# Assignment 3
# Training robust Neural Networks

**Team :** Art-Attack

December 11, 2024

**Authors :**
**Sabina Askerova; Emmarius DELAR; Marwa NAIR**

## 1 Introduction

Building neural network classifiers that remain reliable under adversarial perturbations is a central challenge in robust machine learning. In this report, we explore strategies to defend against adversarial inputs on the CIFAR-10 dataset. We start with two foundational attack methods—FGSM [2] and PGD [3]—to perform adversarial training and establish initial baselines. We then implement Boosted Adversarial Training [5] to leverage randomization techniques, and finally explore the DeepFool attack [4], both as a new attack and a defense mechanism using adversarial training.

## 2 Baselines

Our approach involves first training a standard convolutional neural network classifier on CIFAR-10 without any adversarial robustness measures and then evaluating its vulnerability to adversarial attacks. Building upon this initial setup, we introduced two well-known gradient-based attacks whose target function for an $\epsilon$-bounded attack is

$$\max_{\|\delta\|\leq\epsilon} \ell_f(x+\delta, y)$$

- **FGSM [2]** : We implemented it for the $\ell_\infty$ where we have $\delta^* = \epsilon \operatorname{sign}\left(\nabla_x \ell_f\left(x_t, y\right)\right)$
- **PGD [3]**: Which is an iterative version of FGSM that starts with $x_0 = x$ and performs updates as follows: $x_{t+1} = \Pi_{B(x_0,\epsilon)}\left(x_t + \delta \operatorname{sign}\left(\nabla_x \ell_f\left(x_t, y\right)\right)\right)$, with  representing the projection operato rand $B\left(x_0, \epsilon\right)$ the hyperball centered in $x_0$ with radius $\epsilon$

The attacks are implemented in the functions *pgd_attack* and *fgsm_attack* from the **utils_ADV.py** code source.

To improve the robustness of our standard model, we implemented Adversarial Training (AT) using both attacks. Specifically, at each training step, we generated adversarial examples using FGSM/PGD and trained the model on these examples. The adversarial training processes are implemented in the functions *train_adversarial_fgsm* and *train_adversarial_pgd* within **utils_ADV.py**.

The resulting adversarially trained baseline models are saved in the **models** directory under the names **AT_FGSM.pth** and **AT_PGD.pth**, corresponding to FGSM-based and PGD-based adversarial training, respectively. These models serve as baselines for comparing more advanced defense mechanisms.

## 3 Randomization matters : Boosted Adversarial Training

As a different way, Boosted Adversarial Training (BAT) [5] uses a zero-sum game theory approach between two players: a class of classifiers $\mathcal{H}$ and a class of attacks $\Phi_\epsilon$.

### 3.1 Principle of BAT

By a Classifier point of view, we try to give the better response for any attacks. This game define as sup-inf problem such as :

$$\inf_{h\in\mathcal{H}} \sup_{\phi\in(\Phi_\epsilon)} \mathcal{R}_{adv}(h, \phi)$$

where

$$\mathcal{R}_{adv}(h,\phi) := E_{Y\sim\nu}\left[E_{X\sim\phi_Y\#\mu_Y}\left[1\{h(X)\neq Y\}\right]\right]$$

design the risk to a classifier to be fooled. Assume the computational capacity is limited, it is necessary to add some penalizer to efficiently utilize the computational resources. First, by reducing the number of useless image attacks by defining:

$$\Omega_{\text{mass}}(\phi) := E_{Y\sim\nu}\left[E_{X\sim\mu_Y}\left[1\{X\neq\phi_Y(X)\}\right]\right].$$

and by limit high variation of pixel modification :

$$\Omega_{\text{norm}}(\phi) := E_{Y\sim\nu}\left[E_{X\sim\mu_Y}\left[\|X-\phi_Y(X)\|_2\right]\right].$$

where $\nu$ is the probability measure that defines the law of the random variable Y, and $\mu_Y$ the conditional law of $X|(Y=y)$.

However, Pinot et al. demonstrate that no Nash equilibrium exists in a deterministic setup. To address this limitation, randomization has been employed as a mixture of classifiers.

Next, the algorithm proposed for Boosted Adversarial Training is defined as follows:

**Algorithm 1** Boosted Adversarial Training [5]

```
1: **Input:** $D$ the training data set and $\alpha$ the weight parameter.
2: Create and adversarially train $h_1$ on $D$
3: Generate the adversarial data set $\tilde{D}$ against $h_1$
4: Create and naturally train $h_2$ on $\tilde{D}$
5: $q \leftarrow (1-\alpha, \alpha)$
6: $h \leftarrow (1-\alpha) * h_1 + (\alpha) * h_2$
7: **Return** $h$
```

## 3.2 BAT's implementation

Starting with the same architecture as our first Adversarial Training model (Sec.2), we train successively the classifiers $h_1$ and $h_2$. First, $h_1$ with a classical adversarial method under both attack mechanisms PGD and FGSM. For each, we generate a perturbed dataset using respectively the functions *pgd_attack* and *fgsm_attack* from the "**utils_ADV.py**" code source, to train the second classifier $h_2$ in the classical way. The training function and the test function are implemented in the script "**utils_BAT.py**". We denote each model as **BAT + the approach used to generate perturbed data**. All the models we have trained are located in the directory "**models**". Here is a summary table 1:

| Model | $h_1$ | $h_2$ |
|---|---|---|
| BAT + PGD | `default_model_BAT_PGD.pth` | `default_model_c2_BAT_PGD.pth` |
| BAT + FGSM ($\epsilon = 0.1$) | `default_model_BAT_FGSM.pth` | `default_model_c2_BAT_FGSM.pth` |
| BAT + FGSM ($\epsilon = 0.6$) | `default_model_BAT_FGSM_e06.pth` | `default_model_c2_BAT_FGSM_e06.pth` |

Table 1: Location of saved classifiers.

**Remark:** Note that we define $\epsilon$ by grid-search and using classical litterature value mentionned by [5], so after all we retain $\epsilon = 0.1$ and $\epsilon = 0.6$ especially fort the BAT + FGSM methods who give a little improvement for natural accuracy but not necessarily for accuracy under attack.

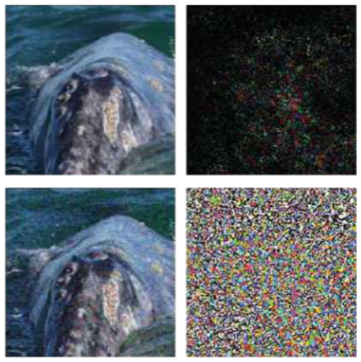

Figure 1: First row: perturbations introduced by DeepFool. Second row: perturbations introduced by FGSM. Excerpt from [4]

## 4 DeepFool attack mechanism and training

While FGSM and PGD attacks are pretty fast while still having good performance on fooling a classification network, they introduce a substantial amount of noise to the images, even though they are unnoticable for a human eye.

It was interesting to analyze attack mechanisms that introduce smaller perturbations because models aversarially trained with simpler attacks like FGSM could potentially be less effective in face of fine-grained attacks.

### 4.1 DeepFool attack

DeepFool attack mechanism was introduced in order to inject minimal perturbations to the image required for it to be misclassified. It is a general attack method designed to iteratively perturbate inputs until they cross the decision boundary of the classifier. This is a reasonable assumption for small perturbations in neural networks, including CNNs. The authors of the DeepFool paper have found that the perturbations made with their method were more accurate and efficient compared to FGSM.

Figure 1 shows noise generated for image perturbation by DeepFool and FGSM attacks.

The DeepFool algorithm aims to find the minimal adversarial perturbation required to alter the classification of an input image. The classifier is defined as $f : R^n \to R^c$, where $f$ is the multiclass classifier function, $n$ is the dimensionality of the input image, and $c$ is the number of classes. The classification rule is given by $\hat{k}(x) = \arg\max_k f_k(x)$, where $\hat{k}(x)$ is the predicted class label for the input image $x$, and $f_k(x)$ is the output of the classifier for class $k$.

The DeepFool algorithm uses an iterative approach to compute the minimal perturbation. At each iteration $i$, the algorithm linearizes the classifier around the current point $x_i$ and approximates the decision boundary as a polyhedron $\tilde{P}_i$. The minimal perturbation $r_i$ at iteration $i$ is computed as the vector that projects the current point $x_i$ onto the closest face of the polyhedron $\tilde{P}_i$. This involves calculating the distance between $x_i$ and each hyperplane defining the faces of $\tilde{P}_i$, and selecting the hyperplane with the smallest distance. The perturbation is given by $r_i = \frac{|f'_{\hat{l}}|}{\|w'_{\hat{l}}\|_2^2} w'_{\hat{l}}$, where $\hat{l}$ is the index of the closest hyperplane, and $f'_{\hat{l}}$ and $w'_{\hat{l}}$ are derived from the linearized classifier at $x_i$.

The perturbed image at iteration $i+1$ is updated as $x_{i+1} = x_i + r_i$. This process is repeated until the perturbed image $x_{i+1}$ is classified differently from the original image $x_0$. The final perturbation vector $\hat{r}$ is the sum of all perturbations computed during the iterations: $\hat{r} = \sum_i r_i$.

**Algorithm 2** DeepFool: multi-class case [4]

```
Require: Image $\boldsymbol{x}$, classifier $f$.
Ensure: Perturbation $\hat{\boldsymbol{r}}$.
 1: Initialize $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}$, $i \leftarrow 0$.
 2: while $\hat{k}(\boldsymbol{x}_i) = \hat{k}(\boldsymbol{x}_0)$ do
 3:     for $k \neq \hat{k}(\boldsymbol{x}_0)$ do
 4:         $\boldsymbol{w}'_k \leftarrow \nabla f_k(\boldsymbol{x}_i) - \nabla f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_i)$
 5:         $f'_k \leftarrow f_k(\boldsymbol{x}_i) - f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_i)$
 6:     end for
 7:     $\hat{l} \leftarrow_{k \neq \hat{k}(\boldsymbol{x}_0)} \frac{|f'_k|}{\|\boldsymbol{w}'_k\|_2}$
 8:     $\boldsymbol{r}_i \leftarrow \frac{|f'_{\hat{l}}|}{\|\boldsymbol{w}'_{\hat{l}}\|_2^2} \boldsymbol{w}'_{\hat{l}}$
 9:     $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + \boldsymbol{r}_i$
10:     $i \leftarrow i + 1$
11: end while
12: return $\hat{\boldsymbol{r}} = \sum_i \boldsymbol{r}_i$
```

The overshoot parameter in the DeepFool algorithm is used to ensure the perturbed image crosses the classification boundary. By slightly pushing the perturbed image past the decision boundary, the model may learn to be more robust to smaller perturbations. This is because the model encounters examples that are more confidently misclassified during training. We wanted to check how the efficiency of the attack would change with different overshoot parameters. The overshoot parameter should be much lower than 1. In our experiments we used overshoot parameter values of 0.01, 0.02, 0.06 and 0.1.

According to our local metrics, we managed to decrease the accuracy of different adversarial models on the whole validation dataset. You can see the natural accuracy results obtained locally using auto-attack [1] in the table

| **Model** | **nat accuracy** | **nat accuracy for perturbed image set** |
|---|---|---|
| default | 78.125 | 18.55 |
| BAT+PGD | 66.503 | 58.89 |
| BAT+FGSM | 58.203 | 37.01 |
| AT+DeepFool (overshoot of 0.02) | 75.407 | 37.5 |
| AT+FGSM | 72.143 | 35.742 |
| AT+PGD | 46.875 | 18.65 |

Table 2: DeepFool attack vs. our models (tested locally)

We see that our models are more robust to deepfool attacks than the default model without adversarial training.

## 4.2 DeepFool adversarial training

The authors of the DeeFool paper discuss how augmenting training data with adversarial examples crafted by DeepFool can significantly increase the robustness of models. They experimentally demonstrate this by fine-tuning pre-trained networks on datasets augmented with DeepFool-generated adversarial examples and observing improvements in robustness. We also wanted to check the impact this parameter would have on the performance of a model adversarially trained with DeepFool attack.

We can see that the overshoot parameter did not have significant influence on the natural accuracy of the model. However, there was some difference in PGD L and PGD L2 accuracies. The results obtained locally

| **Overshoot $\eta$** | **nat accuracy** | **PGD L accuracy** | **PGD L2 accuracy** | **agg** |
|---|---|---|---|---|
| 0.02 | 76.953125 | 0.0 | 1.93 | 1.93 |
| 0.01 | 78.90625 | 0.00 | 3.89 | 3.89 |
| 0.06 | 77.734375 | 0.00 | 3.11 | 3.11 |
| 0.1 | 78.90625 | 0.01 | 3.04 | 3.05 |

Table 3: Performance of AT+DeepFool attack for different overshoot parametervalues. Evaluated on a local machine.

might not correspond to those on the platform. These result did not showcase better accuracy scores than FGSM.

## 5 Results

In this section, we report the performance of our models. Table 1 summarizes the results obtained on the test platform.

| Model | nat accuracy | pgdling accuracy | pgdl2 accuracy | agg | time |
|---|---|---|---|---|---|
| Standard Training | 56.25 | 6.03 | 25.15 | 31.18 | 153.46 |
| AT + PGD | 43.75 | 24.53 | 30.01 | 54.54 | 137.6s |
| AT + FGSM | 63.75 | 15.92 | 17.85 | 33.77 | 444.51 |
| BAT + PGD | 65 | 49.83 | 57.05 | 106.88 | 780s |
| BAT + FGSM | 63.75 | 50.03 | 57.79 | 107.81 | 247.3s |

Table 4: Results on the test platform

The "standard training" model refers to a model trained with no defense mechanism. As expected, it is highly susceptible to adversarial perturbations, achieving low robustness under both PGD $\ell_\infty$ and $\ell_2$ attacks. It will, however, serve as a reference for comparison. Introducing adversarial training (AT) improves robustness. Specifically, AT with FGSM shows some gains in robustness against the PGD $\ell_2$ attack. In contrast, AT with PGD trades off a lower net accuracy for better robustness to both PGD perturbations compared to the standard model.

Boosted Adversarial Training (BAT) further enhances robustness by incorporating randomization. BAT with PGD outperforms AT with PGD, achieving a higher net accuracy and showing a marked improvement in robustness against both $\ell_\infty$ and $\ell_2$ attacks. Similarly, BAT with FGSM shows an increase in robustness over AT with FGSM, while maintaining comparable net accuracy. Notably, BAT with FGSM achieves this improvement with significantly reduced training time compared to BAT with PGD.

Overall, BAT demonstrates a more effective balance between robustness and accuracy, outperforming standard adversarial training and in both robustness and computational efficiency.

While theoretically promising for minimal perturbation generation. Our implementation of DeepFool adversarial training did not yield significant improvements. Local experiments showed limited effectiveness in terms of PGD L-$\infty$. and PGD L2 metrics, compared to FGSM-based approaches. We conjecture that the DeepFool method's focus on minimal perturbations could inadvertently create adversarial examples that are too subtle to effectively challenge the neural network's decision boundaries, unlike FGSM's more aggressive perturbation strategy.

**Remark 1:** All of these methods use the same classifier architecture, trained with the same number of epochs (10), the same batch size (32), the same dataset, and the same $\epsilon = 0.1$ when required.

## References

[1] fra31. GitHub - fra31/auto-attack: Code relative to "Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks" — github.com. `https://github.com/fra31/auto-attack`.

[2] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[3] Aleksander Madry. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[4] Fawzi A. Frossard P. Moosavi-Dezfooli, S. Deepfool: a simple and accurate method to fool deep neural networks. *https://openaccess.thecvf.com/content*$_cvpr_2016/html/Moosavi-Dezfooli_DeepFool_ASimple_CVPR_2016_paper.html$, 2016.

[5] Rafael Pinot, Raphael Ettedgui, Geovani Rizk, Yann Chevaleyre, and Jamal Atif. Randomization matters. how to defend against strong adversarial attacks. *https://arxiv.org/abs/2002.11565*, 2020.