Revisiting and Advancing Fast Adversarial Training Through the Lens of Bi-Level Optimization

Yihua Zhang [1 *] **Guanhua Zhang** [2 *] **Prashant Khanduri** [3] **Mingyi Hong** [3] **Shiyu Chang** [2] **Sijia Liu** [1 4]

Abstract

Adversarial training (AT) is a widely recognized defense mechanism to gain the robustness of deep neural networks against adversarial attacks. It is built on min-max optimization (MMO), where the minimizer (*i.e.*, defender) seeks a robust model to minimize the worst-case training loss in the presence of adversarial examples crafted by the maximizer (*i.e.*, attacker). However, the conventional MMO method makes AT hard to scale. Thus, FAST-AT (Wong et al., 2020) and other recent algorithms attempt to simplify MMO by replacing its maximization step with the single gradient sign-based attack generation step. Although easy to implement, FAST-AT lacks theoretical guarantees, and its empirical performance is unsatisfactory due to the issue of robust catastrophic overfitting when training with strong adversaries. In this paper, we advance FAST-AT from the fresh perspective of bi-level optimization (BLO). We first show that the commonly-used FAST-AT is equivalent to using a stochastic gradient algorithm to solve a linearized BLO problem involving a sign operation. However, the discrete nature of the sign operation makes it difficult to understand the algorithm performance. Inspired by BLO, we design and analyze a new set of robust training algorithms termed **Fast Bi**-level **AT** (FAST-BAT), which effectively defends sign-based projected gradient descent (PGD) attacks without using any gradient sign method or explicit robust regularization. In practice, we show our method yields substantial robustness improvements over baselines across multiple models and datasets. Codes are available at `https://github.com/OPTML-Group/Fast-BAT`.

[*]Equal contribution [1] Michigan State University [2] UC Santa Barbara [3] University of Minnesota [4]MIT-IBM Watson AI Lab, IBM Research. Correspondence to: Yihua Zhang <zhan1908@msu.edu>, Guanhua Zhang <guanhua@ucsb.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

1. Introduction

Given the fact that machine learning (ML) models can be easily fooled by tiny adversarial perturbations (also known as adversarial attacks) on the input (Goodfellow et al., 2014; Carlini & Wagner, 2017; Papernot et al., 2016), training robust deep neural networks is now a major focus in research. Nearly all existing effective defense mechanisms (Madry et al., 2018; Zhang et al., 2019b; Shafahi et al., 2019; Wong et al., 2020; Zhang et al., 2019a; Athalye et al., 2018a) are built on the adversarial training (AT) recipe, introduced by (Szegedy et al., 2014) and later formalized in (Madry et al., 2018) using *min-max optimization* (MMO), where a minimizer (*i.e.* defender) seeks to update model parameters against a maximizer (*i.e.* attacker) that aims to increase the training loss by perturbing the training data.

Despite the effectiveness of the AT-type defenses in various application domains, the min-max nature makes them difficult to scale, because of the *multiple* maximization steps required by the iterative attack generator at each training step. The resulting prohibitive computation cost makes AT not suitable in practical settings. For example, (Xie et al., 2019) used 128 GPUs to run AT on ImageNet. Thereby, how to speed up AT without losing accuracy and robustness is now a *grand challenge* for adversarial defense (Wong et al., 2020).

Recently, (Wong et al., 2020; Shafahi et al., 2019; Zhang et al., 2019a; Andriushchenko & Flammarion, 2020) attempted to develop computationally-efficient alternatives of AT, that is, the 'fast' versions of AT. To the best of our knowledge, FAST-AT (Wong et al., 2020) and FAST-AT with gradient alignment (GA) regularization, termed FAST-AT-GA (Andriushchenko & Flammarion, 2020), are the two state-of-the-art (SOTA) 'fast' versions of AT, since they achieve the most significant reduction in computational complexity and preserve accuracy and robustness to a large extent. In particular, the inner-maximization in FAST-AT (Wong et al., 2020) only calls for single-step attack generation. However, different from the direct application of fast gradient sign method (Goodfellow et al., 2015), the empirical success of FAST-AT also relies on a series of heuristics-based strategies, *e.g.*, using large attack step size, cyclic learning rate schedule, and mixed-precision train-

Table 1. Performance overview of proposed FAST-BAT vs. the baselines FAST-AT (Wong et al., 2020) and FAST-AT-GA (Andriushchenko & Flammarion, 2020) on CIFAR-10, CIFAR-100 and Tiny-ImageNet with PreActResNet-18. All methods are robustly trained under two perturbation budgets $\epsilon = 8/255$ and $16/255$ over 20 epochs. We use the early-stopping policy (Rice et al., 2020) to report the model with best robustness for each method. The evaluation metrics include robust accuracy (RA) against PGD-50-10 attacks (50-step PGD attack with 10 restarts) (Madry et al., 2018) at $\epsilon = 8/255$ and $16/255$ (the test-time ϵ is the *same* as the train-time), RA against AutoAttack (AA) (Croce & Hein, 2020) at $\epsilon = 8/255$ and $16/255$, and computation time (per epoch). The result $a_{\pm b}$ represents mean a and standard deviation b over 10 random trials. All experiments are run on a single GeForce RTX 3090 GPU.

Method	SA (%) ($\epsilon = 8/255$)	RA-PGD (%) ($\epsilon = 8/255$)	RA-AA (%) ($\epsilon = 8/255$)	SA (%) ($\epsilon = 16/255$)	RA-PGD (%) ($\epsilon = 16/255$)	RA-AA (%) ($\epsilon = 16/255$)	Time (s/epoch)
CIFAR-10, PreActResNet-18							
FAST-AT	**82.39**±0.14	45.49±0.21	41.87±0.15	44.15±7.27	21.83±1.32	12.49±0.33	23.1
FAST-AT-GA	79.71±0.24	47.27±0.22	43.24±0.27	58.29±1.32	26.01±0.16	17.97±0.33	75.3
FAST-BAT	79.97±0.12	**48.83**±0.17	**45.19**±0.12	**68.16**±0.25	**27.69**±0.16	**18.79**±0.24	61.4
CIFAR-100, PreActResNet-18							
FAST-AT	**52.62** ±0.18	24.66±0.21	21.72±0.17	21.32±3.27	8.62±1.03	6.22±0.61	23.8
FAST-AT-GA	50.06±0.27	24.97±0.23	21.82±0.21	32.51±1.27	12.27±0.36	9.43±0.19	77.1
FAST-BAT	50.19±0.21	**26.49**±0.20	**23.97**±0.15	**39.29**±0.53	**13.97**±0.17	**11.32**±0.22	61.6
Tiny-ImageNet, PreActResNet-18							
FAST-AT	41.37±3.08	17.05±3.25	12.31±2.73	31.38±0.19	5.42±2.17	3.13±0.24	284.6
FAST-AT-GA	45.52±0.24	20.39±0.19	16.25±0.17	29.17±0.32	6.79±0.27	4.27±0.15	592.7
FAST-BAT	**45.80**±0.22	**21.97**±0.21	**17.64**±0.15	**33.78**±0.23	**8.83**±0.22	**5.52**±0.14	572.4

ing. Yet, FAST-AT suffers from two main issues: *(i)* lack of stability, *i.e.*, it has a large variance in performance (Li et al., 2020), and *(ii)* catastrophic overfitting, *i.e.*, when training with strong adversaries, the robustness of the resulting model can drop significantly. To alleviate these problems, Andriushchenko & Flammarion (2020) proposed FAST-AT-GA by penalizing FAST-AT using an explicit GA regularization. However, we will show that FAST-AT-GA encounters another problem: *(iii)* It improves robust accuracy (RA) at the cost of a sharp drop in standard accuracy (SA), leading to a poor accuracy-robustness tradeoff for large attack budget (*e.g.*, $\epsilon = 16/255$ in Table 1). Moreover, *(iv)* there has been no theoretical guarantee for the optimization algorithms used in FAST-AT and FAST-AT-GA. Given the limitations *(i)- (iv)*, we ask:

How to design a 'fast' AT *with improved stability, mitigated catastrophic overfitting, enhanced RA-SA tradeoff, and some theoretical guarantees?*

To address the above question, we formulate the AT problem as a unified **bi-level optimization (BLO)** problem (Dempe, 2002). In the new formulation, the attack generation is cast as a constrained *lower-level* optimization problem, while the defense serves as the *upper-level* optimization problem. To the best of our knowledge, this is the first work that makes a rigorous connection between adversarial defense and BLO. Technically, we show that FAST-AT can be *interpreted* as BLO with linearized lower-level problems. Delving into the linearization of BLO, we propose a novel, theoretically-grounded 'fast' AT framework, termed **fast b**i-level **AT** (FAST-BAT). Practically, Table 1 highlights some achieved improvements over FAST-AT and FAST-AT-GA: When a stronger train-time attack (*i.e.*, $\epsilon = 16/255$) is adopted, FAST-AT suffers from a large degradation of RA and SA, together with a very high variances. Although FAST-AT-GA outperforms FAST-AT, it still incurs a significant SA loss at $\epsilon = 16/255$. In contrast, FAST-BAT produces the best robustness with a more graceful SA-RA tradeoff: As FAST-BAT brings an improvement of over 1.5% on RA in all experimental settings, its SA still remains at a high level, *e.g.* a significant improvement on SA by 9.9% and 6.7% at $\epsilon = 16/255$ for CIFAR-10 and CIFAR-100.

Contributions. We summarize our contributions below.

① (Formulation-wise) We propose a unified BLO-based formulation for the robust training problem. Within this formulation, we show the conventional FAST-AT method is solving a low-level linearized BLO problem, rather than the vanilla min-max problem. This key observation not only provides a new interpretation of FAST-AT, but most importantly, also explains why FAST-AT is difficult to possess strong theoretical guarantees.

② (Methodology-wise) We propose the new FAST-BAT algorithm based on our new understanding of FAST-AT. The key enabling technique is to introduce a new *smooth* lower-level objective of BLO for robust training. In contrast to MMO, BLO adopts a different optimization routine that requires implicit gradient (IG) computation. We derive the closed-form of IG for FAST-BAT.

③ (Experiment-wise) We made a comprehensive experimental study to demonstrate the effectiveness of FAST-BAT over SOTA baselines across datasets and model types. Besides its merit in robustness enhancement, we also show its improved stability, lifted accuracy-robustness trade-off, and mitigated catastrophic overfitting.

2. Related work

Adversarial attack. Adversarial attacks are techniques to generate malicious perturbations that are imperceptible to humans but can mislead the ML models (Goodfellow et al., 2014; Carlini & Wagner, 2017; Croce & Hein, 2020; Xu et al., 2019; Athalye et al., 2018b). The adversarial attack has become a major approach to evaluate the robustness of deep neural networks and thus, help build safe artificial intelligence in many high-stakes applications such as autonomous driving (Deng et al., 2020; Kumar et al., 2020) and surveillance (Thys et al., 2019; Xu et al., 2020).

Adversarial defense and robust training at scale. Our work falls into the category of robust training, which was mostly built on MMO. For example, (Madry et al., 2018) established the framework of AT for the first time, always recognized as one of the most powerful defenses (Athalye et al., 2018a). Extended from AT, TRADES (Zhang et al., 2019b) sought the optimal balance between robustness and generalization ability. Further, AT-type defense has been generalized to the semi-/self-supervised settings (Carmon et al., 2019; Chen et al., 2020b) and integrated 1 with certified defense techniques such as randomized smoothing (Salman et al., 2019).

Despite the effectiveness of AT and its variants, how to speed up AT without losing performance remains an open question. Some recent works attempted to impose algorithmic simplifications to AT, leading to *fast but approximate* AT algorithms, such as 'free' AT (Shafahi et al., 2019), you only propagate once (YOPO) (Zhang et al., 2019a), FAST-AT (Wong et al., 2020), and FAST-AT regularized by gradient alignment (termed FAST-AT-GA) (Andriushchenko & Flammarion, 2020). In particular, FAST-AT and FAST-AT-GA are the baselines most relevant to ours due to their low computational complexity. However, their defense performance is still unsatisfactory. For example, FAST-AT has poor training stability (Li et al., 2020) and suffers catastrophic overfitting when facing strong attacks (Andriushchenko & Flammarion, 2020). In contrast, FAST-AT-GA yields improved robustness but has a poor accuracy-robustness tradeoff (see Table 1).

Bi-level optimization (BLO). BLO is a unified hierarchical learning framework, where the objective and variables of the upper-level problem depend on the lower-level problems. The BLO problem in its most generic form is very challenging, and thus, the design of algorithms and theory for BLO focuses on special cases (Vicente et al., 1994; White & Anandalingam, 1993; Gould et al., 2016; Ghadimi & Wang, 2018; Khanduri et al., 2021; Ji et al., 2020; Hong et al., 2020). In practice, BLO has been successfully applied to meta-learning (Rajeswaran et al., 2019), data poisoning attack design (Huang et al., 2020), and reinforcement learning (Chen et al., 2019). However, as will be evident later, the existing BLO approach is not directly applied to adversarial defense due to the presence of the *constrained* non-convex lower-level problem (for attack generation). To the best of our knowledge, our work makes a rigorous connection between adversarial defense and BLO for the first time.

3. A Bi-Level Optimization View on FAST-AT

Preliminaries on AT and FAST-AT. Let us consider the following standard min-max formulation for the robust adversarial training problem (Madry et al., 2018)

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \ \mathbb{E}_{(\mathbf{x},y)\in\mathcal{D}} \left[\underset{\boldsymbol{\delta}\in\mathcal{C}}{\text{maximize}} \ell_{\text{tr}}(\boldsymbol{\theta}, \mathbf{x}+\boldsymbol{\delta}, y) \right], \quad (1)$$

where $\boldsymbol{\theta} \in \mathbb{R}^n$ denotes model parameters; $\mathcal{D}$ is the training set consisting (a finite number) of labeled data pairs with feature $\mathbf{x}$ and label y; $\boldsymbol{\delta} \in \mathbb{R}^d$ represents adversarial perturbations subject to the perturbation constraint $\mathcal{C}$, *e.g.*, $\mathcal{C} = \{\boldsymbol{\delta} \,|\, \|\boldsymbol{\delta}\|_\infty \le \epsilon, \boldsymbol{\delta} \in [\mathbf{0}, \mathbf{1}]\}$ for ϵ-toleration ℓ_∞-norm constrained attack (normalized to $[\mathbf{0}, \mathbf{1}]$); $\ell_{\text{tr}}(\cdot)$ is the training loss; $(\mathbf{x}+\boldsymbol{\delta})$ represents an adversarial example.

The standard solver for problem (1) is known as AT (Madry et al., 2018). However, it has to call an *iterative* optimization method (*e.g.*, K-step PGD attack) to solve the inner maximization problem of (1), which is computationally expensive. To improve its scalability, the FAST-AT algorithm (Wong et al., 2020) was proposed to take a *single-step* PGD attack for inner maximization. The algorithm backbone of FAST-AT is summarized below.

FAST-AT algorithm Let $\boldsymbol{\theta}_t$ be parameters at iteration t. The $(t+1)$th iteration becomes (Wong et al., 2020):

• *Inner maximization by* 1*-step PGD:*

$$\boldsymbol{\delta} \leftarrow \mathcal{P}_{\mathcal{C}} \left(\boldsymbol{\delta}_0 + \alpha \cdot \text{sign} \left(\nabla_{\boldsymbol{\delta}} \ell_{\text{tr}}(\boldsymbol{\theta}_t, \mathbf{x}+\boldsymbol{\delta}_0, y) \right) \right),$$

where $\mathcal{P}_{\mathcal{C}}(\mathbf{a})$ denotes the projection operation that projects the point $\mathbf{a}$ onto $\mathcal{C}$, *i.e.*, $\mathcal{P}_{\mathcal{C}}(\mathbf{z}) = \arg\min_{\boldsymbol{\delta}\in\mathcal{C}} \|\boldsymbol{\delta} - \mathbf{z}\|_2^2$, $\boldsymbol{\delta}_0$ is a random uniform initialization within $[\mathbf{0}, \mathbf{1}]$, $\alpha > 0$ is a proper learning rate (*e.g.*, 1.25ϵ), and $\text{sign}(\cdot)$ is the element-wise sign operation.

• *Outer minimization for model training:* This can be conducted by any standard optimizer, *e.g.*,

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \beta \nabla_{\boldsymbol{\theta}} \ell_{\text{tr}}(\boldsymbol{\theta}_t, \mathbf{x}+\boldsymbol{\delta}, y)$$

where $\beta > 0$ is a proper learning rate (*e.g.*, cyclic learning rate), and $\boldsymbol{\delta}$ is provided from the inner maximization step.

A few remarks about FAST-AT are given below.

① Roughly speaking, FAST-AT is a simplification of AT using 1-step PGD for inner maximization. However, it is not entirely clear which problem FAST-AT is actually solving. If we take a closer look at the algorithm, we will see that the

inner update only changes to the *initial* $\boldsymbol{\delta}_0$, not the most recent $\boldsymbol{\delta}$. Clearly, this scheme is fundamentally *different* from the classical gradient descent-ascent algorithm for min-max optimization (Razaviyayn et al., 2020), which alternatively updates the inner and outer variables. *Therefore, it remains elusive if* FAST-AT *is solving the original problem (1).*

② Andriushchenko & Flammarion (2020) demonstrated that FAST-AT could lead to catastrophic overfitting when using strong adversaries for training. However, there was no grounded theory to justify the pros and cons of FAST-AT. We will show that BLO provides a promising solution.

BLO: Towards a unified formulation of robust training. BLO (bi-level optimization) is an optimization problem that involves two levels of optimization tasks, where the *lower-level* task is nested inside the *upper-level* task. More precisely, it has the following generic form:

$$\begin{array}{ll} \min\limits_{\mathbf{u}\in\mathcal{U}} & f(\mathbf{u}, \mathbf{v}^*(\mathbf{u})) \\ \text{s.t.} & \mathbf{v}^*(\mathbf{u}) = \arg\min_{\mathbf{v}\in\mathcal{V}}\ g(\mathbf{v}, \mathbf{u}) \end{array} \tag{2}$$

where $\mathcal{U}$ and $\mathcal{V}$ are the feasible sets for the variables $\mathbf{u}$ and $\mathbf{v}$, respectively; $f(\cdot)$ and $g(\cdot)$ are the upper- and the lower-level objective functions, respectively. Intuitively, the BLO (2) can be used to formulate the adversarial training problem as the latter also involves two problems, one nested in the other. Importantly, it is more powerful than the min-max formulation (1) as it allows the two problems to have *different* objective functions. This flexibility provided by BLO is the key to the generality of our proposed framework.

To make the above intuition precise, we use the upper-level problem to model the training loss minimization, while the lower-level problem to model the attack generation process, and consider the following BLO problem:

$$\begin{array}{ll} \min\limits_{\boldsymbol{\theta}} & \mathbb{E}_{(\mathbf{x},y)\in\mathcal{D}}[\ell_{\text{tr}}(\boldsymbol{\theta}, \mathbf{x}+\boldsymbol{\delta}^*(\boldsymbol{\theta};\mathbf{x},y); y)] \\ \text{s.t.} & \boldsymbol{\delta}^*(\boldsymbol{\theta};\mathbf{x},y) = \arg\min\limits_{\boldsymbol{\delta}\in\mathcal{C}} \ell_{\text{atk}}(\boldsymbol{\theta}, \boldsymbol{\delta}; \mathbf{x}, y), \end{array} \tag{3}$$

where the training loss function $\ell_{\text{tr}}(\cdot)$ has been defined in (1), and $\ell_{\text{atk}}(\cdot)$ denotes an attack objective. For the notation simplicity, in the subsequent discussion, we will not indicate the dependency of the functions ℓ_{tr}, ℓ_{atk}, and $\boldsymbol{\delta}^*$ with respect to (w.r.t.) the data samples $(\mathbf{x}, y)$. The formulation (3) has two key differences from (1):

① When we choose $\ell_{\text{atk}} = -\ell_{\text{tr}}$, problem (3) becomes equivalent to the min-max formulation (1). It follows that the BLO is suitable to formulate the adversarial training problem. Moreover, we will see shortly that the flexibility provided by choosing the lower-level objective *independently* of the upper-level one enables us to interpret FAST-AT as solving a certain special form of the BLO problem. Note that prior to our work, it was not entirely clear what is the problem that FAST-AT is trying to solve.

② BLO calls a different optimization routine from those to solve the original min-max problem (1). As will be evident later, even if we set $\ell_{\text{atk}} = -\ell_{\text{tr}}$ in (3), the BLO-enabled solver does not simplify to FAST-AT (see more details in Appendix B). This is because for a given data sample $(\mathbf{x}, y)$, the gradient for the upper-level problem of (3) yields:

$$\frac{d\ell_{\text{tr}}(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))}{d\boldsymbol{\theta}} = \nabla_{\boldsymbol{\theta}}\ell_{\text{tr}}(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})) + \underbrace{\frac{d\boldsymbol{\delta}^*(\boldsymbol{\theta})^\top}{d\boldsymbol{\theta}}}_{\text{IG}} \nabla_{\boldsymbol{\delta}}\ell_{\text{tr}}(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})), \tag{4}$$

where the superscript $\top$ denotes the transpose operation, and $\nabla_{\boldsymbol{\theta}}\ell_{\text{tr}}(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))$ denotes the partial derivative w.r.t. the first input argument $\boldsymbol{\theta}$; and $\frac{d\boldsymbol{\delta}^*(\boldsymbol{\theta})^\top}{d\boldsymbol{\theta}} \in \mathbb{R}^{n\times d}$, if exists, is referred to as *implicit gradient (IG)* because it is defined through an implicitly constrained optimization problem $\min_{\boldsymbol{\delta}\in\mathcal{C}} \ell_{\text{atk}}$. The dependence on IG is a 'fingerprint' of BLO (1) in contrast to AT or FAST-AT.

BLO-enabled interpretation of FAST-AT. Next, we demonstrate how FAST-AT relates to BLO. FAST-AT can be interpreted as an approximated stochastic gradient algorithm for solving the following **lower-level linearized BLO**. That is to say, FAST-AT is not solving the original min-max problem (1), but its linearized version below:

$$\begin{array}{l} \min\limits_{\boldsymbol{\theta}}\ \mathbb{E}_{(\mathbf{x},y)\in\mathcal{D}}[\ell_{\text{tr}}(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))] \\ \text{s.t. } \boldsymbol{\delta}^*(\boldsymbol{\theta}) = \arg\min\limits_{\boldsymbol{\delta}\in\mathcal{C}}\ [(\boldsymbol{\delta}-\mathbf{z})^\top \text{sign}(\nabla_{\boldsymbol{\delta}=\mathbf{z}}\ell_{\text{atk}}(\boldsymbol{\theta}, \boldsymbol{\delta})) + (\lambda/2)\|\boldsymbol{\delta}-\mathbf{z}\|_2^2], \end{array} \tag{5}$$

where $\mathbf{z} = \boldsymbol{\delta}_0$ and $\lambda = 1/\alpha$. Our justification for the above interpretation is elaborated on below.

① The simplified lower-level problem of (5) leads to the **closed-form** solution:

$$\begin{aligned} \boldsymbol{\delta}^*(\boldsymbol{\theta}) &= \arg\min_{\boldsymbol{\delta}\in\mathcal{C}} (\lambda/2)\|\boldsymbol{\delta} - \mathbf{z} + (1/\lambda)\text{sign}(\nabla_{\boldsymbol{\delta}=\mathbf{z}}\ell_{\text{atk}}(\boldsymbol{\theta}, \boldsymbol{\delta}))\|_2^2 \\ &= \mathcal{P}_{\mathcal{C}}\left(\mathbf{z} - (1/\lambda)\text{sign}(\nabla_{\boldsymbol{\delta}=\mathbf{z}}\ell_{\text{atk}}(\boldsymbol{\theta}, \boldsymbol{\delta}))\right), \end{aligned} \tag{6}$$

which is exactly given by the 1-step PGD attack with initialization $\mathbf{z}$ and learning rate $(1/\lambda)$. In the linearization used in (5), a quadratic regularization term (with regularization parameter λ) is introduced to ensure the strong convexity of the lower-level objective within the constraint set $\boldsymbol{\delta} \in \mathcal{C}$ so as to achieve the unique minimizer (6). Note that imposing such a strongly convex regularizer is also commonly used to stabilize the convergence of MMO (min-max optimization) and BLO (Qian et al., 2019; Hong et al., 2020). If we set $\mathbf{z} = \boldsymbol{\delta}_0$ and $\lambda = 1/\alpha$, (6) precisely depicts the inner maximization step used in FAST-AT.

② By substituting (6) into the upper-level problem of (5), we can then use (4) to compute the stochastic gradients of the upper-level problem. If the stochastic gradient can be precisely computed, we can update the model parameters $\boldsymbol{\theta}$ using SGD based on (4). That is, $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \beta \frac{d\ell_{\text{tr}}(\boldsymbol{\theta}_t, \boldsymbol{\delta}^*(\boldsymbol{\theta}))}{d\boldsymbol{\theta}}$ (with learning rate β). However, generally speaking, the IG function $\frac{d\boldsymbol{\delta}^*(\boldsymbol{\theta})^\top}{d\boldsymbol{\theta}}$ involved in (4) may not be differentiable, and even it is, the computation may not be easy. For our case, $\boldsymbol{\delta}^*(\boldsymbol{\theta})$ expressed in (6) involves both a projection and a sign

operation, which can be particularly difficult to compute. To proceed, let us make the following approximations. We assume that the chain rule of the derivative of $\boldsymbol{\delta}^*(\boldsymbol{\theta})$ holds w.r.t. $\boldsymbol{\theta}$, implying the differentiability of the projection operation and the sign operation. Then, based on the closed-form of $\boldsymbol{\delta}^*(\boldsymbol{\theta})$ in (6), IG is approximately equal to $\frac{d\boldsymbol{\delta}^*(\boldsymbol{\theta})^\top}{d\boldsymbol{\theta}}=\mathbf{0}$, where we use two facts: (1) The linearization point $\mathbf{z}$ is independent of $\boldsymbol{\theta}$, *i.e.* $\mathbf{z}=\boldsymbol{\delta}_0$; And (2) $\frac{d\text{sign}(\cdot)}{d\boldsymbol{\theta}}=\mathbf{0}$ holds almost everywhere. Clearly, the use of the gradient sign method simplifies the IG computation. Further, the upper-level gradient is approximated by $\frac{d\ell_{\text{tr}}(\boldsymbol{\theta},\boldsymbol{\delta}^*(\boldsymbol{\theta}))}{d\boldsymbol{\theta}}\approx\nabla_{\boldsymbol{\theta}}\ell_{\text{tr}}(\boldsymbol{\theta},\boldsymbol{\delta}^*(\boldsymbol{\theta})):=\tilde{\mathbf{h}}(\boldsymbol{\theta})$, and the upper-level optimization of problem (5) becomes $\boldsymbol{\theta}\leftarrow\boldsymbol{\theta}-\beta\tilde{\mathbf{h}}(\boldsymbol{\theta})$, which is the same as the outer minimization step in FAST-AT. In a nutshell, the BLO solver of problem (5), which calls the IG computation based on (6), eventually reduces to the FAST-AT algorithm.

The aforementioned analysis shows that FAST-AT can be viewed as using an approximated stochastic gradient algorithm to solve the linearized BLO (5), with the linearization point $\mathbf{z}$ and the regularization parameter λ set as $\mathbf{z}=\boldsymbol{\delta}_0$ and $\lambda=1/\alpha$. However, since a series of approximations have been used when arriving at the approximated gradient $\widetilde{\mathbf{h}}(\boldsymbol{\theta})$ used by FAST-AT, it is no longer clear if the resulting algorithm can still sufficiently reduce the objective function of the upper-level problem. Additionally, based on the fact that the lower-level problem of (5) involves the *discrete* sign operator, it is unlikely (if not impossible) that any approximated stochastic gradient-based algorithms developed for it can possess any strong theoretical guarantees.

4. FAST-BAT: Advancing FAST-AT by BLO

FAST-BAT and rationale. The key take-away from (5) is that the conventional FAST-AT adopts the *sign of input gradient* to linearize the lower-level attack objective. However, a more natural choice is to use the first-order Taylor expansion for linearization. By doing so, problem (5) can be modified to the following form:

$$\begin{aligned}&\min_{\boldsymbol{\theta}}\ \mathbb{E}_{(\mathbf{x},y)\in\mathcal{D}}[\ell_{\text{tr}}(\boldsymbol{\theta},\boldsymbol{\delta}^*(\boldsymbol{\theta}))] \qquad (7)\\ &\text{s.t.}\ \ \boldsymbol{\delta}^*(\boldsymbol{\theta})=\arg\min_{\boldsymbol{\delta}\in\mathcal{C}}\ [(\boldsymbol{\delta}-\mathbf{z})^\top(\nabla_{\boldsymbol{\delta}=\mathbf{z}}\ell_{\text{atk}}(\boldsymbol{\theta},\boldsymbol{\delta}))+(\lambda/2)\|\boldsymbol{\delta}-\mathbf{z}\|_2^2],\end{aligned}$$

Similar to (6), problem (7) can be solved as:

$$\boldsymbol{\delta}^*(\boldsymbol{\theta})=\mathcal{P}_{\mathcal{C}}\left(\mathbf{z}-(1/\lambda)\nabla_{\boldsymbol{\delta}=\mathbf{z}}\ell_{\text{atk}}(\boldsymbol{\theta},\boldsymbol{\delta})\right). \qquad (8)$$

In contrast to (6), the IG associated with (7) cannot be approximated by zeros since the gradient sign operation is not present in (8). To compute IG, the auto-differentiation (the chain rule) can be applied to the closed-form of $\boldsymbol{\delta}^*(\boldsymbol{\theta})$. However, this will not give us an accurate and generalizable IG solution since the projection operation $\mathcal{P}_{\mathcal{C}}$ is *not* smooth everywhere and thus, the use of chain rule does not yield a rigorous derivation. In the following subsection, we address the *IG challenge* in a theoretically-grounded manner.

IG theory for FAST-BAT. The problem of FAST-BAT (7) falls into a class of very challenging BLO problems, which requires *constrained* lower-level optimization. The unconstrained case is easier to handle since one can apply the implicit function theory to the stationary condition of the lower-level problem to obtain IG (Hong et al., 2020). Yet, in our case with *constrained* problems, a stationary point could violate the constraints, and thus, the stationary condition becomes non-applicable. Fortunately, in problem (7), we are dealing with a special class of lower-level constraints – *linear constraints*. Let us rewrite the constraints below:

$$\|\boldsymbol{\delta}\|_\infty\leq\epsilon,\boldsymbol{\delta}\in[-\mathbf{x},\mathbf{1}-\mathbf{x}]\Longleftrightarrow\mathbf{B}\boldsymbol{\delta}\leq\mathbf{b}, \qquad (9)$$

where $\mathbf{B}:=\begin{bmatrix}\mathbf{I}\\-\mathbf{I}\end{bmatrix}$, $\mathbf{b}:=\begin{bmatrix}\min\{\epsilon\mathbf{1},\mathbf{1}-\mathbf{x}\}\\-\max\{-\epsilon\mathbf{1},-\mathbf{x}\}\end{bmatrix}$. By exploiting the above linearly constrained problem structure, we show that the IG challenge associated with (7) can be addressed via *Karush–Kuhn–Tucker (KKT)* conditions. We present our main theoretical results (Proposition 1 and Corollary 1) below and refer readers to Appendix A for detailed derivation.

Proposition 1 *For the BLO problem (7), let $g(\boldsymbol{\theta},\boldsymbol{\delta}^*)$ denote its lower-level objective function evaluated at $\boldsymbol{\delta}^*$ given $\boldsymbol{\theta}$, then the analytical form of IG (implicit gradient) is given by*

$$\begin{aligned}\frac{d\boldsymbol{\delta}^*(\boldsymbol{\theta})^\top}{d\boldsymbol{\theta}}&=-\nabla_{\boldsymbol{\theta}\boldsymbol{\delta}}g(\boldsymbol{\theta},\boldsymbol{\delta}^*)\nabla_{\boldsymbol{\delta}\boldsymbol{\delta}}g(\boldsymbol{\theta},\boldsymbol{\delta}^*)^{-1} \qquad (10)\\ &+\nabla_{\boldsymbol{\theta}\boldsymbol{\delta}}g(\boldsymbol{\theta},\boldsymbol{\delta}^*)\nabla_{\boldsymbol{\delta}\boldsymbol{\delta}}g(\boldsymbol{\theta},\boldsymbol{\delta}^*)^{-1}\mathbf{B}_0^\top[\mathbf{B}_0\nabla_{\boldsymbol{\delta}\boldsymbol{\delta}}g(\boldsymbol{\theta},\boldsymbol{\delta}^*)^{-1}\mathbf{B}_0^\top]^{-1}\mathbf{B}_0\nabla_{\boldsymbol{\delta}\boldsymbol{\delta}}g(\boldsymbol{\theta},\boldsymbol{\delta}^*)^{-1},\end{aligned}$$

where $\boldsymbol{\delta}^*$ is given by (8) (the dependence on $\boldsymbol{\theta}$ is omitted for ease of notation), $\nabla_{\boldsymbol{\theta}\boldsymbol{\delta}}g(\boldsymbol{\theta},\boldsymbol{\delta}^*)\in\mathbb{R}^{n\times d}$ denotes the second-order partial derivatives evaluated at $(\boldsymbol{\theta},\boldsymbol{\delta}^*)$, and $\mathbf{B}_0$ denotes the sub-matrix of $\mathbf{B}$ that only corresponds to the active constraints at $\boldsymbol{\delta}^*$, i.e., those in $\mathbf{B}\boldsymbol{\delta}^*\leq\mathbf{b}$ satisfied with equality.

It is clear from (10) that the computation of IG requires the second-order derivatives as well as matrix inversion. This is computationally intensive. Recall from (7) that the Hessian matrix $\nabla_{\boldsymbol{\delta}\boldsymbol{\delta}}g$ of the lower-level objective function is given by $\nabla_{\boldsymbol{\delta}\boldsymbol{\delta}}g(\boldsymbol{\theta},\boldsymbol{\delta}^*)=\nabla_{\boldsymbol{\delta}\boldsymbol{\delta}}\ell_{\text{atk}}+\lambda\mathbf{I}$. This inspires us to impose the Hessian-free approximation, *i.e.*, $\nabla_{\boldsymbol{\delta}\boldsymbol{\delta}}\ell_{\text{atk}}=\mathbf{0}$. The rationale behind the Hessian-free assumption is that ReLU-based neural networks commonly lead to a piecewise linear decision boundary w.r.t. the inputs (Moosavi-Dezfooli et al., 2019; Alfarra et al., 2020), and thus, its second-order derivative (Hessian) $\nabla_{\boldsymbol{\delta}\boldsymbol{\delta}}\ell_{\text{atk}}$ is close to zero. In Sec. 5.2 and Appendix D, we will empirically show that the Hessian-free assumption is reasonable for both ReLU and non-ReLU neural networks. Thus, the Hessian matrix is approximated as:

$$\nabla_{\boldsymbol{\delta}\boldsymbol{\delta}}g(\boldsymbol{\theta},\boldsymbol{\delta}^*(\boldsymbol{\theta}))=\nabla_{\boldsymbol{\delta}\boldsymbol{\delta}}\ell_{\text{atk}}+\lambda\mathbf{I}=\mathbf{0}+\lambda\mathbf{I}. \qquad (11)$$

With (11), we can simplify closed-form of IG as below:

Corollary 1 *With the Hessian-free assumption, namely* $\nabla_{\boldsymbol{\delta}\boldsymbol{\delta}}\ell_{\text{atk}} = \mathbf{0}$*, the IG (implicit gradient) of (7) is*

$$\frac{d\boldsymbol{\delta}^*(\boldsymbol{\theta})^\top}{d\boldsymbol{\theta}} = -(1/\lambda)\nabla_{\boldsymbol{\theta}\boldsymbol{\delta}}\ell_{\text{atk}}(\boldsymbol{\theta}, \boldsymbol{\delta}^*)\mathbf{H}_{\mathcal{C}}, \tag{12}$$

where $\mathbf{H}_{\mathcal{C}} := \left[1_{p_1<\delta_1^*<q_1}\mathbf{e}_1 \cdots 1_{p_1<\delta_d^*<q_d}\mathbf{e}_d\right] \in \mathbb{R}^{d\times d}$ *and the function* $1_{p_i<\delta_i^*<q_i} \in \{0,1\}$ *denotes the indicator over the constraint of* $\{\delta_i \,|\, p_i < \delta_i^* < q_i\}$*, which returns* 1 *if the constraint is satisfied,* δ_i^* *denotes the* i*th entry of* $\boldsymbol{\delta}^*(\boldsymbol{\theta})$*,* $p_i = \max\{-\epsilon, -x_i\}$ *and* $q_i = \min\{\epsilon, 1-x_i\}$ *characterize the boundary of the linear constraint (9) for the variable* δ_i*, and* $\mathbf{e}_i \in \mathbb{R}^d$ *denotes the basis vector with the* i*th entry being* 1 *and others being* 0*.*

FAST-BAT algorithm and implementation. Similar to FAST-AT or AT, the FAST-BAT algorithm also follows the principle of alternating optimization. Specifically, it consists of the IG-based upper-level gradient descent (4), interlaced with the lower-level optimal attack (8). We summarize the FAST-BAT algorithm below.

• *Lower-level solution*: Obtain $\boldsymbol{\delta}^*(\boldsymbol{\theta}_t)$ from (8);

$$\boldsymbol{\delta}^*(\boldsymbol{\theta}) = \mathcal{P}_{\mathcal{C}}\left(\mathbf{z} - (1/\lambda)\nabla_{\boldsymbol{\delta}=\mathbf{z}}\ell_{\text{atk}}(\boldsymbol{\theta}, \boldsymbol{\delta})\right).$$

• *Upper-level model training*: Integrating the IG (12) into (4), call SGD to update the model parameters as:

$$\begin{aligned}\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t &- \alpha_1\nabla_{\boldsymbol{\theta}}\ell_{\text{tr}}(\boldsymbol{\theta}_t, \boldsymbol{\delta}^*) \\ &- \alpha_2(-\frac{1}{\lambda})\nabla_{\boldsymbol{\theta}\boldsymbol{\delta}}\ell_{\text{atk}}(\boldsymbol{\theta}_t, \boldsymbol{\delta}^*)\mathbf{H}_{\mathcal{C}}\nabla_{\boldsymbol{\delta}}\ell_{\text{tr}}(\boldsymbol{\theta}_t, \boldsymbol{\delta}^*),\end{aligned} \tag{13}$$

where $\alpha_1, \alpha_2 > 0$ are learning rates associated with the model gradient and the IG-augmented descent direction.

It is clear from (13) that to train a robust model, FAST-BAT can be decomposed into the regular FAST-AT update (*i.e.*, the term multiplied by α_1) and the additional update that involves IG, (*i.e.*, the term multiplied by α_2). To implement FAST-BAT, we highlight some key hyper-parameter setups that are different from FAST-AT (Wong et al., 2020) and FAST-AT-GA (Andriushchenko & Flammarion, 2020).

Remark on implementation details Next, we discuss the practical setups used in Fast-BAT (more details in Appendix C). First, $1/\lambda$ serves as the attack step size as shown in (8). We refer readers to Table 7 for a sensitivity analysis of λ. In Fast-BAT, the hyperparameter α_2 is newly introduced, and is set differently from α_1 so as to control the descent error associated with the coupled second-order/first-order stochastic derivatives, *i.e.*, the α_2-term in (13). In Sec. 7, we show that a proper α_2 helps mitigate catastrophic overfitting. In practice, we set $\alpha_2/\lambda = 0.1\alpha_1$. To specify the linearization point $\mathbf{z}$ in (7), we investigate two types of linearization schemes: (1) the random constant linearization (random uniform and random corner linearization) and (2) the 1-step perturbation warm-up-based linearization (1-step sign-based and 1-step w/o sign PGD). These linearization schemes have computational complexities up to the one-step attack generation. Empirically, we find that FAST-BAT using "1-step PGD w/o sign" leads to the best defensive performance (see Table 9). We follow this experiment setup in the sequel.

Remark on convergence analysis In Appendix E, we prove that under some assumptions on the gradient bias, Fast-BAT converges to a first-order stationary point or its small neighborhood in the rate of $\mathcal{O}(1/\sqrt{T})$, where T is the iteration number of model updates. The main analysis difficulty lies in the last term of the model updating rule (13), which involves two coupled derivatives built upon the same mini-batch.

5. Experiments

5.1. Experiment Setup

Datasets and model architectures. We will evaluate the effectiveness of our proposal under CIFAR-10 (Krizhevsky & Hinton, 2009), CIFAR-100 (Krizhevsky & Hinton, 2009), Tiny-ImageNet (Deng et al., 2009), and ImageNet (Deng et al., 2009). Unless specified otherwise, we will train DNN models PreActResNet (PARN)-18 (He et al., 2016b) for all datasets except ImageNet, and ResNet (RN)-50 (He et al., 2016a) for ImageNet. As a part of the ablation study, we also train larger models PARN-50 and WideResNet (WRN)-16-8 (Zagoruyko & Komodakis, 2016) on CIFAR-10. Some preliminary ImageNet results are reported in Appendix D.

Baselines. We focus on three baselines, namely, FAST-AT (Wong et al., 2020), FAST-AT-GA (Andriushchenko & Flammarion, 2020), and PGD-2-AT (Madry et al., 2018), and refer readers to comparisons with more baselines (PGD-7-AT (Madry et al., 2018), BACKSMOOTH (Chen et al., 2020a), FREE-AT (Shafahi et al., 2019), ATTA (Zheng et al., 2020), YOPO (Zhang et al., 2019a)) in Appendix D. Here PGD-2-AT and PGD-7-AT stand for the 2-step and 7-step PGD attack-based AT, respectively. The primal criterion of baseline selection is computational complexity. All the methods except PGD-7-AT consume the training time of the same order, while PGD-7-AT serves as a reference to the performance of the non-accelerated robust training method. We stress that FAST-AT-GA is the strongest baseline to the best of our knowledge in terms of improving robustness and mitigating robust catastrophic overfitting (Andriushchenko & Flammarion, 2020).

Training details. We choose the training perturbation strength $\epsilon \in \{8, 16\}/255$ for CIFAR-10, CIFAR-100, and Tiny-ImageNet; and $\epsilon = 2/255$ for ImageNet following (Wong et al., 2020; Andriushchenko & Flammarion, 2020). Throughout the experiments, we utilize an SGD optimizer

Table 2. SA, RA-PGD, and RA-AA of different robust training methods in the setup (CIFAR-10, PARN-18 training with $\epsilon = 8/255$) and (CIFAR-10, PARN-18 training with $\epsilon = 16/255$), respectively. All the results are averaged over 10 independent trials with different random seeds.

Method	SA (%)	RA-PGD (%)		RA-AA (%)	
		$\epsilon = 8$	$\epsilon = 16$	$\epsilon = 8$	$\epsilon = 16$
CIFAR-10, PARN-18 trained with $\epsilon = 8/255$					
FAST-AT	**82.39**±0.44	45.49 ±0.41	9.56 ±0.26	41.87 ±0.15	7.91 ±0.06
FAST-AT-GA	79.71±0.44	47.27 ±0.42	11.57±0.32	43.24 ±0.27	9.48 ±0.15
PGD-2-AT	81.97±0.41	44.62 ±0.39	9.39 ±0.32	41.73 ±0.20	7.54±0.25
FAST-BAT	79.97 ±0.12	**48.83** ±0.17	**14.00** ±0.21	**45.19** ±0.12	**11.51** ±0.20
CIFAR-10, PARN-18 trained with $\epsilon = 16/255$					
FAST-AT	44.15±7.27	37.17 ±0.74	21.83 ±1.32	31.66 ±0.27	12.49 ±0.33
FAST-AT-GA	58.29 ±1.32	43.86 ±0.67	26.01 ±0.16	38.69 ±0.56	17.97 ±0.33
PGD-2-AT	68.04 ±0.30	48.79 ±0.31	24.30 ±0.46	41.59 ±0.22	15.40 ±0.29
FAST-BAT	**68.16** ±0.25	**49.05** ±0.12	**27.69** ±0.16	**43.64** ±0.26	**18.79** ±0.24

with a momentum of 0.9 and weight decay of 5×10^{-4}. For CIFAR-10, CIFAR-100 and Tiny-ImageNet, we train each model for 20 epochs in total, where we use the cyclic scheduler to adjust the learning rate. The learning rate linearly ascends from 0 to 0.2 within the first 10 epochs and then reduces to 0 within the last 10 epochs. Our batch size is set to 128 for all settings. In the implementation of FAST-BAT, we follow the dataset-agnostic hyperparameter scheme for λ, such that $\lambda = 255/5000$ for $\epsilon = 8/255$ and $\lambda = 255/2500$ for $\epsilon = 16/255$ for CIFAR-10, CIFAR-100 and Tiny-ImageNet. For ImageNet, we strictly follow the setup given by (Wong et al., 2020) and we choose the train-time attack budget as $\epsilon = 2/255$. For each method, we use the early stopping method to pick the model with the best robust accuracy, following (Rice et al., 2020). All the experiments are conducted on a single GeForce RTX 3090 GPU.All the baselines are trained with the recommended configurations in their official GitHub repos. We refer readers to Appendix C for more details on the training setup.

Evaluation details. For adversarial evaluation, we report robust test accuracy (**RA**) of a learned model against PGD attacks (Madry et al., 2018) (**RA-PGD**). Unless otherwise specified, we set the test-time perturbation strength (ϵ) the same as the train-time value, and take 50-step PGD with 10 restarts all the datasets. Since AutoAttack (Croce & Hein, 2020) is known as the strongest robust benchmark evaluation metric (given as an ensemble attack), we also measure robust accuracy against AutoAttack, termed **RA-AA**. Further, we measure the standard accuracy (**SA**) against natural examples. Results are averaged over 10 independent trials. We would like to highlight that all the methods share the same batch size, epoch number, and training hardware. Thus, the time consumption per epoch reported in Table 1 and Table 5 will serve as a fair indicator of the algorithm complexity of different methods.

Table 3. Performance of different robust training methods under different model types. All the models are both trained and evaluated with the same perturbation strength ϵ.

Model	Method	SA(%) ($\epsilon = 8/255$)	RA-PGD(%) ($\epsilon = 8/255$)	SA(%) ($\epsilon = 16/255$)	RA-PGD(%) ($\epsilon = 16/255$)
PARN-50	FAST-AT	73.15±6.10	41.03±2.99	43.86±4.31	22.08±0.27
	FAST-AT-GA	77.40±0.81	46.16±0.98	42.28±6.69	22.87±1.25
	PGD-2-AT	**83.53**±0.17	46.17±0.59	68.88±0.39	22.37±0.41
	FAST-BAT	78.91±0.68	**49.18**±0.35	**69.01**±0.19	**24.55**±0.06
WRN-16-8	FAST-AT	84.39±0.46	45.80±0.57	49.39±2.17	21.99±0.41
	FAST-AT-GA	81.51±0.38	48.29±0.20	45.95±13.65	23.10±3.90
	PGD-2-AT	**85.52**±0.14	45.47±0.14	**72.11**±0.33	23.61±0.16
	FAST-BAT	81.66±0.54	**49.93**±0.36	68.12±0.47	**25.63**±0.44

5.2. Results

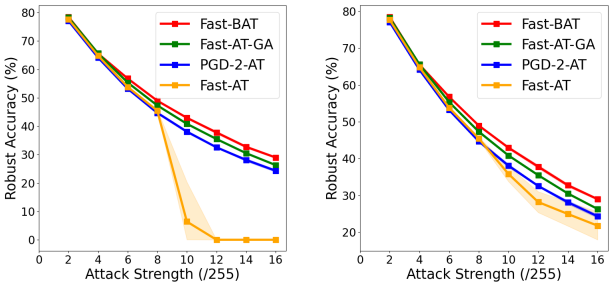
Overall performance of FAST-BAT. In Table 1, Table 2, and Table 6, we compare the overall performance of our proposed FAST-BAT with baselines.

① We find that FAST-BAT consistently outperforms the other baselines across the datasets and attack types. In Table 1, FAST-BAT improves the RA-PGD performance consistently by over 1.5% and RA-AA by around 1% across all the datasets on both attack strengths. For stronger attacks with a larger perturbation budget, the advantage of FAST-BAT is even clearer, *e.g.* a gain of over 2% in Tiny-ImageNet with $\epsilon = 16/255$. On ImageNet, FAST-BAT outperforms FAST-AT by 1.23% with $\epsilon = 2/255$.

② FAST-BAT leads to a much better SA-RA trade-off compared with the baselines. For example, in Table 1, the improvement in RA is not at cost of a huge drop in SA. Instead, when models are trained with $\epsilon = 16/255$, FAST-BAT even enjoys a significant boost over FAST-AT-GA in SA by 9.9%, 6.7%, and 4.6% for CIFAR-10, CIFAR-100, and Tiny-ImageNet respectively.

Performance under different model architectures. Besides PARN-18 reported above, Table 3 presents results on both deeper (PARN-50) and wider (WRN-18-6) models. As we can see, FAST-BAT consistently yields RA improvement over other methods. We also note that PGD-2-AT could be a competitive baseline in terms of SA. In contrast to FAST-AT and FAST-AT-GA, FAST-BAT is the only approach that yields an evident RA improvement over PGD-2-AT.

Mitigation of robust catastrophic overfitting. As shown in (Andriushchenko & Flammarion, 2020), FAST-AT suffers robust catastrophic overfitting when the train-time and test-time attack strength ϵ grows. Following (Andriushchenko & Flammarion, 2020), Figure 1 presents two RA-PGD trajectories, *i.e.*, training without early stopping and training with early stopping, versus the train/test-time ϵ. As we can see, FAST-AT encounters a sharp RA drop when $\epsilon > 8$ when early stopping is not used, consistent with the FAST-AT-GA



(a) Without early stopping (b) With early stopping

Figure 1. RA-PGD of different robust training methods for CIFAR-10 with the same training and evaluation attack strengths.

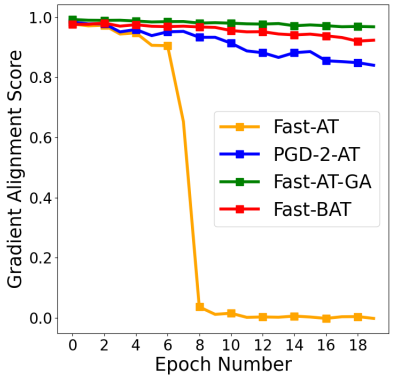


Figure 2. The training curve of GA score over different training methods on CIFAR-10.

paper (Andriushchenko & Flammarion, 2020). With early stopping, the overfitting of RA can be alleviated to some extent for FAST-AT, but its performance still remains the worst. Moreover, different from (Andriushchenko & Flammarion, 2020), we find that PGD-2-AT yields resilient performance against catastrophic overfitting. Our implementation gives a more positive baseline for PGD-2-AT, since the implementation in (Andriushchenko & Flammarion, 2020) did not use random initialization to generate train-time attacks. Furthermore, Figure 1 shows that FAST-BAT mitigates the issue of catastrophic overfitting and yields improved RA over other methods. We highlight that such an achievement is *free* of any robustness stability regularization, like gradient alignment used in FAST-AT-GA.

Gradient alignment for 'free'. As shown by (Andriushchenko & Flammarion, 2020), *catastrophic overfitting* occurs with the local non-linearity of deep networks, which can be measured by the gradient alignment (GA) score: $\mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D},\boldsymbol{\eta}\sim\mathcal{U}([-\epsilon,\epsilon]^d)}[\cos(\nabla_{\mathbf{x}}\ell_{\text{tr}}(\mathbf{x},y;\boldsymbol{\theta}),\nabla_{\mathbf{x}}\ell_{\text{tr}}(\mathbf{x}+\boldsymbol{\eta},y;\boldsymbol{\theta}))]$, where $\mathcal{U}$ denotes the randomly uniform distribution. GA is a key performance indicator to measure the appearance of robustness catastrophic overfitting, as catastrophic overfitting is always accompanied by a sharp GA drop in the training trajectory of the robustly trained model. In our paper, we calculate the GA for each method on the test set at the end of each epoch throughout the training process.

Table 4. Robustness against adaptive attacks (RA-PGD) and transfer attacks (RA-Transfer Attack). Naturally trained PARN-18, PARN-50, and WRN-16-8 serve as source victim models for attack generation with PGD-20 ($\epsilon = 8/255$) and PARN-18 robustified by different methods as target model for transfer attack evaluation.

Method	RA-PGD(%)	RA-Transfer Attack(%)		
		PARN-18	PARN-50	WRN-16-8
FAST-AT	45.44±0.06	76.35±0.12	76.94±0.14	77.23±0.21
PGD-2-AT	44.71±0.04	77.56±0.14	78.64±0.12	78.84±0.17
FAST-AT-GA	47.31±0.05	77.34±0.13	78.34±0.13	78.53±0.12
FAST-BAT(Ours)	**48.67**±0.05	**78.03**±0.15	**79.93**±0.12	**79.21**±0.15

Table 5. Performance of Hessian-free and Hessian-aware FAST-BAT on CIFAR-10. We train and evaluate with the same attack budgets $\epsilon = 8/255$ and $\epsilon = 16/255$ to show the influence brought by Hessian matrix.

Method	**SA(%)** ($\epsilon = 8/255$)	**RA-PGD(%)** ($\epsilon = 8/255$)	**SA(%)** ($\epsilon = 16/255$)	**RA-PGD(%)** ($\epsilon = 16/255$)	**Time (s/epoch)**
Hessian-free	79.97 ±0.12	48.83 ±0.17	68.16 ±0.25	27.69 ±0.16	61.4
Hessian-aware	79.62 ±0.17	49.13 ±0.14	67.82 ±0.23	27.82 ±0.19	82.6

Figure 1 suggested that FAST-BAT can mitigate overfitting without explicit GA regularization, and Figure 2 presents the GA score versus the training epoch number. As can be seen, FAST-BAT automatically enforces GA and remains very close to FAST-AT-GA, which maximizes GA with an explicit regularization. Therefore, a high GA score may just be a necessary but not a sufficient condition for avoiding catastrophic overfitting. As additional evidence, Fig. 3 shows that similar to FAST-AT-GA, FAST-BAT has a flatter loss landscape than FAST-AT as well. Therefore, a direct penalization on the input gradient norm may not achieve the state-of-the-art model robustness.

Sanity check for obfuscated gradients As pointed out by (Athalye et al., 2018a), model robustness could be overestimated due to obfuscated gradients. The model with obfuscated gradients could have 'obfuscated' stronger resilience to white-box (adaptive) attacks than black-box (transfer) attacks. To justify the validity of FAST-BAT, Table 4 summarizes the comparison between our proposal and the other baselines when facing adaptive and transfer attacks. Firstly, for all the methods, RA increases if the transfer attack is applied, implying that the transfer attack is weaker than the adaptive attack. This is desired in the absence of obfuscated gradients. Moreover, FAST-BAT consistently outperforms the other baselines against both adaptive and transfer attacks. The absence of obfuscated gradients can also be justified by the flatness of the adversarial loss landscape in Figure. 3.

The validity of the Hessian-free assumption on ReLU-based neural networks. In Corollary 1, the Hessian-free assumption, *i.e.*$\nabla_{\boldsymbol{\delta}\boldsymbol{\delta}}\ell_{\text{atk}} = 0$, was made to simplify the computation of implicit gradient. To justify this assumption

we conduct experiments to compare the Hessian-free FAST-BAT with the Hessian-aware version. In Hessian-aware FAST-BAT, the implicit gradient is calculated based on (23). In Table 5, the results do not indicate much difference when Hessian is used. However, the extra computations required to evaluate the Hessian heavily slows down FAST-BAT as around 30% more time is needed. Therefore, the Hessian-free assumption is reasonable and also necessary in terms of the efficiency of the algorithm. We also justify this assumption on some non-ReLU neural networks and for more results please refer to Appendix D.

6. Conclusion

In this paper, we introduce a novel bi-level optimization-based fast adversarial training framework, termed FAST-BAT. The rationale behind designing FAST-BAT lies in two aspects. First, from the perspective of implicit gradients, we show that the existing FAST-AT framework is equivalent to the lower-level linearized BLO along the sign direction of the input gradient. Second, we show that FAST-BAT is able to achieve improved stability of performance, mitigated catastrophic overfitting, and enhanced accuracy-robustness trade-off. To the best of our knowledge, we for the first time establish the theory and the algorithmic foundation of BLO for adversarial training. Extensive experiments are provided to demonstrate the superiority of our method to state-of-the-art accelerated AT baselines.

Acknowledgement

Y. Zhang and S. Liu are supported by the Cisco Research grant CG# 70614511. M. Hong and P. Khanduri are supported in part by NSF grants CIF-1910385 and NSF CMMI-1727757.

References

Alfarra, M., Bibi, A., Hammoud, H., Gaafar, M., and Ghanem, B. On the decision boundaries of deep neural networks: A tropical geometry perspective. *arXiv preprint arXiv:2002.08838*, 2020.

Andriushchenko, M. and Flammarion, N. Understanding and improving fast adversarial training. *NeurIPS*, 2020.

Athalye, A., Carlini, N., and Wagner, D. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018a.

Athalye, A., Engstrom, L., Ilyas, A., and Kwok, K. Synthesizing robust adversarial examples. In *International Conference on Machine Learning*, pp. 284–293, 2018b.

Carlini, N. and Wagner, D. Towards evaluating the robustness of neural networks. In *IEEE Symposium on S&P*, 2017.

Carmon, Y., Raghunathan, A., Schmidt, L., Liang, P., and Duchi, J. C. Unlabeled data improves adversarial robustness. *arXiv preprint arXiv:1905.13736*, 2019.

Chen, J., Cheng, Y., Gan, Z., Gu, Q., and Liu, J. Efficient robust training via backward smoothing. *arXiv preprint arXiv:2010.01278*, 2020a.

Chen, T., Liu, S., Chang, S., Cheng, Y., Amini, L., and Wang, Z. Adversarial robustness: From self-supervised pretraining to fine-tuning. In *CVPR*, 2020b.

Chen, Z., Liu, D., Wu, X., and Xu, X. Research on distributed renewable energy transaction decision-making based on multi-agent bilevel cooperative reinforcement learning. 2019.

Croce, F. and Hein, M. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *International Conference on Machine Learning*, pp. 2206–2216. PMLR, 2020.

Dempe, S. *Foundations of bilevel programming*. Springer Science & Business Media, 2002.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. IEEE, 2009.

Deng, Y., Zheng, X., Zhang, T., Chen, C., Lou, G., and Kim, M. An analysis of adversarial attacks and defenses on autonomous driving models. In *2020 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pp. 1–10. IEEE, 2020.

Engstrom, L., Ilyas, A., and Athalye, A. Evaluating and understanding the robustness of adversarial logit pairing. *arXiv preprint arXiv:1807.10272*, 2018.

Ghadimi, S. and Wang, M. Approximation methods for bilevel programming. *arXiv preprint:1802.02246*, 2018.

Goodfellow, I., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *ICLR*, 2015.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Gould, S., Fernando, B., Cherian, A., Anderson, P., Cruz, R. S., and Guo, E. On differentiating parameterized argmin and argmax problems with application to bi-level optimization. *arXiv preprint arXiv:1607.05447*, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016a.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016b.

Hong, M., Wai, H.-T., Wang, Z., and Yang, Z. A two-timescale framework for bilevel optimization: Complexity analysis and application to actor-critic. *arXiv preprint arXiv:2007.05170*, 2020.

Huang, W. R., Geiping, J., Fowl, L., Taylor, G., and Goldstein, T. Metapoison: Practical general-purpose clean-label data poisoning. *arXiv preprint arXiv:2004.00225*, 2020.

Ji, K., Yang, J., and Liang, Y. Bilevel optimization: Nonasymptotic analysis and faster algorithms. *arXiv preprint arXiv:2010.07962*, 2020.

Khanduri, P., Zeng, S., Hong, M., Wai, H.-T., Wang, Z., and Yang, Z. A momentum-assisted single-timescale stochastic approximation algorithm for bilevel optimization. *arXiv preprint arXiv:2102.07367*, 2021.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

Kumar, K. N., Vishnu, C., Mitra, R., and Mohan, C. K. Black-box adversarial attacks in autonomous vehicle technology. In *2020 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*, pp. 1–7. IEEE, 2020.

Li, B., Wang, S., Jana, S., and Carin, L. Towards understanding fast adversarial training. *arXiv preprint arXiv:2006.03089*, 2020.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

Moosavi-Dezfooli, S.-M., Fawzi, A., Uesato, J., and Frossard, P. Robustness via curvature regularization, and vice versa. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9078–9086, 2019.

Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., and Swami, A. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pp. 372–387. IEEE, 2016.

Qian, Q., Zhu, S., Tang, J., Jin, R., Sun, B., and Li, H. Robust optimization over multiple domains. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4739–4746, 2019.

Rajeswaran, A., Finn, C., Kakade, S., and Levine, S. Meta-learning with implicit gradients. *arXiv preprint arXiv:1909.04630*, 2019.

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Razaviyayn, M., Huang, T., Lu, S., Nouiehed, M., Sanjabi, M., and Hong, M. Nonconvex min-max optimization: Applications, challenges, and recent theoretical advances. *IEEE Signal Processing Magazine*, 37(5):55–66, 2020.

Rice, L., Wong, E., and Kolter, Z. Overfitting in adversarially robust deep learning. In *International Conference on Machine Learning*, pp. 8093–8104. PMLR, 2020.

Salman, H., Yang, G., Li, J., Zhang, P., Zhang, H., Razenshteyn, I., and Bubeck, S. Provably robust deep learning via adversarially trained smoothed classifiers. *arXiv preprint arXiv:1906.04584*, 2019.

Shafahi, A., Najibi, M., Ghiasi, M. A., Xu, Z., Dickerson, J., Studer, C., Davis, L. S., Taylor, G., and Goldstein, T. Adversarial training for free! In *Advances in Neural Information Processing Systems*, pp. 3353–3364, 2019.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. *International Conference on Learning Representations*, 2014.

Thys, S., Van Ranst, W., and Goedemé, T. Fooling automated surveillance cameras: adversarial patches to attack person detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 0–0, 2019.

Vicente, L., Savard, G., and Júdice, J. Descent approaches for quadratic bilevel programming. *Journal of Optimization Theory and Applications*, 81(2):379–399, 1994.

White, D. J. and Anandalingam, G. A penalty function approach for solving bi-level linear programs. *Journal of Global Optimization*, 3(4):397–419, 1993.

Wong, E., Rice, L., and Kolter, J. Z. Fast is better than free: Revisiting adversarial training. In *International Conference on Learning Representations*, 2020.

Xie, C., Wu, Y., Maaten, L. v. d., Yuille, A. L., and He, K. Feature denoising for improving adversarial robustness. In *Proceedings of the IEEE/CVF Conference on*

Computer Vision and Pattern Recognition, pp. 501–509, 2019.

Xu, K., Liu, S., Zhao, P., Chen, P.-Y., Zhang, H., Fan, Q., Erdogmus, D., Wang, Y., and Lin, X. Structured adversarial attack: Towards general implementation and better interpretability. In *ICLR*, 2019.

Xu, K., Zhang, G., Liu, S., Fan, Q., Sun, M., Chen, H., Chen, P.-Y., Wang, Y., and Lin, X. Adversarial T-Shirt! evading person detectors in a physical world. In *ECCV*, 2020.

Zagoruyko, S. and Komodakis, N. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Zhang, D., Zhang, T., Lu, Y., Zhu, Z., and Dong, B. You only propagate once: Accelerating adversarial training via maximal principle. *arXiv preprint arXiv:1905.00877*, 2019a.

Zhang, H., Yu, Y., Jiao, J., Xing, E. P., Ghaoui, L. E., and Jordan, M. I. Theoretically principled trade-off between robustness and accuracy. *ICML*, 2019b.

Zheng, H., Zhang, Z., Gu, J., Lee, H., and Prakash, A. Efficient adversarial training with transferable adversarial examples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1181–1190, 2020.

A. Proof of Proposition 1 and Corollary 1

Proof: Upon defining $g(\boldsymbol{\theta}, \boldsymbol{\delta}) = (\boldsymbol{\delta} - \mathbf{z})^\top \nabla_{\boldsymbol{\delta}=\mathbf{z}} \ell_{\text{atk}}(\boldsymbol{\theta}, \boldsymbol{\delta}) + (\lambda/2)\|\boldsymbol{\delta} - \mathbf{z}\|_2^2$, we repeat (7) as

$$\begin{aligned} \underset{\boldsymbol{\theta}}{\text{minimize}} \quad & \mathbb{E}_{(\mathbf{x},y)\in\mathcal{D}}[\ell_{\text{tr}}(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))] \\ \text{subject to} \quad & \boldsymbol{\delta}^*(\boldsymbol{\theta}) = \underset{\mathbf{B}\boldsymbol{\delta}\le\mathbf{b}}{\arg\min}\, g(\boldsymbol{\theta}, \boldsymbol{\delta}), \end{aligned} \tag{14}$$

where we have used the expression of linear constraints in (9).

Our goal is to derive the IG $\frac{d\boldsymbol{\delta}^*(\boldsymbol{\theta})^\top}{d\boldsymbol{\theta}}$ shown in (4). To this end, we first build implicit functions by leveraging KKT conditions of the lower-level problem of (14). We say $\boldsymbol{\delta}^*(\boldsymbol{\theta})$ and $\boldsymbol{\lambda}^*(\boldsymbol{\theta})$ (Lagrangian multipliers) satisfy the KKT conditions:

$$\begin{aligned} \text{Stationarity:} \quad & \nabla_{\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})) + \mathbf{B}^\top \boldsymbol{\lambda}^*(\boldsymbol{\theta}) = \mathbf{0}, \\ \text{Complementary slackness :} \quad & \boldsymbol{\lambda}^*(\boldsymbol{\theta}) \cdot (\mathbf{B}\boldsymbol{\delta}^*(\boldsymbol{\theta}) - \mathbf{b}) = \mathbf{0} \\ \text{Dual feasibility:} \quad & \boldsymbol{\lambda}^*(\boldsymbol{\theta}) \ge \mathbf{0} \end{aligned} \tag{15}$$

where $\cdot$ denotes the elementwise product.

Active constraints & definition of $\mathbf{B}_0$: Let $\mathbf{B}_0$ denote the sub-matrix of $\mathbf{B}$ and $\mathbf{b}_0$ the sub-vector of $\mathbf{b}$, which consists of only the *active constraints* at $\boldsymbol{\delta}^*(\boldsymbol{\theta})$, *i.e.*, those satisfied with the equality $\mathbf{B}_0\boldsymbol{\delta}^*(\boldsymbol{\theta}) = \mathbf{b}_0$ (corresponding to *nonzero* dual variables). The determination of active constraints is done given $\boldsymbol{\theta}$ at each iteration.

With the aid of $(\mathbf{B}_0, \mathbf{b}_0)$, KKT (15) becomes

$$\nabla_{\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})) + \mathbf{B}_0^\top \boldsymbol{\lambda}^*(\boldsymbol{\theta}) = \mathbf{0}, \;\; \text{and} \;\; \mathbf{B}_0\boldsymbol{\delta}^*(\boldsymbol{\theta}) - \mathbf{b}_0 = \mathbf{0}, \tag{16}$$

where the nonzero $\boldsymbol{\lambda}^*(\boldsymbol{\theta})$ only corresponds to the active constraints. We take derivatives w.r.t. $\boldsymbol{\theta}$ of (16), and thus obtain the following

$$\begin{aligned} & \frac{d\nabla_{\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))^\top}{d\boldsymbol{\theta}} + \nabla_{\boldsymbol{\theta}} \boldsymbol{\lambda}^*(\boldsymbol{\theta})^\top \mathbf{B}_0 = \mathbf{0} \\ \Longrightarrow & \nabla_{\boldsymbol{\theta}\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})) + \underbrace{\frac{d\boldsymbol{\delta}^*(\boldsymbol{\theta})^\top}{d\boldsymbol{\theta}}}_{\text{IG}} \nabla_{\boldsymbol{\delta}\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})) + \nabla_{\boldsymbol{\theta}} \boldsymbol{\lambda}^*(\boldsymbol{\theta})^\top \mathbf{B}_0 = \mathbf{0}, \end{aligned} \tag{17}$$

$$\text{and} \quad \underbrace{\frac{d\boldsymbol{\delta}^*(\boldsymbol{\theta})^\top}{d\boldsymbol{\theta}}}_{\text{IG}} \mathbf{B}_0^\top = \mathbf{0}, \tag{18}$$

where $\nabla_{\boldsymbol{\theta}\boldsymbol{\delta}} \in \mathbb{R}^{|\boldsymbol{\theta}|\times|\boldsymbol{\delta}|}$ denotes second-order partial derivatives (recall that $|\boldsymbol{\theta}| = n$ and $|\boldsymbol{\delta}| = d$). According to (17), we have

$$\frac{d\boldsymbol{\delta}^*(\boldsymbol{\theta})^\top}{d\boldsymbol{\theta}} = -[\nabla_{\boldsymbol{\theta}\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})) + \nabla_{\boldsymbol{\theta}} \boldsymbol{\lambda}^*(\boldsymbol{\theta})^\top \mathbf{B}_0] \nabla_{\boldsymbol{\delta}\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))^{-1}. \tag{19}$$

Substituting the above into (18), we obtain

$$\nabla_{\boldsymbol{\theta}\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})) \nabla_{\boldsymbol{\delta}\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))^{-1} \mathbf{B}_0^\top + \nabla_{\boldsymbol{\theta}} \boldsymbol{\lambda}^*(\boldsymbol{\theta})^\top \mathbf{B}_0 \nabla_{\boldsymbol{\delta}\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))^{-1} \mathbf{B}_0^\top = \mathbf{0}, \tag{20}$$

which yields:

$$\nabla_{\boldsymbol{\theta}} \boldsymbol{\lambda}^*(\boldsymbol{\theta})^\top = -\nabla_{\boldsymbol{\theta}\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})) \nabla_{\boldsymbol{\delta}\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))^{-1} \mathbf{B}_0^\top [\mathbf{B}_0 \nabla_{\boldsymbol{\delta}\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))^{-1} \mathbf{B}_0^\top]^{-1}, \tag{21}$$

and thus,

$$\nabla_{\boldsymbol{\theta}} \boldsymbol{\lambda}^*(\boldsymbol{\theta})^\top \mathbf{B}_0 = -\nabla_{\boldsymbol{\theta}\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})) \nabla_{\boldsymbol{\delta}\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))^{-1} \mathbf{B}_0^\top [\mathbf{B}_0 \nabla_{\boldsymbol{\delta}\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))^{-1} \mathbf{B}_0^\top]^{-1} \mathbf{B}_0. \tag{22}$$

Substituting (22) into (19), we obtain the IG

$$\begin{aligned} & \frac{d\boldsymbol{\delta}^*(\boldsymbol{\theta})^\top}{d\boldsymbol{\theta}} = -\nabla_{\boldsymbol{\theta}\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})) \nabla_{\boldsymbol{\delta}\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))^{-1} - \nabla_{\boldsymbol{\theta}} \boldsymbol{\lambda}^*(\boldsymbol{\theta})^\top \mathbf{B}_0 \nabla_{\boldsymbol{\delta}\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))^{-1} \\ & = -\nabla_{\boldsymbol{\theta}\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})) \nabla_{\boldsymbol{\delta}\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))^{-1} \\ & + \nabla_{\boldsymbol{\theta}\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})) \nabla_{\boldsymbol{\delta}\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))^{-1} \mathbf{B}_0^\top [\mathbf{B}_0 \nabla_{\boldsymbol{\delta}\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))^{-1} \mathbf{B}_0^\top]^{-1} \mathbf{B}_0 \nabla_{\boldsymbol{\delta}\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))^{-1}. \end{aligned} \tag{23}$$

The above completes the proof of Proposition 1. □

To further compute (23), the Hessian matrix $\nabla_{\boldsymbol{\delta}\boldsymbol{\delta}}\ell_{\text{atk}}$ is needed. Recall from the definition of the lower-level objective that the Hessian matrix is given by

$$\nabla_{\boldsymbol{\delta}\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})) = \nabla_{\boldsymbol{\delta}\boldsymbol{\delta}}\ell_{\text{atk}} + \lambda \mathbf{I} = \mathbf{0} + \lambda \mathbf{I}. \tag{24}$$

Here we used the assumption that $\nabla_{\boldsymbol{\delta}\boldsymbol{\delta}}\ell_{\text{atk}} = \mathbf{0}$. The rationale behind that is neural networks commonly leads to a piece-wise linear decision boundary w.r.t. the inputs (Moosavi-Dezfooli et al., 2019; Alfarra et al., 2020), and thus, its second-order derivative (Hessian) $\nabla_{\boldsymbol{\delta}\boldsymbol{\delta}}\ell_{\text{atk}}$ is close to zero.

Based on the simplification (24), we have

$$\begin{aligned}\frac{d\boldsymbol{\delta}^*(\boldsymbol{\theta})^\top}{d\boldsymbol{\theta}} =& -(1/\lambda)\nabla_{\boldsymbol{\theta}\boldsymbol{\delta}} g(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))\underbrace{\left(\mathbf{I} - \mathbf{B}_0^\top[\mathbf{B}_0\mathbf{B}_0^\top]^{-1}\mathbf{B}_0\right)}_{:=\mathbf{H}_{\mathcal{C}}} \\ & -(1/\lambda)\nabla_{\boldsymbol{\theta}\boldsymbol{\delta}}\ell_{\text{atk}}(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))\mathbf{H}_{\mathcal{C}},\end{aligned} \tag{25}$$

where we have used the fact that $\nabla_{\boldsymbol{\theta}\boldsymbol{\delta}} g = \nabla_{\boldsymbol{\theta}\boldsymbol{\delta}}\ell_{\text{atk}}$.

What is $\mathbf{H}_{\mathcal{C}}$ in (25)? Since $\mathbf{B} = \begin{bmatrix}\mathbf{I} \\ -\mathbf{I}\end{bmatrix}$, we can obtain that $\mathbf{B}_0\mathbf{B}_0^\top = \mathbf{I}$ and $\mathbf{B}_0^\top\mathbf{B}_0$ is a sparse diagonal matrix with diagonal entries being 0 or 1. Thus, $\mathbf{H}_{\mathcal{C}}$ can be first simplified to

$$\mathbf{H}_{\mathcal{C}} = \mathbf{I} - \mathbf{B}_0^\top\mathbf{B}_0. \tag{26}$$

Clearly, $\mathbf{H}_{\mathcal{C}}$ is also a diagonal matrix with either 0 or 1 diagonal entries. The 1-valued diagonal entry of $\mathbf{H}_{\mathcal{C}}$ corresponds to the *inactive constraints* in $\mathbf{B}\boldsymbol{\delta}^*(\boldsymbol{\theta}) < \mathbf{b}$, *i.e.*, those satisfied with *strict inequalities* in $\{\|\boldsymbol{\delta}\|_\infty \le \epsilon, \mathbf{0} \le \boldsymbol{\delta} \le \mathbf{1}\}$. This can be expressed as

$$\mathbf{H}_{\mathcal{C}} = \left[1_{p_1 \le \delta_1^* \le q_1}\mathbf{e}_1, \ldots, 1_{p_1 \le \delta_d^* \le q_d}\mathbf{e}_d\right] \tag{27}$$

where $1_{p_i \le \delta_i^* \le q_i} \in \{0, 1\}$ denotes the indicator function over the constraint $\{p_i \le \delta_i^* \le q_i\}$ and returns 1 if the constraint is satisfied, δ_i^* denotes the ith entry of $\boldsymbol{\delta}^*(\boldsymbol{\theta})$, $p_i = \max\{-\epsilon, -x_i\}$ and $q_i = \min\{\epsilon, 1 - x_i\}$, and $\mathbf{e}_i \in \mathbb{R}^d$ denotes the basis vector with the ith entry being 1 and others being 0s.

Based on the definition of g, (25) and (27), we can eventually achieve the desired IG formula (12). The proof of Corollary 1 is now complete. □

B. Discussion on case $\ell_{\text{atk}} = -\ell_{\text{tr}}$

We provide an in-depth explanation on the fact that even if we set $\ell_{\text{atk}} = -\ell_{\text{tr}}$, the *optimization routine* given by (4) to solve problem (3) does not reduce to the ordinary IG-absent gradient descent to solve problem (1) because of the presence of lower-level constraints.

- In the **absence** of the constraint $\boldsymbol{\delta} \in \mathcal{C}$, if we set $\ell_{\text{atk}} = -\ell_{\text{tr}}$, then solving problem (3) via IG-involved descent method (4) will reduce to the ordinary IG-absent method that solves problem (1).

  This is a known BLO result (*e.g.* (Ghadimi & Wang, 2018)) and can be readily proven using the stationary condition. To be specific, based on the stationary condition of unconstrained lower-level optimization, we have $\nabla_{\boldsymbol{\delta}} \ell_{\text{atk}}(\boldsymbol{\theta}, \boldsymbol{\delta}^*) = 0$. Since $\ell_{\text{atk}} = -\ell_{\text{tr}}$, we have $\nabla_{\boldsymbol{\delta}} \ell_{\text{tr}}(\boldsymbol{\theta}, \boldsymbol{\delta}^*) = 0$. As a result, the second term in (4) becomes $\mathbf{0}$ and solving problem (3) becomes identical to solving the min-max problem (1).

- In the **presence** of the constraint $\boldsymbol{\delta} \in \mathcal{C}$, the stationary condition cannot be applied since the stationary point may not be a feasible point in the constraint. In other words, $\nabla_{\boldsymbol{\delta}} \ell_{\text{atk}}(\boldsymbol{\theta}, \boldsymbol{\delta}^*) = 0$ does not hold in the case of $\ell_{\text{atk}} = -\ell_{\text{tr}}$. As a matter of fact, one has to resort to KKT conditions instead of the stationary condition for a constrained lower-level problem. Similar to our proof in Proposition 1, the implicit gradient (and thus the second term of (4)) cannot be omitted in general. This makes the optimization routine to solve problem (3) different from solving problem (1).

C. Detailed Experiment settings

C.1. Training Set-up

For CIFAR-10, we summarize the training setup for each method. 1) FAST-AT: We use FGSM with an attack step size of 1.25ϵ to generate perturbations; 2) PGD-2-AT: 2-step PGD attacks[1] with an attack step size of 0.5ϵ is implemented; 3) FAST-AT-GA: The gradient alignment regularization parameter is set to the recommended value for each ϵ; 4) FAST-BAT: We select λ from $255/5000$ to $255/2000$ for different ϵ. At the same time, we adjust α_2 accordingly, so that the coefficient of the second term in (13), namely α_2/λ always equals to $0.1\alpha_1$.

For ImageNet, we set ϵ to $2/255$, and we strictly follow the training setting adopted by (Wong et al., 2020). In FAST-BAT, we fix λ at $255/3000$ and adopt the same α_2 selection strategy as CIFAR-10.

Parameter for FAST-AT-GA Regarding FAST-AT-GA with different model types, we adopt the same regularization parameter recommended in its official repo[2] intended for PreActResNet-18 (namely 0.2 for $\epsilon = 8/255$ and 2.0 for $\epsilon = 16/255$).

C.2. The choice of initialization point z

To specify $\mathbf{z}$ in (7), we investigate two classes of linearization schemes. The first class is random constant linearization, including: "uniformly random linearization", i.e., $\mathbf{z} = \boldsymbol{\delta}_0$ similar to FAST-AT, and "random corner linearization" under the ϵ-radius ℓ_∞-ball, i.e., $\mathbf{z} \in \{-\epsilon, \epsilon\}^d$. The second class is 1-step perturbation warm-up-based linearization, including the other two specifications: "1-step sign-based PGD" , and "1-step PGD w/o sign" . We consider this linearization schemes as their computation complexities are less than or close to the complexity of one-step attack generation. As a result, FAST-BAT takes comparable computation cost to the baselines FAST-AT, PGD-2-AT and FAST-AT-GA. Empirically, we find that FAST-BAT using "1-step PGD w/o sign" leads to the best defensive performance; see justification in Table 9. We follow this experiment setup in the sequel.

[1] We use random initialization to generate perturbations for PGD, while in the paper of FAST-AT-GA (Andriushchenko & Flammarion, 2020), 2-step PGD is initialized at zero point, which we believe will underestimate the effect of PGD-2-AT

[2] FAST-AT-GA: https://github.com/tml-epfl/understanding-fast-adv-training/blob/master/sh

D. Additional Experimental Results

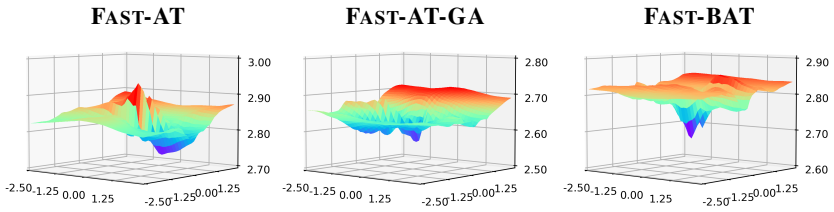


Figure 3. Visualization of adversarial loss landscapes of FAST-AT, FAST-AT-GA and FAST-BAT trained using the ResNet-18 model on the CIFAR-10 dataset. The losses at are calculated w.r.t. the same image example ID #001456, and the landscape is obtained by tracking the loss changes w.r.t. input variations following (Engstrom et al., 2018). That is, the loss landscape is generated by $z = \text{loss}(I + x \cdot \mathbf{r_1} + y \cdot \mathbf{r_2})$, where I denotes an image, and the x-axis and the y-axis correspond to linear coefficients associated with the sign-based attack direction $\mathbf{r_1} = \text{sign}(\nabla_I \text{loss}(I))$ $\hat{\mathbf{x}}$ and a random direction $\mathbf{r_2} \sim \text{Rademacher}(0.5)$, respectively.

Results on ImageNet We train DNN models ResNet (RN)-50 (He et al., 2016a) for ImageNet and we choose the training perturbation strength $\epsilon = 2/255$ strictly following (Wong et al., 2020; Andriushchenko & Flammarion, 2020). We remark that when evaluating on ImageNet, we only compare ours with FAST-AT since as shown in Table 6 of (Andriushchenko & Flammarion, 2020), the other baseline methods did not show any improvement over Fast-AT at the attack budget $\epsilon = 2/255$. RA-PGD stands for the robustness against PGD-50-10 (50-step PGD attack with 10 restarts) with $\epsilon = 2/255$. Table 6 shows the performance comparison between FAST-AT and FAST-BAT. We can see FAST-BAT outperforms FAST-AT by 1.23% when facing attacks with $\epsilon = 2/255$. Since robust training on ImageNet usually takes small ϵ (like 2/255), the benefit of robust catastrophic overfitting alleviation becomes less evident.

Table 6. SA and RA on ImageNet.

Method	SA (%)	RA-PGD (%)
FAST-AT	60.90	43.43
FAST-BAT	60.18	44.64

Sensitivity to regularization parameter λ In Table 7, we show the sensitivity of FAST-BAT to the regularization parameter λ. All the parameters remain the same as the default setting, except that for different λ. We always adjust α_2 so that $\alpha_2/\lambda = 0.1\alpha_1$ holds. Note $1/\lambda$ also serves as the attack step in (8). As λ decreases, the improvement in robust accuracy is evidently strengthened, and there is an obvious trade-off between robust accuracy (SA) and standard accuracy (RA). At a certain level of λ, namely when $\lambda \leq 255/3500$, RA starts to converge and stop surging.

Table 7. Performance of FAST-BAT with different parameter λ. We train and evaluate with the same attack budget $\epsilon = 16/255$ on CIFAR-10 to show the influence brought by λ.

CIFAR-10, PreActResNet-18, $\epsilon = 16/255$					
$1/\lambda$ (/255)	500	1000	1500	2500	3500
SA (%)	**83.20**	75.06	69.31	68.16	64.37
RA-PGD (%)	19.02	21.42	23.34	**27.69**	25.32

Sensitivity to different α_2 choices We consider the case of robust training with the large ϵ choice $(16/255)$. As we can see from Table 8, if α_2 is set too small ($\alpha_2 = 0.008\alpha_1$), then both SA and RA will drop significantly. Here α_1 is set as the cyclic learning rate and thus not a constant parameter. However, in the α_2 interval $[0.0125\alpha_1, 0.025\alpha_1]$, we observed a tradeoff between standard accuracy (SA) and robust accuracy (RA): That is, the improvement in RA corresponds to a loss in SA. In our experiments, we choose α_2 when the tradeoff yields the best RA without suffering a significant drop of SA (which still outperforms the baseline approaches).

Table 8. Performance of FAST-BATwith different α_2 choices on CIFAR-10. Models are trained and evaluated with the same attack budget ($\epsilon = 16/255$). Here α_1 is set as the cyclic learning rate and is not a constant value. α_2 is always set proportionate to α_1 for simplicity.

α_2 **(CIFAR-10, PreActResNet18,** $\epsilon = 16/255$**)**	$0.025\alpha_1$	$0.0167\alpha_1$	$0.0125\alpha_1$	$0.008\alpha_1$
SA (%)	**75.06**	69.31	68.16	57.92
RA-PGD (%)	21.42	23.34	**27.69**	20.53

Sensitivity of linearization schemes Fast-BAT needs a good linearization point $\mathbf{z}$ in (7). In experiments, we adopt the perturbation generated by 1-step PGD without sign as our default linearization scheme. In Table 9, we show the performance of the other possible linearization options. We find that 1-step PGD without sign achieves the best robust accuracy among all the choices. This is not spurring since this linearization point choice is consistent with the first-Taylor expansion that we used along the direction of the input gradient without the sign operation involved. By contrast, FAST-BAT linearized with

uniformly random noise suffers from catastrophic overfitting and reaches a rather low standard accuracy (SA). FAST-BAT with other linearizations also yields a worse SA-RA trade-off than our proposal.

The validity of the Hessian-free assumption on non-ReLU based neural networks. The Hessian-free assumption is based on the fact that the commonly used ReLU activation function is piece-wise linear *w.r.t.* input. We further conduct experiments to verify the feasibility of such an assumption on models with non-ReLU activation functions. We choose two commonly used activation functions, Swish (Ramachandran et al., 2017) and Softplus, as alternatives for the non-smooth ReLU function. We compare the results both calculating Hessian as well as the Hessian-free version to see if the Hessian-free assumption still holds for the non-ReLU neural network. The results are shown in Table 11. As we can see, the use of Hessian does not affect performance much. A similar phenomenon can be observed across different ϵ and different model activation functions (ReLU, Softplus, and Swish). However, the introduction of Hessian leads to an increase in time consumption by more than 30%. Therefore, we can draw the conclusion that the Hessian-free assumption is reasonable across different activation function choices.

Table 9. Performance of FAST-BAT with different linearization schemes. Besides 1-step PGD without sign (**PGD w/o Sign**), we further generate linearization point with the following methods: uniformly random noise $[-\epsilon, \epsilon]^d$ (**Uniformly Random**); uniformly random corner $\{-\epsilon, \epsilon\}^d$ (**Random Corner**); and perturbation from 1-step PGD attack with 0.5ϵ as attack step (**PGD**).

CIFAR-10, PreActResNet-18, $\epsilon = 16/255$				
Linearization Method	PGD w/o Sign	Uniformly Random	Random Corner	PGD
SA (%)	68.16	43.42	62.19	**75.30**
RA-PGD (%)	**27.69**	21.25	16.5	19.42

Ablation studies. In Appendix D, we present additional empirical studies including 1) the sensitivity analysis of the linearization hyperparameter λ, 2) the choice of the linearization point, and 3) the sensitivity analysis of α_2.

Comparisons with more baselines. In Tab. 10, we compare FAST-BAT with more baselines, PGD-7-AT, BACKSMOOTH, ATTA, FREE-AT, and YOPO. The standard PDG-7 AT typically yields the best RA, but causes the highest computation cost (see column 'Time' for time till best model). While in the **fast** robust training paradigm, FAST-BAT stands top for different values of ϵ. FAST-AT-GA is indeed a strong baseline to mitigate robust catastrophic overfitting as train-time ϵ increases (e.g., $16/255$). Ours also outperforms FREE-AT and YOPO in robust catastrophic overfitting alleviation as ϵ grows. FAST-AT-GA paper also identified the incapability of FREE-AT.

Table 10. Performance comparison of FAST-BAT vs. baselines on (CIFAR10, PreActResNet-18). Each baseline follows its original setting. FREE-AT (m=8) and YOPO-5-3 are adopted. ATTA refers to ATTA-1-TRADES. For fair comparison, all the methods are trained with 20 epochs and cyclic learning rate. Minor performance degradation compared to the results reported in the original papers is due to different training setting, that we train all the methods with only 90% of training data, choose the best model on the validation set (10% training data), and evaluate on the test set. All the results are at the same level as the ones reported in the original papers (for those adopting similar training settings). Evaluation settings are consistent with Table 1 in the main paper.

Method	$\epsilon = 8/255$ (Train/test-time attack)			$\epsilon = 16/255$ (Train/test-time attack)			Time
	SA	RA-PGD	RA-AA	SA	RA-PGD	RA-AA	(min)
AT (PGD-7)	81.43±0.13	50.63±0.16	47.05±0.18	61.55±0.17	31.11±0.61	22.99±0.31	121.5
FAST-AT	82.39±0.14	45.49±0.21	41.87±0.15	44.15±7.27	21.83±1.32	12.49±0.33	**7.7**
BACKSMOOTH	79.31±0.17	48.06±0.07	44.55±0.07	64.88±1.75	24.18±1.37	15.47±0.92	20.2
FREE-AT	79.59±0.14	42.84±0.86	39.39±0.20	35.00±12.37	6.07±1.95	0.91±0.42	24.5
FAST-AT-GA	79.71±0.24	47.27±0.22	43.24±0.27	58.29±1.32	26.01±0.16	17.97±0.33	25.1
ATTA	79.43±0.09	48.78±0.62	44.61±0.35	67.37±1.89	0.36±0.12	0.00±0.00	38.8
YOPO-5-3	**83.17**±0.11	44.50±0.28	40.61±0.43	44.04±3.61	23.08±2.30	10.61±0.86	43.8
FAST-BAT	79.97±0.12	**48.83**±0.17	**45.19**±0.12	**68.16**±0.25	**27.69**±0.16	**18.79**±0.24	20.5

Table 11. Performance of FAST-AT and FAST-BAT with different activation functions on CIFAR-10. ReLU, Swish and Softplus are taken into consideration. For FAST-BAT, we compare the Hessian-free and Hessian-aware version to verify the influence of Hessian matrix. The results are averaged over 3 independent trials.

Setting	SA (%) ($\epsilon = 8/255$)	RA-PGD (%) ($\epsilon = 8/255$)	SA (%) ($\epsilon = 16/255$)	RA-PGD (%) ($\epsilon = 16/255$)	Time (s/epoch)
Fast-AT-ReLU	82.39±0.14	45.49±0.21	44.15±7.27	21.83±1.32	23.1
Fast-BAT-ReLU Hessian-free	79.97 ±0.12	48.83 ±0.17	68.16 ±0.25	27.69 ±0.16	61.4
Fast-BAT-ReLU Hessian-aware	79.62 ±0.17	49.13 ±0.14	67.82 ±0.23	27.82 ±0.19	82.6
Fast-AT-Softplus	81.29 ±0.16	47.26 ±0.24	45.39 ±3.27	22.40 ±0.75	23.3
Fast-BAT-Softplus Hessian-free	79.48 ±0.18	49.67 ±0.21	68.57 ±0.27	25.59 ±0.15	61.7
Fast-BAT-Softplus Hessian-aware	79.59 ±0.21	49.74 ±0.12	68.63 ±0.23	25.54 ±0.19	82.8
Fast-AT-Swish	75.61 ±0.15	44.43 ±0.18	52.03 ±4.29	23.08 ±2.23	23.1
Fast-BAT-Swish Hessian-free	73.89 ±0.14	45.90 ±0.23	62.59 ±0.29	23.81 ±0.17	61.7
Fast-BAT-Swish Hessian-aware	73.93 ±0.16	45.97 ±0.19	62.49 ±0.27	23.99 ±0.17	82.6

E. Convergence Analysis

Let us consider we have data samples $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$, where N denotes the number of the training data. Then the goal of FAST-BAT algorithm is to solve:

$$\begin{aligned} \min_{\boldsymbol{\theta}} L_{\text{tr}}(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})) &= \frac{1}{N}\sum_{i=1}^{N} \ell_{\text{tr}}(\boldsymbol{\theta}; \mathbf{x}_i + \boldsymbol{\delta}_i^*(\boldsymbol{\theta}), y_i) \\ \boldsymbol{\delta}^*(\boldsymbol{\theta}) \in \arg\min_{\boldsymbol{\delta}_i \in \mathcal{C}} L_{\text{atk}}(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})) &= \frac{1}{N}\sum_{i=1}^{N} \ell_{\text{atk}}(\boldsymbol{\theta}; \mathbf{x}_i + \boldsymbol{\delta}_i^*(\boldsymbol{\theta}), y_i) \end{aligned} \tag{28}$$

Let us denote the batch size as b. Then the problem can be reformulated as :

$$\begin{aligned} \min_{\boldsymbol{\theta}} L_{\text{tr}}^{b_t}(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})) &= \frac{1}{b_t}\sum_{i=1}^{b_i} \ell_{\text{tr}}(\boldsymbol{\theta}; \mathbf{x}_i + \boldsymbol{\delta}_i^*(\boldsymbol{\theta}), y_i) \\ \boldsymbol{\delta}^*(\boldsymbol{\theta}) \in \arg\min_{\boldsymbol{\delta}_i \in \mathcal{C}} L_{\text{atk}}(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})) &= \underbrace{\frac{1}{b_t}\sum_{i=1}^{b_t} \ell_{\text{atk}}(\boldsymbol{\theta}; \mathbf{x}_i + \boldsymbol{\delta}_i^*(\boldsymbol{\theta}), y_i)}_{L_{\text{atk}}^{b_t}(\boldsymbol{\theta}, \boldsymbol{\delta}_i(\boldsymbol{\theta}))} \end{aligned} \tag{29}$$

The way to computes the gradient (4) is equal to the case where $b = N$.

$$\frac{dL_{\text{tr}}^N(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))}{d\boldsymbol{\theta}} = \nabla_{\boldsymbol{\theta}} L_{\text{tr}}^N(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})) + \frac{d\boldsymbol{\delta}^*(\boldsymbol{\theta})^\top}{d\boldsymbol{\theta}} \nabla_{\boldsymbol{\delta}} L_{\text{tr}}^N(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})) \tag{30}$$

If we approximate the gradient $\nabla_{\boldsymbol{\theta}} L_{\text{tr}(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))}$ using a batch size of b, then we have the following assumptions:

- Bias assumption:

$$\mathbb{E}[\nabla_{\boldsymbol{\theta}} L_{\text{tr}}^b(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))] = \nabla_{\boldsymbol{\theta}} L_{\text{tr}}(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})) + \beta(b), \tag{31}$$

  where $\beta(b)$ is the bias and the expectation is taken *w.r.t.* batches. Note that $\beta(b) = 0$ for $b = N$.

- Variance assumption:

$$\mathbb{E}\|\nabla_{\boldsymbol{\theta}} L_{\text{tr}}^{b}(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})) - \nabla_{\boldsymbol{\theta}} L_{\text{tr}}(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})) - \beta(b)\|^2 = \sigma^2[1 + \|\nabla_{\boldsymbol{\theta}} L_{\text{tr}}(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))\|^2], \tag{32}$$

  Note that the variance equals 0 for $b = N$.

Now the FAST-BAT algorithm with batch size b can be stated as follows:

- Initialization: $\boldsymbol{\theta}_0 \in \mathbb{R}^n$, step size $\{\alpha_t\}_{t=0}^{T-1}$, batch size $\{\alpha_b\}_{t=0}^{T-1}$.
- for $t = 1$ to T:
  - Choose batsh size b_t at each iteration $t \in [T]$.
  - $L_{\text{tr}}(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})) \approx L_{\text{tr}}^{b_t}(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))$, $L_{\text{atk}}(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta})) \approx L_{\text{atk}}^{b_t}(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))$
  - Update:

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha_{t-1} \nabla_{\boldsymbol{\theta}} L_{\text{tr}}^{b_t}(\boldsymbol{\theta}_{t-1}, \boldsymbol{\delta}_{t-1}^*(\boldsymbol{\theta}_{t-1})) \tag{33}$$

    where

$$\nabla_{\boldsymbol{\theta}} L_{\text{tr}}^{b_t}(\boldsymbol{\theta}_{t-1}, \boldsymbol{\delta}_{t-1}^*(\boldsymbol{\theta}_{t-1})) = \nabla_{\boldsymbol{\theta}} L_{\text{tr}}^{b_t}(\boldsymbol{\theta}_{t-1}, \boldsymbol{\delta}^*(\boldsymbol{\theta}_{t-1})) + \nabla_{\boldsymbol{\theta}} \boldsymbol{\delta}^*(\boldsymbol{\theta}_{t-1})^\top \nabla_{\boldsymbol{\delta}} L_{\text{tr}}^{b_t}(\boldsymbol{\theta}_{t-1}, \boldsymbol{\delta}^*(\boldsymbol{\theta}_{t-1})). \tag{34}$$

    Note $\nabla_{\boldsymbol{\theta}} \boldsymbol{\delta}^*(\boldsymbol{\theta}_{t-1})$ is computed using (23) by replacing:
    - $L_{\text{atk}}(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))$ with $L_{\text{atk}}^{b_t}(\boldsymbol{\theta}, \boldsymbol{\delta}^*(\boldsymbol{\theta}))$
    - B_0 with B_0 computed with (29).

Assuming smoothness of $L_{\text{tr}(\boldsymbol{\theta})}$ *w.r.t.* $\boldsymbol{\theta}$, we get:

$$\begin{aligned} L_{\text{tr}}(\boldsymbol{\theta}_{t+1}) &\leq L_{\text{tr}}(\boldsymbol{\theta}_t) + \langle \nabla_{\boldsymbol{\theta}} (\boldsymbol{\theta}_t), \boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t \rangle + \frac{L}{2} \|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\|_2^2 \\ &= L_{\text{tr}}(\boldsymbol{\theta}_t) - \alpha_t \left\langle \nabla_{\boldsymbol{\theta}} L_{\text{tr}}(\boldsymbol{\theta}_t), \nabla_{\boldsymbol{\theta}} L_{\text{tr}}^{b_t}(\boldsymbol{\theta}_t) \right\rangle + \frac{\alpha_t^2 L}{2} \|\nabla_{\boldsymbol{\theta}} L_{\text{tr}}^{b_t}(\boldsymbol{\theta}_t)\|_2^2 \end{aligned} \tag{35}$$

Taking expectation *w.r.t.* random samples, we get:

$$\begin{aligned} \mathbb{E}[L_{\text{tr}}(\boldsymbol{\theta}_{t+1})] \leq \mathbb{E}[L_{\text{tr}}(\boldsymbol{\theta}_t)] - \alpha_t \mathbb{E}[\|\nabla_{\boldsymbol{\theta}} L_{\text{tr}}(\boldsymbol{\theta}_t)\|_2^2] - \alpha_t \mathbb{E}[\langle \nabla_{\boldsymbol{\theta}} (\boldsymbol{\theta}_t), \beta(b_t) \rangle] + \alpha_t^2 \sigma^2 L \mathbb{E}[1 + \|\nabla_{\boldsymbol{\theta}} L_{\text{tr}}(\boldsymbol{\theta}_t)\|_2^2] \\ + 2\alpha_t^2 L \mathbb{E}[\|\nabla_{\boldsymbol{\theta}} L_{\text{tr}}(\boldsymbol{\theta}_t)\|_2^2] + 2\alpha_t^2 L \mathbb{E}[\|\beta(b_t)\|_2^2] \end{aligned}$$

$$\leq \mathbb{E}\left[L_{\text{tr}}(\boldsymbol{\theta}_t) - \left[\frac{\alpha_t}{2} - \alpha_t^2 \sigma^2 L \right] \|\nabla_{\boldsymbol{\theta}} L_{\text{tr}}(\boldsymbol{\theta}_t)\|_2^2 + \left[\frac{\alpha_t}{2} + 2\alpha_t^2 L \right] \|\beta(b_t)\|_2^2 + \alpha_t^2 L \sigma^2 \right]$$

Using $\alpha_t \leq \frac{1}{4L(2+\sigma^2)}$, we get:

$$\mathbb{E}[L_{\text{tr}}(\boldsymbol{\theta}_{t+1})] \leq \mathbb{E}\left[L_{\text{tr}}(\boldsymbol{\theta}_t) - \frac{\alpha_t}{4} \|\nabla_{\boldsymbol{\theta}} L_{\text{tr}}(\boldsymbol{\theta}_t)\|_2^2 + \alpha_t \|\beta(b_t)\|^2 + \alpha_t^2 L \sigma^2 \right]. \tag{36}$$

Finally, after rearranging the terms, we get:

$$\frac{\alpha_t}{4} \mathbb{E}[\|\nabla_{\boldsymbol{\theta}} L_{\text{tr}}(\boldsymbol{\theta}_t)\|_2^2] \leq \mathbb{E}[(L_{\text{tr}}(\boldsymbol{\theta}_t) - L_{\text{tr}}(\boldsymbol{\theta}_{t+1})) + \alpha_t \|\beta(b_t)\|^2 + \alpha_t^2 L \sigma^2] \tag{37}$$

Summing over $t = 1$ to T and multiplying by $1/T$, we get:

$$\frac{1}{T}\sum_{t=1}^{T}\frac{\alpha_t}{4}\mathbb{E}[\|\nabla_{\boldsymbol{\theta}} L_{\mathrm{tr}}(\boldsymbol{\theta}_t)\|_2^2] \leq \frac{L_{\mathrm{tr}}(\boldsymbol{\theta}_0) - L_{\mathrm{tr}}(\boldsymbol{\theta}^*)}{T} + \frac{1}{T}\sum_{t=1}^{T}\alpha_t\|\beta(b_t)\|^2 + \frac{1}{T}\sum_{t=1}^{T}\alpha_t^2\sigma_t^2. \tag{38}$$

Taking $\alpha_t = \alpha$ and $b_t = b$, we get:

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\|\nabla_{\boldsymbol{\theta}} L_{\mathrm{tr}}(\boldsymbol{\theta}_t)\|_2^2] \leq \frac{4[L_{\mathrm{tr}}(\boldsymbol{\theta}_0) - L_{\mathrm{tr}}(\boldsymbol{\theta}^*)]}{\alpha T} + 4\|\beta(b)\|^2 + 4\alpha\sigma^2. \tag{39}$$

Recall if $b = N$, we have $\sigma^2 = 0$ and $\beta(b) = 0$, we can further get:

$$\frac{1}{T}\sum_{t=1}^{T}\|\nabla_{\boldsymbol{\theta}} L_{\mathrm{tr}}(\boldsymbol{\theta}_t)\|_2^2 \leq \frac{4[L_{\mathrm{tr}}(\boldsymbol{\theta}_0) - L_{\mathrm{tr}}(\boldsymbol{\theta}^*)]}{\alpha T}. \tag{40}$$

Case I: We can choose $\alpha = \frac{1}{8L}$ and get:

$$\frac{1}{T}\|\nabla_{\boldsymbol{\theta}} L_{\mathrm{tr}}(\boldsymbol{\theta}_t)\|_2^2 = \mathcal{O}(\frac{1}{T}). \tag{41}$$

Case II: We can choose $\alpha = \sqrt{\frac{1}{T}}$:

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\|\nabla_{\boldsymbol{\theta}} L_{\mathrm{tr}}(\boldsymbol{\theta}_t\|_2^2)] \leq \frac{4[L_{\mathrm{tr}}(\boldsymbol{\theta}_0) - L_{\mathrm{tr}}(\boldsymbol{\theta}^*)]}{\sqrt{T}} + \underbrace{\frac{4\sigma^2}{\sqrt{T}}}_{\mathcal{O}(\frac{1}{\sqrt{T}})} + \underbrace{4\|\beta(b)\|^2}_{\mathcal{O}(1)}$$

Now if $\|\beta(b)\|^2 = \mathcal{O}(\frac{1}{\sqrt{T}})$, then

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\|\nabla_{\boldsymbol{\theta}} L_{\mathrm{tr}}(\boldsymbol{\theta}_t\|_2^2] \leq \mathcal{O}(\frac{1}{\sqrt{T}}), \tag{42}$$

otherwise we get in the worst case:

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\|\nabla_{\boldsymbol{\theta}} L_{\mathrm{tr}}(\boldsymbol{\theta}_t\|_2^2] \leq \mathcal{O}(\frac{1}{\sqrt{T}}) + \mathcal{O}(1). \tag{43}$$