Improved Adversarial Training via Learned Optimizer

Yuanhao Xiong[1] and Cho-Jui Hsieh[1]

University of California, Los Angeles, CA 90024, USA
{yhxiong,chohsieh}@cs.ucla.edu

Abstract. Adversarial attack has recently become a tremendous threat to deep learning models. To improve the robustness of machine learning models, adversarial training, formulated as a minimax optimization problem, has been recognized as one of the most effective defense mechanisms. However, the non-convex and non-concave property poses a great challenge to the minimax training. In this paper, we empirically demonstrate that the commonly used PGD attack may not be optimal for inner maximization, and improved inner optimizer can lead to a more robust model. Then we leverage a learning-to-learn (L2L) framework to train an optimizer with recurrent neural networks, providing update directions and steps adaptively for the inner problem. By co-training optimizer's parameters and model's weights, the proposed framework consistently improves over PGD-based adversarial training and TRADES.

Keywords: Optimization, Adversarial Training, Learning to Learn

1 Introduction

It has been widely acknowledged that deep neural networks (DNN) have made tremendous breakthroughs benefiting both academia and industry. Despite being effective, many DNN models trained with benign inputs are vulnerable to small and undetectable perturbation added to original data and tend to make wrong predictions under such threats. Those perturbed examples, also known as adversarial examples, can be easily constructed by algorithms such as DeepFool [23], Fast Gradient Sign Method (FGSM) [11], and Carlini-Wagner (C&W) attack [4]. Moreover, such adversarial attacks can also be conducted in the black-box setting [3, 5, 6] and can appear naturally in the physical world [12, 16]. This phenomenon can bring about serious consequences in domains such as face recognition and autonomous-driving. Therefore, how to train a model resistant to adversarial inputs has become an important topic.

A variety of defense methods have been proposed to improve the performance of DNNs against adversarial attacks [17, 27, 31, 32, 36, 39]. Among them, adversarial training [17] stands out for its effectiveness. Moreover, [21] shows that adversarial training can be formulated as a minimax optimization problem, resembling a game between the attacker and the defender. The formulation is so intuitive that the inner problem aims at generating adversarial examples by maximizing the training loss while the outer one guides the network in the direction that minimizes the loss to resist attacks. However, directly obtaining the optimal value of the inner maximization is infeasible, so one has to run an iterative optimization algorithm for a fixed number (often 10) iterations to get an approximate inner maximizer.

Existing adversarial training often uses hand-designed general purpose optimizers, such as PGD attack, to (approximately) solve the inner maximization. However, there is an essential property of adversarial training that is rarely explored: the maximization problems associated with each sample share very similar structure, and a good inner maximizer for adversarial training only needs to work well for this set of data-dependent problems. To be specific, there are a finite of n maximization problems need to be solved (where n is number of training samples), and those maximization problems share the same objective function along with identical network structure and weights, and the only difference is their input $\boldsymbol{x}$. Based on this observation, can we have a better optimizer that in particular works well for these very similar and data-dependent problems?

Motivated by this idea, we propose a learned optimizer for improved adversarial training. Instead of using an existing optimizer with a fixed update rule (such as PGD), we aim at learning the inner maximizer that could be faster and more effective for this particular set of maximization problems. We have noticed that two works have already put forward algorithms to combine learning to learn with adversarial training [14, 13]. Both of them adopt a convolutional neural network (CNN) generator to produce malicious perturbations whereas CNN structure might complicate the training process and cannot grasp the essence of the update rule in the long term. In contrast, we propose an L2L-based adversarial training method with recurrent neural networks (RNN). RNN is capable of capturing long-term dependencies and has shown great potentials in predicting update directions and steps adaptively [20]. Thus, following the framework in [1], we leverage RNN as the optimizer to generate perturbations in a coordinate-wise manner. Based on the properties of the inner problem, we tailor our RNN optimizer with removed bias and weighted loss for further elaborations to ameliorate issues like short-horizon in L2L [34].

Specifically, our main contributions in this paper are summarized as follows:

- We first investigate and confirm the improvement in the model robustness from stronger attacks by searching a suitable step size for PGD.
- In replacement of hand-designed algorithms like PGD, an RNN-based optimizer based on the properties of the inner problem is designed to learn a better update rule. In addition to standard adversarial training, the proposed algorithm can also be applied to any other minimax defense objectives such as TRADES [39].
- Comprehensive experimental results show that the proposed method can noticeably improve the robust accuracy of both adversarial training [21] and TRADES [39]. Furthermore, our RNN-based adversarial training significantly outperforms previous CNN-based L2L adversarial training and requires much less number of trainable parameters.

2 Related Work

2.1 Adversarial Attack and Defense

Model robustness has recently become a great concern for deploying deep learning models in real-world applications. Goodfellow *et al.* [11] succeeded in fooling the model to make wrong predictions by Fast Gradient Sign Method (FGSM). Subsequently, to produce adversarial examples, IFGSM and Projected Gradient Descent (PGD) [11, 21] accumulate attack strength through running FGSM iteratively, and Carlini-Wagner (C&W) attack [4] designs a specific objective function to increase classification errors. Besides these conventional optimization-based methods, there are several algorithms [25, 35] focusing on generating malicious perturbations via neural networks. For instance, Xiao *et al.* [35] exploit GAN, which is originally designed for crafting deceptive images, to output corresponding noises added to benign iuput data. The appearance of various attacks has pushed forward the development of effective defense algorithms to train neural networks that are resistant to adversarial examples. The seminal work of adversarial training has significantly improved adversarial robustness [21]. It has inspired the emergence of various advanced defense algorithms: TRADES [39] is designed to minimize a theoretically-driven upper bound and GAT [19] takes generator-based outputs to train the robust classifier. All these methods can be formulated as a minimax problem [21], where the defender makes efforts to mitigate negative effects (outer minimization) brought by adversarial examples from the attacker (inner maximization). Whereas, performance of such an adversarial game is usually constrained by the quality of solutions to the inner problem [13, 14]. Intuitively, searching a better maxima for the inner problem can improve the solution of minimax training, leading to improved defensive models.

2.2 Learning to Learn

Recently, learning to learn emerges as a novel technique to efficiently address a variety of problems such as automatic optimization [1], few-shot learning [10], and neural architecture search [8]. In this paper, we emphasize on the subarea of L2L: how to learn an optimizer for better performance. Rather than using human-defined update rules, learning to learn makes use of neural networks for designing optimization algorithms automatically. It is developed originally from [7] and [37], in which early attempts are made to model adaptive algorithms on simple convex problems. More recently, [1] proposes an LSTM optimizer for some complex optimization problems, such as training a convolutional neural network classifier. Based on this work, elaborations in [20] and [33] further improve the generalization and scalability for learned optimizers. Moreover, [26] demonstrates that a zeroth order optimizer can also be learned using L2L. Potentials of learning-to-learn motivates a line of L2L-based defense which replaces hand-designed methods for solving the inner problem with neural network optimizers. [14] uses a CNN generator mapping clean images into corresponding perturbations. Since it only makes one-step and deterministic attack like FGSM,

[13] modifies the algorithm and produces stronger and more diverse attacks iteratively. Unfortunately, due to the large number of parameters and the lack of ability to capture the long-term dependencies, the CNN generator adds too much difficulty in the optimization, especially for the minimax problem in adversarial training. Therefore, we adopt an RNN optimizer in our method for a more stable training process as well as a better grasp of the update rule.

3 Preliminaries

3.1 Notations

We use bold lower-case letters $\boldsymbol{x}$ and $\boldsymbol{y}$ to represent clean images and their corresponding labels. An image classification task is considered in this paper with the classifier f parameterized by θ. $\mathsf{sign}(\cdot)$ is an elementwise operation to output the sign of a given input with $\mathsf{sign}(0) = 1$. $\mathbb{B}(\boldsymbol{x}, \epsilon)$ denotes the neighborhood of $\boldsymbol{x}$ as well as the set of admissible perturbed images: $\{\boldsymbol{x}' : \|\boldsymbol{x}' - \boldsymbol{x}\|_\infty \leq \epsilon\}$, where the infinity norm is adopted as the distance metric. We denote by Π the projection operator that maps perturbed data to the feasible set. Specifically, $\Pi_{\mathbb{B}(\boldsymbol{x},\epsilon)}(\boldsymbol{x}') = \max(\boldsymbol{x}-\epsilon, \min(\boldsymbol{x}', \boldsymbol{x}+\epsilon))$, which is an elementwise operator. $\mathcal{L}(\cdot, \cdot)$ is a multi-class loss like cross-entropy.

3.2 Adversarial Training

In this part, we present the formulation of adversarial training, together with some hand-designed optimizers to solve this problem. To obtain a robust classifier against adversarial attacks, an intuitive idea is to minimize the robust loss, defined as the worst-case loss within a small neighborhood $\mathbb{B}(\boldsymbol{x}, \epsilon)$. Adversarial training, which aims to find the weights that minimize the robust loss, can be formulated as a minimax optimization problem in the following way [21]:

$$\min_\theta \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim D} \left\{ \max_{\boldsymbol{x}'\in\mathbb{B}(\boldsymbol{x},\epsilon)} \mathcal{L}(f(\boldsymbol{x}'), \boldsymbol{y}) \right\} \tag{1}$$

where D is the empirical distribution of input data. However, (1) only focuses on accuracy over adversarial examples and might cause severe over-fitting issues on the training set. To address this problem, TRADES [39] investigates the trade-off between natural and robust errors and theoretically puts forward a different objective function for adversarial training:

$$\min_\theta \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim D} \left\{ \mathcal{L}(f(\boldsymbol{x}), \boldsymbol{y}) + \max_{\boldsymbol{x}'\in\mathbb{B}(\boldsymbol{x},\epsilon)} \mathcal{L}(f(\boldsymbol{x}), f(\boldsymbol{x}'))/\lambda \right\}. \tag{2}$$

Note that (1) and (2) are both defined as minimax optimization problems, and to solve such saddle point problems, a commonly used approach is to first get an approximate solution $\boldsymbol{x}'$ of inner maximization based on the current θ, and then use $\boldsymbol{x}'$ to conduct updates on model weights θ. The adversarial training procedure then iteratively runs this on each batch of samples until convergence. Clearly, the quality and efficiency of inner maximization is crucial to the performance of adversarial training. The most commonly used inner maximizer is the projected gradient descent algorithm, which conducts a fixed number of updates:

$$\boldsymbol{x}'_{t+1} = \mathit{\Pi}_{\mathbb{B}(\boldsymbol{x},\epsilon)}(\alpha \mathsf{sign}(\nabla_{\boldsymbol{x}'}\mathcal{L}(\boldsymbol{x}'_t)) + \boldsymbol{x}'_t). \tag{3}$$

Here $\mathcal{L}(\boldsymbol{x}'_t)$ represents the maximization term in (1) or (2) with abuse of notation.

3.3 Effects of Adaptive Step Sizes

We found that the performance of adversarial training crucially depends on the optimization algorithm used for inner maximization, and the current widely used PGD algorithm may not be the optimal choice. Here we demonstrate that even a small modification of PGD and without any change to the adversarial training objective can boost the performance of model robustness. We use the CNN structure in [39] to train a classifier on MNIST dataset. When 10-step PGD (denoted by PGD for simplicity) is used for the inner maximization, a constant step size is always adopted, which may not be suitable for the subsequent update. Therefore, we make use of backtracking line search (BLS) to select a step size adaptively for adversarial training (AdvTrain as abbreviation). Starting with a maximum candidate step size value α_0, we iteratively decrease it by $\alpha_t = \rho\alpha_{t-1}$ until the following condition is satisfied:

$$\mathcal{L}(\boldsymbol{x}' + \alpha_t \boldsymbol{p}) \geq \mathcal{L}(\boldsymbol{x}') + c\alpha_t \boldsymbol{p}^{\mathrm{T}}\boldsymbol{p} \tag{4}$$

where $\boldsymbol{p} = \nabla_{\boldsymbol{x}'}\mathcal{L}(\boldsymbol{x}')$ is a search direction. Based on a selected control parameter $c \in (0, 1)$, the condition tests whether the update with step size α_t leads to sufficient increase in the objective function, and it is guaranteed that a sufficiently small α will satisfy the condition so line search will always stop in finite steps. This is standard in gradient ascent (descent) optimization, and see more discussions in [24]. Following the convention, we set $\rho = 0.5$ and $c = 10^{-4}$. As shown in Table 1, defense with AdvTrain+BLS leads to a more robust model than solving the inner problem only by PGD (88.71% vs 87.33%). At the same time the attacker combined with BLS generates stronger adversarial examples: the robust accuracy of the model trained from vanilla adversarial training drops over 1.2% with PGD+BLS, compared to merely PGD attack. This experiment motivates our efforts to find a better inner maximizer for adversarial training.

Table 1. Effects of the inner solution quality on robust accuracy (%)

Attack / Defense	Natural	PGD	PGD+BLS
AdvTrain	96.43	87.33	86.09
AdvTrain+BLS	**96.70**	**88.71**	**88.00**

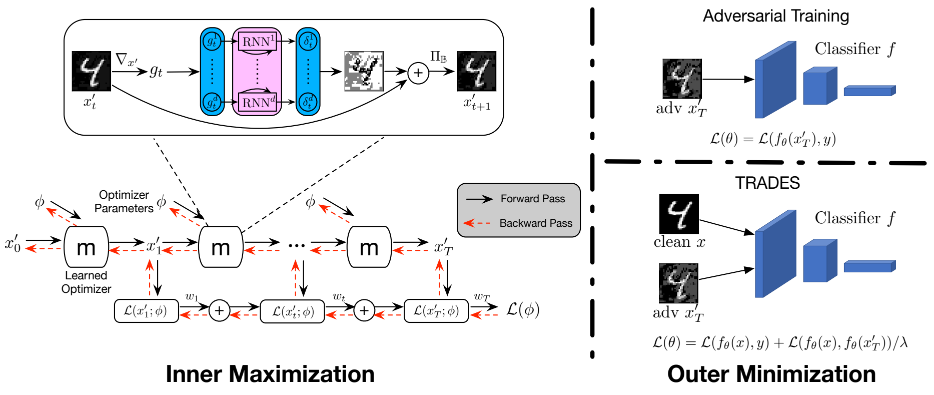


Fig. 1. Model architecture of our defense method with an RNN optimizer

4 Proposed Algorithm

4.1 Learning to Learn for Adversarial Training

As mentioned in the previous section, it can be clearly seen that the inner maximizer plays an important role in the performance of adversarial training. However, despite the effectiveness of BLS introduced in Section 3.3, it is impractical to combine it with adversarial training as multiple line searches together with loss calculation in this algorithms increase its computational burden significantly. Then a question arises naturally: is there any automatic way for determining a good step size for inner maximization without too much computation overhead? Moreover, apart from the step size, the question can be extended to whether such a maximizer can be learned for a particular dataset and model in replacement of a general optimizer like PGD. Recently, as a subarea of learning-to-learn, researchers have been investigating whether it is possible to use machine learning, especially neural networks, to learn improved optimizer to replace the hand-designed optimizer [1, 20, 33]. However, it is commonly believed that those ML-learned general-purpose optimizers are still not practically useful due to several unsolved issues. For instance, the exploded gradient [22] in unrolled optimization impedes generalization of these learned optimizers to longer steps and truncated optimization on the other hand induces short-horizon bias [34].

In this paper, we show it is possible and practical to learn an optimizer for inner maximization in adversarial training. Note that in adversarial training, the maximization problems share very similar form: $\max_{\boldsymbol{x}' \in \mathbb{B}(\boldsymbol{x},\epsilon)} \mathcal{L}(f(\boldsymbol{x}'), \boldsymbol{y})$, where they all have the same loss function $\mathcal{L}$ and the same network (structure and weights) f, and the only difference is their input $\boldsymbol{x}$ and label $\boldsymbol{y}$. Furthermore, we only need the maximizer to perform well on a fixed set of n optimization problems for adversarial training. These properties thus enable us to learn a better optimizer that outperforms PGD.

To allow a learned inner maximizer, we parameterize the learned optimizer by an RNN network. This is following the literature of learning-to-learn [1], but we propose several designs as shown below that works better for our inner maximization problem which is a constrained optimization problem instead of a standard unconstrained training task in [1]. We then jointly optimize the classifier parameters (θ) as well as the parameters of the inner maximizer (ϕ). The overall framework can be found in Figure 1.

Specifically, the inner problem is to maximize vanilla adversarial training loss in (1) or TRADES loss in (2), with a constraint that $\boldsymbol{x}' \in \mathbb{B}(\boldsymbol{x}, \epsilon)$. We expand on adversarial training here and more details about TRADES can be found in Appendix A. With an RNN optimizer m parameterized by ϕ, we propose the following parameterized update rule to mimic the PGD update rule in (3):

$$\boldsymbol{\delta}_t, \boldsymbol{h}_{t+1} = m_\phi(\boldsymbol{g}_t, \boldsymbol{h}_t), \quad \boldsymbol{x}'_{t+1} = \Pi_{\mathbb{B}(\boldsymbol{x},\epsilon)}\left(\boldsymbol{x}'_t + \boldsymbol{\delta}_t\right). \tag{5}$$

Here, $\boldsymbol{g}_t$ is the gradient $\nabla_{\boldsymbol{x}'}\mathcal{L}(f(\boldsymbol{x}'), \boldsymbol{y})$ and $\boldsymbol{h}_t$ is the hidden state representation. It has to be emphasized that our RNN optimizer generates perturbations coordinate-wisely, in contrast to other L2L based methods which take as input the entire image. This property reduces trainable parameters significantly, making it much easier and faster for training. In addition, note that the hidden state of our RNN optimizer plays an important role in the whole optimization. A separate hidden state for each coordinate guarantees the different update behavior. And it contains richer information like the trajectory of loss gradients mentioned in [13] but can produce a recursive update with a simpler structure.

For the RNN design, we mainly follow the structure in [1] but with some modifications to make it more suitable to adversarial training. We can expand the computation of perturbation for each step as:

$$\boldsymbol{\delta}_t = \tanh(\boldsymbol{V}\boldsymbol{h}_t + \boldsymbol{b}_1), \tag{6}$$
$$\boldsymbol{h}_{t+1} = \tanh(\boldsymbol{U}\boldsymbol{g}_t + \boldsymbol{W}\boldsymbol{h}_t + \boldsymbol{b}_2) \tag{7}$$

where $\boldsymbol{h}_t \in \mathbb{R}^d$, $\boldsymbol{V} \in \mathbb{R}^{1\times d}$, $\boldsymbol{U} \in \mathbb{R}^{d\times 1}$, $\boldsymbol{W} \in \mathbb{R}^{d\times d}$, $\boldsymbol{b}_1 \in \mathbb{R}$ and $\boldsymbol{b}_2 \in \mathbb{R}^d$ in the coordinate-wise update manner. As the optimization proceeds, the gradient will become much smaller when approaching the local maxima. At that time, a stable value of the perturbation is expected without much change between two consecutive iterations. However, from (6) and (7), we can clearly see that despite small $\boldsymbol{g}_t$, the update rule will still produce an update with magnitude proportional to $\tanh(\boldsymbol{b}_1)$. Imagine the case where the exact optimal value is found with an all-zero hidden state ($\boldsymbol{b}_2$ needs to be zero as well), $\boldsymbol{\delta}_t = \tanh(\boldsymbol{b}_1)$ with a non-zero bias will push the adversarial example away from the optimal one. Thus, two bias terms $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are problematic for optimization close to the optimal solution. Due to the short horizon of the inner maximization in adversarial training, it is unlikely for the network to learn zero bias terms. Therefore, to ensure stable training, we remove the bias terms in the vanilla RNN in all implementations.

With an L2L framework, we simultaneously train the RNN optimizer parameters ϕ and the classifier weights θ together. The joint optimization problem can be formulated as follows:

$$\min_{\theta} \ \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim D} \left\{\mathcal{L}(f_\theta(\boldsymbol{x}'_T(\phi^*)),\boldsymbol{y})\right\} \tag{8}$$

$$\text{s.t.} \ \ \phi^* = \arg\max \mathcal{L}(\phi) \tag{9}$$

where $\boldsymbol{x}'_T(\phi^*)$ is computed by running Eq.(5) T times iteratively. Since the learned optimizer aims at finding a better solution to the inner maximization term, the objective function for training it in the horizon T is defined as:

$$\mathcal{L}(\phi) = \sum_{t=1}^{T} w_t \mathcal{L}(f_\theta(\boldsymbol{x}'_t(\phi)),\boldsymbol{y}). \tag{10}$$

Note that if we set $w_t = 0$ for all $t < T$ and $w_T = 1$, then (10) implies that our learned maximizer m_ϕ will maximize the loss after T iterations. However, in practice we found that considering intermediate iterations can further improve the performance since it will make the maximizer converges faster even after conducting one or few iterations. Therefore in the experiments we set an increased weights $w_t = t$ for $t = 1,\dots,T$. Note that [22] showed that this kind of unrolled optimization may lead to some issues such as exploded gradients which is still an unsolved problem in L2L. However, in adversarial training we only need to set a relative small T (e.g., $T = 10$) so we do not encounter that issue.

While updating the learned optimizer, corresponding adversarial examples are produced together. We can then train the classifier by minimizing the loss accordingly. The whole algorithm is presented in Algorithm 1.

4.2 Advantages over Other L2L-based Methods

Previous methods have proposed to use a CNN generator [13, 14] to produce perturbations in adversarial training. However, CNN-based generator has a larger number of trainable parameters, which makes it hard to train. In Table 2, the detailed properties including the number of parameter and training time per epoch are provided for different learning-to-learn based methods. We can observe that our proposed RNN approach stands out with the smallest parameters as well as efficiency in training. Specifically, our RNN optimizer only has 120 parameters, almost 5000 times fewer than L2LDA while the training time per epoch is 268.50s (RNN-TRADES only consumes 443.52s per training epoch) v.s. 1972.41s. Furthermore, our method also leads to better empirical performance, as shown in our main comparison in Table 3, 4 and 5. Comparison of our variants and original adversarial training methods can be found in Appendix B.

5 Experimental Results

In this section, we present experimental results of our proposed RNN-based adversarial training. We compare our method with various baselines against both white-box and black-box attack. In addition, different datasets and network architectures are also evaluated.

Algorithm 1 RNN-based adversarial training

```
1: Input: clean data $\{(\boldsymbol{x}, \boldsymbol{y})\}$, batch size $B$, step sizes $\alpha_1$ and $\alpha_2$, number of inner
   iterations $T$, classifier parameterized by $\theta$, RNN optimizer parameterized by $\phi$
2: Output: Robust classifer $f_\theta$, learned optimizer $m_\phi$
3: Randomly initialize $f_\theta$ and $m_\phi$, or initialize them with pre-trained configurations
4: repeat
5:     Sample a mini-batch $M$ from clean data.
6:     for $(\boldsymbol{x}, \boldsymbol{y})$ in $B$ do
7:         Initialization: $\boldsymbol{h}_0 \leftarrow 0$, $\mathcal{L}_\theta \leftarrow 0$, $\mathcal{L}_\phi \leftarrow 0$
8:         Gaussian augmentation: $\boldsymbol{x}'_0 \leftarrow \boldsymbol{x} + 0.001 \cdot \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
9:         for $t = 0, \ldots, T-1$ do
10:            $\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{x}'} \mathcal{L}(f_\theta(\boldsymbol{x}'_t), \boldsymbol{y})$
11:            $\boldsymbol{\delta}_t, \boldsymbol{h}_{t+1} \leftarrow m_\phi(\boldsymbol{g}_t, \boldsymbol{h}_t)$, where coordinate-wise update is applied
12:            $\boldsymbol{x}'_{t+1} \leftarrow \Pi_{\mathbb{B}(\boldsymbol{x},\epsilon)}(\boldsymbol{x}'_t + \boldsymbol{\delta}_t)$
13:            $\mathcal{L}_\phi \leftarrow \mathcal{L}_\phi + w_{t+1} \mathcal{L}(f_\theta(\boldsymbol{x}'_{t+1}), \boldsymbol{y})$, where $w_{t+1} = t + 1$
14:        end for
15:        $\mathcal{L}_\theta \leftarrow \mathcal{L}_\theta + \mathcal{L}(f_\theta(\boldsymbol{x}'_T), \boldsymbol{y})$
16:    end for
17:    Update $\phi$ by $\phi \leftarrow \phi + \alpha_1 \nabla_\phi \mathcal{L}_\phi / B$
18:    Update $\theta$ by $\theta \leftarrow \theta - \alpha_2 \nabla_\theta \mathcal{L}_\theta / B$
19: until training converged
```

Table 2. Comparion among different L2L-based methods

	Number of parameters	Training time per epoch (s)
RNN-Adv	**120**	**268.50**
RNN-TRADES	**120**	443.52
L2LDA	500944	1972.41

5.1 Experimental Settings

- **Datasets and classifier networks.** We mainly use MNIST [18] and CIFAR-10 [15] datasets for performance evaluation in our experiments. For MNIST, the CNN architecture with four convolutional layers in [4] is adopted as the classifier. For CIFAR-10, we use both the standard VGG-16 [28] and Wide ResNet [38], which has been used in most of the previous defense papers including adversarial training [21] and TRADES [39]. We also conduct an additional experiment on Restricted ImageNet [30] with ResNet-18 and results are presented in Appendix C.
- **Baselines for Comparison.** Note that our method is an optimization framework which is irrelevant to what minimax objective function is used. Therefore we choose two most popular minimax formulations, AdvTrain[1] [21]

[1] https://github.com/xuanqing94/BayesianDefense

and TRADES[2] [39], and substitute the proposed L2L-based optimization for their original PGD-based algorithm. Moreover, we also compare with a previous L2L defense mechanism L2LDA[3] [13] which outperforms other L2L-based methods for thorough comparison. We use the source code provided by the authors on github with their recommended hyper-parameters for all these baseline methods.

– **Evaluation and implementation details.** Defense algorithms are usually evaluated by classification accuracy under different attacks. Effective attack algorithms including PGD, C&W and the attacker of L2LDA are used for evaluating the model robustness, with the maximum ℓ_∞ perturbation strength $\epsilon = 0.3$ for MNIST and $\epsilon = 8/255$ for CIFAR-10. For PGD, we run 10 and 100 iterations (PGD-10 and -100) with the step size $\eta = \epsilon/4$, as suggested in [13]. C&W is implemented with 100 iterations in the infinity norm. For L2LDA attacker, it is learned from L2LDA [13] under different settings with 10 attack steps. In addition, we also uses the learned optimizer of RNN-Adv to conduct 10-step attacks.

For our proposed RNN-based defense, we use a one-layer vanilla RNN with the hidden size of 10 as the optimizer for the inner maximization. Since we test our method under two different minimax losses, we name them as RNN-Adv and RNN-TRADES respectively. The classifier and the optimizer are updated alternately according to the Algorithm 1. All algorithms are implemented in PyTorch-1.1.0 with four NVIDIA 1080Ti GPUs. Note that all adversarial training methods adopt 10-step inner optimization for fair comparison. We run each defense method five times with different random seeds and report the lowest classification accuracy.

5.2 Performance on White-box Attacks

We demonstrate the robustness of models trained from different defense methods under the white-box setting in this part. Experimental results are shown in Table 3, 4 and 5. From these three tables, we can observe that our proposed L2L-based adversarial training with RNN always outperforms its counterparts.

To be specific, our method achieves 95.80% robust accuracy among various attacks on MNIST dataset. On CIFAR-10, RNN-TRADES reaches 47.23% and 54.11 for VGG-16 and Wide ResNet with 1.28% and 1.43% gain over other baselines. It should be stressed that our method surpasses L2LDA (the previous CNN-based L2L method) noticeably. For conventional defense algorithms, our L2L-based variant improves the original method by $1\% - 2\%$ percents under different attacks from comparison of robust accuracy in AdvTrain and RNN-Adv. A similar phenomenon can also be observed in TRADES and RNN-TRADES. Since previous works of L2L-based defense only concentrate on PGD-based adversarial training, the substantial performance gain indicates that the learned optimizer

[2] https://github.com/yaodongyu/TRADES
[3] https://github.com/YunseokJANG/l2l-da

Table 3. Robust accuracy under white-box attacks (MNIST, 4-layer CNN)

Defense \ Attack	Natural	PGD-10	PGD-100	CW100	L2LDA	RNN-Adv	Min
Plain	99.46	1.04	0.42	83.63	5.94	0.79	0.42
AdvTrain	99.17	94.89	94.28	98.38	95.83	94.39	94.28
TRADES	**99.52**	95.77	95.50	98.72	96.03	95.50	95.50
L2LDA	98.76	94.73	93.22	97.69	95.28	93.16	93.16
RNN-Adv	99.20	95.80	95.62	98.75	96.05	95.51	95.51
RNN-TRADES	99.46	**96.09**	**95.83**	**98.85**	**96.56**	**95.80**	**95.80**

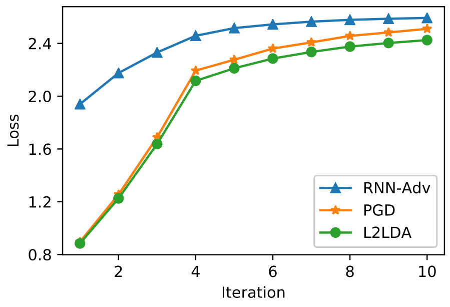
Table 4. Robust accuracy under white-box attacks (CIFAR-10, VGG-16)

Defense \ Attack	Natural	PGD-10	PGD-100	CW100	L2LDA	RNN-Adv	Min
Plain	**93.66**	0.74	0.09	0.08	0.89	0.43	0.08
AdvTrain	81.11	42.32	40.75	42.26	43.55	41.07	40.75
TRADES	78.08	48.83	48.30	45.94	49.94	48.38	45.95
L2LDA	77.47	35.49	34.27	35.31	36.27	34.54	34.27
RNN-Adv	81.22	44.98	42.89	43.67	46.20	43.21	42.89
RNN-TRADES	80.76	**50.23**	**49.42**	**47.23**	**51.29**	**49.49**	**47.23**

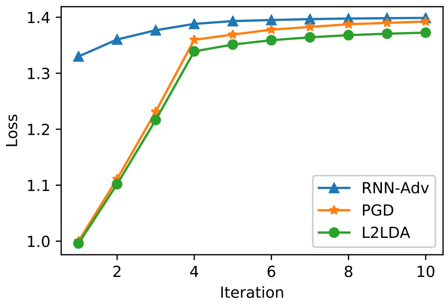
Table 5. Robust accuracy under white-box attakcs (CIFAR-10, WideResNet)

Defense \ Attack	Natural	PGD-10	PGD-100	CW100	L2LDA	RNN-Adv	Min
Plain	**95.14**	0.01	0.00	0.00	0.02	0.00	0.00
AdvTrain	86.28	46.64	45.13	46.64	48.46	45.41	45.13
TRADES	85.89	54.28	52.68	53.68	56.49	53.00	52.68
L2LDA	85.30	45.47	44.35	44.19	47.16	44.54	44.19
RNN-Adv	85.92	47.62	45.98	47.26	49.40	46.23	45.98
RNN-TRADES	84.21	**56.35**	**55.68**	**54.11**	**58.86**	**55.80**	**54.11**

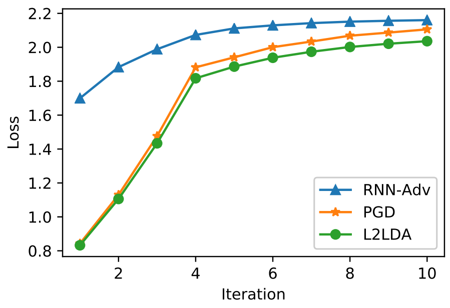
can contribute to the minimax problem in TRADES as well. Furthermore, apart from traditional attack algorithms, we leverage our RNN optimizer learned from adversarial training as the attacker (the column RNN-Adv). Results in three experiments show that compared with other general attackers when conducting



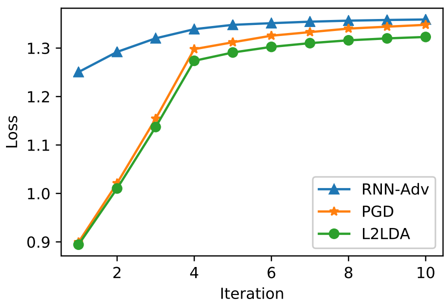
(a) AdvTrain



(b) TRADES



(c) RNN-Adv



(d) RNN-TRADES

Fig. 2. Comparison of optimization trajectories among various attack algorithms. We evaluate four defense mechanisms, AdvTrain TRADES, RNN-Adv and RNN-TRADES, under three attackers including PGD, L2LDA and our proposed RNN-Adv. All attackers conduct 10-step perturbing process

10 iterations such as PGD-10 and L2LDA, ours is capable of producing much stronger perturbations which lead to low robust accuracy.

5.3 Analysis

Learned Optimizer. As mentioned in Section 5.2, the optimizer learned from PGD-based adversarial training can be regarded as an special attacker. Thus, we primarily investigate the update trajectories of different attackers to obtain an in-depth understanding of our RNN optimizer. For VGG-16 models trained from four defense methods, three attacker are used to generate perturbations in 10 steps respectively and losses are recorded as shown in Figure 2.

We can see clearly from these four figures that the losses obtained from RNN-Adv are always larger than others within 10 iterations, reflecting stronger attacks produced by our proposed optimizer. Moreover, it should be noted that the loss gap between RNN-Adv and other attackers is much more prominent at some very beginning iterations. This in fact demonstrates an advantage of the learning-to-learn framework that the optimizer can converge faster than hand-designed algorithms.

Generalization to more attack steps. Although our learned RNN optimizer is only trained under 10 steps, we show that it can generalize to more steps as an attacker. From Table 6, we can observe that the attacker is capable of producing much stronger adversarial examples by extending its attack steps to 40. Performance of our attacker is even comparable with that of PGD-100, which further demonstrates the superiority of our proposed method.

5.4 Performance on Black-box Transfer Attacks

We further test the robustness of the proposed defense method under transer attack. As suggested by [2], this can be served as a sanity check to see whether our defense leads to obfuscated gradients and gives a false sense of model robustness. Following procedures in [2], we first train a surrogate model with the same architecture of the target model using a different random seed, and then generate adversarial examples from the surrogate model to attack the target model.

Table 6. Generalization to more steps of learned optimizer

Defense \ Step	10	40
Plain	0.43	**0.03**
AdvTrain	41.07	**40.70**
TRADES	48.38	**48.27**
L2LDA	34.54	**34.19**
RNN-Adv	43.21	**42.89**
RNN-TRADES	49.49	**49.28**

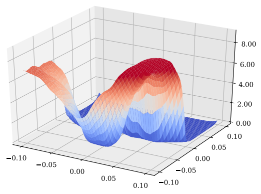
Table 7. Robust accuracy under black-box attack settings

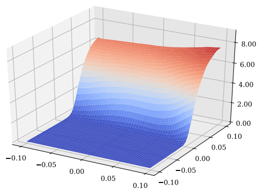
Defense \ Surrogate	Plain-Net	PGD-Net
AdvTrain	79.94	62.57
TRADES	77.01	65.41
L2LDA	76.37	60.32
RNN-Adv	**80.58**	63.17
RNN-TRADES	79.54	**67.09**

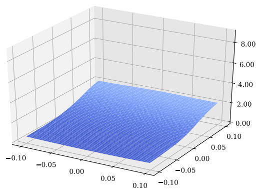
Specifically, we choose VGG-16 models obtained from various defense algorithms as our target models. In the meanwhile, we train two surrogate models: one is Plain-Net with natural training and the other is PGD-Net with 10-step PGD-based adversarial training. Results are presented in Table 7. We can observe that our method outperforms all other baselines, with RNN-PGD and RNN-TRADES standing out in defending attacks from Plain-Net and PGD-Net respectively. It suggests great resistance of our L2L defense to transfer attacks.

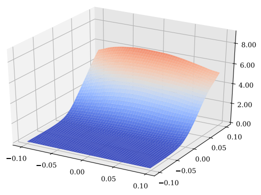
5.5 Loss Landscape Exploration

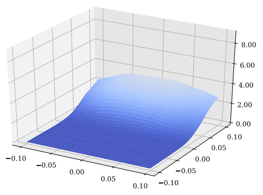
To further verify the superior performance of the proposed algorithm, we visualize the loss landscapes of VGG-16 models trained under different defense


(a) Plain


(b) Adversarial Training


(c) TRADES


(d) L2LDA


(e) RNN-Adv

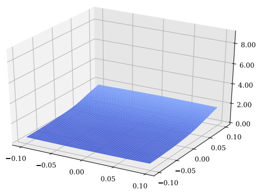

(f) RNN-TRADES

Fig. 3. Comparison of loss landscapes among different training methods. The color gradually changes from blue (low loss) to red (high loss)

strategies, as shown in Figure 3. According to the implementation in [9], we modify the input along a linear space defined by the sign of the gradient and a random Rademacher vector, where the x and y axes represent the magnitude of the perturbation added in each direction and the z axis represents the loss. It can be observed that loss surfaces of models trained from RNN-Adv and RNN-TRADES in Figure 3e and 3f are much smoother than those of their counterparts in Figure 3b and 3c. Besides, our method significantly reduces the loss value of perturbed data close to the original input. In particular, the maximum loss decreases roughly from 7.61 in adversarial training to 3.66 in RNN-Adv. Compared with L2LDA in Figure 3d, the proposed RNN optimizer can contribute to less bumpier loss landscapes with smaller variance, which further demonstrates the stability and superiority of our L2L-based adversarial training.

6 Conclusion

For defense mechanisms that can be formulated as a minimax optimization problem, we propose to replace the inner PGD-based maximizer with a automatically learned RNN maximizer, and show that jointly training the RNN maximizer and classifier can significantly improve the defense performance. Empirical results demonstrate that the proposed approach can be combined with several minimax defense objectives, including adversarial training and TRADES. For future work, it can be a worthwhile direction to address the inadequacy of L2L in dealing with a long-horizon problem. Then we can substitute the learned optimizer for hand-designed algorithms in both inner and outer problems, which enables an entirely automatic process for adversarial training.

Acknowledgements This work is supported by NSF under IIS-1901527 and IIS-2008173.

References

1. Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M.W., Pfau, D., Schaul, T., Shillingford, B., De Freitas, N.: Learning to learn by gradient descent by gradient descent. In: Advances in neural information processing systems. pp. 3981–3989 (2016)
2. Athalye, A., Carlini, N., Wagner, D.: Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In: International Conference on Machine Learning. pp. 274–283 (2018)
3. Brendel, W., Rauber, J., Bethge, M.: Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. arXiv preprint arXiv:1712.04248 (2017)
4. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: 2017 ieee symposium on security and privacy (sp). pp. 39–57. IEEE (2017)
5. Cheng, M., Le, T., Chen, P.Y., Yi, J., Zhang, H., Hsieh, C.J.: Query-efficient hard-label black-box attack: An optimization-based approach. arXiv preprint arXiv:1807.04457 (2018)
6. Cheng, M., Singh, S., Chen, P.H., Chen, P.Y., Liu, S., Hsieh, C.J.: Sign-opt: A query-efficient hard-label adversarial attack. In: ICLR (2020)
7. Cotter, N.E., Conwell, P.R.: Fixed-weight networks can learn. In: 1990 IJCNN International Joint Conference on Neural Networks. pp. 553–559. IEEE (1990)
8. Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: A survey. arXiv preprint arXiv:1808.05377 (2018)
9. Engstrom, L., Ilyas, A., Athalye, A.: Evaluating and understanding the robustness of adversarial logit pairing. arXiv preprint arXiv:1807.10272 (2018)
10. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. pp. 1126–1135. JMLR. org (2017)
11. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
12. Hendrycks, D., Zhao, K., Basart, S., Steinhardt, J., Song, D.: Natural adversarial examples. arXiv preprint arXiv:1907.07174 (2019)
13. Jang, Y., Zhao, T., Hong, S., Lee, H.: Adversarial defense via learning to generate diverse attacks. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 2740–2749 (2019)
14. Jiang, H., Chen, Z., Shi, Y., Dai, B., Zhao, T.: Learning to defense by learning to attack. arXiv preprint arXiv:1811.01213 (2018)
15. Krizhevsky, A., Nair, V., Hinton, G.: Cifar-10 (canadian institute for advanced research). URL http://www. cs. toronto. edu/kriz/cifar. html **8** (2010)
16. Kurakin, A., Goodfellow, I., Bengio, S.: Adversarial examples in the physical world. arXiv preprint arXiv:1607.02533 (2016)
17. Kurakin, A., Goodfellow, I., Bengio, S.: Adversarial machine learning at scale. arXiv preprint arXiv:1611.01236 (2016)
18. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
19. Lee, H., Han, S., Lee, J.: Generative adversarial trainer: Defense to adversarial perturbations with gan. arXiv preprint arXiv:1705.03387 (2017)
20. Lv, K., Jiang, S., Li, J.: Learning gradient descent: Better generalization and longer horizons. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. pp. 2247–2255. JMLR. org (2017)

21. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083 (2017)
22. Metz, L., Maheswaranathan, N., Nixon, J., Freeman, D., Sohl-Dickstein, J.: Understanding and correcting pathologies in the training of learned optimizers. In: International Conference on Machine Learning. pp. 4556–4565 (2019)
23. Moosavi-Dezfooli, S.M., Fawzi, A., Frossard, P.: Deepfool: a simple and accurate method to fool deep neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2574–2582 (2016)
24. Nocedal, J., Wright, S.: Numerical optimization. Springer Science & Business Media (2006)
25. Reddy Mopuri, K., Ojha, U., Garg, U., Venkatesh Babu, R.: Nag: Network for adversary generation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 742–751 (2018)
26. Ruan, Y., Xiong, Y., Reddi, S., Kumar, S., Hsieh, C.J.: Learning to learn by zeroth-order oracle. arXiv preprint arXiv:1910.09464 (2019)
27. Samangouei, P., Kabkab, M., Chellappa, R.: Defense-gan: Protecting classifiers against adversarial attacks using generative models. arXiv preprint arXiv:1805.06605 (2018)
28. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
29. Sinha, A., Singh, M., Kumari, N., Krishnamurthy, B., Machiraju, H., Balasubramanian, V.N.: Harnessing the vulnerability of latent layers in adversarially trained models. arXiv preprint arXiv:1905.05186 (2019)
30. Tsipras, D., Santurkar, S., Engstrom, L., Turner, A., Madry, A.: Robustness may be at odds with accuracy. arXiv preprint arXiv:1805.12152 (2018)
31. Wang, H., Yu, C.N.: A direct approach to robust deep learning using adversarial networks. arXiv preprint arXiv:1905.09591 (2019)
32. Wang, J., Zhang, H.: Bilateral adversarial training: Towards fast training of more robust models against adversarial attacks. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 6629–6638 (2019)
33. Wichrowska, O., Maheswaranathan, N., Hoffman, M.W., Colmenarejo, S.G., Denil, M., de Freitas, N., Sohl-Dickstein, J.: Learned optimizers that scale and generalize. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. pp. 3751–3760. JMLR. org (2017)
34. Wu, Y., Ren, M., Liao, R., Grosse, R.: Understanding short-horizon bias in stochastic meta-optimization. arXiv preprint arXiv:1803.02021 (2018)
35. Xiao, C., Li, B., Zhu, J.Y., He, W., Liu, M., Song, D.: Generating adversarial examples with adversarial networks. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence. pp. 3905–3911 (2018)
36. Xie, C., Wu, Y., Maaten, L.v.d., Yuille, A.L., He, K.: Feature denoising for improving adversarial robustness. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 501–509 (2019)
37. Younger, A.S., Hochreiter, S., Conwell, P.R.: Meta-learning with backpropagation. In: IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222). vol. 3. IEEE (2001)
38. Zagoruyko, S., Komodakis, N.: Wide residual networks. arXiv preprint arXiv:1605.07146 (2016)
39. Zhang, H., Yu, Y., Jiao, J., Xing, E.P., Ghaoui, L.E., Jordan, M.I.: Theoretically principled trade-off between robustness and accuracy. arXiv preprint arXiv:1901.08573 (2019)

A Algorithm for TRADES

As we have emphasized, our proposed method can be incorporated into any adversarial training which can be formulated as a minimax optimization problem. Here we provide the detailed algorithm of RNN-TRADES in Algorithm 2.

Algorithm 2 RNN-based TRADES

```
1: Input: clean data $\{(\boldsymbol{x}, \boldsymbol{y})\}$, batch size $B$, step sizes $\alpha_1$ and $\alpha_2$, number of inner
   iterations $T$, classifier parameterized by $\theta$, RNN optimizer parameterized by $\phi$
2: Output: Robust classifer $f_\theta$, learned optimizer $m_\phi$
3: Randomly initialize $f_\theta$ and $m_\phi$, or initialize them with pre-trained configurations
4: repeat
5:     Sample a mini-batch $M$ from clean data.
6:     for $(\boldsymbol{x}, \boldsymbol{y})$ in $B$ do
7:         Initialization: $\boldsymbol{h}_0 \leftarrow 0$, $\mathcal{L}_\theta \leftarrow 0$, $\mathcal{L}_\phi \leftarrow 0$
8:         Gaussian augmentation: $\boldsymbol{x}'_0 \leftarrow \boldsymbol{x} + 0.001 \cdot \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
9:         for $t = 0, \ldots, T-1$ do
10:            $\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{x}'} \mathcal{L}(f_\theta(\boldsymbol{x}), f_\theta(\boldsymbol{x}'_t))$
11:            $\boldsymbol{\delta}_t, \boldsymbol{h}_{t+1} \leftarrow m_\phi(\boldsymbol{g}_t, \boldsymbol{h}_t)$, where coordinate-wise update is applied
12:            $\boldsymbol{x}'_{t+1} \leftarrow \Pi_{\mathbb{B}(\boldsymbol{x},\epsilon)}(\boldsymbol{x}'_t + \boldsymbol{\delta}_t)$
13:            $\mathcal{L}_\phi \leftarrow \mathcal{L}_\phi + w_{t+1} \mathcal{L}(f_\theta(\boldsymbol{x}), f_\theta(\boldsymbol{x}'_{t+1}))$, where $w_{t+1} = t+1$
14:        end for
15:        $\mathcal{L}_\theta \leftarrow \mathcal{L}_\theta + \mathcal{L}(f_\theta(\boldsymbol{x}), \boldsymbol{y}) + \mathcal{L}(f_\theta(\boldsymbol{x}), f_\theta(\boldsymbol{x}'_T))/\lambda$
16:    end for
17:    Update $\phi$ by $\phi \leftarrow \phi + \alpha_1 \nabla_\phi \mathcal{L}_\phi / B$
18:    Update $\theta$ by $\theta \leftarrow \theta - \alpha_2 \nabla_\theta \mathcal{L}_\theta / B$
19: until training converged
```

B Additional Analysis

B.1 Time comparison

In this section, we compared training time of our proposed methods with the original adversarial training. We still conduct analysis of VGG-16 on CIFAR-10 dataset. From results in Table 8, it can be observed that our methods approximately double the overall training time per epoch, which is not a heavy burden with improved performance taken into account.

B.2 Trajectory

A similar trajectory can be observed in terms of classification accuracy as well when the model is attacked by different attackers. In Figure 4, the robust accuracy under RNN-Adv drops most rapidly and also achieves the lowest point after

Table 8. Time comparison with original adversarial training. Here we report the ratio of our proposed method to its original counterpart, for example, $T_{\text{RNN-Adv}}/T_{\text{AdvTrain}}$. In the parentheses, we report the training time per epoch of our proposed method including RNN-Adv and RNN-TRADES

	Training Time	Ratio of RNN counterpart
AdvTrain	122.24	2.20 (268.50)
TRADES	189.43	2.34 (443.52)

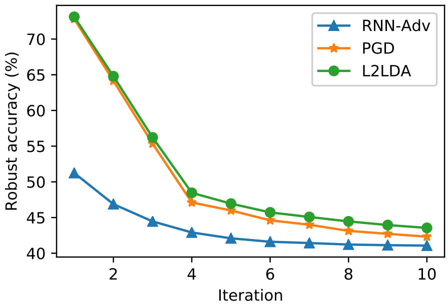


Fig. 4. Accuracy trajectory

the entire 10-step attack. It further verifies that our learned optimizer can guide the optimization along a better trajectory for the inner problem, meaning that crafted adversarial examples are much more powerful. This in turn contributes to a more robust model.

Table 9. Robust accuracy under white-box attacks (Rest. ImageNet, ResNet18)

Defense \ Attack	Natural	PGD-10	PGD-100	CW100	Min
Plain	**97.66**	0.00	0.00	0.00	0.00
AdvTrain	91.43	8.50	4.88	9.04	4.88
TRADES	72.51	6.96	4.80	10.47	4.80
RNN-Adv	90.28	**10.13**	**6.04**	12.98	**6.04**
RNN-TRADES	73.84	9.76	5.79	**13.22**	5.79

C Experiments on Restricted ImageNet

Without loss of generality, we conduct extra experimental analysis on a larger scale dataset, Restricted ImageNet [30]. It is a subset of 9 different super-classes extracted from the entire ImageNet to reduce the computational burden for adversarial training. We adopt the structure of ResNet-18 as the classifier for this dataset. Following the literature [29, 30], the inner attack strength of all defense methods is set to be 0.005 in l_∞ ball while models trained from these mechanisms are evaluated under the attack with the radius of 0.025. Note that in this experiment we only fine-tune the model starting from the naturally trained one for 2 epochs to observe different performance of defense strategies.

From Table 9, we can clearly see that our proposed methods such as RNN-Adv and RNN-TRADES consistently improve the robust accuracy compared with their original adversarial training algorithms. Specifically, models trained by our methods are always $2\% - 3\%$ percents more robust that others under various attacks.