Defense and Adversarial Training

Yunzhen Feng, Haocheng Ju, Zehao Wang, Haotong Yang, Pu Yang

School of Mathematical Science, Peking University

Questions:

Does there exist the "most adversarial" example? And does there exist the "most defensive" model/algorithm?

> *We can never be certain that a given attack finds the "most adversarial" example in the context, or that a particular defense mechanism prevents the existence of some well-defined class of adversarial attacks. This makes it difficult to navigate the landscape of adversarial robustness or to fully evaluate the possible security implications.*

A Beginning: why we should consider those questions?

A Beginning: why we should consider those questions?

Towards Evaluating the Robustness of Neural Networks

Table of contents

Towards Evaluating the Robustness of Neural Networks

Proposed a new targeted-attack type: CW attack which is more effective than before.

By those new attacks, they found that:

- **Defensive distillation**[1] does not significantly increase the robustness of neural networks.
- The attack examples generated by unsafe network can transfer to so-called safe(distilled) network.
- An explanation for why defensive distillation does not remove the adversarial examples (linearity hypothesis): the locally-linear nature of neural networks.

[1] based on knowledge distillation

Towards Evaluating the Robustness of Neural Networks

Definition. Adversarial examples considered in this paper is defined as follows,

$$\begin{array}{ll} \text{minimize} & \mathcal{D}(x, x+\delta) \\ \text{such that} & C(x+\delta) = t \\ & x+\delta \in [0,1]^n \end{array},$$

where D is some distance metric, C is the classifier and t is the target of x, here they choose $L_0, L_2,$ or L_∞.

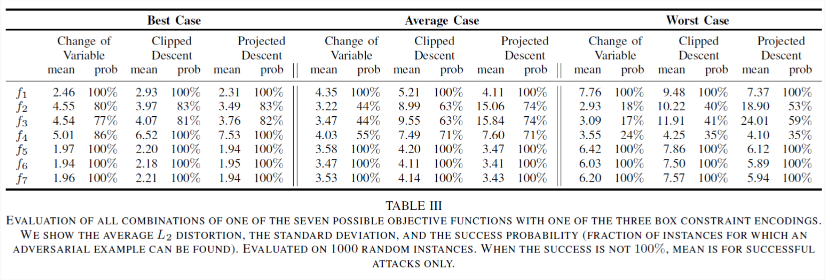
But C is always sophisticated, to simplify the process of optimization, there are some alternatives (f) for this constraint.

Then the definition can be reformulated as follows,

$$\begin{array}{ll} \text{minimize} & \|\delta\|_p + c \cdot f(x+\delta) \\ \text{such that} & x+\delta \in [0,1]^n \end{array}.$$

To deal with the box constraint, they used three methods: Projected gradient descent, Clipped gradient descent, Change of variables.

Towards Evaluating the Robustness of Neural Networks



	Best Case						Average Case						Worst Case					
	Change of Variable		Clipped Descent		Projected Descent		Change of Variable		Clipped Descent		Projected Descent		Change of Variable		Clipped Descent		Projected Descent	
	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob
f_1	2.46	100%	2.93	100%	2.31	100%	4.35	100%	5.21	100%	4.11	100%	7.76	100%	9.48	100%	7.37	100%
f_2	4.55	80%	3.97	83%	3.49	83%	3.22	44%	8.99	63%	15.06	74%	2.93	18%	10.22	40%	18.90	53%
f_3	4.54	77%	4.07	81%	3.76	82%	3.47	44%	9.55	63%	15.84	74%	3.09	17%	11.91	41%	24.01	59%
f_4	5.01	86%	6.52	100%	7.53	100%	4.03	55%	7.49	71%	7.60	71%	3.55	24%	4.25	35%	4.10	35%
f_5	1.97	100%	2.20	100%	1.94	100%	3.58	100%	4.20	100%	3.47	100%	6.42	100%	7.86	100%	6.12	100%
f_6	1.94	100%	2.18	100%	1.95	100%	3.47	100%	4.11	100%	3.41	100%	6.03	100%	7.50	100%	5.89	100%
f_7	1.96	100%	2.21	100%	1.94	100%	3.53	100%	4.14	100%	3.43	100%	6.20	100%	7.57	100%	5.94	100%

TABLE III
Evaluation of all combinations of one of the seven possible objective functions with one of the three box constraint encodings. We show the average L_2 distortion, the standard deviation, and the success probability (fraction of instances for which an adversarial example can be found). Evaluated on 1000 random instances. When the success is not 100%, mean is for successful attacks only.

Figure 1: Result of three methods and 7 alternative constraints.

Towards Evaluating the Robustness of Neural Networks

Three attacks:

- L_2:

$$\text{minimize} \ \left\| \tfrac{1}{2}(\tanh(w)+1) - x \right\|_2^2 + c \cdot f\left(\tfrac{1}{2}(\tanh(w)+1)\right)$$

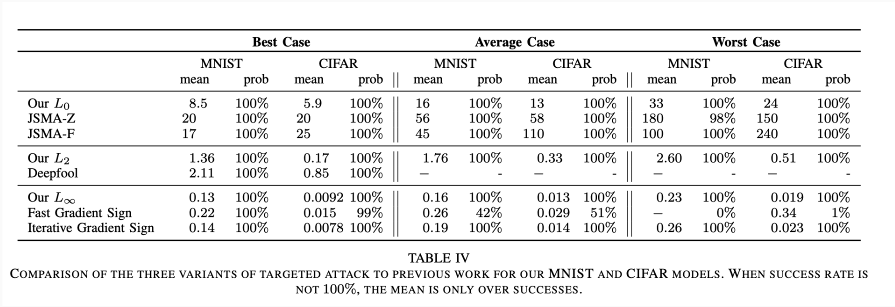
with f defined as

$$f(x') = \max\left(\max\left\{Z(x')_i : i \neq t\right\} - Z(x')_t, -\kappa\right)$$

- L_0: 0-norm is not differentiable, so they used an iterative algorithm to implement it. Detailed can be found in this paper.
- L_∞:

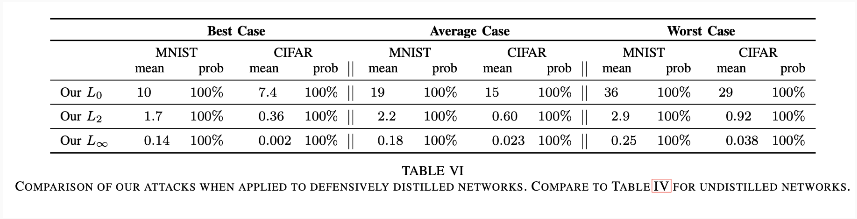
$$\text{minimize} \quad c \cdot f(x+\delta) + \|\delta\|_\infty$$

Towards Evaluating the Robustness of Neural Networks



	Best Case				Average Case				Worst Case			
	MNIST		CIFAR		MNIST		CIFAR		MNIST		CIFAR	
	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob
Our L_0	8.5	100%	5.9	100%	16	100%	13	100%	33	100%	24	100%
JSMA-Z	20	100%	20	100%	56	100%	58	100%	180	98%	150	100%
JSMA-F	17	100%	25	100%	45	100%	110	100%	100	100%	240	100%
Our L_2	1.36	100%	0.17	100%	1.76	100%	0.33	100%	2.60	100%	0.51	100%
Deepfool	2.11	100%	0.85	100%	—	-	—	-	—	-	—	-
Our L_∞	0.13	100%	0.0092	100%	0.16	100%	0.013	100%	0.23	100%	0.019	100%
Fast Gradient Sign	0.22	100%	0.015	99%	0.26	42%	0.029	51%	—	0%	0.34	1%
Iterative Gradient Sign	0.14	100%	0.0078	100%	0.19	100%	0.014	100%	0.26	100%	0.023	100%

TABLE IV
COMPARISON OF THE THREE VARIANTS OF TARGETED ATTACK TO PREVIOUS WORK FOR OUR MNIST AND CIFAR MODELS. WHEN SUCCESS RATE IS NOT 100%, THE MEAN IS ONLY OVER SUCCESSES.



	Best Case				Average Case				Worst Case			
	MNIST		CIFAR		MNIST		CIFAR		MNIST		CIFAR	
	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob
Our L_0	10	100%	7.4	100%	19	100%	15	100%	36	100%	29	100%
Our L_2	1.7	100%	0.36	100%	2.2	100%	0.60	100%	2.9	100%	0.92	100%
Our L_∞	0.14	100%	0.002	100%	0.18	100%	0.023	100%	0.25	100%	0.038	100%

TABLE VI
COMPARISON OF OUR ATTACKS WHEN APPLIED TO DEFENSIVELY DISTILLED NETWORKS. COMPARE TO TABLE IV FOR UNDISTILLED NETWORKS.

Defense

Defense

To improve the robustness of neural networks to adversarial examples, many defense strategies and models have been proposed, such as adversarial training, orthogonal regularization, Bayesian method, TRADES, rejecting adversarial examples, Jacobian regularization, generative model based defense, pixel defense, ordinary differential equation (ODE) viewpoint, ensemble via an intriguing stochastic differential equation perspective, and feature denoising, etc. Among all these approaches, adversarial training and its variants tend to be most effective since it largely avoids the the obfuscated gradient problem.

Here we will mainly introduce defense with network architecture or regularization and adversarial training.

Defense

Defense with Network Architecture

Table of contents

On Robustness of Neural ODE

Main contribution:

- Study the inherent robustness of Neural ODE.
- Design TisODE (time-invariant steady neural ODE) for more robust neural ODE-based neural network.

On Robustness of Neural ODE

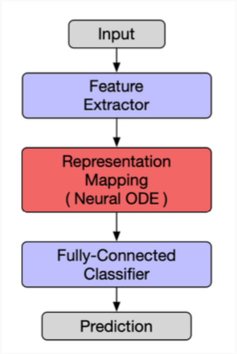

Input
Feature Extractor
Representation Mapping (Neural ODE)
Fully-Connected Classifier
Prediction

Figure 2: The architecture of ODE-Net

They claimed that due to this non-intersecting property of ODE integral curves, ODENets are intrinsically robust.

On Robustness of Neural ODE

Experiments on vanilla neural ODE:

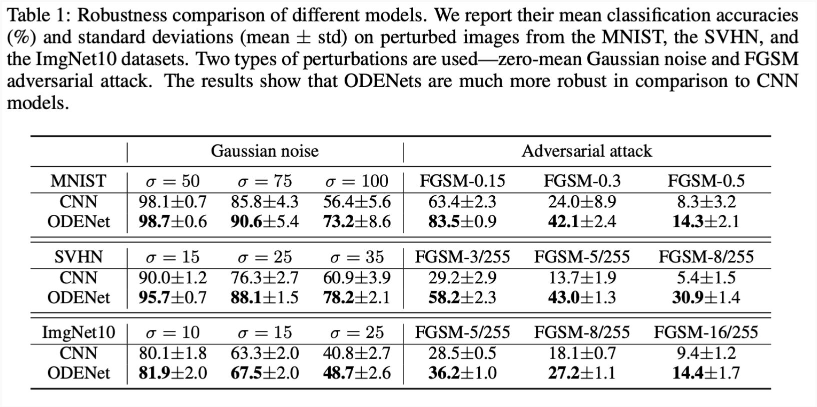


Table 1: Robustness comparison of different models. We report their mean classification accuracies (%) and standard deviations (mean ± std) on perturbed images from the MNIST, the SVHN, and the ImgNet10 datasets. Two types of perturbations are used—zero-mean Gaussian noise and FGSM adversarial attack. The results show that ODENets are much more robust in comparison to CNN models.

	Gaussian noise			Adversarial attack		
MNIST	$\sigma = 50$	$\sigma = 75$	$\sigma = 100$	FGSM-0.15	FGSM-0.3	FGSM-0.5
CNN	98.1±0.7	85.8±4.3	56.4±5.6	63.4±2.3	24.0±8.9	8.3±3.2
ODENet	**98.7**±0.6	**90.6**±5.4	**73.2**±8.6	**83.5**±0.9	**42.1**±2.4	**14.3**±2.1
SVHN	$\sigma = 15$	$\sigma = 25$	$\sigma = 35$	FGSM-3/255	FGSM-5/255	FGSM-8/255
CNN	90.0±1.2	76.3±2.7	60.9±3.9	29.2±2.9	13.7±1.9	5.4±1.5
ODENet	**95.7**±0.7	**88.1**±1.5	**78.2**±2.1	**58.2**±2.3	**43.0**±1.3	**30.9**±1.4
ImgNet10	$\sigma = 10$	$\sigma = 15$	$\sigma = 25$	FGSM-5/255	FGSM-8/255	FGSM-16/255
CNN	80.1±1.8	63.3±2.0	40.8±2.7	28.5±0.5	18.1±0.7	9.4±1.2
ODENet	**81.9**±2.0	**67.5**±2.0	**48.7**±2.6	**36.2**±1.0	**27.2**±1.1	**14.4**±1.7

On Robustness of Neural ODE

Experiments on vanilla neural ODE:

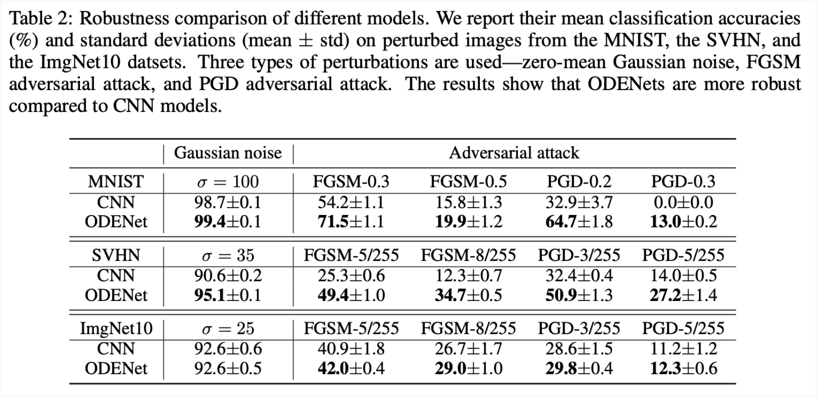


Table 2: Robustness comparison of different models. We report their mean classification accuracies (%) and standard deviations (mean ± std) on perturbed images from the MNIST, the SVHN, and the ImgNet10 datsets. Three types of perturbations are used—zero-mean Gaussian noise, FGSM adversarial attack, and PGD adversarial attack. The results show that ODENets are more robust compared to CNN models.

	Gaussian noise	Adversarial attack			
MNIST	$\sigma = 100$	FGSM-0.3	FGSM-0.5	PGD-0.2	PGD-0.3
CNN	98.7±0.1	54.2±1.1	15.8±1.3	32.9±3.7	0.0±0.0
ODENet	**99.4**±0.1	**71.5**±1.1	**19.9**±1.2	**64.7**±1.8	**13.0**±0.2
SVHN	$\sigma = 35$	FGSM-5/255	FGSM-8/255	PGD-3/255	PGD-5/255
CNN	90.6±0.2	25.3±0.6	12.3±0.7	32.4±0.4	14.0±0.5
ODENet	**95.1**±0.1	**49.4**±1.0	**34.7**±0.5	**50.9**±1.3	**27.2**±1.4
ImgNet10	$\sigma = 25$	FGSM-5/255	FGSM-8/255	PGD-3/255	PGD-5/255
CNN	92.6±0.6	40.9±1.8	26.7±1.7	28.6±1.5	11.2±1.2
ODENet	92.6±0.5	**42.0**±0.4	**29.0**±1.0	**29.8**±0.4	**12.3**±0.6

On Robustness of Neural ODE

Time-invariant steady neural ODEs

They proposed so-called TisODE by removed the time dependence of the vanilla neural ODE,

$$\begin{cases} \frac{\mathrm{d}\mathbf{z}(t)}{\mathrm{d}t} = {}^{2} f_\theta(\mathbf{z}(t)) \\ \mathbf{z}(0) = \mathbf{z}_{\mathrm{in}} \\ \mathbf{z}_{\mathrm{out}} = \mathbf{z}(T) \end{cases}$$

[2]The right side of this equation is $f_\theta(\mathbf{z}(t), t)$ in the vanilla neural ODE.

On Robustness of Neural ODE

Experiments on TisODE:

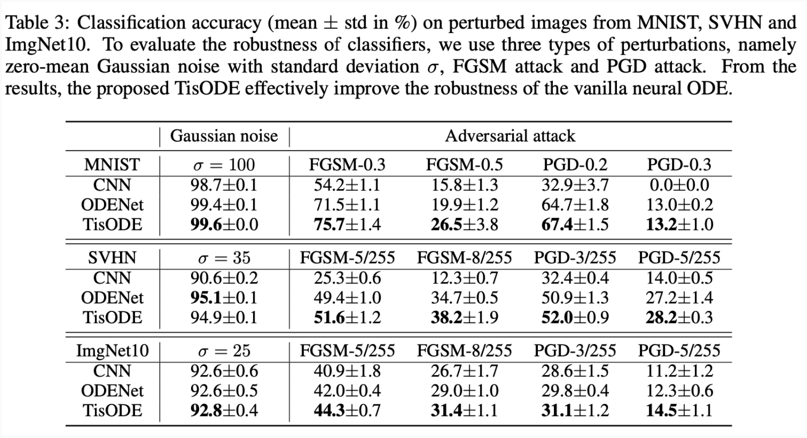


Table 3: Classification accuracy (mean ± std in %) on perturbed images from MNIST, SVHN and ImgNet10. To evaluate the robustness of classifiers, we use three types of perturbations, namely zero-mean Gaussian noise with standard deviation σ, FGSM attack and PGD attack. From the results, the proposed TisODE effectively improve the robustness of the vanilla neural ODE.

	Gaussian noise	Adversarial attack			
MNIST	$\sigma = 100$	FGSM-0.3	FGSM-0.5	PGD-0.2	PGD-0.3
CNN	98.7±0.1	54.2±1.1	15.8±1.3	32.9±3.7	0.0±0.0
ODENet	99.4±0.1	71.5±1.1	19.9±1.2	64.7±1.8	13.0±0.2
TisODE	**99.6**±0.0	**75.7**±1.4	**26.5**±3.8	**67.4**±1.5	**13.2**±1.0
SVHN	$\sigma = 35$	FGSM-5/255	FGSM-8/255	PGD-3/255	PGD-5/255
CNN	90.6±0.2	25.3±0.6	12.3±0.7	32.4±0.4	14.0±0.5
ODENet	**95.1**±0.1	49.4±1.0	34.7±0.5	50.9±1.3	27.2±1.4
TisODE	94.9±0.1	**51.6**±1.2	**38.2**±1.9	**52.0**±0.9	**28.2**±0.3
ImgNet10	$\sigma = 25$	FGSM-5/255	FGSM-8/255	PGD-3/255	PGD-5/255
CNN	92.6±0.6	40.9±1.8	26.7±1.7	28.6±1.5	11.2±1.2
ODENet	92.6±0.5	42.0±0.4	29.0±1.0	29.8±0.4	12.3±0.6
TisODE	**92.8**±0.4	**44.3**±0.7	**31.4**±1.1	**31.1**±1.2	**14.5**±1.1

On Robustness of Neural ODE

Experiments on TisODE:

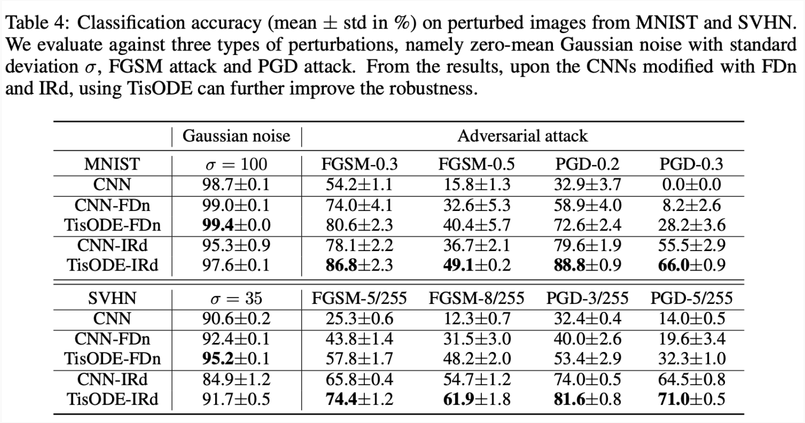


Table 4: Classification accuracy (mean ± std in %) on perturbed images from MNIST and SVHN. We evaluate against three types of perturbations, namely zero-mean Gaussian noise with standard deviation σ, FGSM attack and PGD attack. From the results, upon the CNNs modified with FDn and IRd, using TisODE can further improve the robustness.

	Gaussian noise	Adversarial attack			
MNIST	$\sigma = 100$	FGSM-0.3	FGSM-0.5	PGD-0.2	PGD-0.3
CNN	98.7±0.1	54.2±1.1	15.8±1.3	32.9±3.7	0.0±0.0
CNN-FDn	99.0±0.1	74.0±4.1	32.6±5.3	58.9±4.0	8.2±2.6
TisODE-FDn	**99.4**±0.0	80.6±2.3	40.4±5.7	72.6±2.4	28.2±3.6
CNN-IRd	95.3±0.9	78.1±2.2	36.7±2.1	79.6±1.9	55.5±2.9
TisODE-IRd	97.6±0.1	**86.8**±2.3	**49.1**±0.2	**88.8**±0.9	**66.0**±0.9
SVHN	$\sigma = 35$	FGSM-5/255	FGSM-8/255	PGD-3/255	PGD-5/255
CNN	90.6±0.2	25.3±0.6	12.3±0.7	32.4±0.4	14.0±0.5
CNN-FDn	92.4±0.1	43.8±1.4	31.5±3.0	40.0±2.6	19.6±3.4
TisODE-FDn	**95.2**±0.1	57.8±1.7	48.2±2.0	53.4±2.9	32.3±1.0
CNN-IRd	84.9±1.2	65.8±0.4	54.7±1.2	74.0±0.5	64.5±0.8
TisODE-IRd	91.7±0.5	**74.4**±1.2	**61.9**±1.8	**81.6**±0.8	**71.0**±0.5

Defense

Defense with Regularization

Table of contents

Jacobian Regularization[3]

Jacobian regularization term:

$$\|J(x_i)\|_F^2 = \sum_{d=1}^{D} \sum_{k=1}^{K} \left(\frac{\partial}{\partial x_d} z_k^{(L)}(x_i) \right)^2 = \sum_{k=1}^{K} \left\| \nabla_x z_k^{(L)}(x_i) \right\|_2^2$$

$$Loss = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log f_k(x_i) + \lambda \sqrt{\sum_{d=1}^{D} \sum_{k=1}^{K} \sum_{i=1}^{N} \left(\frac{\partial}{\partial x_d} z_k^{(L)}(x_i) \right)^2}$$

[3] Proposed by paper *Improving DNN Robustness to Adversarial Attacks using Jacobian Regularization*

Input Gradient Regularization[4]

Regularization term:

$$\sum_{d=1}^{D} \sum_{i=1}^{N} \left(\frac{\partial}{\partial x_d} \sum_{k=1}^{K} -y_{ik} \log f_k \left(x_i \right) \right)^2$$

[4]Proposed by paper *Formal guarantees on the robustness of a classi- fier against adversarial manipulation*

Cross-Lipschitz Regularization[5]

Regularization term:

$$\sum_{i=1}^{N} \sum_{j,k=1}^{K} \left\| \nabla_x z_k^{(L)}(x_i) - \nabla_x z_j^{(L)}(x_i) \right\|_2^2$$

[5]Proposed by paper *Improving the adversarial robustness and inter- pretability of deep neural networks by regularizing their input gradients*

Defense with Regularization

Research on defense with regularization aims to provide a more interpretable and theoretical way to achieve and analysis robustness, maybe some of them will cost less than adversarial training in time. It's a trade-off between interpretability(theoretical) and effectiveness.

Defense

Why do some fancy defensive methods fail?

Table of contents

Obfuscated Gradients

Many defense methods, intentionally or unintentionally, use a "gradient masking" method. Most white box attacks are performed by calculating the gradient of the model, so if the effective gradient cannot be calculated, the attack will be invalid.

Gradient masking makes gradients useless, usually by changing the model to some extent to make it non-differentiable, or making it have zero gradient in most cases, or the gradient point is far from the decision boundary. In fact, what gradient masking does is to make the optimizer not work without actually moving the decision boundary.

Obfuscated Gradients

However, since the decision boundary of the model trained with the same distribution data set will not be much different, the defense method based on gradient masking is easily broken by black box attacks, because the attacker can get the gradient in the "substitute". Some defense strategies (such as replacing smooth sigmoid units with hard thresholds) are directly designed to perform masking and masking. Other defense measures, such as many forms of confrontation training, are not designed with gradient masking as the target, but actually what they did is similar to gradient masking.

Obfuscated Gradients

Three ways in which defenses obfuscate gradients:

- **Shattered Gradients** are caused when a defense is non- differentiable, introduces numeric instability, or otherwise causes a gradient to be nonexistent or incorrect. Defenses that cause gradient shattering can do so unintentionally, by using differentiable operations but where following the gradient does not maximize classification loss globally.
- **Stochastic Gradients** are caused by randomized defenses, where either the network itself is randomized or the input is randomly transformed before being fed to the classifier, causing the gradients to become randomized. This causes methods using a single sample of the randomness to incorrectly estimate the true gradient.
- **Exploding & Vanishing Gradients** are often caused by defenses that consist of multiple iterations of neural network evaluation, feeding the output of one computation as the input of the next. This type of computation, when unrolled, can be viewed as an extremely deep neural network evaluation, which can cause vanishing/exploding gradients.

Obfuscated Gradients

Identifying Obfuscated Masked Gradients

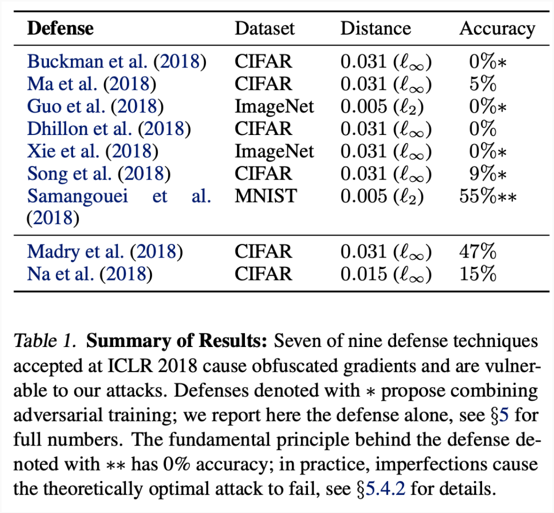
- One-step attacks perform better than iterative attacks.
- Black-box attacks are better than white-box attacks.
- Unbounded attacks do not reach 100% success.
- Random sampling finds adversarial examples.
- Increasing distortion bound does not increase success.

Obfuscated Gradients

- **Backward Pass Differentiable Approximation** for shattered gradients.
- **Expectation over Transformation** for Randomized Classifiers(Stochastic Gradients).
- **Reparameterization** for vanishing/exploding gradients.

Obfuscated Gradients

Case study: ICLR 2018 Defenses



Defense	Dataset	Distance	Accuracy
Buckman et al. (2018)	CIFAR	0.031 (ℓ_∞)	0%$*$
Ma et al. (2018)	CIFAR	0.031 (ℓ_∞)	5%
Guo et al. (2018)	ImageNet	0.005 (ℓ_2)	0%$*$
Dhillon et al. (2018)	CIFAR	0.031 (ℓ_∞)	0%
Xie et al. (2018)	ImageNet	0.031 (ℓ_∞)	0%$*$
Song et al. (2018)	CIFAR	0.031 (ℓ_∞)	9%$*$
Samangouei et al. (2018)	MNIST	0.005 (ℓ_2)	55%$**$
Madry et al. (2018)	CIFAR	0.031 (ℓ_∞)	47%
Na et al. (2018)	CIFAR	0.015 (ℓ_∞)	15%

Table 1. **Summary of Results:** Seven of nine defense techniques accepted at ICLR 2018 cause obfuscated gradients and are vulnerable to our attacks. Defenses denoted with $*$ propose combining adversarial training; we report here the defense alone, see §5 for full numbers. The fundamental principle behind the defense denoted with $**$ has 0% accuracy; in practice, imperfections cause the theoretically optimal attack to fail, see §5.4.2 for details.

Adversarial Training

Adversarial Training

Some basic methods

Table of contents

Towards Deep Learning Models Resistant to Adversarial Attacks

Main contribution:

- They proposed a strong first-order attack: PGD (projected gradient descent), which is the strongest attack utilizing the local first order information about the network.
- With PGD as a reliable first-order adversary, their adversarially trained models are robust to a wide range of adversarial attacks.

Towards Deep Learning Models Resistant to Adversarial Attacks

An optimization view on adversarial robustness:

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\max_{\delta \in \mathcal{S}} L(\theta, x + \delta, y) \right].$$

Inspired by the optimization algorithms of the above problem for the inner-loop (max), some attack methods are proposed:

- FGSM: $x + \varepsilon \operatorname{sgn}(\nabla_x L(\theta, x, y))$, an l_∞ bounded adversary.
- **PGD**: $x^{t+1} = \Pi_{x+\mathcal{S}}\left(x^t + \alpha \operatorname{sgn}(\nabla_x L(\theta, x, y))\right)$.

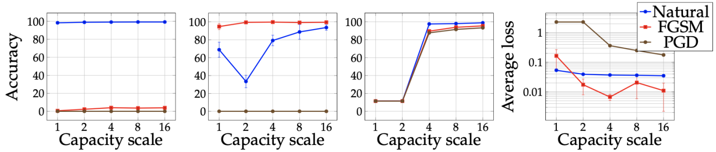
Towards Deep Learning Models Resistant to Adversarial Attacks

Network Capacity and Adversarial Robustness:

- Capacity alone helps.
- FGSM adversaries don't increase robustness (for large ϵ).
- Weak models may fail to learn non-trivial classifiers.
- The value of the saddle point problem decreases as we increase the capacity.
- More capacity and stronger adversaries decrease transferability.

Towards Deep Learning Models Resistant to Adversarial Attacks

MNIST



CIFAR10

	Simple	Wide
Natural	92.7%	95.2%
FGSM	27.5%	32.7%
PGD	0.8%	3.5%

(a) Standard training

Simple	Wide
87.4%	90.3%
90.9%	95.1%
0.0%	0.0%

(b) FGSM training

Simple	Wide
79.4%	87.3%
51.7%	56.1%
43.7%	45.8%

(c) PGD training

Simple	Wide
0.00357	0.00371
0.0115	0.00557
1.11	0.0218

(d) Training Loss

Figure 4: The effect of network capacity on the performance of the network. We trained MNIST and CIFAR10 networks of varying capacity on: (a) natural examples, (b) with FGSM-made adversarial examples, (c) with PGD-made adversarial examples. In the first three plots/tables of each dataset, we show how the standard and adversarial accuracy changes with respect to capacity for each training regime. In the final plot/table, we show the value of the cross-entropy loss on the adversarial examples the networks were trained on. This corresponds to the value of our saddle point formulation (2.1) for different sets of allowed perturbations.

Towards Deep Learning Models Resistant to Adversarial Attacks

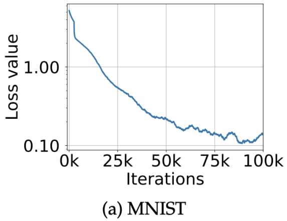
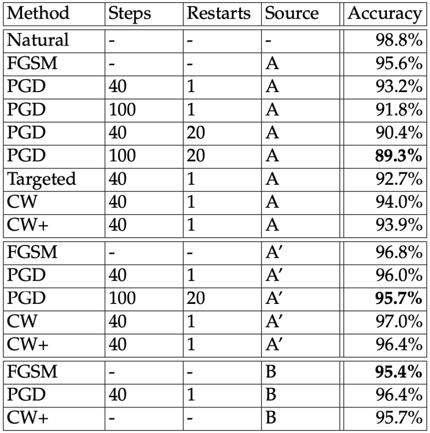


Figure 5: Cross-entropy loss on adversarial examples during training. The plots show how the adversarial loss on training examples evolves during training the MNIST and CIFAR10 networks against a PGD adversary. The sharp drops in the CIFAR10 plot correspond to decreases in training step size. These plots illustrate that we can consistently reduce the value of the inner problem of the saddle point formulation (2.1), thus producing an increasingly robust classifier.

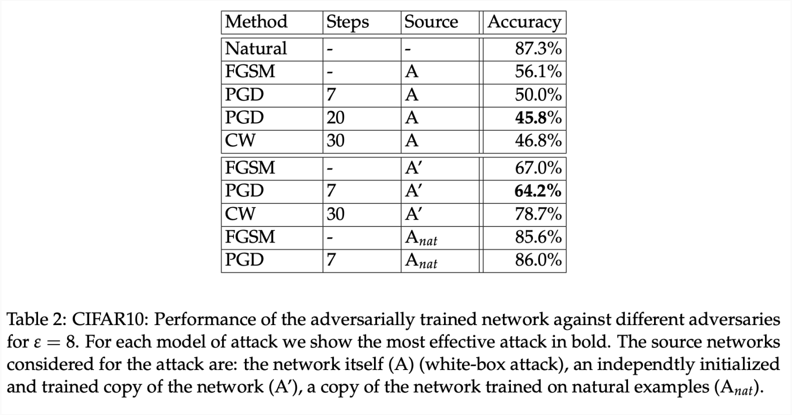
Towards Deep Learning Models Resistant to Adversarial Attacks



Method	Steps	Restarts	Source	Accuracy
Natural	-	-	-	98.8%
FGSM	-	-	A	95.6%
PGD	40	1	A	93.2%
PGD	100	1	A	91.8%
PGD	40	20	A	90.4%
PGD	100	20	A	**89.3%**
Targeted	40	1	A	92.7%
CW	40	1	A	94.0%
CW+	40	1	A	93.9%
FGSM	-	-	A′	96.8%
PGD	40	1	A′	96.0%
PGD	100	20	A′	**95.7%**
CW	40	1	A′	97.0%
CW+	40	1	A′	96.4%
FGSM	-	-	B	**95.4%**
PGD	40	1	B	96.4%
CW+	-	-	B	95.7%

Table 1: MNIST: Performance of the adversarially trained network against different adversaries for $\varepsilon = 0.3$. For each model of attack we show the most successful attack with bold. The source networks used for the attack are: the network itself (A) (white-box attack), an indepentenly initialized and trained copy of the network (A′), architecture B from [29] (B).

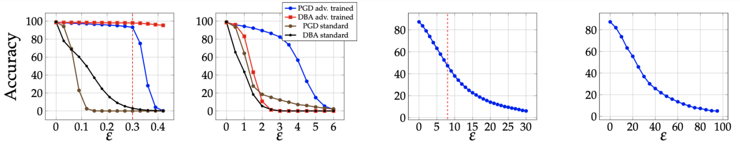
Towards Deep Learning Models Resistant to Adversarial Attacks



Method	Steps	Source	Accuracy
Natural	-	-	87.3%
FGSM	-	A	56.1%
PGD	7	A	50.0%
PGD	20	A	**45.8%**
CW	30	A	46.8%
FGSM	-	A′	67.0%
PGD	7	A′	**64.2%**
CW	30	A′	78.7%
FGSM	-	A_{nat}	85.6%
PGD	7	A_{nat}	86.0%

Table 2: CIFAR10: Performance of the adversarially trained network against different adversaries for $\varepsilon = 8$. For each model of attack we show the most effective attack in bold. The source networks considered for the attack are: the network itself (A) (white-box attack), an independtly initialized and trained copy of the network (A′), a copy of the network trained on natural examples (A_{nat}).

Towards Deep Learning Models Resistant to Adversarial Attacks



(a) MNIST, ℓ_∞-norm (b) MNIST, ℓ_2-norm (c) CIFAR10, ℓ_∞-norm (d) CIFAR10, ℓ_2-norm

Figure 6: Performance of our adversarially trained networks against PGD adversaries of different strength. The MNIST and CIFAR10 networks were trained against $\varepsilon = 0.3$ and $\varepsilon = 8$ PGD ℓ_∞ adversaries respectively (the training ε is denoted with a red dashed lines in the ℓ_∞ plots). In the case of the MNIST adversarially trained networks, we also evaluate the performance of the Decision Boundary Attack (DBA) [4] with 2000 steps and PGD on standard and adversarially trained models. We observe that for ε less or equal to the value used during training, the performance is equal or better. For MNIST there is a sharp drop shortly after. Moreover, we observe that the performance of PGD on the MNIST ℓ_2-trained networks is poor and significantly overestimates the robustness of the model. This is potentially due to the threshold filters learned by the model masking the loss gradients (the decision-based attack does not utilize gradients).

MMA Training

This paper proposed a margin maximization method for input space inspired by margin theory in SVM.

Definition. For an input (x, y), we define its margin w.r.t. the classifier $f_\theta(\cdot)$ as:

$$d_\theta(x, y) = \|\delta^*\| = \min \|\delta\| \quad \text{s.t. } \delta : L_\theta^{01}(x + \delta, y) = 1,$$

where $\delta^* = \arg\min_{L_\theta^{01}(x+\delta,y)=1} \|\delta\|$ is the "shortest successful perturbation".

For classification results, don't just look at the error rate, you should pay attention to Margin. Margin represents confidence, and confidence has a significant effect on generalization ability.

MMA Training

The proposed method is the following optimization problem,

$$\min_{\theta} \left\{ \sum_{i \in \mathcal{S}_\theta^+} \max\left\{0, d_{\max} - d_\theta\left(x_i, y_i\right)\right\} + \beta \sum_{j \in \mathcal{S}_\theta^-} \mathcal{J}_\theta\left(x_j, y_j\right) \right\},$$

where $\mathcal{S}_\theta^+ = \left\{i : L_\theta^{\mathrm{LM}}\left(x_i, y_i\right) < 0\right\}$ is the set of correctly classified examples, $\mathcal{S}_\theta^+ = \left\{i : L_\theta^{\mathrm{LM}}\left(x_i, y_i\right) \geq 0\right\}$ is the set of wrongly classified examples, $\mathcal{J}_\theta$ is a regular classification loss function, e.g. cross entropy loss, $d_\theta\left(x_i, y_i\right)$ is the margin for correctly classified samples.

MMA Training

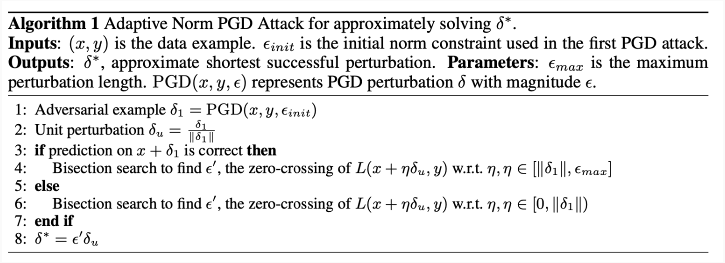
Unlike SVM and robust linear model, neural network based objective function is not easy to calculated in such a min-max optimization problem.

The following theorem provides a viable way to increase $d_\theta\left(x_i, y_i\right)$.

Theorem. Gradient descent on $L_\theta^{LM}\left(x+\delta^*, y\right)$ w.r.t. θ with a proper step size increases $d_\theta\left(x_i, y_i\right)$, where $\delta^* = \arg\min_{L_\theta^{LM}(x+\delta, y) \geq 0} \|\delta\|$ is the shortest successful perturbation given the current θ.

MMA Training

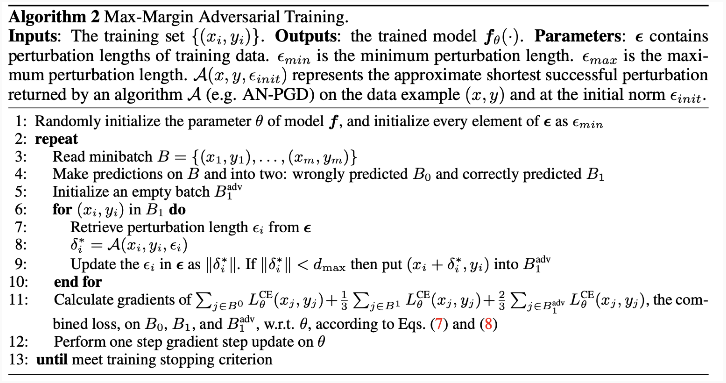
Note that in the above theorem, we have to find the optimal δ^*. The following algorithm proposed a possible way.



Algorithm 1 Adaptive Norm PGD Attack for approximately solving δ^*.
Inputs: (x, y) is the data example. ϵ_{init} is the initial norm constraint used in the first PGD attack. **Outputs**: δ^*, approximate shortest successful perturbation. **Parameters**: ϵ_{max} is the maximum perturbation length. $\mathrm{PGD}(x, y, \epsilon)$ represents PGD perturbation δ with magnitude ϵ.

```
1: Adversarial example $\delta_1 = \mathrm{PGD}(x, y, \epsilon_{init})$
2: Unit perturbation $\delta_u = \frac{\delta_1}{\|\delta_1\|}$
3: **if** prediction on $x + \delta_1$ is correct **then**
4:     Bisection search to find $\epsilon'$, the zero-crossing of $L(x + \eta\delta_u, y)$ w.r.t. $\eta, \eta \in [\|\delta_1\|, \epsilon_{max}]$
5: **else**
6:     Bisection search to find $\epsilon'$, the zero-crossing of $L(x + \eta\delta_u, y)$ w.r.t. $\eta, \eta \in [0, \|\delta_1\|)$
7: **end if**
8: $\delta^* = \epsilon'\delta_u$
```

MMA Training



Algorithm 2 Max-Margin Adversarial Training.

Inputs: The training set $\{(x_i, y_i)\}$. **Outputs**: the trained model $\boldsymbol{f}_\theta(\cdot)$. **Parameters**: $\boldsymbol{\epsilon}$ contains perturbation lengths of training data. ϵ_{min} is the minimum perturbation length. ϵ_{max} is the maximum perturbation length. $\mathcal{A}(x, y, \epsilon_{init})$ represents the approximate shortest successful perturbation returned by an algorithm $\mathcal{A}$ (e.g. AN-PGD) on the data example (x, y) and at the initial norm ϵ_{init}.

```
1: Randomly initialize the parameter $\theta$ of model $\boldsymbol{f}$, and initialize every element of $\boldsymbol{\epsilon}$ as $\epsilon_{min}$
2: repeat
3:     Read minibatch $B = \{(x_1, y_1), \ldots, (x_m, y_m)\}$
4:     Make predictions on $B$ and into two: wrongly predicted $B_0$ and correctly predicted $B_1$
5:     Initialize an empty batch $B_1^{\text{adv}}$
6:     for $(x_i, y_i)$ in $B_1$ do
7:         Retrieve perturbation length $\epsilon_i$ from $\boldsymbol{\epsilon}$
8:         $\delta_i^* = \mathcal{A}(x_i, y_i, \epsilon_i)$
9:         Update the $\epsilon_i$ in $\boldsymbol{\epsilon}$ as $\|\delta_i^*\|$. If $\|\delta_i^*\| < d_{\max}$ then put $(x_i + \delta_i^*, y_i)$ into $B_1^{\text{adv}}$
10:    end for
11:    Calculate gradients of $\sum_{j \in B^0} L_\theta^{\text{CE}}(x_j, y_j) + \frac{1}{3} \sum_{j \in B^1} L_\theta^{\text{CE}}(x_j, y_j) + \frac{2}{3} \sum_{j \in B_1^{\text{adv}}} L_\theta^{\text{CE}}(x_j, y_j)$, the combined loss, on $B_0$, $B_1$, and $B_1^{\text{adv}}$, w.r.t. $\theta$, according to Eqs. (7) and (8)
12:    Perform one step gradient step update on $\theta$
13: until meet training stopping criterion
```

The above algorithm summarizes the practical MMA training algorithm. During training for each minibatch, 1) separate it into 2 batches based on if the current prediction matches the label; 2) find δ^* for each example in the "correct batch"; 3) calculate the gradient of θ.

MMA Training

Understanding the MMA Training

For brevity, fixing (x, y), let
$L(\delta, \theta) = L_\theta^{\mathrm{LM}}(x+\delta, y), d_\theta = d_\theta(x, y),$ and $\epsilon_\theta^*(\rho) = \min_{\delta: L(\delta,\theta)\geq\rho} \|\delta\|$.

Theorem. Assuming an update from adversarial training changes θ_0 to θ_1, s.t. $\rho^* = \max_{\|\delta\|\leq\epsilon} L(\delta, \theta_0) > \max_{\|\delta\|\leq\epsilon} L(\delta, \theta_1)$, then

- if $\epsilon = d_{\theta_0}$, then $\rho^* = 0, \epsilon_{\theta_1}^*(\rho^*) = d_{\theta_1} \geq d_{\theta_0} = \epsilon_{\theta_0}^*(\rho^*)$;
- if $\epsilon < d_{\theta_0}$, then
  $\rho^* \leq 0, \epsilon_{\theta_0}^*(\rho^*) \leq d_{\theta_0}, \epsilon_{\theta_1}^*(\rho^*) \leq d_{\theta_1},$ and $\epsilon_{\theta_1}^*(\rho^*) \geq \epsilon_{\theta_0}^*(\rho^*)$;
- if $\epsilon > d_{\theta_0}$, then
  $\rho^* \geq 0, \epsilon_{\theta_0}^*(\rho^*) \geq d_{\theta_0}, \epsilon_{\theta_1}^*(\rho^*) \geq d_{\theta_1},$ and $\epsilon_{\theta_1}^*(\rho^*) \geq \epsilon_{\theta_0}^*(\rho^*)$.

Remark. The above theorem implies that, adversarial training, with the logit margin loss and a fixed ε

- exactly maximizes the margin, if ε is equal to the margin;
- maximizes a lower bound of the margin, if ε is less than the margin;
- maximizes an upper bound of the margin, if ε is greater than the margin.

MMA Training

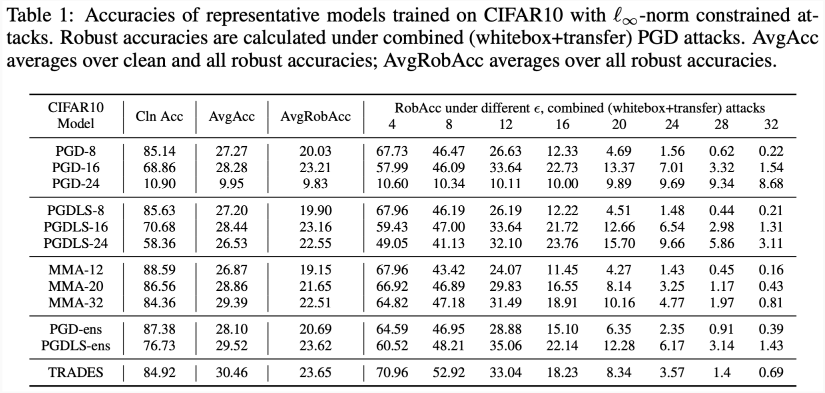


Table 1: Accuracies of representative models trained on CIFAR10 with ℓ_∞-norm constrained attacks. Robust accuracies are calculated under combined (whitebox+transfer) PGD attacks. AvgAcc averages over clean and all robust accuracies; AvgRobAcc averages over all robust accuracies.

CIFAR10 Model	Cln Acc	AvgAcc	AvgRobAcc	RobAcc under different ϵ, combined (whitebox+transfer) attacks							
				4	8	12	16	20	24	28	32
PGD-8	85.14	27.27	20.03	67.73	46.47	26.63	12.33	4.69	1.56	0.62	0.22
PGD-16	68.86	28.28	23.21	57.99	46.09	33.64	22.73	13.37	7.01	3.32	1.54
PGD-24	10.90	9.95	9.83	10.60	10.34	10.11	10.00	9.89	9.69	9.34	8.68
PGDLS-8	85.63	27.20	19.90	67.96	46.19	26.19	12.22	4.51	1.48	0.44	0.21
PGDLS-16	70.68	28.44	23.16	59.43	47.00	33.64	21.72	12.66	6.54	2.98	1.31
PGDLS-24	58.36	26.53	22.55	49.05	41.13	32.10	23.76	15.70	9.66	5.86	3.11
MMA-12	88.59	26.87	19.15	67.96	43.42	24.07	11.45	4.27	1.43	0.45	0.16
MMA-20	86.56	28.86	21.65	66.92	46.89	29.83	16.55	8.14	3.25	1.17	0.43
MMA-32	84.36	29.39	22.51	64.82	47.18	31.49	18.91	10.16	4.77	1.97	0.81
PGD-ens	87.38	28.10	20.69	64.59	46.95	28.88	15.10	6.35	2.35	0.91	0.39
PGDLS-ens	76.73	29.52	23.62	60.52	48.21	35.06	22.14	12.28	6.17	3.14	1.43
TRADES	84.92	30.46	23.65	70.96	52.92	33.04	18.23	8.34	3.57	1.4	0.69

Bilateral Adversarial Training

Bilateral Adversarial Training: perturb both the image and the label during training.

Generate the adversarial label: an closed-form heuristic solution.

Generate the adversarial image: one-step targeted attack(PGD) with the target label being the most confusing class.

Bilateral Adversarial Training

Generate the adversarial label

Solve $\max_{y' \in \mathcal{S}_y} L(x, y'; \theta)$.

Instead of the one-step PGD method, they proposed a heuristic solution as follows.

Notations: $v_k = \nabla_{y_k} L(x, y; \theta), v_{MC} = \min_{k \neq c} v_k, v_{LL} = \max_{k \neq c} v_k$. Here "MC" (most confusing) corresponds to the non-grountruth class with the highest probability, and "LL" (least likely) corresponds to the non-grountruth class with the lowest probability.

$$y'_k = \frac{\epsilon_y}{n-1} \cdot \frac{v_k - v_{MC} + \gamma}{\frac{\sum_{k \neq c} v_k}{n-1} - v_{MC} + \gamma}, \quad k \neq c.$$

Here γ is a very small value, e.g., 0.01. Note that ϵ_y controls the perturbation budget of y.

Take CIFAR10 for example ($n = 10$). We have $\epsilon_y = 0.1$, $\beta = 81$, or $\epsilon_y = 0.5$, $\beta = 9$, or $\epsilon_y = 0.9$, $\beta = 1$.

Bilateral Adversarial Training

Generate the adversarial image

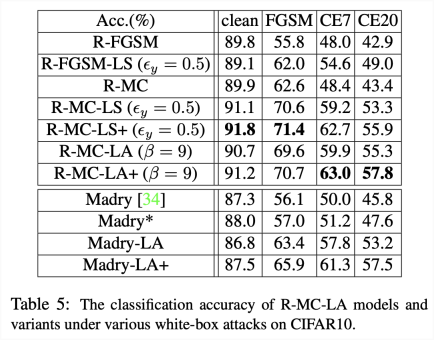
One-step targeted attack(PGD) with the target label being the most confusing class.

The most confusing class(MC): $y' = \text{argmin}_{\hat{y} \neq y} L(x, \hat{y}; \theta)$.

For targeted attacks, they simply replaced the groundtruth label y by the targeted label y, and also replace the plus sign by the minus sign, in order to minimize the loss with respect to the targeted label.

Bilateral Adversarial Training

Results-1

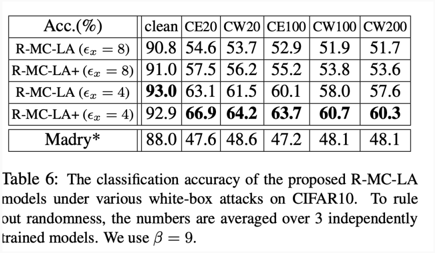


Acc.(%)	clean	FGSM	CE7	CE20
R-FGSM	89.8	55.8	48.0	42.9
R-FGSM-LS ($\epsilon_y = 0.5$)	89.1	62.0	54.6	49.0
R-MC	89.9	62.6	48.4	43.4
R-MC-LS ($\epsilon_y = 0.5$)	91.1	70.6	59.2	53.3
R-MC-LS+ ($\epsilon_y = 0.5$)	**91.8**	**71.4**	62.7	55.9
R-MC-LA ($\beta = 9$)	90.7	69.6	59.9	55.3
R-MC-LA+ ($\beta = 9$)	91.2	70.7	**63.0**	**57.8**
Madry [34]	87.3	56.1	50.0	45.8
Madry*	88.0	57.0	51.2	47.6
Madry-LA	86.8	63.4	57.8	53.2
Madry-LA+	87.5	65.9	61.3	57.5

Table 5: The classification accuracy of R-MC-LA models and variants under various white-box attacks on CIFAR10.

Bilateral Adversarial Training

Results-2

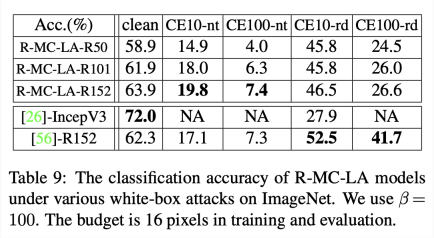


Acc.(%)	clean	CE20	CW20	CE100	CW100	CW200
R-MC-LA ($\epsilon_x = 8$)	90.8	54.6	53.7	52.9	51.9	51.7
R-MC-LA+ ($\epsilon_x = 8$)	91.0	57.5	56.2	55.2	53.8	53.6
R-MC-LA ($\epsilon_x = 4$)	**93.0**	63.1	61.5	60.1	58.0	57.6
R-MC-LA+ ($\epsilon_x = 4$)	92.9	**66.9**	**64.2**	**63.7**	**60.7**	**60.3**
Madry*	88.0	47.6	48.6	47.2	48.1	48.1

Table 6: The classification accuracy of the proposed R-MC-LA models under various white-box attacks on CIFAR10. To rule out randomness, the numbers are averaged over 3 independently trained models. We use $\beta = 9$.

Bilateral Adversarial Training

Results-3



Acc.(%)	clean	CE10-nt	CE100-nt	CE10-rd	CE100-rd
R-MC-LA-R50	58.9	14.9	4.0	45.8	24.5
R-MC-LA-R101	61.9	18.0	6.3	45.8	26.0
R-MC-LA-R152	63.9	**19.8**	**7.4**	46.5	26.6
[26]-IncepV3	**72.0**	NA	NA	27.9	NA
[56]-R152	62.3	17.1	7.3	**52.5**	**41.7**

Table 9: The classification accuracy of R-MC-LA models under various white-box attacks on ImageNet. We use $\beta = 100$. The budget is 16 pixels in training and evaluation.

Adversarial Training

Accelerated Adversarial Learning

Table of contents

YOPO

A NN–specific algorithm for adversarial defense based on the Pontryagin's Maximum Principle for differential game. They split the adversary computation and weighted updating and the adversary computation is focused on the first layer, finally achieved about 45 times speed up than the original PGD training.

YOPO

The Optimal Control Perspective and Differential Game:

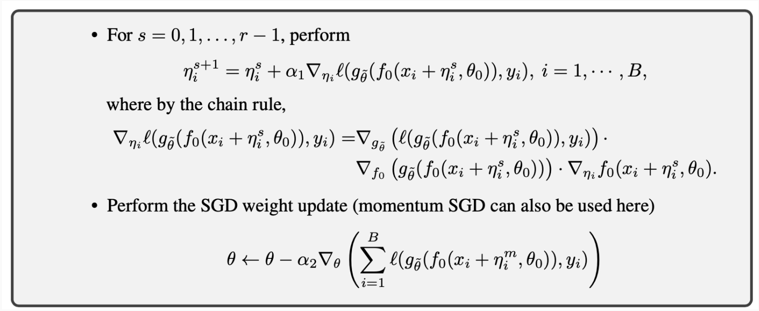
$$\min_\theta \max_{\|\eta_i\|_\infty \le \epsilon} J(\theta, \eta) := \tfrac{1}{N} \textstyle\sum_{i=1}^N \ell_i\left(x_{i,T}\right) + \tfrac{1}{N} \sum_{i=1}^N \sum_{t=0}^{T-1} R_t\left(x_{i,t}; \theta_t\right)$$
$$\text{subject to } x_{i,1} = f_0\left(x_{i,0} + \eta_i, \theta_0\right), i = 1, 2, \cdots, N$$
$$x_{i,t+1} = f_t\left(x_{i,t}, \theta_t\right), t = 1, 2, \cdots, T-1$$

Gradient based YOPO:

$$\min_\theta \max_{\|\eta_i\| \le \epsilon} \sum_{i=1}^B \ell\left(g_{\tilde\theta}\left(f_0\left(x_i + \eta_i, \theta_0\right)\right), y_i\right)$$

YOPO

PGD$-r$:



- For $s = 0, 1, \ldots, r-1$, perform

$$\eta_i^{s+1} = \eta_i^s + \alpha_1 \nabla_{\eta_i} \ell(g_{\tilde{\theta}}(f_0(x_i + \eta_i^s, \theta_0)), y_i), \ i = 1, \cdots, B,$$

  where by the chain rule,

$$\begin{aligned}\nabla_{\eta_i} \ell(g_{\tilde{\theta}}(f_0(x_i + \eta_i^s, \theta_0)), y_i) =& \nabla_{g_{\tilde{\theta}}} \left(\ell(g_{\tilde{\theta}}(f_0(x_i + \eta_i^s, \theta_0)), y_i)\right) \cdot \\ & \nabla_{f_0} \left(g_{\tilde{\theta}}(f_0(x_i + \eta_i^s, \theta_0))\right) \cdot \nabla_{\eta_i} f_0(x_i + \eta_i^s, \theta_0).\end{aligned}$$

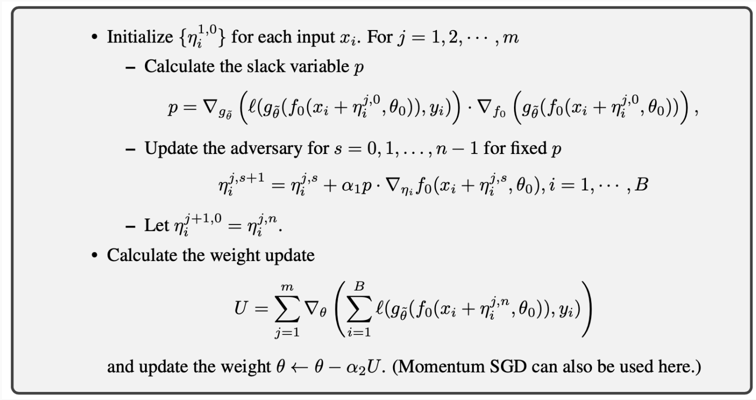
- Perform the SGD weight update (momentum SGD can also be used here)

$$\theta \leftarrow \theta - \alpha_2 \nabla_\theta \left(\sum_{i=1}^{B} \ell(g_{\tilde{\theta}}(f_0(x_i + \eta_i^m, \theta_0)), y_i) \right)$$

Note that this method conducts r sweeps of forward and backward propagation for each update of θ. This is the main reason why adversarial training using PGD-type algorithms can be very slow.

YOPO

YOPO$-m-n$:



- Initialize $\{\eta_i^{1,0}\}$ for each input x_i. For $j = 1, 2, \cdots, m$
  - Calculate the slack variable p

$$p = \nabla_{g_{\tilde{\theta}}} \left(\ell(g_{\tilde{\theta}}(f_0(x_i + \eta_i^{j,0}, \theta_0)), y_i) \right) \cdot \nabla_{f_0} \left(g_{\tilde{\theta}}(f_0(x_i + \eta_i^{j,0}, \theta_0)) \right),$$

  - Update the adversary for $s = 0, 1, \ldots, n-1$ for fixed p

$$\eta_i^{j,s+1} = \eta_i^{j,s} + \alpha_1 p \cdot \nabla_{\eta_i} f_0(x_i + \eta_i^{j,s}, \theta_0), i = 1, \cdots, B$$

  - Let $\eta_i^{j+1,0} = \eta_i^{j,n}$.
- Calculate the weight update

$$U = \sum_{j=1}^{m} \nabla_\theta \left(\sum_{i=1}^{B} \ell(g_{\tilde{\theta}}(f_0(x_i + \eta_i^{j,n}, \theta_0)), y_i) \right)$$

and update the weight $\theta \leftarrow \theta - \alpha_2 U$. (Momentum SGD can also be used here.)

Intuitively, YOPO freezes the values of the derivatives of the network at level $1, 2 \ldots, T1$ during the s-loop of the adversary updates.

YOPO

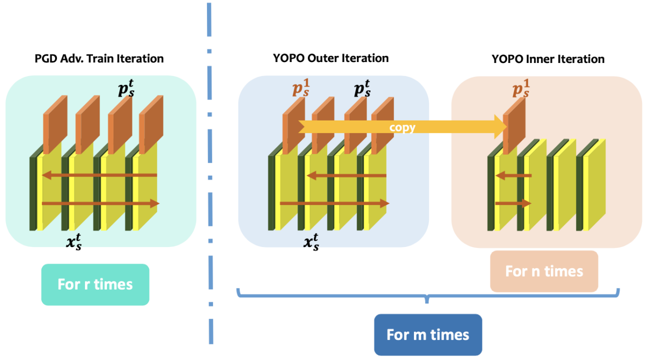
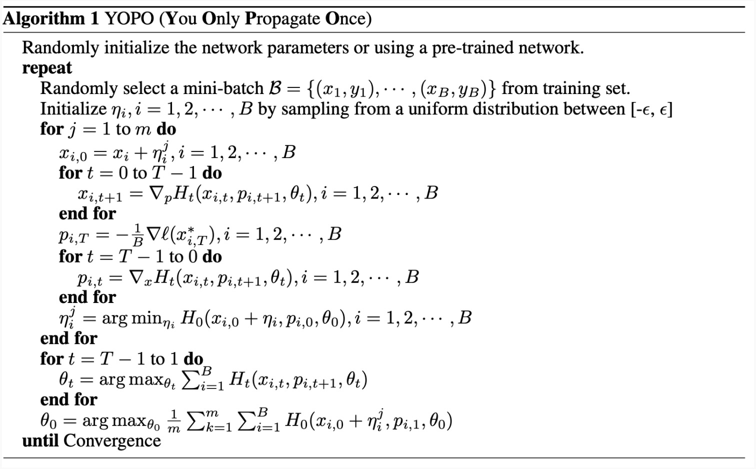


Figure 2: Pipeline of YOPO-m-n described in Algorithm 1. The yellow and olive blocks represent feature maps while the orange blocks represent the gradients of the loss w.r.t. feature maps of each layer.

YOPO



Algorithm 1 YOPO (**Y**ou **O**nly **P**ropagate **O**nce)

```
Randomly initialize the network parameters or using a pre-trained network.
repeat
    Randomly select a mini-batch $\mathcal{B} = \{(x_1, y_1), \cdots, (x_B, y_B)\}$ from training set.
    Initialize $\eta_i, i = 1, 2, \cdots, B$ by sampling from a uniform distribution between [-$\epsilon$, $\epsilon$]
    for $j = 1$ to $m$ do
        $x_{i,0} = x_i + \eta_i^j, i = 1, 2, \cdots, B$
        for $t = 0$ to $T - 1$ do
            $x_{i,t+1} = \nabla_p H_t(x_{i,t}, p_{i,t+1}, \theta_t), i = 1, 2, \cdots, B$
        end for
        $p_{i,T} = -\frac{1}{B} \nabla \ell(x_{i,T}^*), i = 1, 2, \cdots, B$
        for $t = T - 1$ to $0$ do
            $p_{i,t} = \nabla_x H_t(x_{i,t}, p_{i,t+1}, \theta_t), i = 1, 2, \cdots, B$
        end for
        $\eta_i^j = \arg\min_{\eta_i} H_0(x_{i,0} + \eta_i, p_{i,0}, \theta_0), i = 1, 2, \cdots, B$
    end for
    for $t = T - 1$ to $1$ do
        $\theta_t = \arg\max_{\theta_t} \sum_{i=1}^{B} H_t(x_{i,t}, p_{i,t+1}, \theta_t)$
    end for
    $\theta_0 = \arg\max_{\theta_0} \frac{1}{m} \sum_{k=1}^{m} \sum_{i=1}^{B} H_0(x_{i,0} + \eta_i^j, p_{i,1}, \theta_0)$
until Convergence
```

YOPO

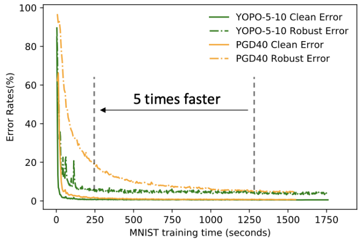
Theoretical understanding

Through the Pontryagin's Maximum Principle, they observed that the adversarial perturbation is only coupled with the first layer of neural networks.

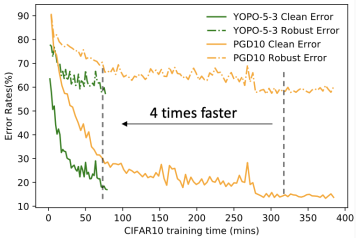
Some detailed theoretical results and theorems can be found in the paper.

YOPO

Result-1



(a) "Small CNN"[43] Result on MNIST

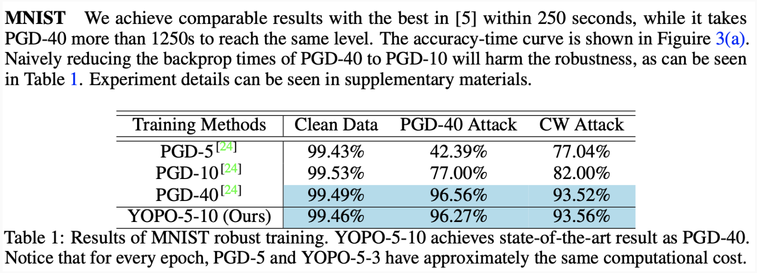


(b) PreAct-Res18 Results on CIFAR10

Figure 3: Performance w.r.t. training time

YOPO

Result-2



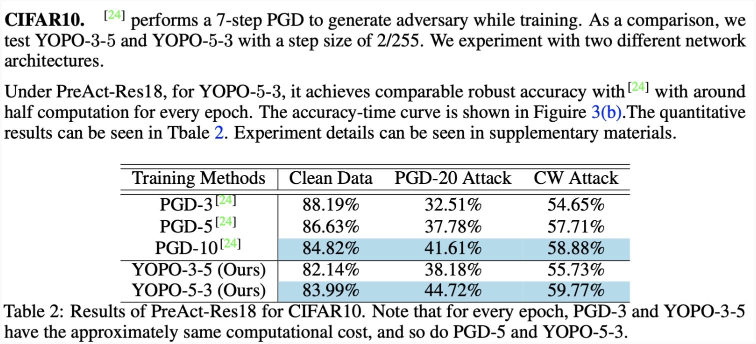
MNIST We achieve comparable results with the best in [5] within 250 seconds, while it takes PGD-40 more than 1250s to reach the same level. The accuracy-time curve is shown in Figuire 3(a). Naively reducing the backprop times of PGD-40 to PGD-10 will harm the robustness, as can be seen in Table 1. Experiment details can be seen in supplementary materials.

Training Methods	Clean Data	PGD-40 Attack	CW Attack
PGD-5[24]	99.43%	42.39%	77.04%
PGD-10[24]	99.53%	77.00%	82.00%
PGD-40[24]	99.49%	96.56%	93.52%
YOPO-5-10 (Ours)	99.46%	96.27%	93.56%

Table 1: Results of MNIST robust training. YOPO-5-10 achieves state-of-the-art result as PGD-40. Notice that for every epoch, PGD-5 and YOPO-5-3 have approximately the same computational cost.

YOPO

Result-3



CIFAR10. [24] performs a 7-step PGD to generate adversary while training. As a comparison, we test YOPO-3-5 and YOPO-5-3 with a step size of 2/255. We experiment with two different network architectures.

Under PreAct-Res18, for YOPO-5-3, it achieves comparable robust accuracy with[24] with around half computation for every epoch. The accuracy-time curve is shown in Figiure 3(b).The quantitative results can be seen in Tbale 2. Experiment details can be seen in supplementary materials.

Training Methods	Clean Data	PGD-20 Attack	CW Attack
PGD-3[24]	88.19%	32.51%	54.65%
PGD-5[24]	86.63%	37.78%	57.71%
PGD-10[24]	84.82%	41.61%	58.88%
YOPO-3-5 (Ours)	82.14%	38.18%	55.73%
YOPO-5-3 (Ours)	83.99%	44.72%	59.77%

Table 2: Results of PreAct-Res18 for CIFAR10. Note that for every epoch, PGD-3 and YOPO-3-5 have the approximately same computational cost, and so do PGD-5 and YOPO-5-3.