CS 182/282: TEST-TIME SEARCH METHODS FOR ADVERSARIAL ROBUSTNESS

EVYN MACHI ZITAO FANG KIAN ZOHOURY

1 Introduction

Adversarial attacks and defenses for computer vision models have been widely studied and remain a great interest for research. Informally, adversarial attacks are targeted examples where an adversary has changed very little about the example yet computer vision models classify them wrong. A good deal of work has formally defined such an adversarial attack in terms of bounded infinity-norm adversaries. In this paper, we adopt this formal definition and focus on adversarial defenses. To this end, we will use the Tiny ImageNet dataset and compare different baseline models using test-time data augmentation, as well as different test-time adversaries. Many defenses proposed have focused on training-time methods to train robust models against different adversaries (such as FGSM or PGD). We propose a novel *test-time* method that modestly protects models with or without adversarial training. We also propose minor improvements to PGD training. In sum, we believe that while adversarial training is surely the best way to achieve robustness, test-time methods are scantly explored and provide non-negligible benefits to improve vision systems overall.

2 Literature Survey/Related Work

Goodfellow et al. (2014) first introduced the *Generative Adversarial Network* (GAN), which aimed to train a robust image classifier, through a joint optimization problem represented as a minimax game between an adversarial generator model, and a discriminator that attempts to discriminate between true versus noise-generated images [1]. Huang et al. (2016) proposed a stronger adversarial training method that focused on classification error as the main objective to minimize. Adversarial examples were generated by adding a small perturbation — namely, the gradient of the network derived by linear approximation assumption — to the clean samples, making the adversarial training more inherent to the classifier [2]. However, due to the nature of the minimax paradigm, where both minimization and maximization are combined, the optimization problem often exhibits a saddle point — the non-convexity results in high empirical risk against strong adversarial attacks. To address this, Madry et al. (2019) introduced a way of escaping these saddle points by utilizing *Projected Gradient Descent* (PGD) [3]. By constraining the optimization problem to a strictly convex set, the adversary can iteratively travel in the direction that maximizes classification error. During training, the weights of the classifier will update to reflect these attacks, building a stronger "first-order" defense.

3 Background

(Definition) Let $x \in \mathbb{R}^d$ be an image classified as C_i. An *adversarial attack* applies a function $F : \mathbb{R}^d \to \mathbb{R}^d$

$$x^{adv} = F(x)$$

to x, where x^{adv} will now be misclassified as C_j.

(Definition) The *maximum attack* an adversary can inflict on any sample is arbitrarily limited,

$$||x - x^{adv}||_\infty \leq \epsilon$$

where ϵ is the maximum size of perturbation along any pixel dimension. However, an optimal ϵ is the smallest degree of perturbation that successfully tricks the classifier. For our case, we chose $\epsilon = 0.0225$.

3.1 Adversaries

Given an image set $\mathcal{D}$, an adversary solves the following optimization problem

$$F^* = \max_F \mathcal{L}(F(\mathcal{D}))$$

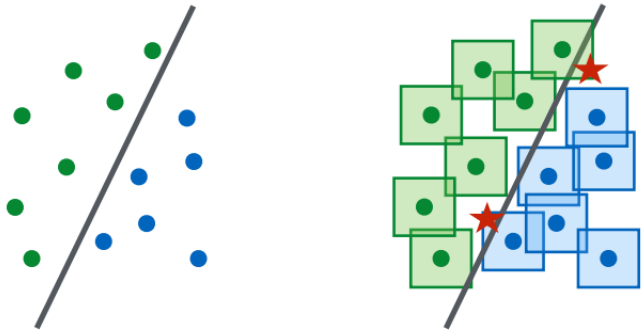


Figure 1: Decision Boundary and l_∞ Sphere, from [3]

where the optimization is over all perturbations F.

An FGSM Adversary approximates this optimization as

$$F^*(x) \approx x + \epsilon \cdot \text{sgn}(\nabla \mathcal{L}(x, y))$$

A PGD Adversary approximates this optimization as

$$F^*(x) \approx \Pi_k \left[x \leftarrow x + \frac{\epsilon}{k} \cdot \text{sgn}(\nabla \mathcal{L}(x, y)) \right]$$

where Π_k indicates k-steps of gradient descent. In this report, we work with $k = 10$.

Note that many popular adversarial attack methods are gradient-based. Attacker will add a carefully chosen tensor to the image, where the tensor is in the direction of the gradient of another class and is restricted in terms of their norm (for example, infinity-norm). This method works if the original images are close to the decision boundary, and the tensor added will bring the point across the boundary. The figure above demonstrates this, where green and blue points are the images in the training set, and the stars are the attack images generated by pushing toward the other side of the boundary while staying in the corresponding l_∞ sphere.

Adversarial attacks can be transferred between models, therefore the decision boundary is not overfitted. From this, we conjecture that the decision boundary between classes and the probability landscape are smooth.

4 Methods/Approach

4.1 Base model

We used a pretrained VGG16 model with batch normalization from the torchvision package as our base model architecture. We utilized the already learned features of the VGG16 model, freezing the convolution layers, but built our own classification layers. Our classifier was a simple yet effective 3-layer feed-forward neural network (intermediate dimension size 4096), with dropout(p=0.5) and ReLU activation layers in between. Although the pretrained model was trained on larger images, the VGG16's adaptive pooling mechanism handled the smaller images from the Tiny ImageNet dataset seamlessly. The images were centered, scaled, and transformed (through a random series of horizontal flips, rotations, and padding/cropping) before being fed into the network. The weights of the classifier were optimized using the Adam optimizer (with weight decay, lr = 0.0001) along with cross-entropy loss.

4.2 Random Search Defense

We utilize the method of test-time random search in addition to regular adversarial training.

The goal of the random search is to recover the perturbation tensor and try to reverse it. The original image will be used to correct our classification result of the input image. We guess a norm that the attacker might use, then generate i (of our choice) random tensors and add them to the test images. We then ask the model to classify them.

From then, we propose two variants. The simple method will add the average value of the classification with a chosen coefficient to the output logit and apply softmax as usual.

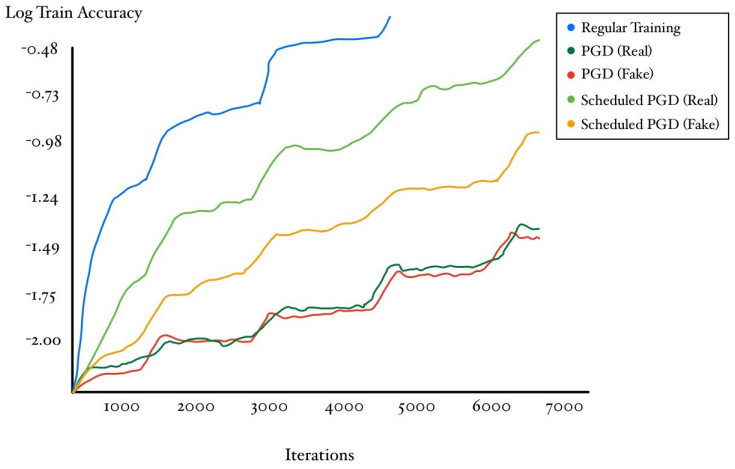
For the second variant we also choose k classes with the highest average logit across the samples, weighted-average the tensor on their probability of these classes and produce k new tensors. These tensors represents inverses of the possible perturbation tensors if the original image is of one of the classes we picked previously. We add the k new tensors to the class and classify them again, then add the output logit of the corresponding class (with a coefficient) to the logit output of our original image.

4.3 PGD Training Optimization

Traditionally, PGD training proceeds similarly to FGSM training, i.e. half the examples are perturbed using PGD and then passed to the classifier. The PGD adversary is generally known beforehand with fixed epsilon and fixed steps (for example 10 steps). Recall the goal of PGD training is to make the classification boundaries more robust. However, we've found that introducing such a boundary with an untrained classifier makes training relatively slow.

We were able to significantly improve training time with a scheduled PGD adversary. While epsilon is always fixed, we let the number of steps taken by the PGD adversary vary by epoch. In particular, we let the number of steps taken by the PGD adversary be equal to the epoch number (e.g. at epoch 1, PGD takes 1 step which is equivalent to FGSM, and at epoch 10 PGD takes 10 steps).

Figure 2: PGD Training Accuracy Over Time



4.4 Trial & Error

Originally, we tried several different ideas. This included implementing a form of GAN training wherein an adversary tries to trick the classifier while still fooling a discriminator. Similar to that of GAN training all three networks–the adversary, the discriminator, and the classifier–would all be trained together. This would allow a general adversary to do whatever they want to perturb a picture (rotate, blur, etc.) without being bound to the ϵ definition above (subject to the contraint that they also fool the discriminator). Unfortunately, the GAN training was not stable and attempts at regularization and stability were fruitless. Perhaps a better attempt at stabilization could make this idea successful.

5 Results

5.1 Baseline Results

We trained three baseline models: a model that was trained with only the vanilla dataset, a model that was adversarially trained with FGSM ($\epsilon = 0.0225$), and a model that was trained with PGD ($\epsilon = 0.0225$, $k = 10$). We then validated all models on the vanilla dataset, an FGSM adversary, and a PGD adversary (where all examples were perturbed). The results are below.

Table 1: Baseline Results

Model Name	Dataset	FGSM	PGD
VGG	0.6207	0.2864	0.0019
FGSM_VGG	0.6125	0.4011	0.1403
PGD_VGG	0.5402	0.4309	0.3582

5.2 Random Search Results

We observed that our random search method improves the performance of FGSM adversarially trainned model on PGD perturbed data, but there are no significant impact on other results.

Table 2: Random Search ($\alpha = 0.5$, branch $= 10$)

Model Name	Dataset	FGSM	PGD
VGG	0.6109	0.2753	0.03
FGSM_VGG	0.6118	0.3917	0.2372
PGD_VGG	0.5314	0.4221	0.3519

Tuning hyper-parameters we defined did not produce any meaningful impact on the performance.

We also experimented with the additional queries after random search method. Our result indicates that applying second round of model queries will only improve the accuracy for about 0.7%.

Table 3: Random Search with Additional Queries($\alpha = 0.5$, branch $= 10$, $k = 3$)

Model Name	Dataset	FGSM	PGD
VGG	0.6124	0.028	0.034
FGS_VGG	0.6149	0.3942	0.2341
PGD_VGG	0.5395	0.4269	0.3563

6 Conclusion

PGD Adversaries are routinely seen as a robust universal adversary as they tend to dramatically decrease accuracy, moreso than its FGSM counterpart. In this report we proposed a test-time defense against adversarial images and found that it works particularly well to improve performance against a PGD Adversary from disastrous (0.0019) to terrible (0.03) in the case of the vanilla model, and from terrible (0.1403) to bad (0.2372) in the case of the FGSM model. Notably it works even when classifiers have not been adversarially trained against PGD. While this is only a modest improvement, it does show the potential for test-time methods in general to improve a model's performance outside of adversarial training.

7 Team Contributions

Zitao Fang worked on early exploration of transfer learning with pretrained models. He also proposed and help implement the random search method.

Evyn Machi worked on early exploration of possible adversarial training methods, implemented the adversarial methods (FGSM, PGD), and implemented PGD optimization. He also helped implement and run random search.

Kian Zohoury helped choose and implement the final vanilla classifier through cross-validation, implemented an early randomized defense strategy [4], created a Google Colab notebook for easier training/testing, added to the preprocessing pipeline, and fixed bugs in validation script.

References

[1] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[2] Ruitong Huang et al. *Learning with a Strong Adversary*. 2016. arXiv: 1511.03034 [cs.LG].

[3] Aleksander Madry et al. *Towards Deep Learning Models Resistant to Adversarial Attacks*. 2019. arXiv: 1706.06083 [stat.ML].

[4] Cihang Xie et al. "Mitigating Adversarial Effects Through Randomization". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=Sk9yuql0Z.