Int. J. Appl. Math. Comput. Sci., 2024, Vol. 34, No. 3, 425–438
DOI: 10.61822/amcs-2024-0030

DEVELOPING HESSIAN–FREE SECOND–ORDER ADVERSARIAL EXAMPLES FOR ADVERSARIAL TRAINING

YAGUAN QIAN [a], LIANGJIAN ZHANG [a], YUQI WANG [a], BOYUAN JI [a], TENGTENG YAO [a], BIN WANG [b,*]

[a]College of Science
Zhejiang University of Science and Technology
Hangzhou 310023, China
e-mail: {qianyaguan,212209701008,222109252007}@zust.edu.cn, 1445318469@qq.com,yaotengteng718@163.com

[b]Zhejiang Key Laboratory of Artificial Intelligence of Things (AIoT) Network and Data Security
Hangzhou 310052, China
e-mail: wbin2006@gmail.com

Recent studies show that deep neural networks (DNNs) are extremely vulnerable to elaborately designed adversarial examples. Adversarial training, which uses adversarial examples as training data, has been proven to be one of the most effective methods of defense against adversarial attacks. However, most existing adversarial training methods use adversarial examples relying on first-order gradients, which perform poorly against second-order adversarial attacks and make it difficult to further improve the robustness of the model. In contrast to first-order gradients, second-order gradients provide a more accurate approximation of the loss landscape relative to natural examples. Therefore, our work focuses on constructing second-order adversarial examples and utilizing them for training DNNs. However, second-order optimization involves computing the Hessian inverse, which typically consumes considerable time. To address this issue, we propose an approximation method that transforms the problem into optimization within the Krylov subspace. Compared with the Euclidean space, the Krylov subspace method typically does not require storing the entire matrix. It only needs to store vectors and intermediate results, avoiding explicitly calculating the complete Hessian matrix. We approximate the adversarial direction by a linear combination of Hessian-vector products in the Krylov subspace to reduce the computation cost. Because of the non-symmetrical Hessian matrix, we use the generalized minimum residual to search for an approximate polynomial solution of the matrix. Our method efficiently reduces computational complexity and accelerates the training process. Extensive experiments conducted on the MNIST, CIFAR-10, and ImageNet-100 datasets demonstrate that our adversarial learning using second-order adversarial samples outperforms other first-order methods, leading to improved model robustness against various attacks.

Keywords: adversarial examples, adversarial machine learning, Krylov subspace, deep neural networks.

1. Introduction

Deep neural networks (DNNs) have been successfully applied in many image-related tasks, such as image classification (Ding *et al*., 2022), target detection (Long *et al*., 2023), and super-resolution (Yin *et al*., 2023). However, DNNs' vulnerability to *adversarial examples* has drawn significant attention in the computer vision community (Szegedy *et al*., 2013; Goodfellow *et al*., 2015). In general, an adversarial example is an image added by an imperceptible perturbation, which can successfully fool a classifier. The existence of adversarial examples will lead to disastrous consequences, especially in safety-sensitive scenarios, such as industrial process control (Pozdnyakov *et al*., 2024), face recognition (Yang *et al*., 2023), and automatic drive systems (Lu *et al*., 2024). Numerous countermeasures are proposed to improve DNNs' robustness against adversarial examples (Huang *et al*., 2023; Athalye *et al*., 2018; Tejankar

*Corresponding author

et al., 2023), among which adversarial training is considered one of the most effective methods (Athalye *et al.*, 2018). Adversarial training as essentially data augmentation with various adversarial examples (Li *et al.*, 2022; Wang and Wang, 2022; Liu *et al.*, 2023) to train a DNN. Madry *et al.* (2017) first formulated adversarial training as a saddle-point problem, where the inner maximization corresponds to the generation of adversarial examples while the outer minimization is supposed to achieve robust network parameters. However, the effectiveness of adversarial training depends on the intensity of adversarial examples corresponding to the inner-maximization problem. In this paper, we hope to generate stronger adversarial examples to improve the effectiveness of adversarial training, i.e, optimization for inner maximization.

In practice, achieving internal maximization of adversarial training necessitates multiple iterations to search for more potent perturbations. To date, first-order projected gradient descent (PGD) adversarial attacks, introduced by Madry *et al.* (2017), remain a prominent approach for generating adversarial examples. However, it hinders the development of significantly improved attack strategies. Models trained using first-order adversarial examples face difficulties in enhancing robustness against more sophisticated attacks as first-order gradients often lack a comprehensive and global perspective of the optimization landscape. To address the quest for more potent adversarial examples, attention has shifted towards second-order gradients. Unlike first-order gradients, methods leveraging second-order gradients offer additional insights, including the trends of first-order gradients, facilitating the generation of stronger adversarial examples (Tsiligkaridis and Roberts, 2020). Hence, the model trained with stronger adversarial examples can achieve more adversarial robustness.

Though several second-order optimization methods were proposed for adversarial robustness (Jin *et al.*, 2022; Bertolace *et al.*, 2024), they did not consider crafting adversarial examples for adversarial training and consume a significant amount of computing resources and time. We propose a novel approach to craft adversarial examples with second-order optimization, named SOAEs (second-order adversarial examples), for adversarial training. Our SOAE method constructs a new perturbation direction based on the second-order gradient information of the loss function in the Krylov subspace (Yeom and Reddy, 2001; Shimonishi *et al.*, 2002; Jea and Young, 1980). Specifically, we use the inverse of the Hessian to determine the adversarial direction, i.e., the direction of maximizing a loss function. However, directly determining the inverse of the Hessian of a loss function with respect to an image (in general, with high resolution) scales quadratically for storage and cubically for the number of operations. It is computationally expensive to handle the inverse Hessian matrix directly. Then we find, compared to the Euclidean space, that the Krylov subspace methods typically do not require storing the entire matrix; they only need to store vectors and intermediate results, avoiding explicitly calculating the complete Hessian matrix. Thus, to address the issue of computational complexity in second-order optimization, we approximate adversarial direction by a liner combination of Hessian-vector products in the Krylov subspace to reduce the computation cost. Because of the nonsymmetrical Hessian matrix, we use the generalized minimum residual to search for an approximate polynomial solution of the matrix polynomial approximation. In addition, optimization conducted in the Krylov subspace can achieve a more accurate approximation. We test the SOAE's effectiveness on different models trained by various adversarial training techniques. Meanwhile, we use SOAEs to train several models and evaluate their robustness against various adversarial attack methods. Extensive experimental results demonstrate that adversarial training with SOAEs yields state-of-the-art performance. Finally, we provide a deep insight into the effectiveness of our method from the theoretical perspective of computational complexity and attack-strength bound.

Our contribution is summarized as follows.

- We propose a second-order gradient-based method to generate more powerful adversarial examples. Adversarial training through these adversarial examples enhances the models with higher robustness than those training with first-order adversarial examples.
- We address the problem of the inverse Hessian matrix occurring in second-order gradients by transforming it into optimization in the Krylov subspace, which remarkably reduces the computational complexity.
- We theoretically prove the superiority of our method over the project gradient descent (PGD). Extensive experiments conducted on MNIST and CIFAR-10 also show that our second-order method outperforms other approaches, including the state-of-the-art AutoAttack.

The paper is organized as follows. In Section 2, we provide an update on the progress of the related work. In Section 3, we provide pre-requisite knowledge before the introduction of our methodology. Then, our method is presented in Section 4 and an experimental proof is given in Section 5. Finally, we summarize our work in Section 6.

List of acronyms.

DNN: deep neural network
PGD: projected gradient descent
SOAE: second-order adversarial examples
SOAR: second-order adversarial regularizer
APGD: auto projected gradient descent
FAB: fast adaptive boundary
FGSM: fast gradient signed method
ERM: empirical risk minimization
GMRES: generalized minimum residual
FC: fully connected
SOAT: second-order adversarial training
STD: standard training
PSNR: peak signal noise ratio
SSIM: structural similarity
MSE: mean square error

2. Related work

To react to the threat of adversarial examples against DNNs, researchers have developed numerous defense mechanisms. Generally, adversarial defense can be categorized into three directions: model optimization, data manipulation, and auxiliary networks. Model optimization involves adjusting the parameters within the model, including defensive distillation (Badjie *et al.*, 2023), gradient regularization (Li and Spratling, 2023), and others. Data manipulation entails fine-tuning the input data, such as adversarial training (Madry *et al.*, 2017), image compression (Song *et al.*, 2024), and similar techniques. Auxiliary networks involve augmenting the trained model with an additional network, such as adversarial example detectors (Guo *et al.*, 2023). Among them, adversarial training is considered one of the most effective methods. However, in most existing adversarial training methods, that of generating adversarial samples relies on a first-order gradient. Although the first-order gradient optimization problem has high computational efficiency, it is weak in the face of second-order adversarial attacks (Zhang *et al.*, 2023; Wu *et al.*, 2024).

Recently, there have been several second-order optimization methods applied to adversarial robustness. Li *et al.* (2018) proposed an attack method based on an approximated second-order derivative of the loss function and showed it can effectively reduce the accuracy of adversarially trained models. However, there was still a noticeable gap between theoretical analysis and empirical results. Tsiligkaridis and Roberts (2020) revealed that adversarial attack based on second-order approximation of the loss is more effective in models with regular landscapes and decision boundaries. Ma *et al.* (2020) further studied the *upper bound* and proposed the SOAR, a second-order adversarial regularizer based on the Taylor approximation of the inner maximization in the robust optimization objective. They showed that SOAR training significantly improves adversarial robustness under L_∞ and L_2 attacks. However, they also found its vulnerability to AutoAttack. A possible explanation is that the SOAR overfits a specific type of attack, i.e., loss function dependent. In order to avoid the overfitting of the model, the generalization ability of the model is improved. Jin *et al.* (2023) performed Taylor expansion of the loss function on random weights, optimized the zeroth, first-order and second-order expansions at the same time, and proposed adversarial training under the randomized model.

Recently, Croce and Hein (2020b) proposed a reliable and stable attack method called AutoAttack, which is an automatic parameter-free method integrating four attack methods: i.e., three white-box attacks: an Auto-PGD (APGD) (Croce and Hein, 2020b) with cross-entropy loss, targeted APGD with difference-of-logits-ratio loss, targeted fast adaptive boundary (FAB) attack (Croce and Hein, 2020a), and a black-box attack named SquareAttack (Andriushchenko *et al.*, 2020). However, the methods mentioned above are all based on first-order gradient information. The fast gradient signed method (FGSM), lacks accuracy due to its one-shot operation. Though the PGD fixes this issue through k iterations, it undoubtedly prolongs the convergence time. The same situation occurs in AutoAttack since it uses the combination of several variants of the PGD and other types of attacks. Besides, those first-order methods have some natural defects in approximating the loss landscape around the neighborhood of the input images.

The limited application of the second-order methods in adversarial robustness is mainly due to high computation cost and large storage consumption of the Hessian-related problem in the optimization process. In order to avoid the high computational cost of Hessian matrices, Ge *et al.* (2023) adopt a first-order procedure to approximate the curvature of the second-order Hessian matrix, which makes computing more efficient by interpolating two Jacobian matrices. Zhao *et al.* (2024) used differential approximation to approximate the Hessian-vector product low-curvature integrated defense model. In addition to the method of approximating the Hessian matrix, a new second-order optimizer named Shampoo was proposed by Anil *et al.* (2020). With the efficiently utilized heterogeneous hardware architecture consisting of multi-core CPUs and multi-accelerator units, it outperformed the state-of-the-art first-order gradient descent methods in the field of image classification, large-scale machine translation, etc. The success of Shampoo undoubtedly reveals the potential for the development of the second-order method in deep learning tasks. This encourages us to explore more possibilities of its application in adversarial robustness.

3. Background

A clean example-label pair $(\boldsymbol{x}, y) \sim \mathcal{D}$ is extracted from an underlying data distribution $\mathcal{D}$ in the standard classification task. The classifier $f_\theta(\cdot)$ with parameters $\boldsymbol{\theta}$ turns $\boldsymbol{x}$ into logits, namely, the unnormalized probability values. The logits are normalized after a softmax layer and become a probability score. The softmax layer can be represented as a function $p_k(\boldsymbol{x}) = e^{f_{\theta,k}(\boldsymbol{x})} / \sum_l e^{f_{\theta,l}(\boldsymbol{x})}$. Thus, the predicted class label is obtained by $\hat{y}_k(\boldsymbol{x}) = \arg\max_k f_{\theta,k}(\boldsymbol{x})$. A standard training procedure is the empirical risk minimization (ERM) (Zhang, 2016). For the loss function $L(\boldsymbol{x}, y, \boldsymbol{\theta})$, the goal of standard training is

$$\min_\theta \mathbb{E}_{(x,y)\sim\mathcal{D}}[L(\boldsymbol{x}, y, \boldsymbol{\theta})]. \tag{1}$$

Training the classifier through the ERM principle guarantees a high accuracy on test sets but leads to an unavoidable vulnerability against adversarial attacks (Madry *et al.*, 2017). To measure the immunity of a classifier $f_\theta(\cdot)$ against perturbations, adversarial robustness is defined with respect to a metric. In practice, the most used metric is an L_p-norm ($p = 1$, 2, or ∞), combined with an L_p-ball $B_p(\epsilon) = \{\boldsymbol{\delta} \mid \|\boldsymbol{\delta}\|_p \le \epsilon\}$. An adversarial example $\boldsymbol{x}^{\text{adv}}$ is obtained by adding perturbation $\boldsymbol{\delta}$ to the original example $\boldsymbol{x}$, where the upper bound of $\boldsymbol{\delta}$ is ϵ. Practically, there are many adversarial example generation algorithms proposed to find $\boldsymbol{x}^{\text{adv}} = \boldsymbol{x} + \boldsymbol{\delta}$ such that $\boldsymbol{\delta}$ is very small but the model misclassifies $\boldsymbol{x}^{\text{adv}}$ to class label $\hat{y} \neq y$. Here y is the ground-truth label of $\boldsymbol{x}$. A classifier is evaluated as robust to adversarial perturbation size ϵ if the label of the given input example $\boldsymbol{x}$ does not change for all perturbations of size up to ϵ, i.e., $f_\theta(\boldsymbol{x}) = f_\theta(\boldsymbol{x}^{\text{adv}}) = f_\theta(\boldsymbol{x} + \boldsymbol{\delta})$, where $\boldsymbol{\delta} \in B_p(\epsilon)$. In this scenario, ϵ is often called a perturbation budget.

There are many algorithms to generate adversarial examples. Goodfellow *et al.* (2015) first proposed an FGSM that multiplies the sign of the loss gradient with respect to the inputs to obtain the perturbation, i.e., $\boldsymbol{x}^{\text{adv}} = \boldsymbol{x} + \alpha \text{sign} \nabla L(\boldsymbol{x}, \boldsymbol{\theta})$. Madry *et al.* (2017) further developed this single-iteration FGSM into a k-iteration method PGD with a projection step to restrict the size of perturbation, i.e., $\boldsymbol{x}^{(t+1)} = \Pi(\boldsymbol{x}^{(t)} + \alpha \text{sign} \nabla L(\boldsymbol{x}, \boldsymbol{\theta}))$.

4. Methodology

4.1. Second-order perturbation. For a classifier $f_\theta(\cdot)$ and an input $\boldsymbol{x}$ with label y, our purpose is to find a small perturbation $\boldsymbol{\delta}$ that leads to a misclassification of $f_\theta(\cdot)$, i.e.,

$$f_\theta(\boldsymbol{x} + \boldsymbol{\delta}) = \hat{y} \tag{2a}$$

subject to

$$y \neq \hat{y} \wedge \boldsymbol{\delta} \in B_p(\epsilon). \tag{2b}$$

Specifically, our goal is to maximize the loss function $L(\boldsymbol{x} + \boldsymbol{\delta}, y)$. Almost all the methods utilize the first-order gradient direction, i.e., $\partial L / \partial \boldsymbol{x}$, to increase the loss function. However, we think the second-order gradient includes more global information for obtaining more powerful adversarial examples. For a clean example $\boldsymbol{x}$, the Taylor expansion of the loss function of its perturbated example $\boldsymbol{x}^{\text{adv}} = \boldsymbol{x} + \boldsymbol{\delta}$ can be written as

$$\begin{aligned} L(\boldsymbol{x}^{\text{adv}}) &\approx Q(\boldsymbol{\delta}) \\ &= L(\boldsymbol{x}) + \nabla L(\boldsymbol{x})^T \boldsymbol{\delta} + \frac{1}{2} \boldsymbol{\delta}^T \nabla^2 L(\boldsymbol{x}) \boldsymbol{\delta}, \end{aligned} \tag{3}$$

where the higher-order terms are omitted. This loss function is a quadratic function $Q(\boldsymbol{\delta})$ of perturbation $\boldsymbol{\delta}$. Since we want the loss to increase as much as possible in each iteration, the final optimal perturbation can be presented by

$$\boldsymbol{\delta}^* = \arg \max_{\boldsymbol{\delta} \in B_p(\epsilon)} Q(\boldsymbol{\delta}). \tag{4}$$

We set the derivative of $Q(\boldsymbol{\delta})$ with respect to $\boldsymbol{\delta}$ to zero and then obtain a feasible optimization direction at the t-th iteration:

$$\boldsymbol{\delta}^{(t)} = \left[\nabla^2 L(\boldsymbol{x} + \boldsymbol{\delta}^{(t-1)})\right]^{-1} \nabla L(\boldsymbol{x} + \boldsymbol{\delta}^{(t-1)}). \tag{5}$$

For notational simplicity let $\mathbf{H} = \nabla^2 L(\cdot)$ and $\mathbf{g} = \nabla L(\cdot)$; then $\boldsymbol{\delta}^{(t)} = \mathbf{H}^{-1}\mathbf{g}$. To limit the perturbation size, a step-size factor α is adopted. If the final perturbation is out of the limitation of the given L_p-ball, referring to the method in PGD, the final form of the generated adversarial example yields

$$\boldsymbol{x}^{\text{adv}} = \Pi(\boldsymbol{x} + \alpha \mathbf{H}^{-1}\mathbf{g}), \tag{6}$$

where $\Pi(\cdot)$ is a projection operator.

4.2. Approximating $\mathbf{H}^{-1}\mathbf{g}$. Calculating the Hessian matrix scales quadratically for storage and cubically for the number of operations, so that obtaining its inverse in (6) is nontrivial. This motivates why we introduce below approximating $\mathbf{H}^{-1}\mathbf{g}$ in (6) by a linear combination of Hessian-vector products in the Krylov subspace, which yields a computationally feasible and efficient gradient approximation:

$$\begin{aligned} \boldsymbol{\delta}^{(t)} = \mathbf{H}^{-1}\mathbf{g} &\approx \sum_{i=0}^{m-1} \beta_i \mathbf{H}^i \mathbf{g} \\ &= \beta_0 \mathbf{g} + \beta_1 \mathbf{H}\mathbf{g} + \beta_2 \mathbf{H}^2 \mathbf{g} + \cdots \\ &\quad + \beta_{m-1} \mathbf{H}^{m-1} \mathbf{g}, \end{aligned} \tag{7}$$

where β_i are coefficients and m is a hyperparameter ($m \ll \dim \mathbf{H}$). Here, an m-dimensional Krylov subspace $\mathcal{K}(\mathbf{H}, \mathbf{g}) \triangleq \operatorname{span}\{\mathbf{g}, \mathbf{Hg}, \dots, \mathbf{H}^{m-1}\mathbf{g}\}$ is employed.

The Krylov subspace method has an important feature, i.e., at each iteration it yields only one matrix and a low number of vector operations. By pre-multiplying the matrix $\mathbf{H}$, we can get larger eigenvectors in matrix $\mathbf{H}$. The eigenvectors are used to find the possible subspace building feasible optimization and approximating $\mathbf{H}^{-1}\mathbf{g}$ during iterations.

We apply the generalized minimum residual (GMRES) (Jea and Young, 1980) to deal with Eqn. (7), which is one of the most effective orthogonalization methods in the Krylov subspace. In an m-dimensional Krylov subspace $\mathcal{K}(\mathbf{H}, \mathbf{g})$, there exists a $\tilde{\boldsymbol{\delta}} \in \boldsymbol{\delta}^{(0)} + \mathcal{K}$ for a minimal residual, where $\boldsymbol{\delta}^{(0)}$ is a randomly initialized perturbation. Hence, (7) is transformed into the following optimization to approximate $\boldsymbol{\delta}^{(t)} = \mathbf{H}^{-1}\mathbf{g}$:

$$\boldsymbol{\delta}^{(t)} = \arg \min_{\tilde{\boldsymbol{\delta}} \in \boldsymbol{\delta}^{(0)} + \mathcal{K}} \left\|\mathbf{g} - \mathbf{H}\tilde{\boldsymbol{\delta}}\right\|_2 \tag{8a}$$

subject to

$$(\mathbf{g} - \mathbf{H}\tilde{\boldsymbol{\delta}}) \perp \mathbf{H}\mathcal{K}, \tag{8b}$$

where $\left\|\mathbf{g} - \mathbf{H}\tilde{\boldsymbol{\delta}}\right\|_2$ is the residual in affine space $\boldsymbol{\delta}^{(0)} + \mathcal{K}$, and $\mathbf{H}\mathcal{K}$ is a constraint space. However, directly solving this optimization is nontrivial. For any $\tilde{\boldsymbol{\delta}} \in \boldsymbol{\delta}^{(0)} + \mathcal{K}_m$, there exists $\boldsymbol{\gamma} \in \mathbb{R}^m$ such that $\tilde{\boldsymbol{\delta}} = \boldsymbol{\delta}^{(0)} + \mathbf{V}_m\boldsymbol{\gamma}$, where $\mathbf{V}_m$ is an m-dimensional unitary matrix (Shimonishi *et al.*, 2002). Then we further transform (8a) into an equivalent optimization problem. Let

$$\begin{aligned} \mathbf{g} - \mathbf{H}\tilde{\boldsymbol{\delta}} &= \mathbf{g} - \mathbf{H}(\boldsymbol{\delta}^{(0)} + \mathbf{V}_m\boldsymbol{\gamma}) \\ &= \mathbf{r}_0 - \mathbf{H}\mathbf{V}_m\boldsymbol{\gamma} \\ &= \beta\mathbf{v}_1 - \mathbf{V}_{m+1}\mathbf{D}_{m+1,m}\boldsymbol{\gamma} \\ &= \mathbf{V}_{m+1}(\beta\mathbf{e}_1 - \mathbf{D}_{m+1,m}\boldsymbol{\gamma}), \end{aligned} \tag{9}$$

where $\mathbf{r}_0 = \mathbf{g} - \mathbf{H}\boldsymbol{\delta}^{(0)}$, $\mathbf{v}_i$ is the i-th column vector of $\mathbf{V}_m$, $\mathbf{D}_{m+1,m} = [d_{ij}]_{m+1\times m}$, with $d_{ij} = (\mathbf{v}_i, \mathbf{H}\mathbf{v}_i)$, and $\mathbf{e}_1 = [1, 0, \dots, 0]^T \in \mathbb{R}^{m+1}$. Since the column vectors of the unitary matrix $\mathbf{V}_{m+1}$ are orthonormal, we have

$$\begin{aligned} \|\mathbf{g} - \mathbf{H}\boldsymbol{\delta}\|_2 &= \|\mathbf{V}_{m+1}(\beta\mathbf{e}_1 - \mathbf{D}_{m+1,m}\boldsymbol{\gamma})\|_2 \\ &= \|\beta\mathbf{e}_1 - \mathbf{D}_{m+1,m}\boldsymbol{\gamma}\|_2. \end{aligned} \tag{10}$$

Hence, the problem (8a) is turned into the following optimization:

$$\begin{aligned} \tilde{\boldsymbol{\delta}} &= \boldsymbol{\delta}^{(0)} + \mathbf{V}_m\boldsymbol{\gamma}^* \\ \boldsymbol{\gamma}^* &= \arg \min_{\boldsymbol{\gamma} \in \mathbb{R}^m} \|\beta\mathbf{e}_1 - \mathbf{D}_{m+1,m}\boldsymbol{\gamma}\|_2. \end{aligned} \tag{11}$$

Since m is sufficiently small in our work, we can further use the QR-decomposition method to solve this least-squares problem. Let $\mathbf{D}_{m+1,m} = \mathbf{Q}_{m+1}^T\mathbf{R}_{m+1,m}$ be the QR decomposition of $\mathbf{D}_{m+1,m}$, where $\mathbf{Q}_{m+1}^T \in \mathbb{R}^{(m+1)\times(m+1)}$ is an orthogonal matrix and $\mathbf{R}_{m+1,m} \in \mathbb{R}^{(m+1)\times m}$ is an upper triangular matrix. Then we have

$$\begin{aligned} \|\beta\mathbf{e}_1 - \mathbf{D}_{m+1,m}\boldsymbol{\gamma}\|_2 &= \left\|\beta\mathbf{e}_1 - \mathbf{Q}_{m+1}^T\mathbf{R}_{m+1,m}\boldsymbol{\gamma}\right\|_2 \\ &= \|\beta\mathbf{Q}_{m+1}\mathbf{e}_1 - \mathbf{R}_{m+1,m}\boldsymbol{\gamma}\|_2 \\ &= \left\|\beta\mathbf{q}_1 - \begin{bmatrix} \mathbf{R}_m \\ 0 \end{bmatrix}\boldsymbol{\gamma}\right\|_2, \end{aligned} \tag{12}$$

where $\mathbf{q}_1$ is the first column of $\mathbf{Q}_{m+1}$ and $\mathbf{R}_m$ represents the first m rows of $\mathbf{R}_{m+1,m}$. Then, $\boldsymbol{\gamma}$ can be solved by the upper triangular equations:

$$\mathbf{R}_m\boldsymbol{\gamma} = \beta\mathbf{q}_1(1:m), \tag{13}$$

where $\mathbf{q}_1(1 : m)$ represents the vector consisting of the first m elements of $\mathbf{q}_1$. Through solving these m-dimensional upper trangular equations by Numpy, the optimal $\boldsymbol{\gamma}^*$ is obtained. Substituting $\boldsymbol{\gamma}^*$ back into Eqn. (11), we finally obtain the optimal perturbation $\boldsymbol{\delta}^*$.

In practice, the QR decomposition of $\mathbf{D}_{j+1,j}$ in Eqn. (12) can be implemented by the recursive Givens transformation method (Thornton and Bierman, 1977), i.e., obtain the QR decomposition of $\mathbf{D}_{j+1,j}$ through a Givens transformation based on the QR decomposition of $\mathbf{D}_{j-1,j}$. The QR decomposition of $\mathbf{D}_{j,j-1} \in \mathbb{R}^{j\times(j-1)}$ is presented as

$$\begin{aligned} \mathbf{D}_{j,j-1} &= (\mathbf{G}_{j-1}\mathbf{G}_{j-2}\dots\mathbf{G}_1)^T\mathbf{R}_{j,j-1} \\ &= \mathbf{Q}_j^T\begin{bmatrix} \mathbf{R}_{j-1} \\ 0 \end{bmatrix}_{j\times(j-1)}, \end{aligned} \tag{14}$$

where $\mathbf{R}_{j-1} \in \mathbb{R}^{(j+1)\times(j-1)}$ is an upper triangular matrix and $\mathbf{G}_i$ represents the Givens transformation. To ensure the consistency of the matrix product, we suppose that the dimension of $\mathbf{G}_i$ will automatically increase according to our calculation requirement, i.e., expanding $\mathbf{G}_i$ by adding the identity matrix to the lower right corner. Briefly, we can obtain the last column of $\mathbf{R}_j$ by applying the Givens transformation $\mathbf{G}_1, \mathbf{G}_2, \dots, \mathbf{G}_j$ to the last column of $\mathbf{D}_{j+1,j}$, respectively.

4.3. Second-order adversarial examples. We illustrate the entire procedure of generating our second-order adversarial example (SOAE) in Algorithm 1. Our goal is to construct a second-order adversarial perturbation $\boldsymbol{\delta}^* = \mathbf{H}^{-1}\mathbf{g}$ and output the final adversarial example $\boldsymbol{x}^{\text{adv}} = \boldsymbol{x} + \alpha\boldsymbol{\delta}^*$, which is implemented by multi-iterations to approximate $\boldsymbol{\delta}^*$ in our algorithm. To perform better, the step size is divided into α/N in each iteration, where N is the number of iterations. If the calculated perturbation is larger than the perturbation budget ϵ, we apply a projection step like PGD (Madry *et al.*, 2017). To avoid directly computing

Algorithm 1. Second-order adversarial example (SOAE).

```
Input: Original image $x$, difference step-size $\eta > 0$ and $\tau > 0$.
Output: $x^{\text{adv}} = x_N$
1: for $n = 1, 2, \cdots, N$ do
2:   $\mathbf{g} \leftarrow \nabla L(x_n)$, $\delta^{(0)} \leftarrow \mathbf{g}$
3:   $\mathbf{Hg} \leftarrow (\nabla L(x_n + \eta\mathbf{g}) - \nabla L(x_n))/\eta$
4:   $\mathbf{r}_0 \leftarrow \delta^{(0)} - \mathbf{Hg}$, $\beta \leftarrow \|\mathbf{r}_0\|_2$, $\mathbf{v}_1 \leftarrow \mathbf{r}_0/\beta$
5:   for $j = 1, 2, \ldots$ do
6:     $\mathbf{w}_j \leftarrow \mathbf{H}\mathbf{v}_j$
7:     for $i = 1$ to $j$ do
8:       $h_{ij} \leftarrow (\mathbf{w}_j, \mathbf{v}_i)$
9:       $\mathbf{w}_j \leftarrow \mathbf{w}_j - h_{ij}\mathbf{v}_i$
10:    end for
11:    $h_{j+1,j} \leftarrow \|\mathbf{w}_j\|_2$, $\mathbf{v}_{j+1} \leftarrow \mathbf{w}_j/h_{j+1,j}$
12:    if $\|\tilde{\mathbf{r}}\|_2/\beta < \tau$ then
13:      $m \leftarrow j$, break
14:    end if
15:  end for
16:  Obtain $\gamma^*$ by solving $\mathbf{R}_m\gamma^* = \beta\mathbf{q}_1(1:m)$
17:  $\tilde{\delta} \leftarrow \delta^{(0)} + \mathbf{V}_m\gamma^*$
18:  $x_{n+1} \leftarrow \mathrm{Clip}\{x_n + \frac{\alpha}{N}\tilde{\delta}\}$
19: end for
```

the Hessian matrix, we apply a difference approximation $\mathbf{Hg} = (\nabla L(x + \eta\mathbf{g}) - \nabla L(x))/\eta$ in Step 3, which is proved to be an accurate approximation to the Hessian-vector product (Tsiligkaridis and Roberts, 2020). Moreover, it has been proved that the gradient direction is well aligned with the direction of the maximum curvature of models (Jetly *et al.*, 2018; Fawzi *et al.*, 2018), which makes $\delta^{(0)} \leftarrow \mathbf{g}$ a strong initialization for our algorithm. The inner loop of our algorithm (Steps 5 to 15) is the Hessian inverse approximation, which controls the dimension of the Krylov subspace through a threshold τ. In practice, the dimension m can be compressed to a sufficiently small size, that is $1/20$ of the original Hessian dimension, which retains the approximation accuracy while significantly reducing the computation cost. Our algorithm involves two hyperparameters: (i) the difference step-size η to control the accuracy of the difference approximation and (ii) the approximation threshold τ to determine the approximating precision of the inverse Hessian matrix.

4.4. Theoretical analysis on attack strength.

Recall in Eqn. (3) that we consider the loss function $L(x + \delta)$ as a quadratic function $Q(\delta)$ with respect to our second-order perturbation δ and the final perturbation is obtained through the following optimization:

$$Q(\delta) = L(x) + \nabla L(x)^T\delta + \frac{1}{2}\delta^T\nabla^2 L(x)\delta,$$

$$\delta^* = \arg\max_{\delta \in B_p(\epsilon)} Q(\delta). \tag{15}$$

Note that in the construction of our second-order perturbation δ^* we use a quadratic function $Q(\delta)$ instead of maximizing the original loss $L(x + \delta)$. Assume that there exists an optimal attack $\hat{\delta}$ which directly comes from maximizing $\hat{L} = L(x + \hat{\delta})$. In such cases,

$$\begin{aligned}\hat{L} - Q &= L(x + \hat{\delta}) - L(x) \\ &\quad - (L(x + \delta^*) - L(x)) \\ &= Q(\hat{\delta}) - Q(\delta^*) + o(x^3),\end{aligned} \tag{16}$$

where $\hat{\delta}$ and δ^* represent the "optimal" perturbation and our second-order perturbation respectively, and $o(x^3)$ is the third-order Taylor polynomial. Hence,

$$\begin{aligned}Q(\hat{\delta}) - Q(\delta^*) &= \mathbf{g}^T\hat{\delta} + \frac{1}{2}\hat{\delta}^T\mathbf{H}\hat{\delta} \\ &\quad - (\mathbf{g}^T\delta^* + \frac{1}{2}\delta^{*T}\mathbf{H}\delta^*) \\ &= (\hat{\delta} - \delta^*, \mathbf{g}) \\ &\quad + (\hat{\delta} - \delta^*, \frac{1}{2}\mathbf{H}(\hat{\delta} + \delta^*)).\end{aligned} \tag{17}$$

By applying the Hölder inequality to the last term of Eqn. (17), we get

$$\begin{aligned}|(\hat{\delta} - \delta^*, \frac{1}{2}\mathbf{H}(\hat{\delta} + \delta^*))| &\le \frac{1}{2}\|\hat{\delta} - \delta^*\|_p \\ &\quad \times (\|\mathbf{H}\hat{\delta}\|_q + \|\mathbf{H}\delta^*\|_q) \\ &\le \epsilon^2\,\|\mathbf{H}\|_{p,q},\end{aligned} \tag{18}$$

where ϵ is a perturbation budget. Hence, the bound of real adversarial loss and our second-order loss yields

$$|\hat{L} - Q| \le 2\epsilon\,\|\mathbf{g}\|_q + 2\epsilon^2\,\|\mathbf{H}\|_{p,q} + \frac{\epsilon^3 M}{3}, \tag{19}$$

where M is the supremum-norm of all derivatives of $L(x)$ of order three. This inequality reveals that the proximity of our second-order adversarial perturbation to the optimal perturbation is controlled by the attack strength and the network regularity. A greater regularity of the network will result in a smaller gap between our attack and the optimal attack. It has been experimentally demonstrated that various robust methods, such as adversarial training and geometric regularization models, lead to a more regular loss landscape than their non-robust counterparts (Lyu *et al.*, 2015; Ros and Doshi-Velez, 2018; Moosavi-Dezfooli *et al.*, 2019). Therefore, our SOAE attack is effective in various adversarially trained models, experimentally demonstrated in Section 5.

4.5. Time complexity analysis.

For the convenience of theoretical analysis, we take a three-layer fully connected (FC) neural network as an example since the construction of adversarial examples contains multiple forward and backward propagations. Our FC network contains l_1, l_2, and l_3 neurons in the input, hidden, and output layers, respectively, where $l_2 > l_1$ and $l_2 > l_3$. With one original image resized to $(l_1, 1)$, the forward propagation requires two multiplications of the weight matrix and the activation vector, that is, $l_1 \times l_2 + l_2 \times l_3$ operations. Its time complexity is $\mathcal{O}(l_1 l_2 + l_2 l_3)$. Similarly, the time complexity of back-propagation is $\mathcal{O}(l_1 l_2 + l_2 l_3)$ as well. Hence, the gradient calculation for the SOAE's initialization has $\mathcal{O}(l_2^2)$ time complexity.

Our calculated gradient $\mathbf{g}$ has the same dimension as the input, i.e., $\dim \mathbf{g} = l_1$. Steps 3 and 4 only contain the multiplication of numbers and l_1-dimensional vectors. Thus, their complexity is $\mathcal{O}(l_1)$. Note that Step 6 does not require Hessian $\mathbf{H}$; instead, we have

$$\begin{aligned}\mathbf{H}\mathbf{v}_1 &= \frac{1}{\beta}\mathbf{H}(\boldsymbol{\delta}^{(0)} - \mathbf{H}\mathbf{g}) = \frac{1}{\beta}(\mathbf{H}\mathbf{g} - \mathbf{H}\mathbf{H}\mathbf{g}) \\ &= \frac{1}{\beta}\Bigg[\frac{\nabla L(\boldsymbol{x}_n + \eta\mathbf{g}) - \nabla L(\boldsymbol{x}_n)}{\eta} \\ &\quad - \frac{\nabla L(\boldsymbol{x}_n + \eta\mathbf{H}\mathbf{g}) - \nabla L(\boldsymbol{x}_n)}{\eta}\Bigg],\end{aligned} \tag{21}$$

where the Hessian-vector product $\mathbf{H}\mathbf{g}$ is approximated through the finite difference:

$$\mathbf{H}\mathbf{g} = \frac{1}{\eta}\left(\nabla L(\boldsymbol{x}_n + \eta\mathbf{g}) - \nabla L(\boldsymbol{x}_n)\right), \tag{22}$$

where the same finite difference can be applied to $\mathbf{H}\mathbf{H}\mathbf{g}$ by considering it to be another Hessian-vector product, in which case the corresponding vector is the product $\mathbf{H}\mathbf{g}$, i.e.,

$$\mathbf{H}^2\mathbf{g} = \frac{1}{\eta}\left(\nabla L(\boldsymbol{x}_n + \eta\mathbf{H}\mathbf{g}) - \nabla L(\boldsymbol{x}_n)\right). \tag{23}$$

Similarly, all $\mathbf{w}_j = \mathbf{H}\mathbf{v}_j$ can be calculated through

$$\begin{aligned}\mathbf{w}_j &= \mathbf{H}\mathbf{v}_j = \mathbf{H}(\mathbf{H}\mathbf{v}_{j-1}) = \cdots \\ &= \mathbf{H}(\mathbf{H}\cdots(\mathbf{4}(\mathbf{H}\mathbf{v}_1))).\end{aligned} \tag{24}$$

In j loops, where j is up to m, each calculation of $\mathbf{w}_j$ requires one forward and one backward propagation. Its time complexity is $\mathcal{O}(ml_2^2)$.

The i loop (Steps 7 to 10) only contains the inner products of l_1-dimensional vectors and number-vector multiplications. The time complexity of one loop is $\mathcal{O}(l_1)$. Note that i is up to j, where j is up to m, and the total time complexity of the i loop amounts to $\mathcal{O}(l_1(1 + 2 + \cdots + m)) = \mathcal{O}(m^2 l_1)$.

After the j loop, there is an m-dimensional upper triangular linear equation in Step 16 which comes from the QR decomposition of $\mathbf{D}_{m+1,m} = [d_{ij}]_{m+1\times m}$ where $d_{ij} = (\mathbf{v}_i, \mathbf{H}\mathbf{v}_i)$, referring to Eqn. (9). The time complexity of the QR decomposition of an m-dimensional matrix is $\mathcal{O}(m^3)$ when applying the Givens transformation (Thornton and Bierman, 1977). Solving the upper triangular equations in Step 16 has $\mathcal{O}(m^2)$ time complexity. The rest of the operations followed by Step 16 are simple operations on l_1-dimensional vectors with $\mathcal{O}(l_1)$ complexity.

Now we can give a formal estimate of our SOAE. For a three-layer FC network with input size $(l_1, 1)$ and a hidden layer with l_2 neurons, the time complexity of the SOAE is

$$\mathcal{O}(l_2^2 + ml_2^2 + m^2 l_1 + m^2). \tag{25}$$

Here, m is controlled by a threshold τ in Step 12, which we will discuss in Section 5.3. In practice, m coincides with input size l_1, and a small m value is sufficient to guarantee high performance. This theoretical analysis reveals the SOAE's effectiveness on small-scale datasets. Even for large-scale images such as ImageNet, our method still shows competitive performance in practical experiments.

In addition, there is no extra storage requirement for our algorithm since it is Hessian-free, which means the batch size can be set as the same as other attack methods. Such a design makes our method more competitive than current second-order adversarial methods.

5. Experiment

In this section, extensive experiments are conducted to evaluate the effectiveness of our SOAE and SOAT. We train ResNet18 and WideResNet using various methods, including standard training and adversarial training on MNIST and CIFAR-10, and test them with multiple attack methods. For large-scale image evaluation, we use 100,000 samples of 100 classes from ImageNet to construct our dataset. After the effectiveness evaluation, we analyze the hyperparameters in our methods to find the most proper values.

5.1. Experimental setup.

5.1.1. Datasets. MNIST contains 60,000 training examples and 10,000 test examples of handwriting numbers with the size of 28×28. CIFAR-10 contains 60,000 RGB images of 10 classes with the size of 32×32, which are divided into a training set containing 50,000 examples and a test set containing 10,000 examples. ILSVRC2012 is a subset of ImageNet containing 1.2 million 224×224 images of 1,000 classes, where we randomly select 100,000 images of 100 classes to construct our dataset ImageNet-100 for large-scale image evaluation.

Table 1. Experimental results on MNIST. Here STD represents standard training with clean examples.

Models used	Training methods	Clean examples	FGSM attack	PGD attack	Auto attack	ASOD attack	**SOAE** attack
ResNet18	STD	97.78%	0.07%	0.06%	0.01%	0.01%	**0.01**%
	PGD (Madry *et al.*, 2017)	86.34%	63.40%	55.89%	49.10%	**20.20**%	30.85%
	TRADES (Zhang *et al.*, 2019)	85.98%	65.05%	53.87%	40.27%	39.76%	**38.32**%
	SOAR (Ma *et al.*, 2020)	87.95%	67.15%	56.06%	18.25%	29.12%	**20.14**%
	STN (Li *et al.*, 2018)	86.26%	66.77%	54.90%	29.81%	48.74%	**25.25**%
	SCORPIO (Tsiligkaridis and Roberts, 2020)	87.92%	68.18%	58.44%	41.01%	47.62%	**39.03**%
	SOAT	87.62%	70.89%	60.13%	**39.10**%	49.45%	47.53%
WideResNet	STD	99.90%	0.08%	0.05%	0.01%	0.02%	**0.01**%
	PGD (Madry *et al.*, 2017)	84.55%	66.50%	57.12%	44.51%	**21.76**%	29.84%
	TRADES (Zhang *et al.*, 2019)	80.56%	68.32%	56.49%	42.79%	41.37%	**32.15**%
	SOAR (Ma *et al.*, 2020)	86.99%	68.64%	57.80%	19.13%	33.90%	**27.60**%
	STN (Li *et al.*, 2018)	87.42%	66.81%	53.21%	33.91%	46.25%	**30.45**%
	SCORPIO (Tsiligkaridis and Roberts, 2020)	89.01%	67.43%	55.02%	39.88%	43.69%	**38.24**%
	SOAT	85.23%	69.98%	57.73%	45.22%	45.96%	**43.47**%

5.1.2. Adversarial attack. We use five different methods to generate adversarial examples: (i) an FGSM (Goodfellow *et al.*, 2015) with the perturbation size $\epsilon = 8/255$; (ii) a 20-iteration PGD (Madry *et al.*, 2017) with the step size $2/255$ and the total perturbation size $\epsilon = 8/255$; (iii) AutoAttack (Croce and Hein, 2020b) with L_∞-norm and $\epsilon = 8/255$; (iv) ASOD (Li *et al.*, 2018) with the total perturbation size $\epsilon = 8/255$ (Since their code is not open-source, we re-write its program according to the implementation in the original paper to reproduce the claimed performance.); and (v) our SOAE with the total perturbation size $\epsilon = 8/255$.

5.1.3. Adversarial training. We train ResNet18 and WideResNet with two first-order adversarial training methods: (i) a PGD with seven iterations and the step size of $2/255$, (ii) TRADES with the total perturbation $\epsilon = 8/255$ and the step size of $2/255$; three second-order adversarial training methods: (i) the SOAR proposed by Ma *et al.* (2020), in which we follow the original settings, (ii) the STN proposed by (Li *et al.*, 2018) trained with ASOD, (iii) the SCORPIO regularizer proposed by Tsiligkaridis and Roberts (2020), and (iv) our method SOAT. All training phases have 300 epochs with a batch size of 128. We use SGD with a momentum of 0.9, where the initial learning rate is 0.1 with a weight decay of 0.0003.

5.2. Main results.

5.2.1. Results on MNIST. The results in Table 1 illustrate the effectiveness of our methods. Compared with the first-order adversarial attacks, our SOAE decreases the models' accuracy remarkably, though these models are adversarially trained. Compared with the second-order ASOD attack, our SOAE attack still achieves an equivalent or larger accuracy drop. Nevertheless, ASOD (Li *et al.*, 2018) is merely effective in compromising PGD adversarial training; our SOAE can disrupt a wide range of adversarial training techniques. On the other hand, if the model is adversarially trained with SOAEs, named second-order adversarial training (SOAT), this can significantly improve the network robustness, particularly resistant to second-order adversarial attacks. Even under the strongest AutoAttack, SOAT still shows state-of-the-art performance.

5.2.2. Results on CIFAR-10. We also test our method on the CIFAR-10 dataset. Experimental results illustrate the efficiency of our method as well. Our attack (SOAE) and defense methods (SOAT) achieve state-of-the-art performance. Except for AutoAttack on the SOAR-trained models (it is claimed by Ma *et al.* (2020) that their method is highly vulnerable to AutoAttack), our SOAE outperforms all the other attacks. On the other hand, the models trained by SOAT still guarantee high robustness against various adversarial attacks. It even achieves the highest accuracy on two trained models under AutoAttack.

5.2.3. Results on ImageNet-100. We test our algorithm on our large-scale image dataset ImageNet-100. As analyzed in the previous section, the SOAE's speed is related to the input size. Large-scale images may

Table 2. Experimental results on CIFAR-10. Here STD represents standard training with clean examples.

Models used	Training methods	Clean examples	FGSM attack	PGD attack	Auto attack	ASOD attack	**SOAE** attack
ResNet18	STD	92.54%	30.59%	0.14%	0.05%	0.12%	**0.08**%
	PGD (Madry *et al.*, 2017)	80.64%	50.96%	42.86%	40.59%	40.32%	**38.19**%
	TRADES (Zhang *et al.*, 2019)	85.61%	53.06%	45.38%	41.34%	43.49%	**40.26**%
	SOAR (Ma *et al.*, 2020)	87.95%	55.70%	56.06%	**19.66**%	45.08%	35.45%
	STN (Li *et al.*, 2018)	83.03%	54.83%	52.34%	38.00%	48.15%	**37.98**%
	SCORPIO (Tsiligkaridis and Roberts, 2020)	86.58%	55.07%	49.56%	39.27%	46.82%	**38.75**%
	SOAT	85.47%	56.24%	54.72%	**45.98**%	47.73%	49.27%
WideResNet	STD	93.79%	44.77%	0.03%	0.01%	0.13%	0.09%
	PGD (Madry *et al.*, 2017)	81.83%	51.15%	43.66%	42.62%	41.86%	**39.14**%
	TRADES (Zhang *et al.*, 2019)	86.54%	54.66%	45.70%	43.70%	44.22%	**41.21**%
	SOAR (Ma *et al.*, 2020)	88.02%	67.15%	57.92%	20.41%	47.31%	39.35%
	STN (Li *et al.*, 2018)	84.11%	58.59%	53.16%	38.95%	49.27%	**38.02**%
	SCORPIO (Tsiligkaridis and Roberts, 2020)	86.99%	61.37%	50.50%	41.28%	47.33%	**40.11**%
	SOAT	86.38%	62.33%	57.67%	**47.53**%	50.50%	51.46%

Table 3. Experimental results of ResNet18 on ImageNet-100.

Training methods	Clean examples	PGD attack	Auto attack	**SOAE** attack
STD	91.33%	0.02%	0	0
PGD	78.26%	55.37%	48.66%	**40.31**%
TRADES	82.91%	56.09%	47.25%	**40.05**%
SOAR	80.80%	61.68%	45.12%	**42.35**%
SOAT	84.50%	62.84%	52.99%	**47.90**%

theoretically increase the SOAE's convergence time. This effect is not apparent on MNIST and CIFAR-10 since the approximation dimension m is small when dealing with 28×28 and 32×32 images ($m = 15$ will be enough, as shown in Section 5.3). However, when encountering 224×224 images, the required m value unavoidably increases to more than 100. This situation makes m's effect significant in Eqn. (25). However, additional running time is proved to be worthy, according to experimental results. Our SOAE attack outperforms other attack methods in all situations. It successfully improves accuracy by 8% compared with AutoAttack on a PGD-training model. Its performances on four different adversarial training models are better than other attack methods. Training with the SOAE also yields satisfying results, where our SOAT model has the highest accuracy under all three attacks.

5.3. Hyperparameter analysis. In this subsection, we mainly discuss three hyperparameters in our method: the difference step-size η, the approximation threshold τ, and the iteration time N.

5.3.1. Difference step-size $\boldsymbol{\eta}$. Recall that in Step 3 of Algorithm 1 we use a first-order difference of gradients to approximate the Hessian-vector product, and the step-size η controls the accuracy of this approximation. We evaluate how different η values influence the performance of our algorithm. Specifically, we generate adversarial examples on MNIST with different η values, and use them to test the accuracy of the standard-trained, PGD-trained, and SOAT-trained ResNet18 models, as shown in Fig. 1. Our SOAE attack achieves the best performance as η is less than 10^{-5}. But if η is larger than 10^{-4}, it brings obvious ineffectiveness to our method. As η increases to 10^{-2}, most perturbed examples generated by our algorithm are actually not adversarial examples. This means that a large η will cause an extremely poor approximation to the Hessian-vector product and lead the subsequent step of the algorithm to complete blindness. In conclusion, the proper η value should be no more than 10^{-5}.

5.3.2. Approximation threshold $\boldsymbol{\tau}$. This hyper-parameter controls the approximating precision of the inverse Hessian matrix. Intuitively, a smaller τ leads to a more accurate approximation. However, too small τ will significantly increase the convergence time. As is analyzed in the previous section, the τ value determines the approximation dimension m, which significantly influences our algorithm's running time. We use a series of τ values to generate 1,000 adversarial examples on CIFAR-10 and attack the standard-trained ResNet18 model. The accuracy change and the convergence time are shown in Fig. 3. When $\tau < 10^{-3}$, our SOAE can perform

Table 4. PSNR and SSIM of different adversarial examples.

Method	Dataset	PSNR	SSIM	Dataset	PSNR	SSIM	Dataset	PSNR	SSIM
FGSM	MNIST	74.23	0.86	CIFAR-10	74.70	0.86	ImageNet-100	75.89	0.87
PGD		72.30	0.83		74.11	0.84		78.41	0.86
AutoAttack		71.49	0.82		72.62	0.85		77.07	0.84
SOAE		**76.08**	**0.87**		**79.53**	**0.86**		**81.66**	**0.89**

Table 5. Approximation dimension m under different τ values.

Dataset	τ value				
	10^{-5}	10^{-4}	10^{-3}	10^{-2}	10^{-1}
MNIST(28×28)	152	71	43	26	17
CIFAR-10(32×32)	243	117	65	38	22
ImageNet(224×224)	2196	1324	746	327	221

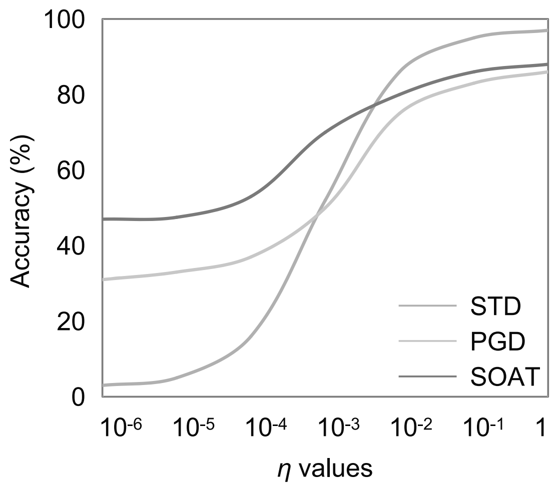


Fig. 1. Accuracy change under different η values. STD, PGD, and SOAT represent different training methods.

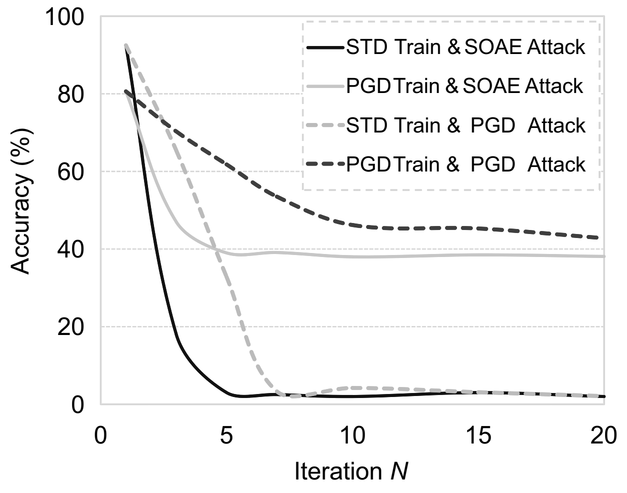


Fig. 2. Accuracy change for the ResNet18 model with the increasing number of iterations N. We run the PGD and the SOAE for 20 iterations and train the model with the standard training and PGD.

well, where the accuracy drops below 16%. We further find that the accuracy barely changes from $\tau = 10^{-6}$ to 10^{-3}; however, the convergence time drops significantly. Therefore, we suggest that $10^{-4} \leq \tau \leq 10^{-3}$ is proper in practice. We also record the m values corresponding to different τ values on different datasets, as shown in Table 5.

5.3.3. Total number of iterations N.

The total number of iterations directly influences attack number of success rates. But a large number of iterations will lead to insufferable time consumption. To find proper numbers of iterations to achieve a good trade-off between attack success rates and computation costs, we run 20 iterations for our algorithm and the PGD, on ResNet18 and record the accuracy changes in every iteration in Fig. 2. Compared with the PGD using 7 or 20 iterations to converge, our method needs only less than four iterations, yielding a competitive performance. This further proves that our second-order method has a perfect convergence property. One may argue that iterations cannot fully measure computation costs. Although the time of a single iteration is a little longer than that of other first-order methods since our method contains a certain number of linear combinations and small matrix multiplications, the total time of generating SOAEs on the entire CIFAR-10 dataset is only one hour longer than that of AutoAttack on an RTX3090 GPU. We believe that our method is practical and it will be faster after further optimization.

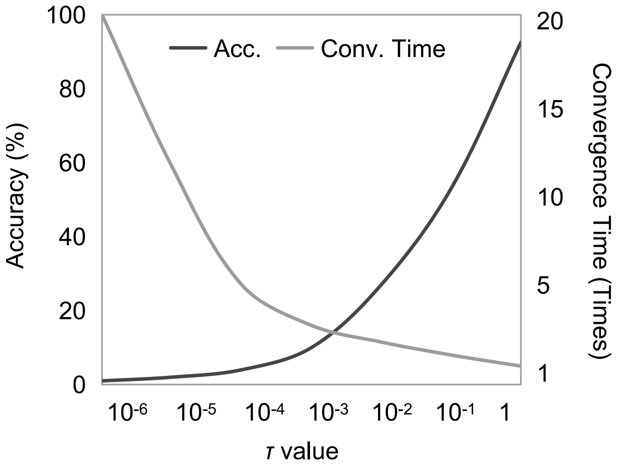


Fig. 3. Accuracy change for ResNet18 and convergence time under different τ values. The convergence time is a multiple of the time when $\tau = 1$.

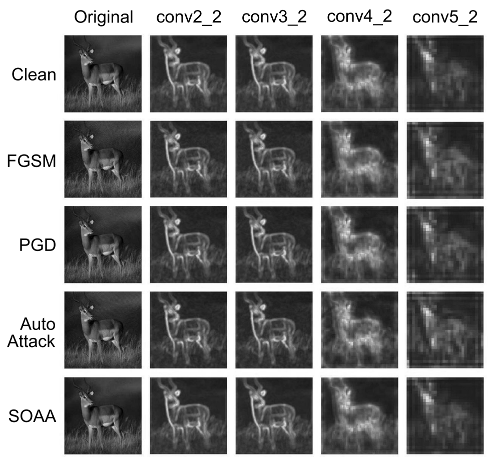


Fig. 4. Visualization of the feature map in different layers of standard-trained ResNet18 with different input examples. The perturbation size of all attack methods is $8/255$.

5.4. Visualization. In this section, we visualize the feature maps of a standard-trained ResNet18 model with different input examples (clean, the FGSM, the PGD, AutoAttack and the SOAE) on the ImageNet dataset, as shown in Fig. 4. Compared with the first-order methods, the adversarial perturbation of the SOAE is more imperceptible to human eyes. Our SOAEs can deceive the model at a relatively low feature distortion level. Besides, we notice that most second-order perturbations occur on the objective edge, where the pixel value has a more rapid change than other image regions. This phenomenon aligns well with the role of a second-order differential operator in image processing, which has a stronger ability to locate the edge information. This visualization further implies the hidden relationship between the adversarial robustness and the edge information of the classification objective. According to Heaven (2019), a disruption of low-level semantics, such as edge information, can significantly weaken a CNN's ability to understand high-level semantics. This provides a reasonable explanation for our method's effectiveness.

To further demonstrate our second-order perturbations' imperceptibility, we use the peak signal noise ratio (PSNR (cf. Faragallah *et al.*, 2021)) and structural similarity (SSIM (cf. Nilsson and Akenine-Möller, 2020)) to quantitatively measure the similarity and distortion between original examples and their corresponding adversarial examples. The formulas for the PSNR and SSIM are as follows:

$$\mathrm{MSE} = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\left[I(i,j) - K(i,j)\right]^2, \qquad (20)$$

$$\mathrm{PSNR} = 10 \cdot \log_{10}\left(\frac{\mathrm{MAX}_I^2}{\mathrm{MSE}}\right),$$

where MSE refers to the mean square error, m and n refer, respectively, to the width and height of the image, I and K refer to the clean and the noisy image, respectively, and MAX is the maximum possible pixel value of the image. If each pixel is represented by an 8-bit binary, i.e., the $\mathrm{MAX} = 2^8 - 1$. We have

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}, \qquad (21)$$

where x and y refers to different samples, μ_x and μ_y are the means of x and y, σ_x^2 and σ_y^2 are the variances of x and y, σ_{xy} is the covariance of x and y, and c_1, c_2 are constants.

In image signal processing, these two matrices are often used to evaluate the quality of the generated images, where the PSNR focuses more on the pixel values and SSIM comprehensively measures the structural similarity, including brightness, contrast, and structure. Generally, the higher the PSNR and SSIM values, the less distortion exists between a perturbated image and its original counterpart. We calculate the average PSNR and SSIM of 1,000 adversarial examples (the FGSM, the PGD, AutoAttack, and the SOAE) with respect to their original examples on MNIST, CIFAR-10, and ImageNet-100, respectively. The experiment results are reported in Table 4. Among all adversarial attack methods, SOAEs achieve the highest PSNR and SSIM values, which indicates that our method can generate low-distortion adversarial examples. This is attributed to the smoother approximation of the loss function of the model to effectively find an adversary point within the $B_p(\epsilon)$ of the input image.

6. Conclusion

This paper demonstrated a Hessian-free second-order adversarial example generation method. We further applied it to adversarial training, which effectively enables the model to demonstrate stronger robustness against various attacks, including first-order and second-order attacks. Benefiting from constructing a Hessian-vector product in the Krylov subspace, our algorithm avoids directly computing the Hessian matrix for gradient updates, which makes second-order methods practical in adversarial training. In the future, a feasible direction of our work is to better understand the landscape of the loss function around the input images, hence selecting a more reasonable initial point of the Taylor expansion and making the approximation more accurate.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (92167203), the Zhejiang Provincial Natural Science Foundation of China (LZ22F020007), and the Foundation of the Zhejiang Key Laboratory of Multi-dimensional Perception Technology Application and Cybersecurity (HIK2022008).

References

Andriushchenko, M., Croce, F., Marion, N.F. and Hein, M. (2020). Square attack: A query-efficient black-box adversarial attack via random search, *European Conference on Computer Vision, Glasgow, UK*, pp. 484–501.

Anil, R., Gupta, V., Koren, T., Regan, K. and Singer, Y. (2020). Second-order optimization made practical, *arXiv:* 2002.09018.

Athalye, A., Carlini, N. and Wagner, D. (2018). Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples, *International Conference on Machine Learning, Stockholm, Sweden*, pp. 274–283.

Badjie, B., Cecílio, J. and Casimiro, A. (2023). Denoising autoencoder-based defensive distillation as an adversarial robustness algorithm, *CoRR*: abs/2303.15901.

Bertolace, A., Gatsis, K. and Margellos, K. (2024). Robust optimization for adversarial learning with finite sample complexity guarantees, *CoRR*: abs/2403.15207.

Croce, F. and Hein, M. (2020a). Minimally distorted adversarial examples with a fast adaptive boundary attack, *International Conference on Machine Learning*, pp. 2196–2205, (virtual event).

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

Ding, X., Zhang, X., Zhou, Y., Han, J., Ding, G. and Sun, J. (2022). Scaling up your kernels to 31x31: Revisiting large kernel design in CNNs, *CoRR*: abs/2203.06717.

Faragallah, O.S., El-Hoseny, H., El-Shafai, W., El-Rahman, W.A., El-Sayed, H.S., El-Rabaie, E.-S.M., El-Samie, F.E.A. and Geweid, G.G.N. (2021). A comprehensive survey analysis for present solutions of medical image fusion and future directions, *IEEE Access* **9**: 11358–11371.

Fawzi, A., Moosavi-Dezfooli, S., Frossard, P. and Soatto, S. (2018). Empirical study of the topology and geometry of deep networks, *2018 IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, USA*, pp. 3762–3770.

Ge, Z., Wang, X., Liu, H., Shang, F. and Liu, Y. (2023). Boosting adversarial transferability by achieving flat local maxima, *Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, USA*.

Goodfellow, I., Shlens, J. and Szegedy, C. (2015). Explaining and harnessing adversarial examples, *International Conference on Learning Representations, San Diego, USA*.

Guo, S., Li, X., Zhu, P. and Mu, Z. (2023). Ads-detector: An attention-based dual stream adversarial example detection method, *Knowledge-Based Systems* **265**: 110388.

Heaven, D. (2019). Why deep-learning AIs are so easy to fool, *Nature* **574**(777): 163–166.

Huang, B., Chen, M., Wang, Y., Lu, J., Cheng, M. and Wang, W. (2023). Boosting accuracy and robustness of student models via adaptive adversarial distillation, *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, Canada*, pp. 24668–24677.

Jea, K.C. and Young, D.M. (1980). Generalized conjugate gradient acceleration of non-symmetrizable iterative methods, *Linear Algebra and Its Applications* **34**: 159–194.

Jetly, S., Lord, N. and Torr, P. (2018). With friend like this, who need adversaries?, *2018 Conference on Neural Information Processing Systems, Montréal, Canada*, pp. 10772–10782.

Jin, G., Yi, X., Huang, W., Schewe, S. and Huang, X. (2022). Enhancing adversarial training with second-order statistics of weights, *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, New Orleans, USA*, pp. 15273–15283.

Jin, G., Yi, X., Wu, D., Mu, R. and Huang, X. (2023). Randomized adversarial training via Taylor expansion, *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, Canada,* pp. 16447–16457.

Li, B., Chen, C., Wang, W. and Carin, L. (2018). Second-order adversarial attack and certifiable robustness. *arXiv:* 1809.03113.

Li, L. and Spratling, M.W. (2023). Understanding and combating robust overfitting via input loss landscape analysis and regularization, *Pattern Recognition* **136**: 109229.

Li, T., Wu, Y., Chen, S., Fang, K. and Huang, X. (2022). Subspace adversarial training, *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, USA*, pp. 13399–13408.

Liu, X., Kuang, H., Lin, X., Wu, Y. and Ji, R. (2023). CAT: Collaborative adversarial training, *CoRR:* abs/2303.14922.

Long, Y., Wen, Y., Han, J., Xu, H., Ren, P., Zhang, W., Zhao, S. and Liang, X. (2023). CAPDET: Unifying dense captioning and open-world detection pretraining, *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, Canada*, pp. 15233–15243.

Lu, Y., Ren, H., Chai, W., Velipasalar, S. and Li, Y. (2024). Time-aware and task-transferable adversarial attack for perception of autonomous vehicles, *Pattern Recognition Letters* **178**: 145–152.

Lyu, C., Huang, K. and Liang, H. (2015). A unified gradient regularization family for adversarial examples, *IEEE International Conference on Data Mining, Atlantic City, USA,* pp. 301–309.

Ma, A., Faghri, F., Papernot, N. and Massoud Farahmand, A. (2020). SOAR: Second-order adversarial regularization. *arXiv:* 2004.01832.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D. and Vladu, A. (2017). Towards deep learning models resistant to adversarial attacks, *arXiv:* 1706.06083.

Moosavi-Dezfooli, S.-M., Fawzi, A., Uesato, J. and Frossard, P. (2019). Robustness via curvature regularization, and vice versa, *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, USA*, pp. 9070–9078.

Nilsson, J. and Akenine-Möller, T. (2020). Understanding SSIM, *arXiv:* 2006.13846.

Pozdnyakov, V., Kovalenko, A., Makarov, I., Drobyshevskiy, M. and Lukyanov, K. (2024). Adversarial attacks and defenses in automated control systems: A comprehensive benchmark, *arXiv:* 2403.13502.

Ros, A.S. and Doshi-Velez, F. (2018). Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients, *AAAI Conference on Artificial Intelligence, New Orleans, USA,* pp. 1660–1669.

Shimonishi, H., MAKI, I., Murase, T. and Murata, M. (2002). Dynamic fair bandwidth allocation for diffserv classes, *IEEE International Conference on Communications, ICC 2002, New York, USA*, pp. 2348–2352.

Song, M., Choi, J. and Han, B. (2024). A training-free defense framework for robust learned image compression, *CoRR:* abs/2401.11902.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. and Fergus, R. (2013). Intriguing properties of neural networks, *arXiv*: 1312.6199.

Tejankar, A., Sanjabi, M., Wang, Q., Wang, S., Firooz, H., Pirsiavash, H. and Tan, L. (2023). Defending against patch-based backdoor attacks on self-supervised learning, *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, Canada*, pp. 12239–12249.

Thornton, C. and Bierman, G. (1977). Givens transformation techniques for Kalman filtering, *Acta Astronautica* **4**(7): 847–863.

Tsiligkaridis, T. and Roberts, J. (2020). Second-order optimization for adversarial robustness and interpretability, *arXiv:* 2009.04923.

Wang, H. and Wang, Y. (2022). Self-ensemble adversarial training for improved robustness, *10th International Conference on Learning Representations, ICLR 2022,* (virtual event).

Wu, T., Luo, T. and Wunsch II, D.C. (2024). LRS: Enhancing adversarial transferability through Lipschitz regularized surrogate, *Proceedings of the AAAI Conference on Artificial Intelligence, Vancouver, Canada*, pp. 6135–6143.

Yang, X., Liu, C., Xu, L., Wang, Y., Dong, Y., Chen, N., Su, H. and Zhu, J. (2023). Towards effective adversarial textured 3D meshes on physical face recognition, *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada*, pp. 4119–4128.

Yeom, I. and Reddy, A. (2001). Modeling TCP behavior in a differentiated services network, *IEEE/ACM Transactions on Networking* **9**(1): 31–46.

Yin, Z., Liu, M., Li, X., Yang, H., Xiao, L. and Zuo, W. (2023). MetaF2N: Blind image super-resolution by learning efficient model adaptation from faces, *IEEE/CVF International Conference on Computer Vision, ICCV 2023, Paris, France*, pp. 12987–12998.

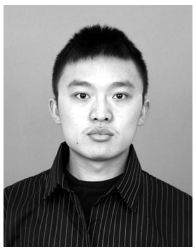
Zhang, H., Yu, Y., Jiao, J., Xing, E., Ghaoui, L.E. and Jordan, M. (2019). Theoretically principled trade-off between robustness and accuracy, *Proceedings of the 36th International Conference on Machine Learning, Long Beach, USA*, pp. 7472–7482.

Zhang, J., Qian, W., Nie, R., Cao, J. and Xu, D. (2023). Generate adversarial examples by adaptive moment iterative fast gradient sign method, *Applied Intelligence* **53**(1): 1101–1114.

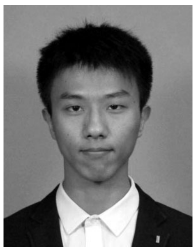
Zhang, X. (2016). Empirical risk minimization, *in* C. Sammut and G.I. Webb (Eds), *Encyclopedia of Machine Learning and Data Mining*, Springer, Berlin/Heidelberg, pp. 392–393.

Zhao, K., Chen, X., Huang, W., Ding, L., Kong, X. and Zhang, F. (2024). Ensemble adversarial defense via integration of multiple dispersed low curvature models, *CoRR:* abs/2403.16405.

Yaguan Qian received a PhD in computer science from Zhejiang University in 2014. He is currently a full professor with the School of Big-Data Science, Zhejiang University of Science and Technology. He has authored or co-authored more than 40 papers in peer-reviewed international conferences and journals. His main research interests include AI security, machine learning, pattern recognition, and machine vision.


Liangjian Zhang received his BS degree in science from Shandong Agricultural University in 2022. He is now a postgraduate student at the School of Big-Data Science, Zhejiang University of Science and Technology. His research interests include adversarial attacks and defenses.

Yuqi Wang is a graduate student at the School of Big-Data Science, Zhejiang University of Science and Technology. He received a BS degree from the School of Mathematics, Harbin Institute of Technology, in 2019. His research interest is adversarial learning in computer vision fields.


Boyuan Ji received his BS degree from Shanghai Maritime University in 2021. He is currently an MS candidate at the School of Science, Zhejiang University of Science and Technology. His research interests include deep learning, machine learning security and machine vision.

Tengteng Yao received a PhD in computational mathematics from Xiamen University in 2016. She is currently an associate professor at the School of Science, Zhejiang University of Science and Technology. She has authored or co-authored several papers in peer-reviewed international journals and conferences.

Bin Wang is a professor at the Zhejiang Key Laboratory of Artificial Intelligence of Things (AIoT) Network and Data Security. He received his PhD degree from the China National Digital Switching System Engineering & Technological R&D Center in 2010. His research interests mainly include Internet of things security, cryptography, artificial intelligence security, and new network security architectures.

Received: 9 February 2024
Revised: 12 May 2024
Accepted: 14 May 2024