Dauphine | PSL
UNIVERSITÉ PARIS

Project Report for Assignement 3

Training Robust Neural Networks

Ryan Belhkir, Emma Covili & Léonard de la Seiglière
– *Titans* –
Benjamin Negrevergne & Laurent Meunier

December 20, 2021

Contents

1 Introduction

Our goal is to study the impact of attack and defense mechanisms on neural networks. To do that, we implemented a neural network and attacked it with two different attacks: FGSM and PGD. We also tried the Carlini & Wagner attack.
After that we tried a defense mechanism on our network : the adversarial training. Later on, we tried the randomized smoothing.
We mainly focused on defense mechanisms since it seemed to be the main focus of the project. Especially because the platform allowed us to compare our networks' resistance.

2 Methods

Firstly, we implemented the FGSM and PGD attacks. We also wanted to try and implement Carlini & Wagner attack, but it took too much time and was not really interesting in the context of this project. Secondly, we implemented the adversarial training, the randomized smoothing and the local winner takes all methods.

2.1 Attacks

2.1.1 FGSM: Fast Gradient Sign Method

One of the simplest yet very efficient method of generating adversarial perturbations. The perturbations are calculated as: $\eta = \epsilon sign(\nabla_x L_\theta(x, y))$.

2.1.2 PGD: Projected Gradient Descent

The projected gradient descent (PGD) method is an iterative version of FGSM running a much finer optimization. It's a more powerful attack compared to FGSM. The PGD attack performs FGSM with a small step size α and projects the updated adversarial sample learned from each iteration onto the ℓ_∞-ball around the original input sample. The iteration of PGD is attack is defined as follows:

$$x_{t+1} = \Pi_{B(0,\epsilon)}(x_t + \eta sign(\nabla_x L_\theta(x, y))$$

2.2 Defenses

2.2.1 Adversarial training

We first used adversarial training as seen in class. We trained our networks on images attacked by PGDℓ_∞ as it turned out to be the best attack to use according to our tests. With this method we minimized the following formula :

$$\min_\theta \mathbb{E}_{(X,Y)}(\max_{\|\tau\| \leq \epsilon} L_\theta(x + \tau, y))$$

2.2.2 Randomized Smoothing

For this method, we found the explanations in the paper *Certified Adversarial Robustness via Randomized Smoothing* (Cohen et al., 2019) [1]. It explains that the advantage of a smoothed classifier is that it is certifiably robust in ℓ_2 norm.
Given a trained ("base") classifier f and an input x, the objective of adversarial robustness is to ensure that an adversary cannot change the classification of x with an imperceptibly small perturbation of x. For randomized smoothing, instead of directly certifying f, we define a new, "smoothed" classifier g whose prediction is the majority vote of f applied to x convolved with some noise distribution p. Formally, for a set of classes $\mathcal{C}$,

$$g(x) = argmax_{c \in \mathcal{C}} \mathbb{P}(f(x + \epsilon) = c)$$

where f outputs a distribution over classes and the subscript indicates indexing. Unlike f, this new classifier g will have provably smooth decision boundaries.
As a result, they are able to guarantee that the output of g will not change within a certain radius R of the input x, where the magnitude of this radius is a function of the margin by which the majority vote wins: the larger the margin, the larger the certified radius. For example, when classifying images, R could be the ℓ_2-norm of a pixel perturbation.
As the code was given, we used it, modified it a bit for it to fit our expectations, (for instance some of it was deprecated and we updated it) and made our own experiments with it. However, we were not able to push it onto the test platform.

2.2.3 Local Winner Takes All (LWTA)

Finally after extensive research, we decided to implement a paper that just came out. Indeed during the project, the paper *Stochastic Local Winner-Takes-All Networks Enable Profound Adversarial Robustness* (Panousis et al., 2021) [2] was published showing excellent results. It is thanks to this network that we were able to finish first on the platform and get similar results locally. But before we talk about our results, let's see how it works.

In this paper, they present a new activation function and a new neural network block. The main idea is that in each LWTA block there is only one neuron that is activated. It is drawn with a probability depending on the activation weight (the higher the weight the more likely they are to be drawn).

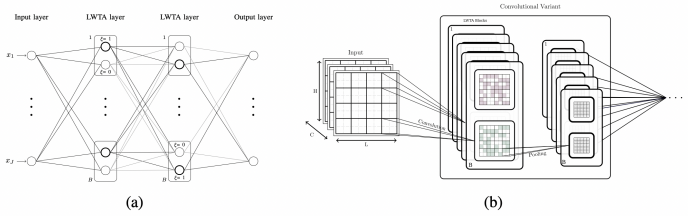

Figure 1: Representation of the activation of a LWTA block

As can be seen in figure 1, only one neuron per LWTA block is activated (the one with bold contour). In the same way, within a convolutional LWTA block, the same position

can only be activated once among the feature maps of the same block.

As we randomly select the winner at each step, different subpaths, stochastically emerges even for the same input, which essentially obstruct the adversary from successfully attacking the model.

After the implementation, we trained the model using adversarial training to reinforce this natural robustness by maximising the loss describe in the paper :

$$L = -\sum_{X_i, Y_i \in \mathcal{D}} CE(Y_i, f(X_i; \hat{\xi})) - \mathrm{KL}(q(\xi)||p(\xi))$$

with ξ the latent variables of the LWTA blocks.

3 Results

3.1 Adversarial training

We tried several networks with several parameters and here are the results we obtained on the one we selected: ResNet18.

As we can see on the plot, we also tried Finetuning on networks already trained on

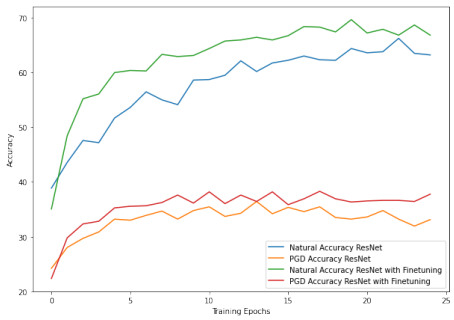

Figure 2: ResNet Accuracy with Adversarial Training and Finetuning on CIFAR-10.

ImageNet and noticed that it increases a little the speed and quality of convergence on both natural and adversarial accuracy.

3.2 Randomized Smoothing

Here are the results we obtained with the randomized smoothing method. Disclaimer : here our results are those of corrupted data not of FGSM or PGD attacks.
We applied randomized smoothing to CIFAR-10, we trained several smoothed classifiers, each with a different σ. Our base classifier was a 50-layer residual network.

model - acc	train	test
$\sigma = 0.25$	82.3%	76.1%
$\sigma = 0.25$	67%	65.2%
$\sigma = 0.25$	48.9%	48.4%

Table 1: Table of the train and test accuracies of models trained with σ corrputed data

Here are the plots of the training and testing losses for models trained with σ corrputed data:

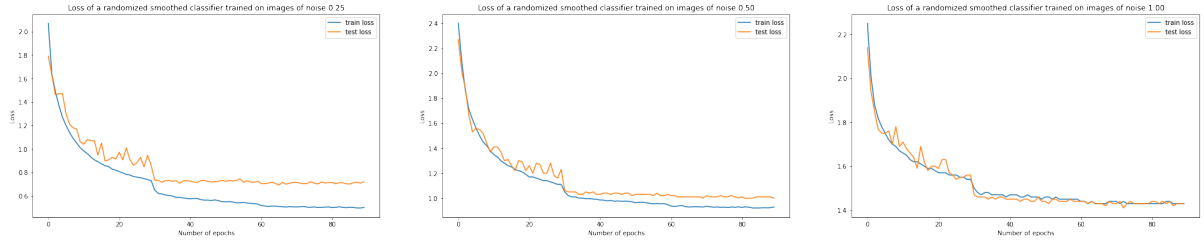

Table 2: Train and test losses for models trained with $\sigma = \{0.25, 0.50, 1.00\}$ corrupted data

	$r = 0.25$	$r = 0.5$	$r = 0.75$	$r = 1.0$	$r = 1.25$	$r = 1.5$
$\sigma = 0.25$	**0.55**	0.40	0.26	0.00	0.00	0.00
$\sigma = 0.50$	0.52	**0.43**	**0.30**	0.19	0.15	0.13
$\sigma = 1.00$	0.39	0.33	0.28	**0.23**	**0.19**	**0.18**

Table 3: Approximated certified accuracies by randomized smoothing on CIFAR-10

In table 3, each row is a setting of the hyperparameter σ, each column is an ℓ_2 radius. The entry of the best σ for each radius is bolded. For comparison, random guessing would attain 0.1 accuracy.

Figure 3 plots the certified accuracy attained by smoothing with each σ.
We see that σ controls a robustness/accuracy tradeoff. When σ is low, small radii can be certified with high accuracy, but large radii cannot be certified. When σ is high, larger radii can be certified, but smaller radii are certified at a lower accuracy.
In table 4 $\sigma = 0.25$ and $\alpha = 0.001$, the Monte Carlo probaility for which the algorithm returns a good result.
Each column shows the fraction of test examples which ended up in one of five categories; the prediction at x is "correct" if PREDICT returned the true label, while the prediction is "accurate" if PREDICT returned $g(x)$. Computing $g(x)$ exactly is not possible, so in order to determine whether PREDICT was accurate, we took the gold standard to be the top class over $n = 100,000$ Monte Carlo samples.

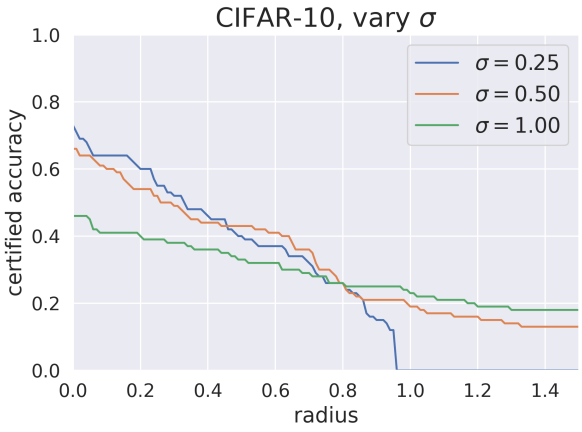

Figure 3: Approximate certified accuracy attained by randomized smoothing on CIFAR-10.

n	correct, accurate	correct, inaccurate	incorrect, accurate	incorrect, inaccurate	abstain
100	0.55	0.00	0.18	0.00	0.27
1000	0.69	0.00	0.25	0.00	0.06
10000	0.72	0.00	0.27	0.00	0.01

Table 4: Performance of PREDICT as n, the number of samples for the Monte Carlo algorithms, varies

3.3 Local Winner Takes All (LWTA)

Finally, as you might expect, we got our best results with our implementation of the LWTA Wide Resnet as described in the article. As we were nearing the end of the project when we discovered this article we couldn't do all the testing we wanted to, such as the impact of the width on our results. We use a LWTA WideResNet34 with width $w = 1$ and trained it using SGD with a learning rate 0.1.

Accuracy	Natural	FGSM	PGD ℓ_∞	PGD ℓ_2
LWTA WideResNet Ours	78.94	73.10	70.23	71.02
LWTA WideResNet pretrained+training	**86.63**	**84.15**	**81.38**	**82.44**
LWTA WideResNet (platform)	98.75	-	85.06	86.31

Table 5: Accuracy on LWTA WideResNet on CIFAR-10

Note that since this is a stochastic network on the same test set the results change a little from one launch to another

Although there were some problems with the results on the platform, we can see that even locally we got much better results than all the articles we could find. Obviously our best results were obtained with the weights already trained that we re-trained on

our attacks. We think that we could have obtained similar results with our network with more training.

4 Conclusion

This project allowed us to discover and learn more about robustness of neural networks and its significance. We discovered methods to attack and defend networks, and had the chance to test our implementations against the rest of the class.
We understood that well designed attacks could easily fool any vanilla networks and it explained the need for defense mechanisms. However, we realized that, apart from adversarial training, defense mechanisms were not easily built and required lots of work in order for them to be efficient.
Finally, during our research we were given the impression that data augmentation was the best way to go, although we considered it more of a trick than a real solution. The discovery of LWTA neural networks has really strengthened our belief that this is not the right solution and we sincerely believe that stochastic neural networks have a lot of potential in this area. That's why if we could have continued to work on this project we would have liked to go deeper into this subject.

References

[1] J. M. Cohen, E. Rosenfeld, and J. Z. Kolter, "Certified adversarial robustness via randomized smoothing," *CoRR*, vol. abs/1902.02918, 2019.

[2] K. P. Panousis, S. Chatzis, and S. Theodoridis, "Stochastic local winner-takes-all networks enable profound adversarial robustness," *CoRR*, vol. abs/2112.02671, 2021.

[3] A. Shafahi, M. Najibi, A. Ghiasi, Z. Xu, J. P. Dickerson, C. Studer, L. S. Davis, G. Taylor, and T. Goldstein, "Adversarial training for free!," *CoRR*, vol. abs/1904.12843, 2019.

[4] S. Zagoruyko and N. Komodakis, "Wide residual networks," *CoRR*, vol. abs/1605.07146, 2016.

[5] N. Carlini and D. A. Wagner, "Towards evaluating the robustness of neural networks," *CoRR*, vol. abs/1608.04644, 2016.

[6] L. Rice, E. Wong, and J. Z. Kolter, "Overfitting in adversarially robust deep learning," *CoRR*, vol. abs/2002.11569, 2020.

[7] H. Zhang, Y. Yu, J. Jiao, E. P. Xing, L. E. Ghaoui, and M. I. Jordan, "Theoretically principled trade-off between robustness and accuracy," *CoRR*, vol. abs/1901.08573, 2019.