Adversarial Projected Gradient Descent

Thomas Hopkins and Andrew Fang

Paper

Title: ***Towards Deep Learning Models Resistant to Adversarial Attacks*** [1]

Authors

- Aleksander Madry, MIT, madry@mit.edu
- Aleksandar Makelov, MIT, amakelov@mit.edu
- Ludwig Schmidt, MIT, ludwigs@mit.edu
- Dimitris Tsipras, MIT, tsipras@mit.edu
- Adrian Vladu, MIT, avladu@mit.edu

Date & Venue

- Poster Paper at the 6th International Conference on Learning Representations (ICLR 2018)

Adversarial Machine Learning

Definition: *a technique that attempts to fool models by supplying deceptive input.*

- Why would someone want to fool a model?
- How big of a problem is this?
- What makes an input deceptive?
  - What are some techniques for finding deceptive inputs?
- What can we do to make our models more robust to deceptive inputs?

Reasons for Attack

The following are some examples for wanting to fool a machine learning model:

- Spam Filtering
  - Find a way to get your spam message through the spam filter
- Virus Protection
  - Finding a way to get a Trojan past virus protection
- Automatic Flagging Systems
  - Trying to get a video/post not get flagged as dangerous/misleading/etc
- Games
  - Trying to take advantage of AI behavior in games like chess

How big of a problem is this?

This largely depends on the choice of model and the problem it is attempting to solve. Empirically, it has been shown that even slight changes to an input can lead to large changes in output (especially for neural networks).

Bad news:

- Hard to characterize the full range of possible attacks (e.g. perceptual similarity of images)
- Models not trained to deal with adversaries are *very* susceptible to attack
- Neural networks are more sensitive to adversarial attacks than other models

Good news:

- There are methods for increasing robustness of machine learning models

What is an “adversarial attack”?

Slightly change the input in a specific way to “fool” the network into producing an output with high loss. From an optimization perspective, this amounts to finding a perturbation δ that maximizes the loss:

$$\underset{\delta \in \Delta}{\text{maximize}}\, \ell(h_\theta(x+\delta), y)$$

where h_θ is the learned model and Δ is any set of perturbations that is “close” to the original input x.

One common choice is: $\Delta = \{\delta : \|\delta\|_\infty \leq \epsilon\}$

This is the type of attack which we will explore further and is known as an ℓ_∞- bounded attack.

Projected Gradient Descent (PGD)

To solve the optimization problem, we can use projected (sub)gradient descent on the negative of the loss function. This takes the form

$$x^{t+1} = \Pi_{x+\Delta}\left(x^t + \alpha\, \mathrm{sgn}(\nabla_x L(\theta, x, y))\right)$$

Instead of computing the gradient w.r.t the model parameters θ, we compute it w.r.t the input x.

Since we are using an ℓ_∞-bounded attack, we take the sign of the gradient to normalize it. This makes the choice of step size α easier since we will always be projecting back onto the ℓ_∞-ball around x.

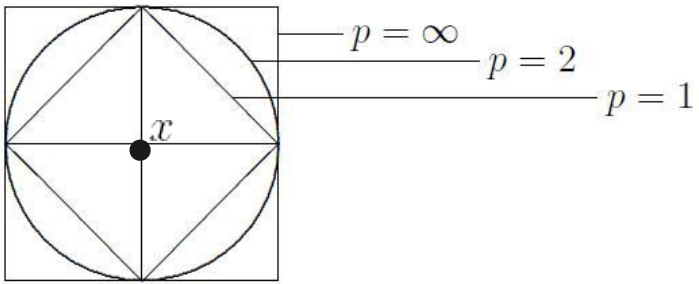
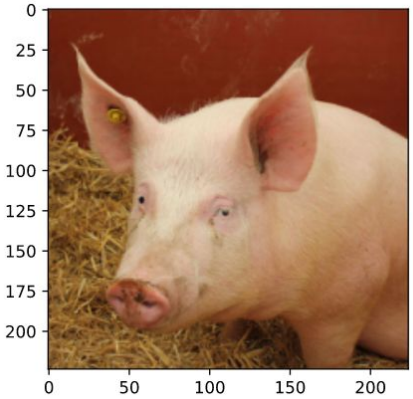


Figure. Different norm-balls around the input point x.

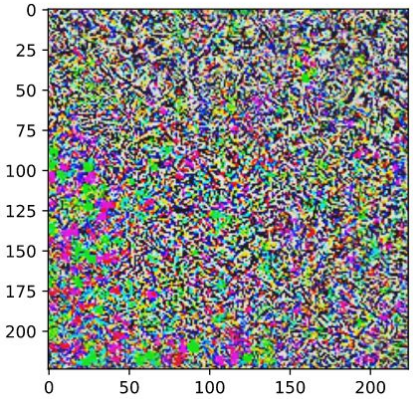
Source: https://math.stackexchange.com/questions/402679/can-we-visualize-the-closed-balls-for-the-space-l-infty-equipped-with-the

Example Attack



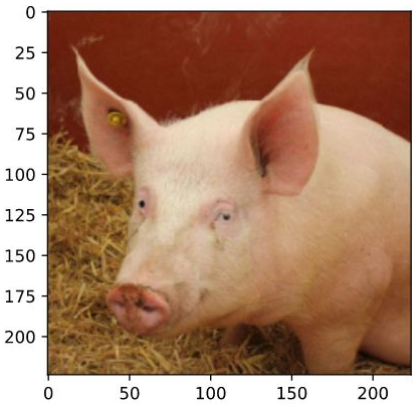
Predicted class: **hog**
Predicted probability: **0.996**
Actual class: **hog**

+



Perturbation found by PGD using ℓ_∞-bounded attack (scaled up by 50x)

=



Predicted class: **wombat**
Predicted probability: **0.9997**
Actual class: **hog**

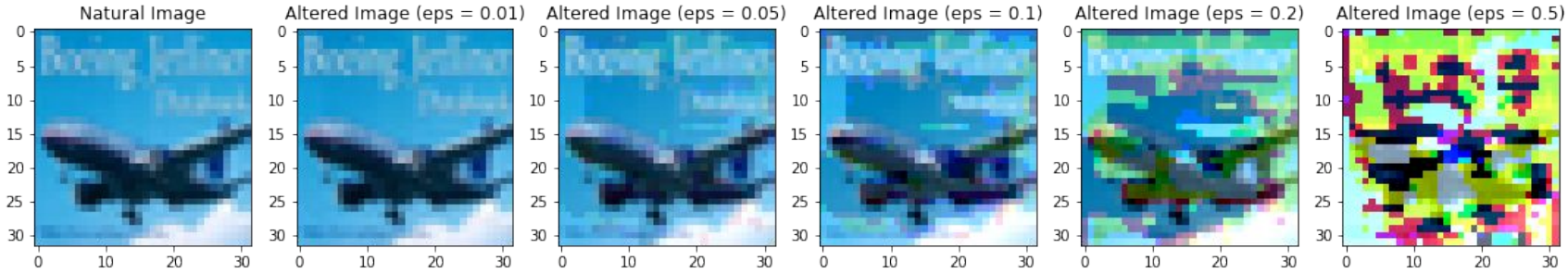
Actual wombat
(not from training set)

Example is from: https://adversarial-ml-tutorial.org/introduction/ [4]

Choice of ϵ = Strength of Attack

Choosing a large value for ϵ means that the allowed perturbation is greater. Now clearly, taken in the limit, an attacker can make an image appear like random noise. This is not the kind of attack we consider here. Instead, we care about attacks that change the image slightly (low ϵ) without changing the meaning of the image.

Here are a few example perturbations with different ϵ values on the same image:



How do we increase the robustness of our model?

From an optimization perspective, we need to minimize the expected maximum loss produced by an adversary over our data set:

$$\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\max_{\delta\in S} \ell(\theta, x+\delta, y)\right]$$

This is a saddle-point problem viewed as the composition of an inner maximization problem and an outer minimization problem. We already defined one method (PGD) for approximately solving the inner maximization. To approximately solve the outer minimization, we can simply train on inputs altered by PGD. This method works due to Danskin's Theorem which states that we must first solve the inner maximization then compute the gradient evaluated at this point. The problem with this is that we are only approximately solving the inner maximization problem so the theory does not technically hold. However, this method still works well in practice and the more iterations of PGD done for the inner maximization, the better gradient we will get for the outer minimization [4].

Computational Cost

Since we are now composing two optimization problems, we incur a much greater computational cost.

That is, for k epochs over the training set, n data points, and t iterations of PGD, we have $O(knt)$ gradient steps to make.

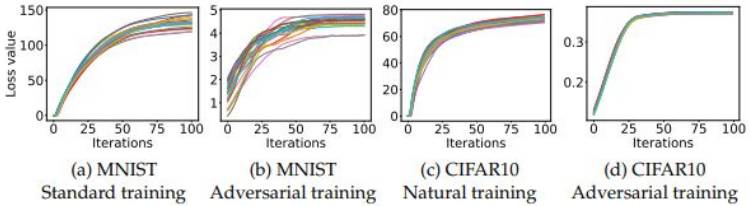
Each gradient step involves a forward pass over the network to compute the logits and a backward pass to compute the gradient with respect to either the network parameters θ (for learning) or the input x (for the attack).

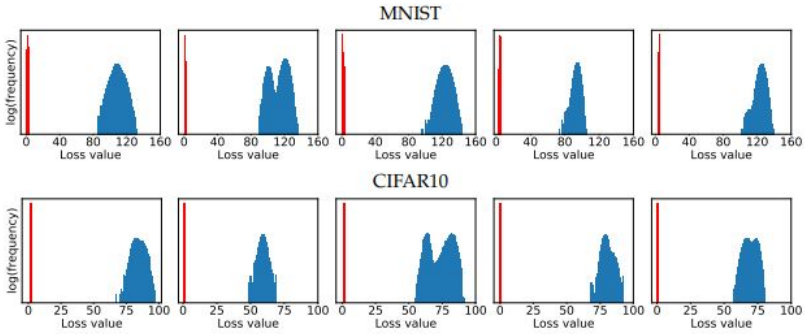
As we can see, adding PGD significantly increases the computational cost of training. For example, if a gradient step takes 1 second and we train for 10 epochs, on 10,000 data points, and for 10 iterations of PGD, we have:

Method	**Cost**
Training without PGD (Natural)	10 * 10,000 * 1 = 100,000s or 27.8 hours
Training with PGD (Adversarial)	10 * 10,000 * 10 * 1 = 1,000,000s or 278 hours or 11.6 days

Performance Guarantees

It can be shown experimentally that loss values for randomly chosen starting points, in $x + \Delta$ where x is a random point in the dataset and Δ is the given bounds (in this case ℓ_∞), increase in a fairly consistent way and plateau rapidly. In addition, we can observe that the loss of the final iterate of PGD over random starting points tend to be very concentrated around similar values. This suggests that PGD can operate as a "universal" adversary against first-order attacks. Regardless of the attack and the starting point, it will not find a significantly better maxima than through PGD. Indeed, the similar maxima of the final iterate suggests that the maximum loss of other first-order attacks should not be significantly different. [1]





Adversarial training is in red, natural training is in blue

Empirical Analysis

The original paper only experiments on MNIST and CIFAR-10, both tasks that involve predicting 10 classes. We experiment using CIFAR-100, which predicts 100 classes, and CIFAR-10 [2]. We observe the different levels of robustness we can achieve for each data set, using a fixed ϵ of 0.035.

Furthermore, we analyze the gap in robustness when the strength of the adversary changes from defense to attack. For example, if we train within a bound $\epsilon = 0.01$ while an attacker uses a much stronger $\epsilon = 0.1$ bounded attack, we should not expect our model to be very robust.

The goal is to see how robust a classifier we can train on two different datasets and what degree of robustness we get based on assumptions we make about our adversary during training.

Experimental Methods

We use the LeNet5 architecture for our experiments [3]. In the table to the right is our experimental setup. We train and evaluate the model on both natural images and adversarially perturbed (by PGD) images. We use the same number of iterations and ℓ_∞-bound in all cases.

Robustness is determined by adversarial evaluation accuracy. A “good” model will have high robustness without having to sacrifice too much natural evaluation accuracy.

We perform these experiments on both CIFAR-10 and CIFAR-100.

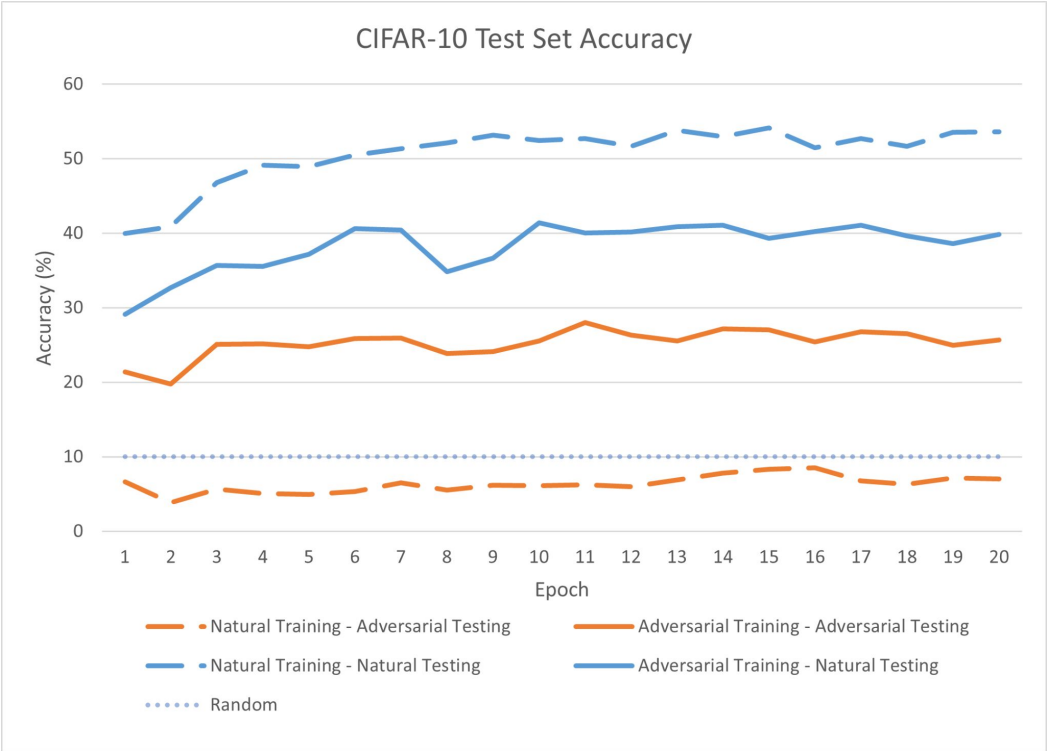
Training Type	Evaluation Type
Natural	Natural
Natural	Adversarial
Adversarial	Natural
Adversarial	Adversarial

CIFAR-10 Accuracy

Here are the accuracy results after training for 20 epochs on the CIFAR-10 data set.

Some observations:

- Natural Training produces the best overall performance but is *very* susceptible to attack
- Adversarial Training does significantly reduce the Natural Testing Accuracy but is much more robust to similar strength attacks
- There is no crossover between any of these methods/evaluations throughout training

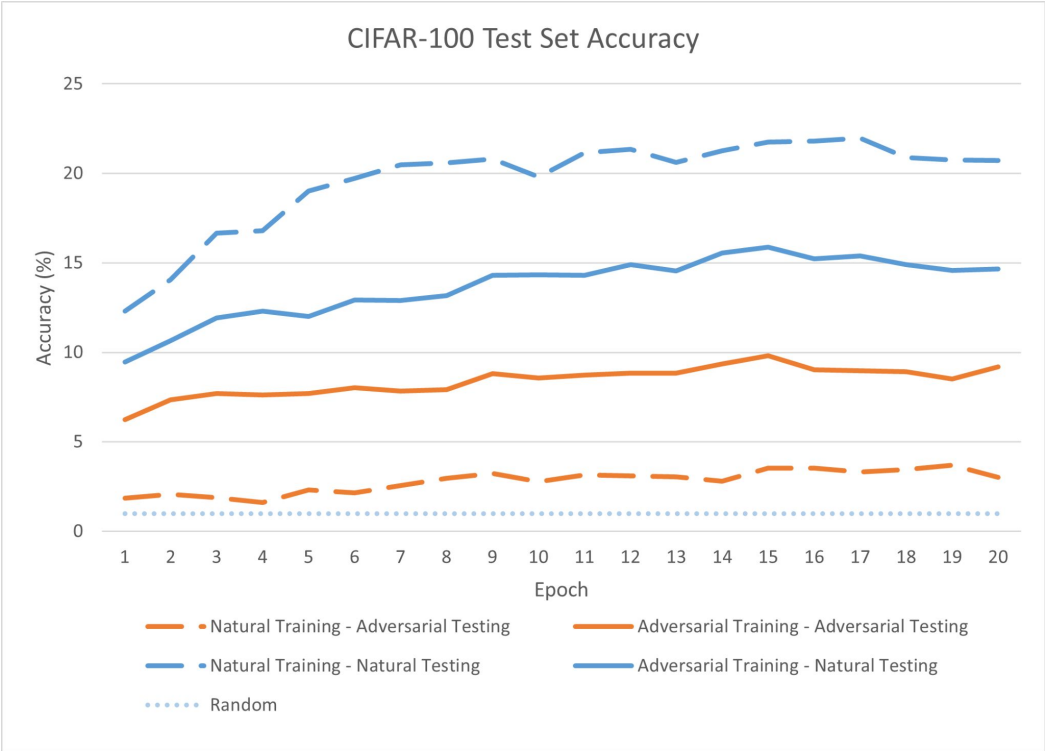


CIFAR-100 Accuracy

Here are the accuracy results after training for 20 epochs on the CIFAR-100 data set.

Some observations:

- Similar to results on CIFAR-10
- Natural Training robustness increases minimally throughout training
- Larger increases in accuracy on the Natural Testing set than the Adversarial Testing set.

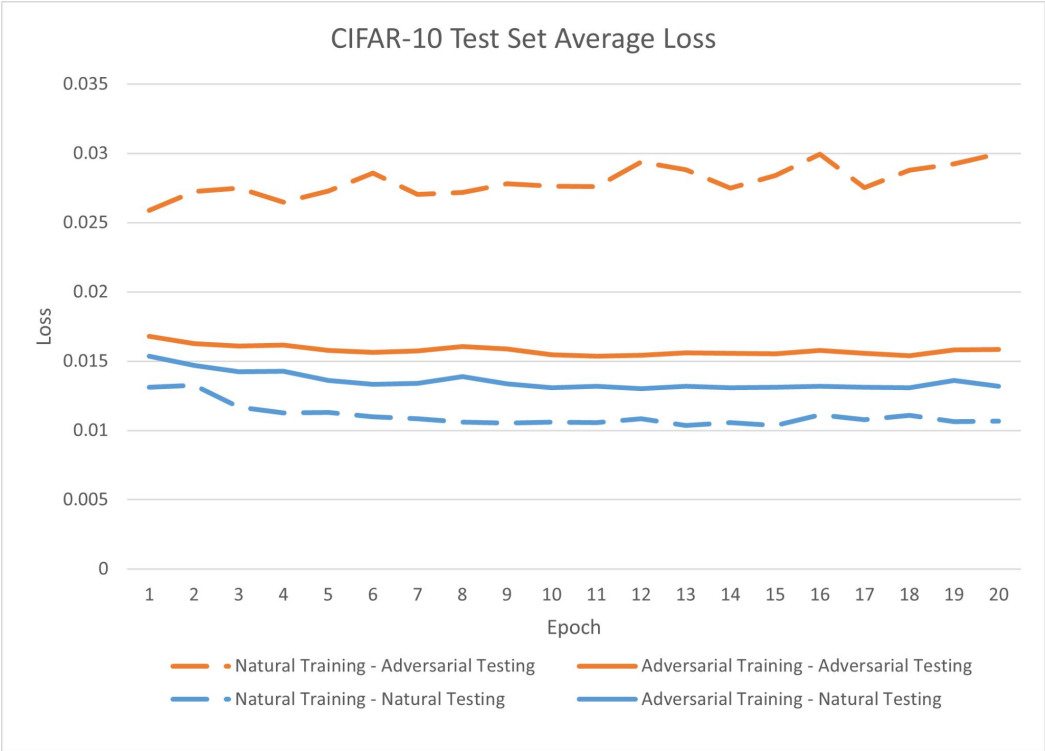


CIFAR-10 Average Loss

Here are the average loss (over individual images) results after training for 20 epochs on the CIFAR-10 data set.

Some observations:

- Natural Training does not yield any decrease in Adversarial Testing Loss
- Adversarial Testing Loss is always greater than Natural Testing Loss
- Adversarial Training has a more stable decrease in average loss.

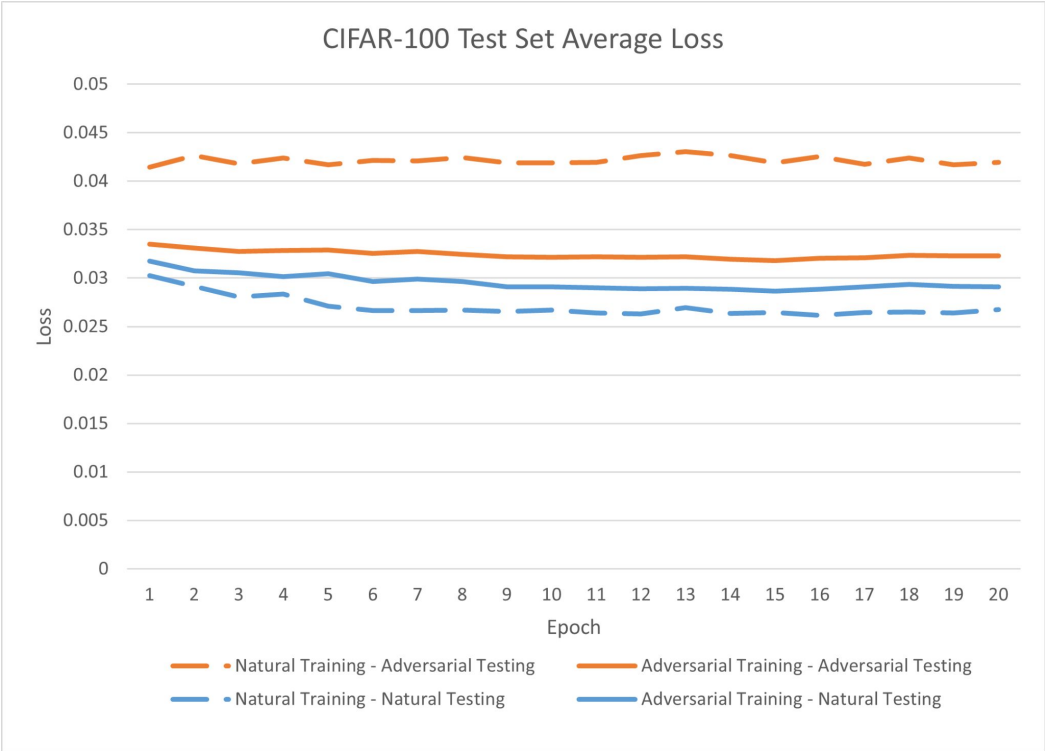


CIFAR-100 Average Loss

Here are the average loss (over individual images) results after training for 20 epochs on the CIFAR-100 data set.

Some observations:

- Similar to results on CIFAR-10
- Average Loss does not improve that much in 20 epochs, most likely due to architecture choice
- There is no crossover in any of these methods/evaluations throughout training



Discussion

Clearly, there exists a tradeoff between choosing to train your model with or without adversarial inputs. The first is that you should expect a decrease in *natural accuracy* but a significant increase in *robustness*. The second is that training on adversarial inputs takes much more computational effort.

These results lead to another question:

> *Can we decrease the strength of our adversary during training to get **better natural accuracy** and **maintain robustness**?*

In the previous experiment, the strength of the adversary during training was the same during testing.

> *What if this isn't the case?*

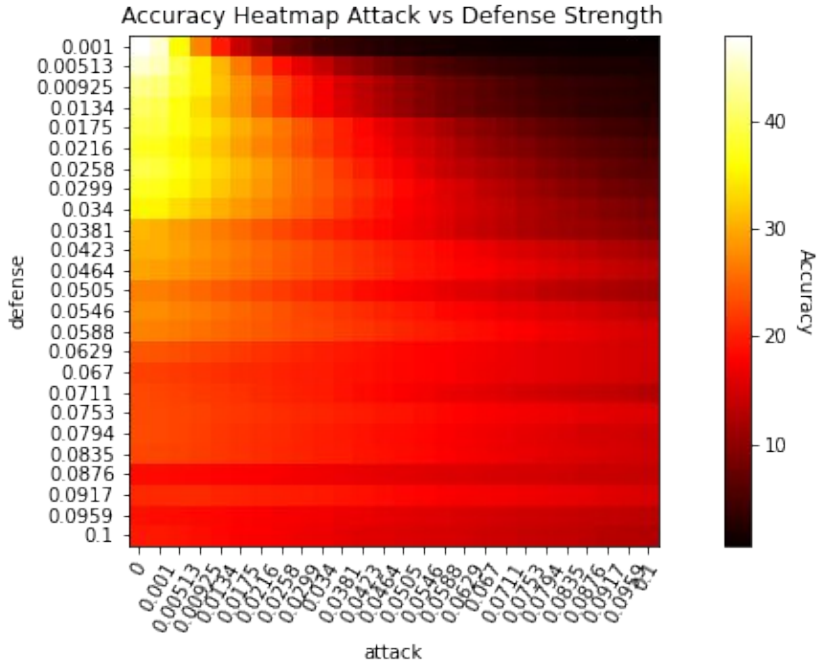
Next, we explore how testing accuracy and average loss change with respect to varying the strength of the adversary between *defense* (training) and *attack* (testing).

CIFAR-10 Attack vs Defense Strength: Accuracy

Here is a heatmap of attack vs defense strength testing accuracy for different values of ϵ.

Some observations:

- Decreasing the strength of adversary during training increases natural accuracy (left-most column)
- Stronger defense leads to being more robust against even stronger attackers
- A good choice of ϵ here might be 0.034 since it maintains good natural accuracy and is robust against adversaries up to around 0.05

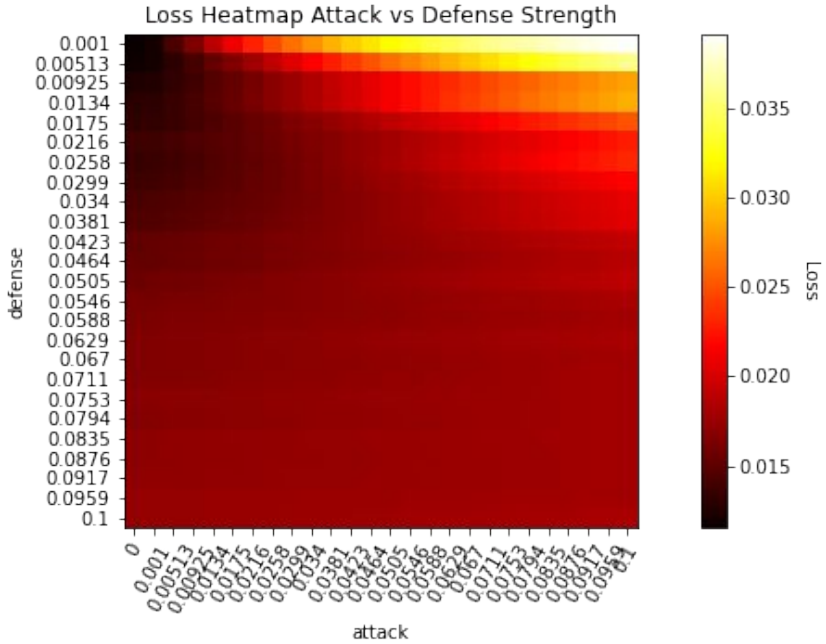


CIFAR-10 Attack vs Defense Strength: Average Loss

Here is a heatmap of attack vs defense strength average testing loss for different values of ϵ.

Some observations:

- Opposite of the accuracy plot (as expected)
- Increasing defense strength leads to stable average loss against all kinds of attacks
- The best loss occurs in the top-left where both attack and defense are weak



Future Work

There are still several areas of future work that we can expand on.

For one, we use the ℓ_∞-bound for all cases. Exploration into other definitions of perceptual similarity are needed. For example, a combination of ℓ_∞ and domain-specific heuristics may provide more robust models.

Further investigation into methods for determining how best to decide ϵ (strength of adversary) to train on. This will most likely be domain-specific.

Lastly, looking at how robustness changes across neural network architectures (size, shape, non-linearities, etc.) could lead to more robust architectures.

References

[1] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. (2017). Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083.

[2] Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images.

[3] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. Neural computation, 1(4), 541-551.

[4] Tutorial on Adversarial Machine Learning: https://adversarial-ml-tutorial.org/

[5] Code Repository: https://github.com/thomashopkins32/PGDAdversarialLearning