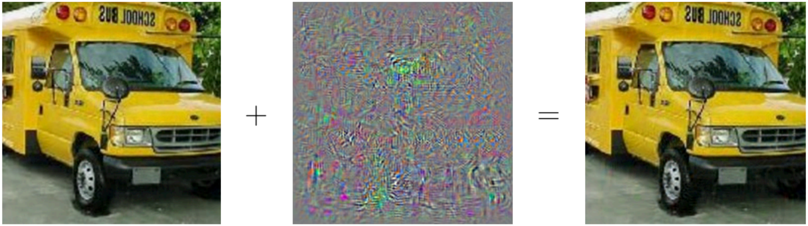
You Only Propagate Once:

Accelerating Adversarial Training via Maximal Principle

Table of Contents

Background

Adversarial Examples



Schoolbus + Perturbation = Ostrich

(rescaled for visualization)

(Szegedy et al, 2013)

Projected Gradient Descent (PGD) Attack

We denote PGD-r attack as doing

$$x^{t+1} = \Pi_{x+\mathcal{S}}\left(x^t + \alpha\,\mathrm{sign}\left(\nabla_x \ell(\theta, x, y)\right)\right)$$

for r times. $\Pi_{x+\mathcal{S}}$ denotes projection to some neighbourhood of x.

Remark

Perform a PGD-r attack requires around r times computation of normal backprop.

Adversarial Training

- We produce adversarial examples with PGD-r attack and use them as training data.

$$\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}} \max_{\|\eta\|\leq\epsilon} \ell(\theta; x+\eta, y), \tag{1}$$

Remark

This requires around r times computation of normal training.

An Optimal Control View: Adversarial Training as a Differential Game

An Optimal Control View of Deep Learning

Deep learning:

$$\begin{aligned} &\min_{\theta} J(\theta) = \ell(x_T) + \sum_{t=0}^{T-1} R_t(x_t; \theta_t) \\ &s.t. \quad x_{t+1} = f_t(x_t, \theta_t), t = 1, 2, \cdots, T-1 \end{aligned} \tag{2}$$

Optimal Control:

$$\begin{aligned} &\min_{\theta(\cdot)} J[\boldsymbol{\theta}(\cdot)] = \ell(\mathrm{x}(T)) + \int_0^T R(\mathrm{x}(t), \boldsymbol{\theta}(t)) dt \\ &s.t. \quad \dot{\mathrm{x}}(t) = \mathrm{f}(\mathrm{x}(t), \boldsymbol{\theta}(t)) \end{aligned} \tag{3}$$

$\theta(\cdot)$ is called a **control**

A Differential Game View of Adversarial Training

Adversarial Training:

$$\min_{\theta} \max_{\|\eta\| \leq \epsilon} J(\theta, \eta) = \ell(x_T) + \sum_{t=0}^{T-1} R_t(x_t; \theta_t, \eta_t)$$
$$s.t. \quad x_1 = f_0(x_0 + \eta, \theta_0), x_{t+1} = f_t(x_t, \theta_t), t = 1, 2, \cdots, T-1 \tag{4}$$

Differential Game:

$$\min_{\theta(\cdot)} \max_{\eta(\cdot)} J[\boldsymbol{\theta}(\cdot), \boldsymbol{\eta}(\cdot)] = \ell(\mathbf{x}(T)) + \int_0^T R(\mathbf{x}(t), \boldsymbol{\theta}(t), \boldsymbol{\eta}(t))dt$$
$$s.t. \quad \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \boldsymbol{\theta}(t), \boldsymbol{\eta}(t)) \tag{5}$$

Differential game is optimal control with 2 controls, **each having opposite target**.

- Pontryagin's Maximal Principle (PMP) is a neccesary condition for optimal control problem

Pontryagin's Maximal Principle (informal)

Define *Hamiltonian*

$$H(x, p, \theta, \eta) := p \cdot \mathbf{f}(x, \theta, \eta) + r(x, \theta, \eta)$$

PMP for differential game tells us there exists an **adjoint dynamic $\mathbf{p}(\cdot)$** satisfying :

$$\dot{\mathbf{x}}^*(t) = \nabla_p H\left(\mathbf{x}^*(t), \mathbf{p}^*(t), \boldsymbol{\theta}^*(t), \boldsymbol{\eta}^*(t)\right) \tag{6}$$

$$\dot{\mathbf{p}}^*(t) = -\nabla_x H\left(\mathbf{x}^*(t), \mathbf{p}^*(t), \boldsymbol{\theta}^*(t), \boldsymbol{\eta}^*(t)\right) \tag{7}$$

$$H\left(\mathbf{x}^*(t), \mathbf{p}^*(t), \boldsymbol{\theta}^*(t), \boldsymbol{\eta}\right) \geq H\left(\mathbf{x}^*(t), \mathbf{p}^*(t), \boldsymbol{\theta}^*(t), \boldsymbol{\eta}^*(t)\right) \tag{8}$$

$$\geq H\left(\mathbf{x}^*(t), \mathbf{p}^*(t), \boldsymbol{\theta}, \boldsymbol{\eta}^*(t)\right), \quad \forall t, \boldsymbol{\eta}, \boldsymbol{\theta} \tag{9}$$

(Here * means optimal situation)

Remark

Only in the first layer there exists η! After discretion, this naturally leads to "splitting" !

Practical Algorithm

1. use Euler scheme to approximate ODEs about $\mathsf{x}^*(t)$ and $\mathsf{p}^*(t)$
2. use SGD to approximate maximal principle of Hamiltonian
3. perform Jacobian iteration to satisfy each cpndition in PMP sequentially

Iteratively discrete the above three PMP conditions, we get the **general YOPO**:

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

---

repeat

Randomly select a mini-batch $\mathcal{B} = \{(x_1, y_1), \cdots, (x_B, y_B)\}$

Initialize $\eta_i, i = 1, 2, \cdots, B$

for $k = 1$ to m **do**

$x_{i,0} = x_i + \eta_i^k, i = 1, 2, \cdots, B$

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

$p_{i,T} = -\frac{1}{B}\nabla\ell(x^*_{i,T}), i = 1, 2, \cdots, B$

General YOPO ii

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

$\eta_i^k = \arg\min_{\eta_i} H_0(x_{i,0} + \eta_i, p_{i,0}, \theta_0), i = 1, 2, \cdots, B$

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

$\theta_0 = \arg\max_{\theta_0} \frac{1}{m} \sum_{k=1}^{m} \sum_{i=1}^{B} H_0(x_{i,0} + \eta_i^k, p_{i,1}, \theta_0)$

until Convergence

Remark

To fully satisfy the PMP conditions, we use a Jacobian approximation, iterating each data for m times. **This naturally leads to the usage of intermediate adversarial examples mentioned before!**

From General YOPO to Gradient Based YOPO

There are more profound results that PMP can lead to commonly used SGD, details can be found in paper

Theorem 1 (informal)
Training NN with SGD-based PMP is equivalent to normal deep learning training method.

Theorem 2 (informal)
Adversarial training NN with SGD-based PMP is equivalent to normal adversarial training method with a "splitting" trick after the first layer.

A Simpler Understanding of YOPO: Splitting PGD

A Splitting View of YOPO

- We split the first layer of network $f_0(\cdot, \theta_0)$ away from other layers $g_{\tilde{\theta}}(\cdot)$
- $\ell = \ell(g_{\tilde{\theta}}(f_0(\cdot)))$
- $\nabla_x \ell = \nabla_{f_0} \ell \cdot \nabla_x f_0 \triangleq p \cdot \nabla_x f_0$

YOPO-m-n

Algorithm 2 pseudocode for YOPO-m-n

1: initialize perturbation η
2: **for** $k = 1$ to m **do**
3: $\quad p \leftarrow \nabla_{f_0} \ell(x + \eta)$
4: $\quad$ **for** $i = 1$ to n **do**
5: $\quad\quad \eta \leftarrow \eta + \alpha \cdot p \cdot \nabla_x f_0(x + \eta)$ splitting
6: $\quad$ **end for**
7: $\quad$ accumulate gradient $U \leftarrow U + \nabla_\theta \ell(x + \eta)$
use intermediate adversarial examples
8: **end for**
9: Use U tp perform SGD / momentum SGD

Remark

Ignoring computation in Step 5, this is about m times computation of normal backprop, but we update perturbation η for $m \times n$ times!

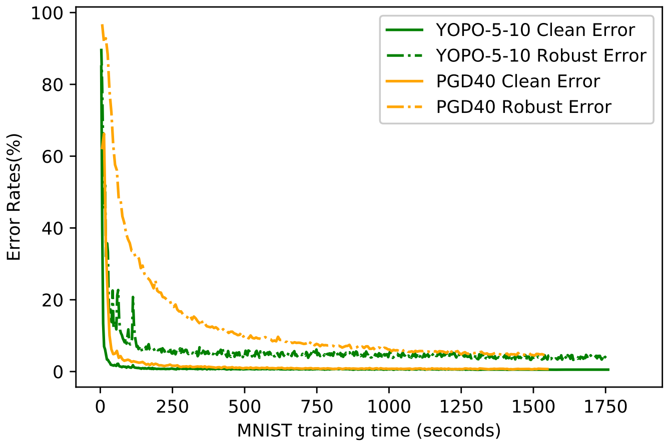
Merits of YOPO

1. Split the network
   Assuming p unchanged in inner iteration, YOPO increase update iteration number with slightly more computation
2. Use intermediate perturbation to update weights θ

These accelerate adversarial training quite a lot!

Empirical Effects

MNIST Results



CIFAR10 PreAct-Res18 Results

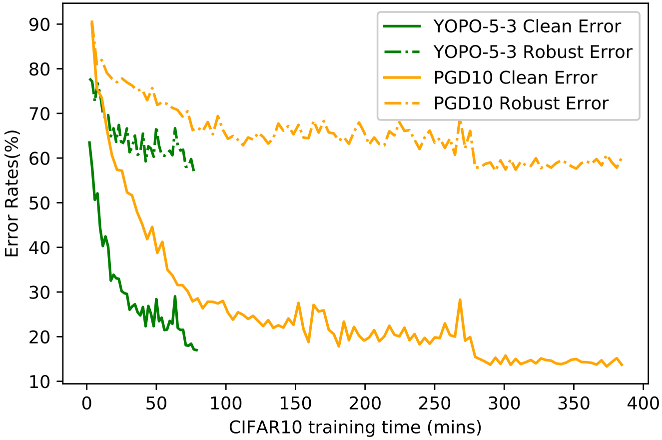


Figure 1: PreAct-Res18 Results on CIFAR10

CIFAR10 WideResNet34 Results

Training Methods	Clean Data	PGD-20 Attack	Training Time (mins)
Natural train	95.03%	0.00%	233
PGD-3	90.07%	39.18%	1134
PGD-5	89.65%	43.85%	1574
PGD-10	87.30%	47.04%	2713
Free-8 [1]	86.29%	47.00%	667
YOPO-3-5 (Ours)	87.27%	43.04%	299
YOPO-5-3 (Ours)	86.70%	47.98%	**476**

Table 1: Results of Wide ResNet34 for CIFAR10.

Thank you!