Spear and Shield: both of them are awful

包清泉-519030910402

1 Introduction

Though DNN (Deep Neural Network) has achieved great success in object detection, classification and e.t.c., vulnerability was found where small perturbations hard to identify by humans can easily fool the model. In this report, we will try to attack 6 given models and propose our own defense model. We will also show much failing work that are quite interesting though.

2 Nomenclature

表 1: Notations

Notations	Description
x	Data sample
x'	Adversarial data sample
y	Ground truth of the data
ϵ	Maximum perturbations we allow
$\|\cdot\|_p$	l_p norm
θ	Parameters of the DNN model
$\mathscr{L}(\cdot)$	Loss function for training DNN
$C(\cdot)$	Classifier outputing a label
$F(\cdot)$	Model outputing a score vector
$Z(\cdot)$	Last layer outputs before softmax,*i.e.*, $F(x) = \text{Softmax}(Z(x))$
$B(x)$	the ϵ-neighbors of x
$\Pi_{B(x)}(\cdot)$	Projection function to the $B(x)$

3 Attack

3.1 Overview of attacks

In this report, we only consider the white-box and non-target attacks, *i.e.*, the attacker knows the architecture and parameters of the model and does not care the changed labels' class. To attack a white-box model, two optimization problems are mainly proposed. Equation 1 was formulated first to attack DNN models. L-BFGS attack and CW attack [2] are based on this.

$$\begin{aligned} \min \quad & \|x' - x\|_p \\ \text{s.t.} \quad & C(x') = t \text{ and } x' \in [0,1]^m \end{aligned} \tag{1}$$

Equation 2 shows another kind of approach which does the opposite optimization in contrast to the model training and elevates the loss between the adversarial data sample x' and its label y conditioned on the perturbation constraints. This optimization is often tackled with gradient-based methods such as Fast Gradient Sign Method [3] and PGD [5]. Most state-of-the-art attacks are based on gradients and our work either.

$$\begin{aligned} \min \quad & \mathscr{L}(\theta, x', y) \\ \text{s.t.} \quad & \|x' - x\|_p \leq \epsilon \text{ and } x' \in [0,1]^m \end{aligned} \tag{2}$$

A classic PGD is shown in algorithm 1, and most of our discussions and experiments are based on it.

Algorithm 1: PGD

Data: data x and their labels y

Parameters: perturb step size η, perturb nums N, random coefficient α

Initialize: $noise \sim \mathscr{N}(0,1)$

$x' \leftarrow x + \alpha * noise$

for $n \leftarrow 1$ **to** N **do**

  $x' \leftarrow x' + \eta * \nabla_{x'} \mathscr{L}(x', y)$

end

return x'

3.2 The choice of the loss function

Cross-entropy loss is widely used as training loss function in multi-classification tasks, which is formulated in Equation 3,

$$\text{CrossEntropy}(F(\mathbf{x}), \mathbf{y}) = -\sum_{i=1}^{M} y_i \log(F(x)_i), \tag{3}$$

where M is the dimension of $\mathbf{x}, \mathbf{y}$. Actually it is the simplified version to optimize the KL-divergence between $\mathbf{x}, \mathbf{y}$ since the entropy of $\mathbf{y}$ can be ignored in the process of optimization.

Another loss function widely used is margin loss which evaluates the difference between the logits in the ground truth and the maximal one in non-truth classes as Equation 4 shows,

$$\text{MarginLoss}(x', y) = \max_{i \neq y} Z(x')_i - Z(x')_y. \tag{4}$$

In the later experiment outcome, we will find that Margin Loss in general cases perform much better than Cross-entropy Loss.

3.3 Output Diversified Sampling

Tashiro *et al.* proposed ODS (Output Diversified Sampling) to better set the start point of PGD [6]. Rather than randomly sampling in the image space, ODS uniformly picks directions in the logits' space and then get the corespondent start point via gradients descent as Equation 5 shows,

$$x_{k+1} = \text{Proj}_{B(x_{org})}(x_k + \eta_{ODI} \text{sign}(v_{ODS}(x_k, f, w_d))) \tag{5}$$

and the gradient direction is based on the inner product between the random fixed direction and logits of the start point, *i.e.*,

$$v_{ODS}(x_k, f, w_d) = \frac{\nabla_x (w_d^T f(x))}{\|\nabla_x (w_d^T f(x))\|_2} \tag{6}$$

, where w_d is uniformly sampled over $[-1, 1]^C$.

3.4 Failed Approach

3.4.1 Langevin Dynamics Sampling

Suppose we have a function f and initialize a random point x_0, we iterate x_t by Equation 7

$$x_{t+1} = x_t - \eta \nabla f(x_t) + \sqrt{2\eta}\xi_t \tag{7}$$

where random variable ξ_t follows a standard Gaussian distribution $\mathcal{N}(0, 1)$ and η is the learning rate. We know the conclusion that $\{x_t\}$ will converges to a distribution $p(x)$ in which $\forall x \in \mathbb{R}, p(x) = \frac{e^{-f(x)}}{\sum_x e^{-f(x)}}$. If we view $f(x)$ as the output likelihood of the correct class or the negative loss, we can sample those 'bad' points with high probability.

We have tried to define $f(x)$ as

- $-1 * \mathscr{L}(\theta, x', y)$
- $1/\mathscr{L}(\theta, x', y)$

with $\mathscr{L}(\cdot)$ being cross-entropy loss and margin loss respectively, however, the outcomes are far worse than the naïve PGD and therefore we do not show the results here. We guess that the points fooling the models are sparse in the feasible constrained space and the sampling will always hit outside the feasible boundary such that this approach does not work.

Also, we have tried to sample a logits' direction via this approach to replace the random sampling in ODS. We suppose that this will fasten the sampling success rate, however, this sampling does not work either.

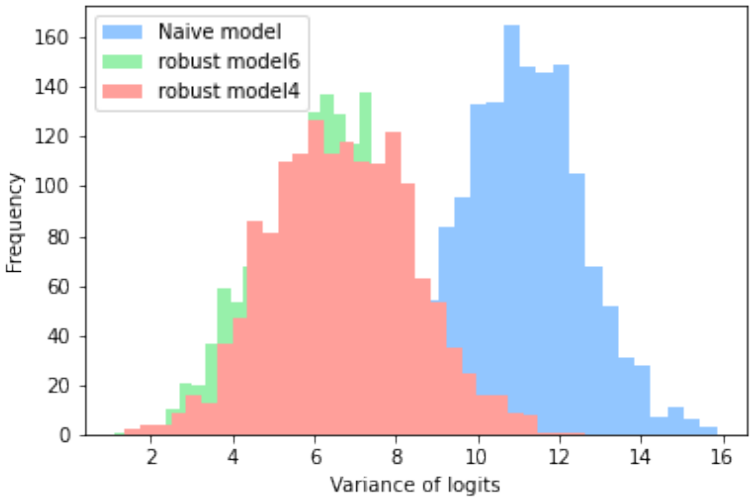

图 1: Variance of logits in various models

3.4.2 Entropy Direction Sampling

Instead of sampling randomly in the logits space, we intuitively suppose that those direction enlarging the entropy of the logits will help finding better start point in the image space. However, the results manifest hindrance of naïve PGD about 1%. We later research on the distribution of logits shown in Figure 1 and discover that robust models have already boast smaller variance in logits space, *i.e.*, the logits may be smooth and well-distributed enough such that the entropy of them is high enough and our approach is likely to fail.

3.5 Attack obfuscated model

Athalye *et al.* point out three types of obfuscated gradients where the model does not *actually* learn from the adversarial samples [1]. In our model zoo, model 3 is obfuscated from the fact that iterative attacks show no improvement and increasing distortion does not increase success rate(we set ϵ from 0.03 to 0.09 and no improvement is viewed).

We propose a mixed loss function to tackle it, *i.e.*, we define the loss function as Equation 8,

$$\mathscr{L}_{mix}(x', y) = \mathscr{L}(x', y) + \beta \cdot \text{Entropy}(Z(x')), \tag{8}$$

where $\mathscr{L}(x', y)$ is the traditional loss function such as cross-entropy loss or margin loss and β is a decaying coefficient in terms of the iteration number in PGD. In real implementation, we set $\beta = \frac{1}{1+i}$ with PGD iteration index i.

Figure 2 shows the experiment results. We set perturb step size 0.01, batch size 300 with two GTX-2080Ti to run the experiment. We can see that the mixed loss indeed performs better than naïve PGD with or without random start, though it is worse than PGD+ODI. But the benefit of this method is obvious that it does not add extra calculation costs in contrast with the PGD+ODI takes much longer time.

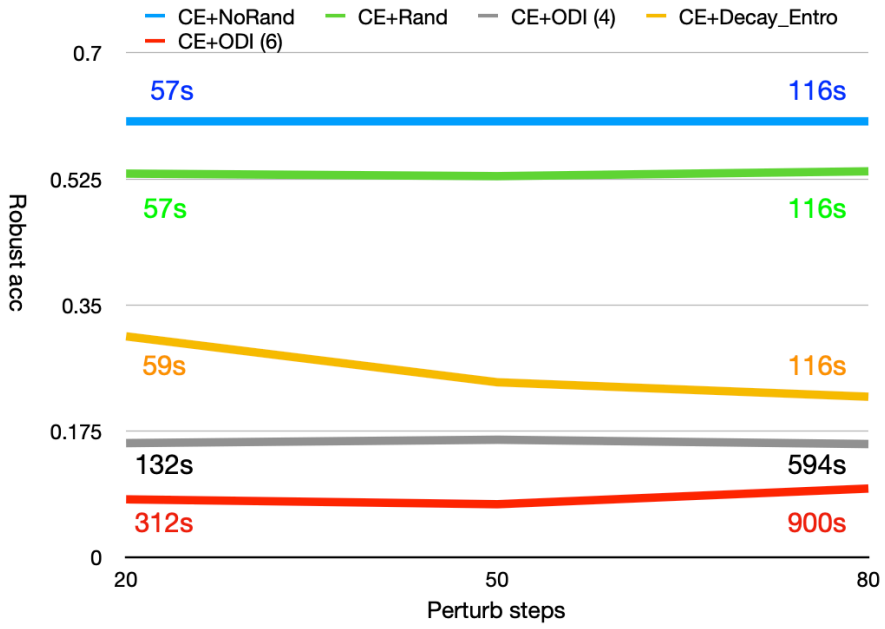

图 2: Robust accuracy on model3 with various attacks. The CE+ODI(4) means implement PGD+ODI 4 times with Cross Entropy Loss and get the accumulated results, similarly for CE+ODI(6).

Also, we run the attack on other models, and the outcomes show the mixed loss function do NOT reduce the attack performance much in naïve PGD. Out of the clearness of the report, we only show the outcome in PGD+Margin, the results in PGD+CE is similar and we skip it. So this kind of criterion do much help in tackling gradient obfuscation without losing the performance in other general tasks.

3.6 Other trivial hyper-parameter

There are some trivial experiment facts. Firstly, perturb steps have limited impact on the result. In general, the decrease of the robust accuracy is not evident though with large increment of perturbation steps. Secondly, rather than the default setting of 0.007, perturb step size can be larger such as 0.01, and the convergence speed can be much faster. Thirdly, the random start is quite important for PGD. With random start, there are much improvement just like shown in Figure 2.

3.7 Our Final Solution

We set perturb steps 20, step size 0.01, random start coefficient 0.01, and batch size 300 with two GTX-2080Ti to run the experiment. Besides, we set ODI steps as 10 with step size0.02, ODI restart number as 4. We put some typical attack method in Table 2. We can derive such conclusions,

- Margin Loss in general performs better than CrossEntropy Loss
- Mix Loss mentioned in 3.5 can tackle the obfuscation of gradients without hindering its original performance
- ODI can improve a little but cost much computation resources and time.

Since TA does not requires time efficiency, we submit our attack method as PGD(40) + Margin + ODI(6).

表 2: Part outcomes of various attacks

Model name	Natural Acc	Robust Acc			
		PGD20+CE	PGD+Margin	PGD+MixMargin	PGD+Margin+ODI(4)
Model1	0.9429	0.0000	0.0001	0.0003	0.0000
Model2	0.8302	0.5019	0.4917	0.4918	**0.4716**
Model3	0.8033	0.6049	0.6020	0.3235	**0.2283**
Model4	0.8492	0.5528	0.5385	0.5386	**0.5310**
Model5	0.8143	0.5329	0.5383	0.5382	**0.5168**
Model6	0.8825	0.6377	0.6092	0.6090	**0.6036**
Forward steps		20	20	20	120
Backward steps		20	20	20	120
Maximal Time (Model4)		195s	193s	198s	1259s

4 Defense

4.1 Overview of the defense model

Many approaches have been proposed to defend the attacks such as . Among them, adversarial training is shown to be the most effective [1]. Adversarial training solves an min-max optimization problems as 9 shows,

$$\min_{\theta} \rho(\theta) \triangleq \frac{1}{n} \sum_{i=1}^{n} \max_{\|x_i' - x_i\|_p \leq \epsilon} \mathscr{L}(f_\theta(x_i'), y_i), \tag{9}$$

where θ is the parameters of DNN model and n is the number of training samples.

TRADES trades off the accuracy and robustness [9] , *i.e.*,

$$\rho^{\text{TRADES}}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \left(\text{CrossEntropy}(f_\theta(x_i), y_i) + \beta \max \text{KL}(f_\theta(x_i) || f_\theta(x_i')) \right), \tag{10}$$

where KL is the Kullback-Leibler divergence and β is the hyper-parameter to control the trade-offs.

AWP explicitly regularizes the weight loss landscape and therefore brings a flatter weight landscape [8]. The combination of TRADES and AWP is shown in algorithm 2, which is our mainstay of our discussion and experiments.

Algorithm 2: TRADES+AWP

Data: data x and their labels y
Parameters: perturb step size η_1, AWP step size η_2, learning rate η_3, perturb nums N, random coefficient α
Initialize: $noise \sim \mathcal{N}(0,1)$
for $i = 1,...,m$ *(in parallel)* **do**
  $x_i' = x_i + \alpha * \mathcal{N}(0,1)$
  for $n = 1,...,N$ **do**
    $x_i' \leftarrow \Pi_\epsilon(x_i' + \eta_1 \text{sign}(\nabla_{x_i'} \text{KL}(f_\theta(x_i) || f_\theta(x_i'))$
  end
  for $k = 1,...,K$ **do**
    $\mathbf{v} \leftarrow \Pi_\gamma \left(\mathbf{v} + \eta_2 \|\theta\| \frac{\nabla_{\mathbf{v}} \sum_i \mathscr{L}(f_{\theta+\mathbf{v}}(x_i', y))/m}{\|\nabla_{\mathbf{v}} \sum_i \mathscr{L}(f_{\theta+\mathbf{v}}(x_i', y))/m\|} \right)$
  end
  $\theta \leftarrow (\theta + \mathbf{v}) - \eta_3 \nabla_{\theta+\mathbf{v}} \frac{1}{m} \sum_i \mathscr{L}(f_{\theta+\mathbf{v}}(x_i'), y) - \mathbf{v}$
end

We choose batch size 512, learning rate 0.1, and PGD20+CE attack with perturb steps 10, perturb step size 0.07, epsilon 8.0/255 in two GTX-2080Ti to run the experiments. Out of clarity, we only show part of the results. Firstly, AT costs much higher computation resources even for the small neural network ResNet18; secondly, AWP indeed diminishes the vulnerability to the attack; thirdly, in our some unshown trial, TRADES + AWP can increase the robustness such as in WideResNet28, but here we see it does not work in ResNet18 (we do not know the reason yet); fourthly, for some big models like WideResNet (here we take WideResNet34 as an example) the computation costs is quite large. In one training process, we train WideResNet34 for about 60 hours with only 0.01 increases in last 30 hours!

4.2 An Observation of Robust model

Table 4 shows the leap from top-one accuracy to top-three accuracy. Firstly, it is natural that natural accuracy increases not much since they are already high. However, the increases of robust accuracy in those robust

表 3: Part outcomes of AT training

ResNet18 Training Mode	Natural Acc	Robust Acc	Time
Normal	0.9396	0.0006	0.38h
AT	0.8311	0.5145	8.33h
AT + AWP	0.8218	0.5355	11.7h
AT + TRADES + AWP	0.7969	0.5122	13.4h
WideResNet28 + TRADES + AWP	0.8485	0.5558	15.6h
WideResNet34 + TRADES +AWP	0.8498	0.5887	47h

model (model2 and model4-6) and non-robust model (model1 and model3) are quite different. For those robust models, above 25% increases are seen, much more than 20% of the stochastic elevation; by contrast, those non-robust models have limited increases in robust accuracy.

So what does that mean? On one hand, it may show that robust models have learned some robust features such that attacks cannot completely 'fool' them and top-3 accuracy are better than stochastic guess; on the other hand, if we calculate Robust Top3 Acc – Robust Top1 Acc over Natural Top3 Acc – Natural Top1 Acc, we will find the rank of their quotients surprisingly matches the rank of their robustness! We temporarily cannot figure out why this coincidence happens, but this occurs to us that we may elevate the robustness of the robust model by introducing a small, simple but quite robust model to strengthen their ability!

表 4: Top1 and Top3 accuracy in different models

Model name	Natural Acc	Robust Acc	Natural Top3 Acc	Robust Top3 Acc	$\frac{\Delta \text{RAcc}}{\Delta \text{NACC}}$
Model1	0.9429	0.0000	0.9933	0.0799	1.585
Model2	0.8302	0.5019	0.9658	0.8298	2.418
Model3	0.8033	0.6049	0.8819	0.6626	0.734
Model4	0.8492	0.5528	0.9675	0.8471	2.488
Model5	0.8143	0.5329	0.9514	0.8767	2.508
Model6	0.8825	0.6377	0.9743	0.8825	**2.623**

4.3 Our Failed Approach

4.3.1 NN Mix Simple Model

From the insight in subsection 4.2, we try to introduce several simple models in logits space. We have tried PCA + KNN, PCA + GMM and PCA + MeansDistance. Concretly, we use PCA to reduce the dimension of raw images to 60 dimensions and classify them with KNN, GMM and MeansDistance. Here MeansDistance means that we calculate the distance between the samples and centers of every class, and then convert distance to scores with function e^{-x} and then introduce Softmax to normalize the score.

We mix those simple models with NN models in two ways: one is to add the simple score with some regularized coefficient, the other is to view the simple model as a weight and multiply the NN score by it. Here we discuss about the experiment outcomes of those model. The first one is not differentiable, so we skip the discussion on it; the second one can be easily beaten by naïve PGD, so we skip it; the results of the third one is shown in Table 5 where we take multiplication method as an example and use PGD20+CE to attack. It is sorry to say that this method serves an obfuscation for the NN model and do not actually strengthen the model. Likewise, other mixing method also obfuscate (or even not, like GMM) gradients of the loss.

表 5: Our Mixdel Model under different attacks

PGD20+CE Attack		Model1	Model2	Model3	Model4	Model5	Model6
Attack naïve model	Natural acc	0.9429	0.8302	0.8033	0.8492	0.8143	0.8825
	Robust acc	0.0006	0.5019	0.6049	0.5528	0.5329	0.6377
Attack mixed model	Natural acc	0.9424	0.8283	0.8033	0.8480	0.8154	0.8797
	Robust acc	0.0013	0.6062	0.6521	0.6196	0.6106	0.6843
Transfer attack mixed model	Natural acc	0.9424	0.8283	0.8033	0.8482	0.8154	0.8797
	Robust acc	0.0006	0.5099	0.5344	0.5530	0.5407	0.6376

4.3.2 Finetuning

To speed up the training process, we wonder whether we can use the model we have trained before to finetune it? We have tried several ways to finetune, *i.e.*,

Finetune Methods	Description
Normal trained Model + AT Finetune	It is totally useless just like Tsipras says the normal model learned some non-robust features [7].
AT trained Model + AWP Finetune	Both are worse than AWP without any pre-training. Maybe different robust model have quite different parameters and cannot directly transfer?
AT trained Model + Freezing shallow layer + AWP Finetune	

4.3.3 Robust Dataset

Ilyas proposes that robustness is determined by features learned but not bugs, and therefore make a robust dataset based on $l2$-norm [4]. We have tried to use this dataset to train but the outcomes is not good, and we guess maybe whether Ilyas method works depends on different perturbation criteria and we do not investigate furthur.

4.4 Our Solution

Though the above method gives us 'false sense of robustness', we can still apply it to our model since TA would not do specific attack:) Our mainstay is a WideResNet34 trained by AWP + TRADES with 60 hours training and the outcome is obfuscated by the outcomes of PCA+MeanDistance.

Some comments: We find that the AT is quite baffling since every bias towards standard AT will lead to the decreases of robust accuracy. Considering the high time cost of AT, we do not delicately investigate on the impact of different parameters.

5 Conclusion

In this project paper, we have tried various kind of weird approaches to attack and defend, and the results show they are quite useless and awful. We must show great respect to those baselines and the state-of-the-art methods. And even with their source code, we can hardly reproduce their outcomes. Finally, we repeatedly express our gratitude for their existence in human intellectual world.

https://github.com/QingquanBao/Spear-Shield contains all author's struggles :)

参考文献

[1] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. In *International Conference on Machine Learning*, pages 274–283. PMLR.

[2] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. *2017 IEEE Symposium on Security and Privacy (SP)*, 2017.

[3] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015.

[4] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features, 2019.

[5] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[6] Yusuke Tashiro, Yang Song, and Stefano Ermon. Diversity can be Transferred: Output Diversification for White- and Black-box Attacks.

[7] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness May Be at Odds with Accuracy.

[8] Dongxian Wu, Shu-Tao Xia, and Yisen Wang. Adversarial weight perturbation helps robust generalization. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 2958–2969. Curran Associates, Inc., 2020.

[9] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P. Xing, Laurent El Ghaoui, and Michael I. Jordan. Theoretically Principled Trade-off between Robustness and Accuracy.