wisdom, which attributes the performance of deep neural networks to properties of the model family or regularization techniques. Specifically, they show that even state-of-the-art CNNs with regularization techniques can easily fit a random labeling of the training data or random noise data. This provides evidence for CNNs lacking inherent architectural choices or regularization techniques that benefit generalization. It also elucidates the fact that even state-of-the-art CNNs can be easily fooled to memorize the training set.