You Only Propagate Once: Accelerating Adversarial Training via Maximal Principle

Dinghuai Zhang*, **Tianyuan Zhang***, **Yiping Lu***
Peking University
{zhangdinghuai, 1600012888, luyiping9712}@pku.edu.cn

Zhanxing Zhu
School of Mathematical Sciences, Peking University
Center for Data Science, Peking University
Beijing Institute of Big Data Research
zhanxing.zhu@pku.edu.cn

Bin Dong
Beijing International Center for Mathematical Research, Peking University
Center for Data Science, Peking University
Beijing Institute of Big Data Research
dongbin@math.pku.edu.cn

Abstract

Deep learning achieves state-of-the-art results in many tasks in computer vision and natural language processing. However, recent works have shown that deep networks can be vulnerable to adversarial perturbations, which raised a serious robustness issue of deep networks. Adversarial training, typically formulated as a robust optimization problem, is an effective way of improving the robustness of deep networks. A major drawback of existing adversarial training algorithms is the computational overhead of the generation of adversarial examples, typically far greater than that of the network training. This leads to the unbearable overall computational cost of adversarial training. In this paper, we show that adversarial training can be cast as a discrete time differential game. Through analyzing the Pontryagin's Maximum Principle (PMP) of the problem, we observe that the adversary update is only coupled with the parameters of the first layer of the network. This inspires us to restrict most of the forward and back propagation within the first layer of the network during adversary updates. This effectively reduces the total number of full forward and backward propagation to only one for each group of adversary updates. Therefore, we refer to this algorithm YOPO (**Y**ou **O**nly **P**ropagate **O**nce). Numerical experiments demonstrate that YOPO can achieve comparable defense accuracy with **approximately 1/5 $\sim$ 1/4 GPU time** of the projected gradient descent (PGD) algorithm [15].[2]

1 Introduction

Deep neural networks achieve state-of-the-art performance on many tasks [16, 8]. However, recent works show that deep networks are often sensitive to adversarial perturbations [33, 25, 46], i.e.,

*Equal Contribution

[2]Our codes are available at https://github.com/a1600012888/YOPO-You-Only-Propagate-Once

Preprint. Under review.

changing the input in a way imperceptible to humans while causing the neural network to output an incorrect prediction. This poses significant concerns when applying deep neural networks to safety-critical problems such as autonomous driving and medical domains. To effectively defend the adversarial attacks, [24] proposed adversarial training, which can be formulated as a robust optimization [36]:

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

where θ is the network parameter, η is the adversarial perturbation, and (x, y) is a pair of data and label drawn from a certain distribution $\mathcal{D}$. The magnitude of the adversarial perturbation η is restricted by $\epsilon > 0$. For a given pair (x, y), we refer to the value of the inner maximization of (1), i.e. $\max_{\|\eta\|\leq\epsilon} \ell(\theta; x+\eta, y)$, as the adversarial loss which depends on (x, y).

A major issue of the current adversarial training methods is their significantly high computational cost. In adversarial training, we need to solve the inner loop, which is to obtain the "optimal" adversarial attack to the input in every iteration. Such "optimal" adversary is usually obtained using multi-step gradient decent, and thus the total time for learning a model using standard adversarial training method is much more than that using the standard training. Considering applying 40 inner iterations of projected gradient descent (PGD [15]) to obtain the adversarial examples, the computation cost of solving the problem (1) is about 40 times that of a regular training.

The main objective of this paper is to reduce the computational burden of adversarial training by limiting the number of forward and backward propagation without hurting the performance of the trained network. In this paper, we exploit the structures that the min-max objectives is encountered with deep neural networks. To achieve this, we formulate the adversarial training problem(1) as a differential game. Afterwards we can derive the Pontryagin's Maximum Principle (PMP) of the problem.

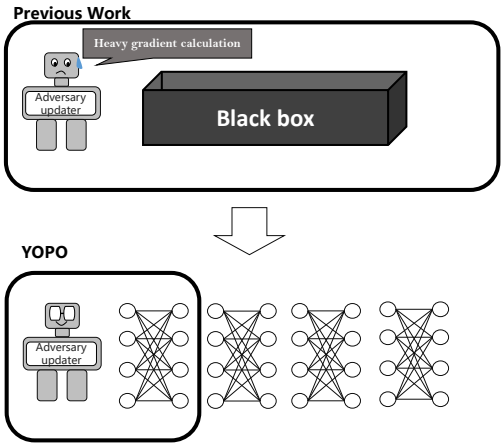


Figure 1: Our proposed YOPO expolits the structure of neural network. To alleviate the heavy computation cost, YOPO focuses the calculation of the adversary at the first layer.

From the PMP, we discover a key fact that the adversarial perturbation is only coupled with the weights of the first layer. This motivates us to propose a novel adversarial training strategy by decoupling the adversary update from the training of the network parameters. This effectively reduces the total number of full forward and backward propagation to only one for each group of adversary updates, significantly lowering the overall computation cost without hampering performance of the trained network. We name this new adversarial training algorithm as **YOPO** (**Y**ou **O**nly **P**ropagate **O**nce). Our numerical experiments show that YOPO achieves approximately 4 $\sim$5 times speedup over the original PGD adversarial training with comparable accuracy on MNIST/CIFAR10. Furthermore, we apply our algorithm to a recent proposed min max optimization objective "TRADES"[43] and achieve better clean and robust accuracy within half of the time TRADES need.

1.1 Related Works

Adversarial Defense. To improve the robustness of neural networks to adversarial examples, many defense strategies and models have been proposed, such as adversarial training [24], orthogonal regularization [6, 21], Bayesian method [42], TRADES [43], rejecting adversarial examples [41], Jacobian regularization [14, 27], generative model based defense [12, 31], pixel defense [29, 23], ordinary differential equation (ODE) viewpoint [44], ensemble via an intriguing stochastic differential equation perspective [37], and feature denoising [40, 32], etc. Among all these approaches, adversarial

training and its variants tend to be most effective since it largely avoids the the obfuscated gradient problem [2]. Therefore, in this paper, we choose adversarial training to achieve model robustness.

Neural ODEs. Recent works have built up the relationship between ordinary differential equations and neural networks [38, 22, 10, 5, 45, 35, 30]. Observing that each residual block of ResNet can be written as $u_{n+1} = u_n + \Delta t f(u_n)$, one step of forward Euler method approximating the ODE $u_t = f(u)$. Thus [19, 39] proposed an optimal control framework for deep learning and [5, 19, 20] utilize the adjoint equation and the maximal principle to train a neural network.

Decouple Training. Training neural networks requires forward and backward propagation in a sequential manner. Different ways have been proposed to decouple the sequential process by parallelization. This includes ADMM [34], synthetic gradients [13], delayed gradient [11], lifted machines [1, 18, 9]. Our work can also be understood as a decoupling method based on a splitting technique. However, we do not attempt to decouple the gradient w.r.t. network parameters but the adversary update instead.

1.2 Contribution

- To the best of our knowledge, it is the first attempt to design *NN–specific* algorithm for adversarial defense. To achieve this, we recast the adversarial training problem as a discrete time differential game. From optimal control theory, we derive the an optimality condition, *i.e.* the Pontryagin's Maximum Principle, for the differential game.
- Through PMP, we observe that the adversarial perturbation is only coupled with the first layer of neural networks. The PMP motivates a new adversarial training algorithm, YOPO. We split the adversary computation and weight updating and the adversary computation is focused on the first layer. Relations between YOPO and original PGD are discussed.
- We finally achieve about **4$\sim$ 5 times speed up** than the original PGD training with comparable results on MNIST/CIFAR10. Combining YOPO with TRADES[43], we achieve both higher clean and robust accuracy within less than half of the time TRADES need.

1.3 Organization

This paper is organized as follows. In Section 2, we formulate the robust optimization for neural network adversarial training as a differential game and propose the gradient based YOPO. In Section 3, we derive the PMP of the differential game, study the relationship between the PMP and the back-propagation based gradient descent methods, and propose a general version of YOPO. Finally, all the experimental details and results are given in Section 4.

2 Differential Game Formulation and Gradient Based YOPO

2.1 The Optimal Control Perspective and Differential Game

Inspired by the link between deep learning and optimal control [20], we formulate the robust optimization (1) as a differential game [7]. A two-player, zero-sum differential game is a game where each player controls a dynamics, and one tries to maximize, the other to minimize, a payoff functional. In the context of adversarial training, one player is the neural network, which controls the weights of the network to fit the label, while the other is the adversary that is dedicated to producing a false prediction by modifying the input.

The robust optimization problem (1) can be written as a differential game as follows,

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

Here, the dynamics $\{f_t(x_t, \theta_t), t = 0, 1, \ldots, T-1\}$ represent a deep neural network, T denote the number of layers, $\theta_t \in \Theta_t$ denotes the parameters in layer t (denote $\theta = \{\theta_t\}_t \in \Theta$), the function

$f_t : \mathbb{R}^{d_t} \times \Theta_t \to \mathbb{R}^{d_{t+1}}$ is a nonlinear transformation for one layer of neural network where d_t is the dimension of the t th feature map and $\{x_{i,0}, i = 1, \ldots, N\}$ is the training dataset. The variable $\eta = (\eta_1, \cdots, \eta_N)$ is the adversarial perturbation and we constrain it in an ∞-ball. Function ℓ_i is a data fitting loss function and R_t is the regularization weights θ_t such as the L_2-norm. By casting the problem of adversarial training as a differential game (2), we regard θ and η as two competing players, each trying to minimize/maximize the loss function $J(\theta, \eta)$ respectively.

2.2 Gradient Based YOPO

The Pontryagin's Maximum Principle (PMP) is a fundamental tool in optimal control that characterizes optimal solutions of the corresponding control problem [7]. PMP is a rather general framework that inspires a variety of optimization algorithms. In this paper, we will derive the PMP of the differential game (2), which motivates the proposed YOPO in its most general form. However, to better illustrate the essential idea of YOPO and to better address its relations with existing methods such as PGD, we present a special case of YOPO in this section based on gradient descent/ascent. We postpone the introduction of PMP and the general version of YOPO to Section 3.

Let us first rewrite the original robust optimization problem (1) (in a mini-batch form) as

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

where f_0 denotes the first layer, $g_{\tilde{\theta}} = f_{T-1}^{\theta_{T-1}} \circ f_{T-2}^{\theta_{T-2}} \circ \cdots f_1^{\theta_1}$ denotes the network without the first layer and B is the batch size. Here $\tilde{\theta}$ is defined as $\{\theta_1, \cdots, \theta_{T-1}\}$. For simplicity we omit the regularization term R_t.

The simplest way to solve the problem is to perform gradient ascent on the input data and gradient descent on the weights of the neural network as shown below. Such alternating optimization algorithm is essentially the popular PGD adversarial training [24]. We summarize the PGD-r (for each update on θ) as follows, i.e. performing r iterations of gradient ascent for inner maximization.

- 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.

To reduce the total number of forward and backward propagation, we introduce a slack variable

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

and freeze it as a constant within the inner loop of the adversary update. The modified algorithm is given below and we shall refer to it as 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 $1, 2 \ldots, T-1$ during the s-loop of the adversary updates. Figure 2 shows the conceptual comprison between YOPO and PGD. YOPO-m-n accesses the data $m \times n$ times while only requires m full forward and backward propagation. PGD-r, on the other hand, propagates the data r times for r full forward and backward propagation. As one can see that, YOPO-m-n has the flexibility of increasing n and reducing m to achieve approximately the same level of attack but with much less computation cost. For example, suppose one applies PGD-10 (i.e. 10 steps of gradient ascent for solving the inner maximization) to calculate the adversary. An alternative approach is using YOPO-5-2 which also accesses the data 10 times but the total number of full forward propagation is only 5. Empirically, YOPO-m-n achieves comparable results only requiring setting $m \times n$ a litter larger than r.

Another benefit of YOPO is that we take full advantage of every forward and backward propagation to update the weights, i.e. the intermediate perturbation $\eta_i^j, j = 1, \cdots, m-1$ are not wasted like PGD-r. This allows us to perform multiple updates per iteration, which potentially drives YOPO to converge faster in terms of the number of epochs. Combining the two factors together, YOPO significantly could accelerate the standard PGD adversarial training.

We would like to point out a concurrent paper [28] that is related to YOPO. Their proposed method, called "Free-m", also can significantly speed up adversarial training. In fact, Free-m is essentially YOPO-m-1, except that YOPO-m-1 delays the weight update after the whole mini-batch is processed in order for a proper usage of momentum [3].

3 The Pontryagin's Maximum Principle for Adversarial Training

In this section, we present the PMP of the discrete time differential game (2). From the PMP, we can observe that the adversary update and its associated back-propagation process can be decoupled. Furthermore, back-propagation based gradient descent can be understood as an iterative algorithm solving the PMP and with that the version of YOPO presented in the previous section can be viewed as an algorithm solving the PMP. However, the PMP facilitates a much wider class of algorithms than gradient descent algorithms [19]. Therefore, we will present a general version of YOPO based on the PMP for the discrete differential game.

3.1 PMP

Pontryagin type of maximal principle [26, 3] provides necessary conditions for optimality with a layer-wise maximization requirement on the Hamiltonian function. For each layer $t \in [T] :=$

[3] Momentum should be accumulated between mini-batches other than different adversarial examples from one mini-batch, otherwise overfitting will become a serious problem.

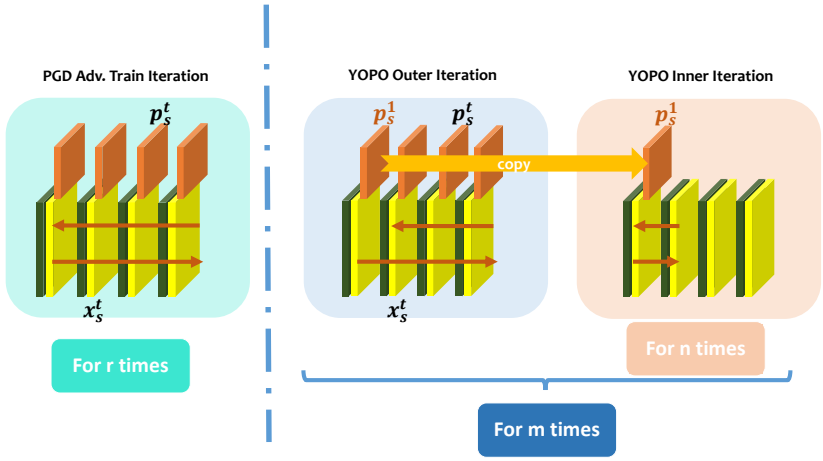


Figure 2: Pipeline of YOPO-m-n described in Algorithm 1. The yellow and olive blocks represent feature maps while the orange blocks represent the gradients of the loss w.r.t. feature maps of each layer.

$\{0, 1, \ldots, T-1\}$, we define the Hamiltonian function $H_t : \mathbb{R}^{d_t} \times \mathbb{R}^{d_{t+1}} \times \Theta_t \to \mathbb{R}$ as

$$H_t(x, p, \theta_t) = p \cdot f_t(x, \theta_t) - \frac{1}{B} R_t(x, \theta_t).$$

The PMP for continuous time differential game has been well studied in the literature [7]. Here, we present the PMP for our discrete time differential game (2).

Theorem 1. *(PMP for adversarial training) Assume ℓ_i is twice continuous differentiable, $f_t(\cdot, \theta), R_t(\cdot, \theta)$ are twice continuously differentiable with respect to x; $f_t(\cdot, \theta), R_t(\cdot, \theta)$ together with their x partial derivatives are uniformly bounded in t and θ, and the sets $\{f_t(x, \theta) : \theta \in \Theta_t\}$ and $\{R_t(x, \theta) : \theta \in \Theta_t\}$ are convex for every t and $x \in \mathbb{R}^{d_t}$. Denote θ^* as the solution of the problem (2), then there exists co-state processes $p_i^* := \{p_{i,t}^* : t \in [T]\}$ such that the following holds for all $t \in [T]$ and $i \in [B]$:*

$$x_{i,t+1}^* = \nabla_p H_t(x_{i,t}^*, p_{i,t+1}^*, \theta_t^*), \qquad x_{i,0}^* = x_{i,0} + \eta_i^* \tag{3}$$

$$p_{i,t}^* = \nabla_x H_t(x_{i,t}^*, p_{i,t+1}^*, \theta_t^*), \qquad p_{i,T}^* = -\frac{1}{B} \nabla \ell_i(x_{i,T}^*) \tag{4}$$

At the same time, the parameters of the first layer $\theta_0^* \in \Theta_0$ and the optimal adversarial perturbation η_i^* satisfy

$$\sum_{i=1}^{B} H_0(x_{i,0}^* + \eta_i, p_{i,1}^*, \theta_0^*) \geq \sum_{i=1}^{B} H_0(x_{i,0}^* + \eta_i^*, p_{i,1}^*, \theta_0^*) \geq \sum_{i=1}^{B} H_0(x_{i,0}^* + \eta_i^*, p_{i,1}^*, \theta_0), \tag{5}$$

$$\forall \theta_0 \in \Theta_0, \|\eta_i\|_\infty \leq \epsilon \tag{6}$$

and the parameters of the other layers $\theta_t^* \in \Theta_t, t \in [T]$ maximize the Hamiltonian functions

$$\sum_{i=1}^{B} H_t(x_{i,t}^*, p_{i,t+1}^*, \theta_t^*) \geq \sum_{i=1}^{B} H_t(x_{i,t}^*, p_{i,t+1}^*, \theta_t), \ \forall \theta_t \in \Theta_t \tag{7}$$

Proof. Proof is in the supplementary materials. □

From the theorem, we can observe that the adversary η is only coupled with the parameters of the first layer θ_0. This key observation inspires the design of YOPO.

3.2 PMP and Back-Propagation Based Gradient Descent

The classical back-propagation based gradient descent algorithm [17] can be viewed as an algorithm attempting to solve the PMP. Without loss of generality, we can let the regularization term $R = 0$, since we can simply add an extra dynamic w_t to evaluate the regularization term R, *i.e.*

$$w_{t+1} = w_t + R_t(x_t, \theta_t),\ w_0 = 0.$$

We append w to x to study the dynamics of a new $(d_t + 1)$-dimension vector and change $f_t(x, \theta_t)$ to $(f_t(x, \theta_t), w + R_t(x, \theta_t))$. The relationship between the PMP and the back-propagation based gradient descent method was first observed by Li et al. [19]. They showed that the forward dynamical system Eq.(3) is the same as the neural network forward propagation. The backward dynamical system Eq.(4) is the back-propagation, which is formally described by the following lemma.

Lemma 1.

$$p_t^* = \nabla_x H_t(x_t^*, p_{t+1}^*, \theta_t^*) = \nabla_x f(x_t^*, \theta_t^*)^T p_{t+1} = (\nabla_{x_t} x_{t+1}^*)^T \cdot -\nabla_{x_{t+1}}(\ell(x_T)) = -\nabla_{x_t}(\ell(x_T)).$$

To solve the maximization of the Hamiltonian, a simple way is the gradient ascent:

$$\theta_t^1 = \theta_t^0 + \alpha \cdot \nabla_\theta \sum_{i=1}^{B} H_t(x_{i,t}^{\theta^0}, p_{i,t+1}^{\theta^0}, \theta_t^0). \tag{8}$$

Theorem 2. *The update (8) is equivalent to gradient descent method for training networks[19, 20].*

3.3 YOPO from PMP's View Point

Based on the relationship between back-propagation and the Pontryagin's Maximum Principle, in this section, we provide a new understanding of YOPO, *i.e.* solving the PMP for the differential game. Observing that, in the PMP, the adversary η is only coupled with the weight of the first layer θ_0. Thus we can update the adversary via minimizing the Hamiltonian function instead of directly attacking the loss function, described in Algorithm 1.

For YOPO-m-n, to approximate the exactly minimization of the Hamiltonian, we perform n times gradient descent to update the adversary. Furthermore, in order to make the calculation of the adversary more accurate, we iteratively pass one data point m times. Besides, the network weights are optimized via performing the gradient ascent to Hamiltonian, resulting in the gradient based YOPO proposed in Section 2.2.

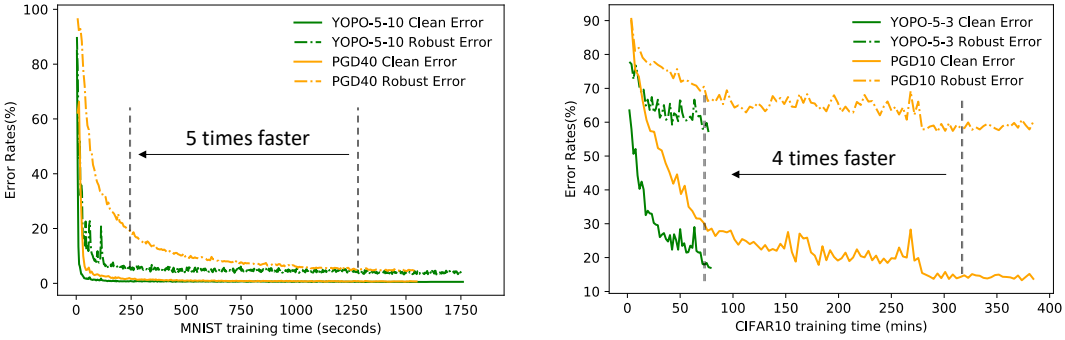
Algorithm 1 YOPO (**Y**ou **O**nly **P**ropagate **O**nce)

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 [-ϵ, ϵ]
  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

4 Experiments

4.1 YOPO for Adversarial Training

To demonstrate the effectiveness of YOPO, we conduct experiments on MNIST and CIFAR10. We find that the models trained with YOPO have comparable performance with that of the PGD adversarial training, but with a much fewer computational cost. We also compare our method with a concurrent method "For Free"[28], and the result shows that our algorithm can achieve comparable performance with around 2/3 GPU time of their official implementation.



(a) "Samll CNN" [43] Result on MNIST (b) PreAct-Res18 Results on CIFAR10

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

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.

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 Figuire 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.

As for Wide ResNet34, YOPO-5-3 still achieves similar acceleration against PGD-10, as shown in Table 3. We also test PGD-3/5 to show that naively reducing backward times for this minmax problem [24] cannot produce comparable results within the same computation time as YOPO. Meanwhile, YOPO-3-5 can achieve more aggressive speed-up with only a slight drop in robustness.

Training Methods	Clean Data	PGD-20 Attack	Training Time (mins)
Natural train	95.03%	0.00%	233
PGD-3 [24]	90.07%	39.18%	1134
PGD-5 [24]	89.65%	43.85%	1574
PGD-10 [24]	87.30%	47.04%	2713
Free-8 [28][1]	86.29%	47.00%	667
YOPO-3-5 (Ours)	87.27%	43.04%	299
YOPO-5-3 (Ours)	86.70%	47.98%	**476**

[1] Code from https://github.com/ashafahi/free_adv_train.

Table 3: Results of Wide ResNet34 for CIFAR10.

4.2 YOPO for TRADES

TRADES[43] formulated a new min-max objective function of adversarial defense and achieves the state-of-the-art adversarial defense results. The details of algorithm and experiment setup are in supplementary material, and quantitative results are demonstrated in Table 4.

Training Methods	Clean Data	PGD-20 Attack	CW Attack	Training Time (mins)
TRADES-10[43]	86.14%	44.50%	58.40%	633
TRADES-YOPO-3-4 (Ours)	87.82%	46.13%	59.48%	259
TRADES-YOPO-2-5 (Ours)	88.15%	42.48%	59.25%	218

Table 4: Results of training PreAct-Res18 for CIFAR10 with TRADES objective

5 Conclusion

In this work, we have developed an efficient strategy for accelerating adversarial training. We recast the adversarial training of deep neural networks as a discrete time differential game and derive a Pontryagin's Maximum Principle (PMP) for it. Based on this maximum principle, we discover that the adversary is only coupled with the weights of the first layer. This motivates us to split the adversary updates from the back-propagation gradient calculation. The proposed algorithm, called YOPO, avoids computing full forward and backward propagation for too many times, thus effectively reducing the computational time as supported by our experiments.

References

[1] Armin Askari, Geoffrey Negiar, Rajiv Sambharya, and Laurent El Ghaoui. Lifted neural networks. *arXiv preprint arXiv:1805.01532*, 2018.

[2] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

[3] Vladimir Grigor'evich Boltyanskii, Revaz Valer'yanovich Gamkrelidze, and Lev Semenovich Pontryagin. The theory of optimal processes. i. the maximum principle. Technical report, TRW SPACE TECHNOLOGY LABS LOS ANGELES CALIF, 1960.

[4] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE, 2017.

[5] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6572–6583, 2018.

[6] Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 854–863. JMLR. org, 2017.

[7] Lawrence C Evans. An introduction to mathematical optimal control theory. *Lecture Notes, University of California, Department of Mathematics, Berkeley*, 2005.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[9] Fangda Gu, Armin Askari, and Laurent El Ghaoui. Fenchel lifted networks: A lagrange relaxation of neural network training. *arXiv preprint arXiv:1811.08039*, 2018.

[10] Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, 2017.

[11] Zhouyuan Huo, Bin Gu, Qian Yang, and Heng Huang. Decoupled parallel backpropagation with convergence guarantee. *arXiv preprint arXiv:1804.10574*, 2018.

[12] Andrew Ilyas, Ajil Jalal, Eirini Asteri, Constantinos Daskalakis, and Alexandros G Dimakis. The robust manifold defense: Adversarial training using generative models. *arXiv preprint arXiv:1712.09196*, 2017.

[13] Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1627–1635. JMLR. org, 2017.

[14] Daniel Jakubovitz and Raja Giryes. Improving dnn robustness to adversarial attacks using jacobian regularization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 514–529, 2018.

[15] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[16] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[17] Yann LeCun, D Touresky, G Hinton, and T Sejnowski. A theoretical framework for back-propagation. In *Proceedings of the 1988 connectionist models summer school*, volume 1, pages 21–28. CMU, Pittsburgh, Pa: Morgan Kaufmann, 1988.

[18] Jia Li, Cong Fang, and Zhouchen Lin. Lifted proximal operator machines. *arXiv preprint arXiv:1811.01501*, 2018.

[19] Qianxiao Li, Long Chen, Cheng Tai, and E Weinan. Maximum principle based algorithms for deep learning. *The Journal of Machine Learning Research*, 18(1):5998–6026, 2017.

[20] Qianxiao Li and Shuji Hao. An optimal control approach to deep learning and applications to discrete-weight neural networks. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2985–2994, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[21] Ji Lin, Chuang Gan, and Song Han. Defensive quantization: When efficiency meets robustness. In *International Conference on Learning Representations*, 2019.

[22] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. *arXiv preprint arXiv:1710.10121*, 2017.

[23] Tiange Luo, Tianle Cai, Mengxiao Zhang, Siyu Chen, and Liwei Wang. RANDOM MASK: Towards robust convolutional neural networks, 2019.

[24] 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.

[25] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582, 2016.

[26] Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. CRC, 1987.

[27] Haifeng Qian and Mark N Wegman. L2-nonexpansive neural networks. *arXiv preprint arXiv:1802.07896*, 2018.

[28] Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Xu Zeng, John Dickerson, Christoph Studer, Larry S. Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! *arXiv preprint arXiv:1904.12843*, 2019.

[29] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. *arXiv preprint arXiv:1710.10766*, 2017.

[30] Sho Sonoda and Noboru Murata. Transport analysis of infinitely deep neural network. *The Journal of Machine Learning Research*, 20(1):31–82, 2019.

[31] Ke Sun, Zhanxing Zhu, and Zhouchen Lin. Enhancing the robustness of deep neural networks by boundary conditional gan. *arXiv preprint arXiv:1902.11029*, 2019.

[32] Jan Svoboda, Jonathan Masci, Federico Monti, Michael Bronstein, and Leonidas Guibas. Peernets: Exploiting peer wisdom against adversarial attacks. In *International Conference on Learning Representations*, 2019.

[33] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[34] Gavin Taylor, Ryan Burmeister, Zheng Xu, Bharat Singh, Ankit Patel, and Tom Goldstein. Training neural networks without gradients: A scalable admm approach. In *International conference on machine learning*, pages 2722–2731, 2016.

[35] Matthew Thorpe and Yves van Gennip. Deep limits of residual neural networks. *arXiv preprint arXiv:1810.11741*, 2018.

[36] Abraham Wald. Contributions to the theory of statistical estimation and testing hypotheses. *The Annals of Mathematical Statistics*, 10(4):299–326, 1939.

[37] Bao Wang, Binjie Yuan, Zuoqiang Shi, and Stanley J Osher. Enresnet: Resnet ensemble via the feynman-kac formalism. *arXiv preprint arXiv:1811.10745*, 2018.

[38] E Weinan. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 2017.

[39] E Weinan, Jiequn Han, and Qianxiao Li. A mean-field optimal control formulation of deep learning. *Research in the Mathematical Sciences*, 6(1):10, 2019.

[40] Cihang Xie, Yuxin Wu, Laurens van der Maaten, Alan Yuille, and Kaiming He. Feature denoising for improving adversarial robustness. *arXiv preprint arXiv:1812.03411*, 2018.

[41] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017.

[42] Nanyang Ye and Zhanxing Zhu. Bayesian adversarial learning. In *Advances in Neural Information Processing Systems*, pages 6892–6901, 2018.

[43] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P Xing, Laurent El Ghaoui, and Michael I Jordan. Theoretically principled trade-off between robustness and accuracy. *arXiv preprint arXiv:1901.08573*, 2019.

[44] Jingfeng Zhang, Bo Han, Laura Wynter, Kian Hsiang Low, and Mohan Kankanhalli. Towards robust resnet: A small step but a giant leap. *arXiv preprint arXiv:1902.10887*, 2019.

[45] Xiaoshuai Zhang, Yiping Lu, Jiaying Liu, and Bin Dong. Dynamically unfolding recurrent restorer: A moving endpoint control method for image restoration. In *International Conference on Learning Representations*, 2019.

[46] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2847–2856. ACM, 2018.

A Proof Of The Theorems

A.1 Proof of Theorem 1

In this section we give the full statement of the maximum principle for the adversarial training and present a proof. Let's start from the case of the natural training of neural networks.

Theorem. *(PMP for adversarial training) Assume ℓ_i is twice continuous differentiable, $f_t(\cdot,\theta), R_t(\cdot,\theta)$ are twice continuously differentiable with respect to x, and $f_t(\cdot,\theta), R_t(\cdot,\theta)$ together with their x partial derivatives are uniformly bounded in t and θ. The sets $\{f_t(x,\theta) : \theta \in \Theta_t\}$ and $\{R_t(x,\theta) : \theta \in \Theta_t\}$ are convex for every t and $x \in \mathbb{R}^{d_t}$. Let θ^* to be the solution of*

$$\min_{\theta\in\Theta} \max_{\|\eta\|_\infty \le \epsilon} J(\theta,\eta) := \frac{1}{N}\sum_{i=1}^{N} \ell_i(x_{i,T}) + \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T-1} R_t(x_{i,t},\theta_t) \tag{9}$$

$$\text{subject to } x_{i,1} = f_0(x_{i,0}+\eta_i;\theta_0), i = 1,2,\cdots,N \tag{10}$$

$$x_{i,t+1} = f_t(x_{i,t},\theta_t), t = 1,2,\cdots,T-1. \tag{11}$$

Then there exists co-state processes $p_i^* := p_{i,t}^* : t = 0,\cdots,T$ such that the following holds for all $t \in [T]$ and $i \in [N]$:

$$x_{i,t+1}^* = \nabla_p H_t(x_{i,t}^*, p_{i,t+1}^*, \theta_t^*), \qquad x_{i,0}^* = x_{i,0} + \eta_i^* \tag{12}$$

$$p_{i,t}^* = \nabla_x H_t(x_{i,t}^*, p_{i,t+1}^*, \theta_t^*), \qquad p_{i,T}^* = -\frac{1}{N}\nabla \ell_i(x_{i,T}^*) \tag{13}$$

Here H is the per-layer defined Hamiltonian function $H_t : \mathbb{R}^{d_t} \times \mathbb{R}^{d_{t+1}} \times \Theta_t \to \mathbb{R}$ as

$$H_t(x,p,\theta_t) = p \cdot f_t(x,\theta_t) - \frac{1}{N} R_t(x,\theta_t)$$

At the same time, the parameter of the first layer $\theta_0^* \in \Theta_0$ and the best perturbation η^* satisfy

$$\sum_{i=1}^{N} H_0(x_{i,0}^* + \eta_i, p_{i,1}^*, \theta_0^*) \geq \sum_{i=1}^{N} H_0(x_{i,0}^* + \eta_i^*, p_{i,1}^*, \theta_0^*) \geq \sum_{i=1}^{N} H_0(x_{i,0}^* + \eta_i^*, p_{i,1}^*, \theta_0), \forall \theta_0 \in \Theta_0, \|\eta_i\|_\infty \leq \epsilon \tag{14}$$

while parameter of the other layers $\theta_t^* \in \Theta_t, t = 1, 2, \cdots, T-1$ will maximize the Hamiltonian functions

$$\sum_{i=1}^{N} H_t(x_{i,t}^*, p_{i,t+1}^*, \theta_t^*) \geq \sum_{i=1}^{N} H_t(x_{i,t}^*, p_{i,t+1}^*, \theta_t), \forall \theta_t \in \Theta_t \tag{15}$$

Proof. We first propose PMP for discrete time dynamic system and utilize it directly gives out the proof of PMP for adversarial training.

Lemma 2. *(PMP for discrete time dynamic system) Assume ℓ is twice continuous differentiable, $f_t(\cdot, \theta), R_t(\cdot, \theta)$ are twice continuously differentiable with respect to x, and $f_t(\cdot, \theta), R_t(\cdot, \theta)$ together with their x partial derivatives are uniformly bounded in t and θ. The sets $\{f_t(x, \theta) : \theta \in \Theta_t\}$ and $\{R_t(x, \theta) : \theta \in \Theta_t\}$ are convex for every t and $x \in \mathbb{R}^{d_t}$. Let θ^* to be the solution of*

$$\min_{\theta \in \Theta} \max_{\|\eta\|_\infty \leq \epsilon} J(\theta, \eta) := \frac{1}{N} \sum_{i=1}^{N} \ell_i(x_{i,T}) + \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T-1} R_t(x_{i,t}, \theta_t) \tag{16}$$

$$\text{subject to } x_{i,t+1} = f_t(x_{i,t}, \theta_t), i \in [N], t = 0, 1, \cdots, T-1. \tag{17}$$

There exists co-state processes $p_i^* := p_{i,t}^* : t = 0, \cdots, T$ such that the following holds for all $t \in [T]$ and $i \in [N]$:

$$x_{i,t+1}^* = \nabla_p H_t(x_{i,t}^*, p_{i,t+1}^*, \theta_t^*), \qquad x_{i,0}^* = x_{i,0} \tag{18}$$

$$p_{i,t}^* = \nabla_x H_t(x_{i,t}^*, p_{i,t+1}^*, \theta_t^*), \qquad p_{i,T}^* = -\frac{1}{N} \nabla \ell_i(x_{i,T}^*) \tag{19}$$

Here H is the per-layer defined Hamiltonian function $H_t : \mathbb{R}^{d_t} \times \mathbb{R}^{d_{t+1}} \times \Theta_t \to \mathbb{R}$ as

$$H_t(x, p, \theta_t) = p \cdot f_t(x, \theta_t) - \frac{1}{N} R_t(x, \theta_t)$$

The parameters of the layers $\theta_t^* \in \Theta_t, t = 0, 1, \cdots, T-1$ will maximize the Hamiltonian functions

$$\sum_{i=1}^{N} H_t(x_{i,t}^*, p_{i,t+1}^*, \theta_t^*) \geq \sum_{i=1}^{N} H_t(x_{i,t}^*, p_{i,t+1}^*, \theta_t), \forall \theta_t \in \Theta_t \tag{20}$$

Proof. Without loss of generality, we let $L = 0$. The reason is that we can simply add an extra dynamic w_t to calculate the regularization term R, *i.e.*

$$w_{t+1} = w_t + R_t(x_t, \theta_t), w_0 = 0.$$

We append w to x to study the dynamic of a new $d_t + 1$ dimension vector and modify $f_t(x, \theta)$ to $(f_t(x, \theta), w + R_t(x, \theta))$. Thus we only need to prove the case when $L = 0$.

For simplicity, we omit the subscript s in the following proof. (Concatenating all x_s into $x = (x_1, \ldots, x_N)$ can justify this.)

Now we begin the proof. Following the linearization lemma in [20], consider the linearized problem

$$\phi_{t+1} = f_t(x_t^*, \theta_t) + \nabla_x f_t(x_t^*, \theta_t)(\phi_t - x_t^*), \phi_0 = x_0 + \eta. \tag{21}$$

The reachable states by the linearized dynamic system is denoted as

$$W_t := \{x \in \mathbb{R}^{d_t} : \exists \theta, \eta = \eta^* \text{ s.t. } \phi_t^\theta = x\}$$

here x_t^θ denotes the the evolution of the dynamical system for x_t under θ. We also define

$$S := \{x \in \mathbb{R}^{d_T} : (x - x_T^*) \nabla \ell(x_T^*) < 0\}$$

The linearization lemma in [20] tells us that $\boldsymbol{W_T}$ and $\boldsymbol{S}$ are separated by $\{x : p_T^* \cdot (x - x_T^*) = 0, p_T^* = -\nabla \ell(x_T^*)\}$, *i.e.*

$$p_T^* \cdot (x - x_T^*) \leq 0, \forall x \in W_t. \tag{22}$$

Thus setting

$$p_t^* = \nabla_x H_t(x_t^*, p_{t+1}^*, \theta_t^*) = \nabla_x f(x_t^*, \theta_t^*)^T \cdot p_{t+1}^*,$$

we have

$$(\phi_{t+1} - x_{t+1}^*) \cdot p_t^* = (\phi_t - x_t^*) \cdot p_t^*. \tag{23}$$

Thus from Eq.22 and Eq.23 we get

$$p_{t+1}^* \cdot (\phi_{t+1}^\theta - x_{t+1}^*) \leq 0, \quad t = 0, \cdots, T-1, \forall \theta \in \Theta := \Theta_0 \times \Theta_1 \times \cdots$$

Setting $\theta_s = \theta_s^*$ for $s < t$ we have $\phi_{t+1}^\theta = f_t(x_t^*, \theta_t)$, which leads to $p_{t+1}^* \cdot (f_t(x_t^*, \theta_t) - x_{t+1}^*) \leq 0$. This finishes the proof of the maximal principle on weight space Θ.

□

We return to the proof of the theorem. The proof of the maximal principle on the weight space, *i.e.*

$$\sum_{i=1}^N H_t(x_{i,t}^*, p_{i,t+1}^*, \theta_t^*) \geq \sum_{i=1}^N H_t(x_{i,t}^*, p_{i,t+1}^*, \theta), \forall \theta_t \in \Theta_t, t = 1, 2, \cdots, T-1$$

and

$$\sum_{i=1}^N H_0(x_{i,0}^* + \eta_i^*, p_{i,1}^*, \theta_0^*) \geq \sum_{i=1}^N H_0(x_{i,0}^* + \eta_i^*, p_{i,1}^*, \theta_0), \forall \theta_0 \in \Theta_0,$$

can be reached with the help of Lemma 2: replacing the dynamic start point $x_{i,0}$ in Eq.18 with $x_{i,0} + \eta_i^*$ makes this maximal principle a direct corollary of Lemma 2.

Next, we prove the Hamiltonian conidition for the adversary, *i.e.*

$$\sum_{i=1}^N H_0(x_{i,0}^* + \eta_i^*, p_{i,1}^*, \theta_0^*) \leq \sum_{i=1}^N H_0(x_{i,0}^* + \eta_i, p_{i,1}^*, \theta_0^*), \forall \|\eta_i\|_\infty \leq \epsilon \tag{24}$$

Assuming $R_{i,t} = 0$ like above, we define a new optimal control problem with target function $\tilde{\ell}_i() = -\ell_i()$ and previous dynamics except $x_{i,1} = \tilde{f}_0(x_{i,0}; \theta_0, \eta_i) = f_0(x_{i,0} + \eta_i; \theta_0)$:

$$\min_{\|\eta\|_\infty \leq \epsilon} \tilde{J}(\theta, \eta) := \frac{1}{N} \sum_{i=1}^N \tilde{\ell}_i(x_{i,T}) \tag{25}$$

$$\text{subject to } x_{i,1} = \tilde{f}_0(x_{i,0}; \theta_0, \eta_i), i = 1, 2, \cdots, N \tag{26}$$

$$x_{i,t+1} = f_t(x_{i,t}, \theta_t), t = 1, 2, \cdots, T-1. \tag{27}$$

However in this time, all the layer parameters θ_t are **fixed** and η_i is the control. From the above Lemma 2 we get

$$\tilde{x}_{i,1}^* = \nabla_p \tilde{H}_0(\tilde{x}_{i,0}^*, \tilde{p}_{i,1}^*, \theta_0, \eta_i^*), \quad \tilde{x}_{i,t+1}^* = \nabla_p H_t(\tilde{x}_{i,t}^*, \tilde{p}_{i,t+1}^*, \theta_t), \quad \tilde{x}_{i,0}^* = x_{i,0}, \tag{28}$$

$$\tilde{p}_{i,0}^* = \nabla_x \tilde{H}_0(\tilde{x}_{i,0}^*, \tilde{p}_{i,1}^*, \theta_0, \eta_i^*), \qquad \tilde{p}_{i,t}^* = \nabla_x H_t(\tilde{x}_{i,t}^*, \tilde{p}_{i,t+1}^*, \theta_t), \quad \tilde{p}_{i,T}^* = \frac{1}{N} \nabla \ell_i(\tilde{x}_{i,T}^*), \tag{29}$$

where $\tilde{H}_0(x, p, \theta_0, \eta) = p \cdot \tilde{f}_0(x; \theta_0, \eta) = p \cdot f_0(x + \eta; \theta_0)$ and $t = 1, \cdots, T-1$. This gives the fact that $\tilde{x}_{i,t}^* = x_{i,t}^*$. Lemma 2 also tells us

$$\sum_{i=1}^N \tilde{H}_0(\tilde{x}_{i,0}^*, \tilde{p}_{i,t+1}^*, \theta_0, \eta_i^*) \geq \sum_{i=1}^N \tilde{H}_0(\tilde{x}_{i,0}^*, \tilde{p}_{i,1}^*, \theta_0, \eta_i), \forall \|\eta_i\|_\infty \leq \epsilon \tag{30}$$

which is

$$\sum_{i=1}^N \tilde{p}_{i,1}^* \cdot f_0(\tilde{x}_{i,0}^* + \eta_i^*; \theta_0) \geq \sum_{i=1}^N \tilde{p}_{i,1}^* \cdot f_0(\tilde{x}_{i,0}^* + \eta_i; \theta_0), \forall \|\eta_i\|_\infty \leq \epsilon \tag{31}$$

On the other hand, Lemma 2 gives

$$\tilde{p}_t^* = -\nabla_{x_t}(\tilde{\ell}(x_T)) = \nabla_{x_t}(\ell(x_T)) = -p_t^*.$$

Then we have

$$\sum_{i=1}^{N} p_{i,1}^* \cdot f_0(x_{i,0}^* + \eta_i^*; \theta_0) \leq \sum_{i=1}^{N} p_{i,1}^* \cdot f_0(x_{i,0}^* + \eta_i; \theta_0), \forall \|\eta_i\|_\infty \leq \epsilon \tag{32}$$

which is

$$\sum_{i=1}^{N} H_0(x_{i,0}^*, p_{i,t+1}^*, \theta_0, \eta_i^*) \leq \sum_{i=1}^{N} H_0(x_{i,0}^*, p_{i,1}^*, \theta_0, \eta_i), \forall \|\eta_i\|_\infty \leq \epsilon \tag{33}$$

This finishes the proof for the adversarial control.

□

Remark. *The additional assumption that the sets $\{f_t(x, \theta) : \theta \in \Theta_t\}$ and $\{R_t(x, \theta) : \theta \in \Theta_t\}$ are convex for every t and $x \in \mathbb{R}^{d_t}$ is extremely weak and is not unrealistic which is already explained in [20].*

B Experiment Setup

B.1 MNIST

Training against PGD-40 is a common practice to get sota results on MNIST. We adopt network architectures from [43] with four convolutional layers followed by three fully connected layers. Following [43] and [24], we set the size of perturbation as $\epsilon = 0.3$ in an infinite norm sense. Experiments are taken on idle NVIDIA Tesla P100 GPUs. We train models for 55 epochs with a batch size of 256, longer than what convergence needs for both training methods. The learning rate is set to 0.1 initially, and is lowered by 10 times at epoch 45. We use a weight decay of $5e-4$ and a momentum of 0.9. To measure the robustness of trained models, we performed a PGD-40 and CW[4] attack with CW coefficient $c = 5e2$ and $lr = 1e-2$.

B.2 CIFAR-10

Following [24], we take Preact-ResNet18 and Wide ResNet-34 as the models for testing. We set the the size of perturbation as $\epsilon = 8/255$ in an infinite norm sense. We perform a 20 steps of PGD with step size $2/255$ when testing. For PGD adversarial training, we train models for 105 epochs as a common practice. The learning rate is set to $5e-2$ initially, and is lowered by 10 times at epoch 79, 90 and 100. For YOPO-m-n, we train models for 40 epochs which is much longer than what convergence needs. The learning rate is set to $0.2/m$ initially, and is lowered by 10 times at epoch 30 and 36. We use a batch size of 256, a weight decay of $5e-4$ and a momentum of 0.9 for both algorithm. We also test our model's robustness under CW attack [4] with $c = 5e2$ and $lr = 1e-2$. The experiments are taken on idle NVIDIA GeForce GTX 1080 Ti GPUs.

B.3 TRADES

TRADES[43] achieves the state-of-the-art results in adversarial defensing. The methodology achieves the 1st place out of the 1,995 submissions in the robust model track of NeurIPS 2018 Adversarial Vision Challenge. TRADES proposed a surrogate loss which quantify the trade-off in terms of the gap between the risk for adversarial examples and the risk for non-adversarial examples and the objective function can be formulated as

$$\min_\theta \mathbb{E}_{(x,y)\sim\mathcal{D}} \max_{\|\eta\|\leq\epsilon} \left(\ell(f_\theta(x), y) + \mathcal{L}\left(f_\theta\left(x\right), f_\theta\left(x+\eta\right)\right)/\lambda\right) \tag{34}$$

where $f_\theta(x)$ is the neural network parameterized by θ, ℓ denotes the loss function, $\mathcal{L}(\cdot,\cdot)$ denotes the consistency loss and λ is a balancing hyper parameter which we set to be 1 as in [43]. To solve the min-max problem, [43] also searched the ascent direction via the gradient of the "adversarial loss", *i.e.* generating the adversarial example before performing gradient descent on the weight. Specifically,

the PGD attack is performed to maximize a consistency loss instead of classification loss. For each clean data x, a single iteration of the adversarial attach can be formulated as

$$x' \leftarrow \Pi_{\|x'-x\|\leq\epsilon}\left(\alpha_1 \operatorname{sign}\left(\nabla_{x'}\mathcal{L}\left(f_\theta\left(x\right), f_\theta\left(x'\right)\right)\right) + x'\right),$$

where Π is projection operator. In the implementation of [43], after 10 such update iterations for each input data x_i, the update for weights is performed as

$$\theta \leftarrow \theta - \alpha_2 \sum_{i=1}^{B} \nabla_\theta \left[\ell\left(f_\theta\left(x_i\right), y_i\right) + \mathcal{L}\left(f_\theta\left(x_i\right), f_\theta\left(x_i'\right)\right)/\lambda\right]/B,$$

where B is the batch size. We name this algorithm as TRADES-10, for it uses 10 iterations to update the adversary.

Following the notation used in previous section, we denote f_0 as the first layer of the neural network and $g_{\tilde{\theta}}$ denotes the network without the first layer. The whole network can be formulated as the compostion of the two parts, *i.e.* $f_\theta = g_{\tilde{\theta}} \circ f_0$. To apply our gradient based YOPO method to TRADES, following Section 2, we decouple the adversarial calculation and network updating as shown in Algorithm 2. Projection operation is omitted. Notice that in Section.2 we take advantage every intermediate perturbation $\eta^j, j = 1, \cdots, m-1$ to update network weights while here we only use the final perturbation $\eta = \eta^m$ to compute the final loss term. In practice, this accumulation of gradient doesn't helps. For TRADES-YOPO, acceleration of YOPO is brought by decoupling the adversarial calculation with the gradient back propagation.

Algorithm 2 TRADES-YOPO-m-n

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 $\eta_i^{1,0}, i = 1, 2, \cdots, B$ by sampling from a uniform distribution between [-ϵ, ϵ]
  for $j = 1$ to m **do**
    $p_i = \nabla_{g_{\tilde{\theta}}}\left(\mathcal{L}\left(g_{\tilde{\theta}}\left(f_0\left(x_i + \eta_i^{j,0}, \theta_0\right)\right), g_{\tilde{\theta}}\left(f_0\left(x_i, \theta_0\right)\right)\right)\right) \cdot \nabla_{f_0}\left(g_{\tilde{\theta}}(f_0(x_i + \eta_i^{j,0}, \theta_0))\right),$
    $i = 1, 2, \cdots, B$
    for $s = 0$ to $n-1$ **do**
      $\eta_i^{j,s+1} \leftarrow \eta_i^{j,s} + \alpha_1 \cdot p_i \cdot \nabla_\eta f_0(x_i + \eta_i^{j,s}, \theta_0), i = 1, 2, \cdots, B$
    end for
    $\eta_i^{j+1,0} = \eta_i^{j,n}, i = 1, 2, \cdots, B$
  end for
  $\theta \leftarrow \theta - \alpha_2 \sum_{i=1}^{B} \nabla_\theta \left[\ell\left(f_\theta\left(x_i\right), y_i\right) + \mathcal{L}\left(f_\theta\left(x_i\right), f_\theta\left(x_i + \eta_i^{m,n}\right)\right)/\lambda\right]/B.$
until Convergence

We name this algorithm as TRADES-YOPO-m-n. With less than half time of TRADES-10, TRADES-YOPO-3-4 achieves even better result than its baseline. Quantitative results is demonstrated in Table 4. The mini-batch size is 256. All the experiments run for 105 epochs and the learning rate set to $2e-1$ initially, and is lowered by 10 times at epoch 70, 90 and 100. The weight decay coefficient is $5e-4$ and momentum coefficient is 0.9. We also test our model's robustness under CW attack [4] with $c = 5e2$ and $lr = 5e-4$. Experiments are taken on idle NVIDIA Tesla P100 GPUs.