IMPROVING ADVERSARIAL ROBUSTNESS OF DEEP NEURAL NETWORKS VIA SELF-ADAPTIVE MARGIN DEFENSE

Anonymous authors
Paper under double-blind review

ABSTRACT

Adversarial training has become the most popular and effective strategy to improve Deep Neural Network (DNN) robustness against adversarial noises. Many adversarial training methods have been proposed in the past few years. However, most adversarial training methods are highly susceptible to hyperparameters, especially the training noise upper bound. Tuning these parameters is expensive for large datasets and difficult for people not in the adversarial robustness research domain, which prevents adversarial training techniques from being used in many application fields. This paper introduces a new adversarial training method with a gradual expansion mechanism to generate adversarial training samples, and it is parameter-free for the user. By gradually expanding the exploration range with self-adaptive and gradient-aware step size, adversarial training samples can be placed into the optimal locations in the input data space. Unlike other defense methods that usually need to fine-tune hyperparameters (e.g., training noise upper bound) by grid-search, our method has no hyperparameters for the user. We name our method Self-adaptive Margin Defense (SMD). We evaluate SMD on three publicly available datasets (CIFAR10, SVHN, and Fashion-MNIST) under the most popular adversarial attacks, AutoAttack and PGD. The results show that: (1) compared with all other competing defense methods, SMD has the best overall performance in robust accuracy on noisy data; (2) the accuracy degradation of SMD on clean data is minor among all competing defense methods.

1 INTRODUCTION AND MOTIVATION

Deep neural networks (DNNs) have become the first choice for automated image analysis due to their superior performance. However, recent studies have shown that DNNs are vulnerable to adversarial noises, which were first discovered by (Szegedy et al., 2014) and then explained by (Goodfellow et al., 2015). To improve the adversarial robustness of a DNN model, adversarial training is the most popular strategy. By generating adversarial training samples to train the model, adversarial training can improve the adversarial robustness of the DNN model. The vanilla adversarial training (Madry et al., 2018; Kurakin et al., 2018) is to generate adversarial training samples with a fixed and unified adversarial noise upper bound. Many advanced adversarial training methods have been proposed. For instance, TRADES (Zhang et al., 2019) and MART (Wang et al., 2019b) use loss regularization terms to make a trade-off between adversarial robustness and standard accuracy. DAT (Wang et al., 2019a) uses converge quality as a criterion to adjust adversarial training perturbations. ATES (Sitawarin et al., 2020) and CAT (Cai et al., 2018) apply curriculum strategy for adversarial training. The misclassification-aware strategy is used in several methods including IAAT (Balaji et al., 2019), FAT(Zhang et al., 2020a), Customized ATCheng et al. (2020) and MMADing et al. (2019a), which adjusts the noise upper bound in the training process. GAIRAT (Zhang et al., 2020b) adds sample-wise weights to the loss for robust training, based on a sample's weakness under an adversarial attack. However, most adversarial training methods are highly susceptible to hyperparameters, among which, the training noise upper bound is the most frequently used. Usually, a very large training noise upper bound leads to a very large accuracy degradation on clean data, while a very small one may not be enough to improve adversarial robustness. Tuning these hyperparameters is expensive for large datasets and difficult for people not in the adversarial robustness research do-

main, which prevents adversarial training techniques from being adopted in application fields, e.g., medical image analysis which often handles large 3D images.

We present a new adversarial training method in this paper, and our contributions are as follows.

(I) We propose a gradual expansion mechanism to generate adversarial samples for training, and show that this mechanism can lead to an optimal state under certain condition.

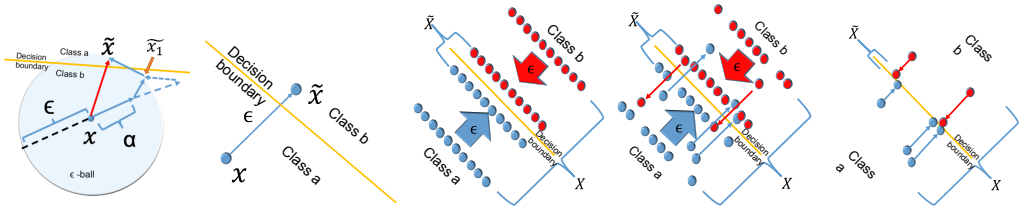
(II) Based on the gradual expansion mechanism, we design and develop Self-adaptive Margin Defense (SMD). This new adversarial training method is parameter-free for the user, by gradually expanding the exploration range with self-adaptive and gradient-aware step sizes. Adversarial training samples can be pushed to the optimal locations in the input data space. SMD has the following advantages. (1) Unlike other defense methods that usually need to fine-tune hyperparameters (e.g., the training noise upper bound), our method has no hyperparameters for the user; (2) Compared with the other competing defense methods, SMD has the best overall performance in robust accuracy on noisy data; (3) The accuracy degradation of SMD on clean data is minor among all the defense methods evaluated in this paper.

2 Related Work

2.1 Terminology

For simplicity, we use "**noise level**" to refer to the upper bound ϵ of the adversarial noise for training or testing. "**clean accuracy**" denotes a model's accuracy on clean data. "**noisy accuracy**" denotes a model's accuracy on noisy/adversarial data, which measures the model's adversarial robustness and is called robust accuracy in some literature. A sample's "**margin**" denotes the distance between the sample and the decision boundary in the input data space. A "**clean sample**" denotes a sample without adversarial noise. A "**noisy sample**" denotes a noisy/adversarial sample. A "**clean model**" denotes a naturally trained model, without using any adversarial training.

2.2 Adversarial Sample Generation



(a) (b) (c) (d) (e)

Figure 1: (a) Create an adversarial sample $\tilde{x}$ by PGD (Eq. (2)): a solid blue arrow shows one PGD iteration to update $\tilde{x}$; a dashed blue arrow shows that when $\tilde{x}$ moves out of the ϵ-ball, it will be projected back onto the ϵ-ball; the solid red arrow denotes the generated adversarial noise from the final iteration. (b): Adding adversarial noise to a sample/data point x is to push it towards a direction in the input space. Once the data point crosses the decision boundary, the prediction from a DNN model is changed. The closer x is to a decision boundary, the more likely it is pushed across the decision boundary. (c) Given a training set (X, Y), where X denotes the set of samples and Y denotes the set of true labels, ideally, adversarial training samples $\tilde{X}$ should be placed closer to the true/optimal decision boundary (Fig. 1 (b)). By training the model with both (X, Y) and $(\tilde{X}, Y)$, the model decision boundary is pushed away from X, and therefore adversarial robustness is improved. (d) However, the distribution of training samples is not always as ideal as that in (c). The margin of each training sample is different. A uniform and fixed adversarial training perturbation can lead to adversarial training samples across the true decision boundary. A model trained on these adversarial samples will have low clean accuracy. (e) The optimal adversarial training samples $\tilde{X}$ should be just about to go across the decision boundary. Training with too small noise $||\tilde{X} - X||_p$ is not effective enough. Training with too large noise $||\tilde{X} - X||_p$ leads to large accuracy degradation.

Without loss of generality, we use the most widely studied untargeted adversarial noise as an example. Let (x, y) be a pair of a sample x and its true label y. The objective function of generating an adversarial sample is:

$$\max_{\tilde{x} \in \{x' | \ \|x-x'\|_p \leq \epsilon\}} Loss(f(\tilde{x}), y) \tag{1}$$

where $Loss(.)$ is the loss function, $\tilde{x}$ is the adversarial sample to be generated, $f(.)$ is a DNN model, ϵ is noise level and $\|.\|_p$ is L_p vector norm.

Given a clean sample (x, y), there are many ways to solve Eq. (1) to obtain an adversarial sample $\tilde{x}$. One popular method is called Projected Gradient Descent (PGD) (Madry et al., 2018; Kurakin et al., 2018), which leverages an iterative way to approximate the optimal $\tilde{x}$:

$$x^{(k)} \leftarrow \Pi_\epsilon(\alpha \cdot h(\frac{\partial}{\partial x^{(k-1)}} L(f(x^{(k-1)}), y)) + x^{(k-1)}) \tag{2}$$

where $h(.)$ is the normalization function, α is the step size, $x^{(k)}$ is the adversarial sample at the iteration k, and $\Pi_\epsilon(.)$ is a projection operation to ensure the generated perturbation to be within an ϵ-ball, i.e., $\|x^{(k)} - x\|_p \leq \epsilon$. After K iterations (called K-PGD), we obtain the generated adversarial sample $\tilde{x}$=$x^{(K)}$. This process is intuitively shown in Fig. 1 (a).

2.3 Adversarial Training (AT)

The objective function of the vanilla Adversarial Training (AT) (Madry et al., 2018; Kurakin et al., 2018) is:

$$min_\theta \ Loss(f_\theta(\tilde{x}), y) \tag{3}$$

where $f_\theta(.)$ is a DNN model with the parameter θ. The objective function is under the constraint:

$$\tilde{x} = argmax_{\tilde{x} \in \{x' | \ \|x-x'\|_p \leq \epsilon_{train}\}} \ Loss(f_\theta(\tilde{x}), y) \tag{4}$$

The AT uses noisy sample $\{\tilde{x}, y\}$ to train the DNN model f_θ. In this way, the DNN model gains robustness against adversarial noise. For AT, adversarial training samples are generated by using the PGD method (Eq. (2)) with a fixed noise level ϵ_{train} for every $\{x, y\}$. The fixed and uniform noise level ϵ_{train} is the weak point of AT, which will be discussed in the next section.

2.4 What is the Optimal Location of an Adversarial Sample for Training?

From Fig. 1 (b) (e), the optimal location of an adversarial training sample $\tilde{x}$ should be very close to the decision boundary (Fig. 1 (e)). And the optimal noise level ϵ_{train} should be the distance between $\tilde{x}$ and the x. This distance is similar to the "margin" in Support Vector Machine (SVM) (Cortes & Vapnik, 1995). So, we will call this optimal noise level "margin" in this paper. The real margin should be the shortest distance between x and the corresponding decision boundary d. This means for a true optimal $\tilde{x}$ that is about to cross the decision boundary, and $\tilde{x} - x \perp d$ holds. However, for DNN models, such a true optimal $\tilde{x}$ can hardly be located precisely in the input space. The estimated margin may be much larger than the real margin. This is an inevitable overestimation problem, and it exists in all adversarial training methods, such as MMA (Ding et al., 2019a) (see Fig. 3).

2.5 MMA and FAT

We have already assumed that the optimal location of the adversarial sample for training is very close to the decision boundary, and the margin should be the optimal noise level for training. How to estimate the margin for each sample? From the margin perspective described in Section 2.4, we can explain two well-known methods: FAT (Zhang et al., 2020a) and MMA (Ding et al., 2019a). FAT uses PGD with an early-stop strategy to generate adversarial samples for training. From the margin perspective, the adversarial sample generated by FAT (when $\tau = 0$) is $\tilde{x}_1$ in Fig. 1 (a). The problem is the estimated margins from FAT are smaller than the real margins, and this is why FAT always has high clean accuracy but low noisy accuracy (Section 5). MMA uses PGD with a large noise level ϵ_{train} to locate $\tilde{x}$ and $\tilde{x}_1$ in Fig. 1 (a), and do exploration between these two points to approximate the real margin. Because of this large noise level for exploration, MMA is very likely to overestimate the sample margins (see Fig. 3).

3 METHOD

3.1 THE GRADUAL EXPANSION MECHANISM AND THE OPTIMAL STATE

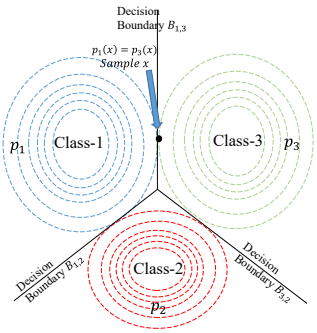


Figure 2: The gradual expansion mechanism (GEM) illustration: in the training process of SMD, the sample-wise noise levels $\mathcal{E}(x)$ expand gradually, until the optimal state is reached (Section 3.1).

Motivated by 2.2, 2.3, 2.4, and 2.5, instead of using a fixed large noise level ϵ_{train} to generate adversarial samples, gradually expanding the noise level for an individual sample is a better strategy (see Fig. 2). We call this Gradual Expansion Mechanism (GEM). GEM not only can largely avoid the margin overestimation problem (Section 2.5) but also has one important property: with GEM, an optimal state in the input data space can be reached, which means that the noise level ϵ_{train} no longer needs to be tuned manually. Under the optimal state, the overall loss is minimized and the adversarial training samples are placed near the decision boundaries.

Theorem 3.1 *If the spatial distributions of the generated adversarial samples for different classes are the same, an optimal state (Fig. 2) in the input data space exists.*

Without loss of generality, we assume there are three classes and three decision boundaries between classes (Fig. 2 (b)). The softmax output of the neural network model $f(.)$ has three components: p_1, p_2 and p_3 corresponding to the three classes. If a data point $\tilde{x}$ is about to cross the decision boundary B_{ij} between class-i (c_i) and class-j (c_j) (i.e., $\tilde{x}$ is on the class-y side of the decision boundary), then $p_i(\tilde{x}) = p_j(\tilde{x})$. The mathematical expectation of the cross-entropy loss of the generated adversarial training samples (i.e., when $\tilde{x}$ is correctly classified) is:

$$E = \mathbf{E}_{\tilde{x}\in c_1}\left(-log\left(p_1\left(\tilde{x}\right)\right)\right) + \mathbf{E}_{\tilde{x}\in c_2}\left(-log\left(p_2\left(\tilde{x}\right)\right)\right) + \mathbf{E}_{\tilde{x}\in c_3}\left(-log\left(p_3\left(\tilde{x}\right)\right)\right) \tag{5}$$

The GEM pushes the generated adversarial training samples towards/onto the decision boundaries. Then, the adversarial samples $\tilde{x} \in c_i$ are split into two parts: those pushed to B_{ij} and denoted by $\tilde{x} \in c_i, B_{ij}$, and those pushed to B_{ik} and denoted by $\tilde{x} \in c_i, B_{ik}$. So, we can get:

$$\mathbf{E}_{\tilde{x}\in c_1}\left(-log\left(p_1\left(\tilde{x}\right)\right)\right) = \mathbf{E}_{\tilde{x}\in c_1, B_{12}}\left(-log\left(p_1\left(\tilde{x}\right)\right)\right) + \mathbf{E}_{\tilde{x}\in c_1, B_{13}}\left(-log\left(p_1\left(\tilde{x}\right)\right)\right) \tag{6}$$

$$\mathbf{E}_{\tilde{x}\in c_2}\left(-log\left(p_2\left(\tilde{x}\right)\right)\right) = \mathbf{E}_{\tilde{x}\in c_2, B_{12}}\left(-log\left(p_2\left(\tilde{x}\right)\right)\right) + \mathbf{E}_{\tilde{x}\in c_2, B_{23}}\left(-log\left(p_2\left(\tilde{x}\right)\right)\right) \tag{7}$$

$$\mathbf{E}_{\tilde{x}\in c_3}\left(-log\left(p_3\left(\tilde{x}\right)\right)\right) = \mathbf{E}_{\tilde{x}\in c_3, B_{13}}\left(-log\left(p_3\left(\tilde{x}\right)\right)\right) + \mathbf{E}_{\tilde{x}\in c_3, B_{23}}\left(-log\left(p_3\left(\tilde{x}\right)\right)\right) \tag{8}$$

If the generated adversarial training samples (random variables) $\tilde{x} \in c_i$ and $\tilde{x} \in c_j$ have the same spatial distribution on the decision boundary B_{ij} between the two classes, then:

$$\begin{aligned} E_{\tilde{x}\in c_i, B_{ij}}(-log(p_i(\tilde{x}))) + E_{\tilde{x}\in c_j, B_{ij}}(-log(p_j(\tilde{x}))) &= \mathbf{E}_{\tilde{x}\in\ B_{ij}}\left(-log\left(p_i\left(\tilde{x}\right)\right) - log\left(p_j\left(\tilde{x}\right)\right)\right) \\ &= \mathbf{E}_{\tilde{x}\in\ B_{ij}}\left(-log\left(p_i\left(\tilde{x}\right)p_j\left(\tilde{x}\right)\right)\right) \\ &\geq \mathbf{E}_{\tilde{x}\in\ B_{ij}}(-log(\frac{p_i\left(\tilde{x}\right) + p_j\left(\tilde{x}\right)}{2})^2) \end{aligned} \tag{9}$$

Therefore, E reaches the minimum when $p_i\left(\tilde{x}\right) = p_j\left(\tilde{x}\right)$, namely, when $\tilde{x}$ is about to cross the decision boundary.

In the above analysis, the loss on clean training samples is ignored because, in experiments, we find out that the loss on clean training samples converges much faster than the loss on noisy samples.

We have shown that an optimal state can be achieved when the generated adversarial training samples have the same spatial distribution on the (final) decision boundary. Here, we outline what will

happen if the initial spatial distributions of the generated adversarial training samples in different classes are not the same on the current decision boundary. We note that our SMD method will actively generate and push the adversarial training samples towards ("close to" due to numerical precision) the current decision boundary of the neural network model. Model training is a dynamic process to adjust the decision boundary of the model. Let's focus on the following two terms.

$$F_i \triangleq \mathbf{E}_{\tilde{x}\in\ c_i, B_{ij}} = -\int q_i(\tilde{x}) log(p_i(\tilde{x}))d\tilde{x} \tag{10}$$

$$F_j \triangleq \mathbf{E}_{\tilde{x}\in\ c_j, B_{ij}} = -\int q_j(\tilde{x}) log(p_j(\tilde{x}))d\tilde{x} \tag{11}$$

where $q_i(x)$ and $q_j(x)$ are the distributions (i.e., densities) of the adversarial/noisy training samples on the current decision boundary between the two classes, and $q_i(x)$ and $q_j(x)$ may not be equal to each other. In fact, F_i and F_j can be interpreted as two forces that try to expand the margins of the samples in the two classes against each other. By dividing the decision boundary into small regions (i.e., linear segments), the two integrals can be evaluated in the individual regions. In a region, if $q_i(\tilde{x}) > q_j(\tilde{x})$ (i.e., more noisy samples are generated in class-i) then the current state is not optimal: after training the model using the noisy samples, most of the noisy samples in class-i will be correctly classified and most of the noisy samples in class-j will be incorrectly-classified (this is a simple result of classification with imbalanced data in the region), which means the decision boundary will shift towards the samples in class-j, and therefore the margins of the corresponding samples in class-i will expand and the margins of the corresponding samples in class-j will shrink. Thus, the decision boundary may shift locally towards the samples in one of the classes. Obviously, the decision boundary will stop shifting when the local densities of noisy samples in different classes are the same along the decision boundary, i.e., $q_i(\tilde{x})$ becomes equal to $q_j(\tilde{x})$, which means an optimal state is reached. We defer a math-rigor analysis of this dynamic process to our future work.

The above analysis implies that GEM finally pushes the noisy samples close to the decision boundaries, which means that the noise level ϵ_{train} for generating adversarial training samples no longer needs to be tuned manually.

3.2 Self-adaptive Margin Defense (SMD)

In this section, we provide the details of SMD, which is parameter-free for the user.

3.2.1 Self-adaptive Step Size for margin expansion

In SMD, the training noise level (i.e., sample-wise margin), $\mathcal{E}(x)$, is individualized for each sample. During training, the training noise level for a sample will gradually expand, and this poses a challenge to the optimizer: it needs to catch up with a loss function that will change its value due to the update of the adversarial samples for training. While pushing an adversarial training sample $\tilde{x}$ away from the clean sample x during training, the terrain of the loss function may vary rapidly. When $\tilde{x}$ is on flat terrain, it may take a few iterations for the optimizer to adjust the model parameters. When $\tilde{x}$ is on steep terrain, the optimizer needs much more iterations to adjust the model parameters. If the optimizer uses the same number of iterations, then the step size for margin expansion must not be a constant and should be adjusted adaptively according to each adversarial sample and the terrain.

Initial step size: Let x be a sample in the input data space. Then the initial step size Δ_ϵ is:

$$\Delta_\epsilon = ceil((h(x)/N)/h(x/255)) \cdot h(x/255) \tag{12}$$

where $h(.)$ is the vector Lp-norm function, $ceil(.)$ gives the smallest nearest integer no smaller than the input, and N is the number of training epochs. The idea is that in the training process with the step size Δ_ϵ, most of the input space should be covered. For example, when L-inf norm is used, the number of epochs is 150, and the pixel value ranges from 0 to 1, then $h(x)$ = 1, and Δ_ϵ = $ceil((1/150)/(1/255)) \times (1/255)$ = 2/255.

To handle the problem of tracking a moving target/loss by the optimizer, we use two types of information to adjust the step size: the noise level and the gradient.

Step size adjusted by using the noise level: The expansion step size Δ_ϵ should be decayed such that a smaller step should be used for a larger sample margin. $\mathcal{E}(x)$ is the current training noise level for a clean sample x. Then the decay rate corresponding to the clean sample x is:

$$\gamma_1(x) = \rho^{floor(\mathcal{E}(x)/h(x/255))} \tag{13}$$

The larger $\mathcal{E}(x)$ is, the smaller $\gamma_1(x)$ is. ρ is derived from a coarse estimation of the margin expansion process (see Appendix D).

Step size adjusted by using the gradient information: We also let the step size Δ_ϵ adjustment be gradient-aware. Let $\tilde{x}$ be the generated adversarial sample from the clean sample x, $f(.)$ be the model, y be the ground truth, and $Loss(.)$ be the loss function for generating adversarial samples. Then the gradient-aware decay rate for the clean sample x is:

$$\gamma_2(x) = 1/(1 + h(abs(\frac{\partial Loss(f(\tilde{x}), y)}{\partial \tilde{x}}))) \tag{14}$$

where $abs(.)$ gives absolute value. The idea is shown in Fig. 4. $\gamma_2(x)$ ranges from 0 to 1: when the gradient at $\tilde{x}$ is 0, $\gamma_2(x)$ is 1, which means no decay; when the gradient at $\tilde{x}$ is ∞, $\gamma_2(x)$ is 0.

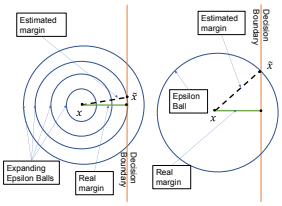


Figure 3: Left: SMD's estimation of margin is closer to the real margin (the green line); Right: Without the gradual expansion mechanism (e.g., MMA), PGD can be used to estimate the margin directly, which can be much larger than the real margin.

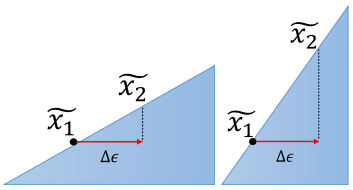


Figure 4: $\tilde{x}_1$ is an adversarial sample in the current iteration. After one step Δ_ϵ expansion, $\tilde{x}_1$ will be "pushed" to $\tilde{x}_2$. If the gradient at $\tilde{x}_1$ is large (right), the loss is significantly changed, which will create a heavier load for the optimizer to adjust model parameters to fit the new data. In this case, the step size Δ_ϵ should be reduced to let the optimizer/model keep up with the loss change.

3.2.2 ALGORITHM

One epoch of the SMD training process is shown in Algorithm 1 (Python-like pseudo-code), which includes two sub-processes: (1) Compute the loss and update the DNN model (Line 2 to 10); (2) Update the sample-wise margin estimation (Line 11 to 12). Algorithm 2 generates the noisy samples for training. Line 3 to 7 explore and record the indexes of noisy samples that cross the decision boundaries. Line 8 to 15 tries to approximate the margins closer to the decision boundary with a binary search (Unlike FAT which just uses $\tilde{X}_1$ in Fig. 1 (a) as the noisy training samples). Note: X_{out} are guaranteed to be close to but not to cross the decision boundary. More detailed explanation is in Appendix A.

Algorithm 1 SMD Training in One Epoch

Input: the training set S; the DNN model $f(.)$; $g(.)$ is the function that transforms the output of $f(.)$ to a predicted class label, e.g., $argmax$; $Loss$ is the loss function for training the model f; $\mathcal{E}$ is the array of the estimated sample margins: $\mathcal{E}(i)$ is the margin of the sample indexed by the unique ID i, every $\mathcal{E}(i)$ is initialized to be Δ_ε.
Parameters: Δ_ε is the expansion step size given by Eq. (12) and adjusted by Eq. (13) and Eq. (14)
Output: Updated model f after this training epoch
Process:

1: **for** each batch of training samples (X, Y) with ID Ids in S **do**
2: Run the model f on clean samples: $Z \leftarrow f(X)$
3: $L_0 \leftarrow Loss(Z,\ Y)$
4: $Flag1 \leftarrow [g(Z) == Y]$ $Flag1$ records the indexes of correctly classified clean samples
5: Generate noisy samples using the **Algorithm 2**: $\tilde{X}, Flag2 \leftarrow EPGD(X[Flag1], Y[Flag1], \mathcal{E}[Flag1], f, Loss)$
6: Use $\tilde{X}$, $f(.)$, $\mathcal{E}$ and Y to compute γ_1 and γ_2 (Eq. (13) and Eq. (14))
7: Run the model f on noisy samples: $\tilde{Z} \leftarrow f(\tilde{X})$
8: $L_1 \leftarrow Loss(\tilde{Z},\ Y[Flag1])$
9: $L \leftarrow (L_0 + L_1)/2$
10: Back-propagate from L and update the model $f(.)$
11: $\mathcal{E}[Flag1\ \ \&\ \ Flag2] \leftarrow \mathcal{E}[Flag1\ \ \&\ \ Flag2] + \Delta_\varepsilon \cdot (\gamma_1 \cdot \gamma_2)[Flag1\ \ \&\ \ Flag2]$
12: $\mathcal{E}[\sim Flag1$ **or** $\sim Flag2] \leftarrow ||X[\sim Flag1$ **or** $\sim Flag2] - \tilde{X}[\sim Flag2]||_p$
13: **end for**

Note: This algorithm is implemented in mini-batches. $||.||_p$ denotes vector Lp norm. $\sim$ is "not in".

Algorithm 2 (Exploration-PGD): generate noisy samples

Input: training sample batch (X, Y); the estimated margins $\mathcal{E}$, currently; the DNN model $f(.)$; $g(.)$ is the function that transforms the output of $f(.)$ to a predicted class label, e.g., $argmax$; the Loss function $L(.)$.
Parameters: maximum PGD iteration number $K \leftarrow 20$; PGD step size $\alpha \leftarrow (4 \times \varepsilon)/K$ for each ε in $\mathcal{E}$
Output: the generated noisy sample batch $\tilde{X}$, the $Flag2$ that records the indexes of correctly classified noisy/adversarial samples;
Function EPGD($X, Y, \mathcal{E}, f, L(.)$):

$X_{out} \leftarrow X, Counter \leftarrow [0, ..., 0]$ (length is the batch size)
while $K > 0$ **do**
  $\tilde{X} \leftarrow \Pi_{\mathcal{E}}(\alpha \cdot h(\nabla_X L(f(X), Y)) + X)$
  $X \leftarrow \tilde{X}$
  $Counter[g(f(X)) \neq Y] \leftarrow 1$
  $K \leftarrow K - 1$
end while
$N \leftarrow 10$
$\tilde{X} \leftarrow (\tilde{X} + X_{out})/2$
while $N > 0$ **do**
  $\tilde{Y} = f(\tilde{X})$
  $X_{out}[\tilde{Y} == Y] \leftarrow \tilde{X}[\tilde{Y} == Y]$
  $\tilde{X} \leftarrow (\tilde{X} + X_{out})/2$
  $N \leftarrow N - 1$
end while
$Flag2 = (Counter == 0)$
return $X_{out}, Flag2$

Note: This algorithm is implemented in mini-batches. $h(.)$ is the normalization function. $||.||_p$ denotes vector Lp norm. N is always set to 10, which is constant. So, this binary search will not enlarge the time complexity of the whole algorithm. $\Pi_{\mathcal{E}}(.)$ is explaned in Eq.(2).

3.3 Comparing with Related Work

TRADES (Zhang et al., 2019) optimizes a loss with a KL divergence-based regularization term to consider both adversarial robustness and clean accuracy. MART (Wang et al., 2019b) uses a similar KL divergence-based regularization term for robustness training. DAT (Wang et al., 2019a) applies convergence quality as the criterion for generating noisy training samples. CAT (Cai et al., 2018) uses adversarial training perturbations of different strengths for every training sample. GAIRAT (Zhang et al., 2020b) applies sample-wise weights in the loss, but the weights cannot prevent adversarial training samples from crossing the decision boundary. So, GAIRAT is very likely to use too-large noise levels for training, and as shown in the experiments (Section 5), GAIRAT's clean accuracy is always lower than SMD's. Obviously, SMD is different from these methods. The FAT method (Zhang et al., 2020a) applies early-stop PGD to generate adversarial training samples with a parameter τ. When $\tau = 0$, the generated adversarial training sample is $\tilde{x}_1$ in Fig. 1, which leads to lower noisy accuracy (Section 2.5); when $\tau > 0$, the generated noisy training samples will go across the decision boundary, which may hurt the model's clean accuracy and therefore is against the idea of SMD. The IAAT method (Balaji et al., 2019) uses a sample-wise noise level to train a model, but the sample-wise noise level is based on heuristics, which may not be optimal; and it uses fixed step size and ϵ_{train}. The MMA method (Ding et al., 2019a) is based on sample-wise margins estimated by PGD (Madry et al., 2018) with a large noise level, which leads to the margin overestimation problem (mentioned in 2.5). Also, as illustrated in Fig. 3, a fixed noise level may lead to margin overestimation for one more reason. With the GEM, the sample margins estimated from SMD can be much closer to the real margin, as illustrated in Fig. 3. So, SMD is different from MMA. In addition, the most significant difference is that SMD is parameter-free for the user.

4 Experiment Configuration

Experiment settings of SVHN and Fashion-MNIST evaluation and more details of the CIFAR10 evaluation are shown in Appendix B. Additional Supportive Results for GEM are in Appendix C

CIFAR10 (Krizhevsky et al., 2009) is used for this evaluation. We train WideResNet-28-4 described in (Zagoruyko & Komodakis, 2016) for this experiment. The data preprocessing is the same as that in (Ding et al., 2019a).

For our SMD: SMD has no hyperparameters to tune. **For TRADES (Zhang et al., 2019):** we use the official code from (Zhang et al., 2019), and the settings are strictly in accordance with (Zhang et al., 2019), except that β and ϵ_{train} are varied in the experiment. **For MMA (Ding et al., 2019a):**

we use the trained MMA models provided officially by (Ding et al., 2019a), and these three models have different ϵ_{train}: 12/255, 20/255, and 32/255. **For FAT (Zhang et al., 2020a):** we use the official code and all settings are the same as those in (Zhang et al., 2020a), except for three different ϵ_{train}: 4/255, 8/255 (endorsed in (Zhang et al., 2020a)) and 12/255. **For GAIRAT(Zhang et al., 2020b):** we use the official code, and the settings are the same as those in (Zhang et al., 2020b), except for three different ϵ_{train}: 4/255, 8/255 (endorsed in (Zhang et al., 2020b)) and 12/255. **For AT(Madry et al., 2018):** three noise levels are used: 4/255, 8/255 (endorsed in (Madry et al., 2018)) and 12/255; the max number of PGD-iterations for training is 20 (endorsed in (Madry et al., 2018), another weaker endorsed choice is 7), and the other settings are the same as those in (Madry et al., 2018).

Settings for attacks: AutoAttack (AA) (Croce & Hein, 2020) is the most reliable attack method currently, because: (1) AA is parameter-free; (2) AA consists of four attacks: AutoPGD (a white-box untargeted attack, stronger than PGD), APGD-t (a white-box targeted attack), FAB-t (a white-box targeted attack), and Square (a black-box attack), which means AA can provide a more sufficient evaluation of the robustness than an individual adversarial attack. **We consider the AA result as the major evaluation in this paper**. All the results of AA are the average value from three runs with different random seeds. Code is from (Croce & Hein, 2020). In addition, we provide two more attacks for evaluation: **PGD with Cross-entropy (CE) and CW loss(Carlini & Wagner, 2017) (CE-CW-PGD)** (this is named as "only white-box attack" in (Ding et al., 2019a)) runs for 100 iterations, the results are from 10 runs: half with CE-PGD and half with CW-PGD; **PGD with Cross-entropy loss (CE-PGD)** runs for 100 iterations, and the results are the average value from five runs with different random seeds. Code is from (Ding et al., 2019b). Results of CE-PGD and CE-CW-PGD show that these two attacks are insufficient, compared with AA, and thus give a wrong impression of robustness, supported by (Croce & Hein, 2020).

5 Results and Discussion

Table 1: Results of CIFAR10 Evaluation: (1) N. L. denotes "Noise Level" for adversarial attacks (2/255, 4/255, and 8/255 are the most popular evaluation noise levels measured by L-inf norm). (2) Method with "*" denotes the settings (ϵ_{train} or β value) endorsed in the corresponding paper. (3) "STD" denotes the clean model; "ATϵ_{train}" denotes adversarial training (Madry et al., 2018) with noise level ϵ_{train}; "FATϵ_{train}" denotes FAT (Zhang et al., 2020a) with noise level ϵ_{train};"GAIϵ_{train}" denotes GAIRAT (Zhang et al., 2020b) with noise level ϵ_{train}; "TR-bβ-ϵ_{train} denotes TRADES(Zhang et al., 2019) with trade-off factor β and noise level ϵ_{train}; "MMAϵ_{train}" denotes MMA (Ding et al., 2019a) with noise level ϵ_{train} (4) Noise level is measured by L-inf norm. (5) The largest value in each column is bolded.

(a) CE-PGD

N. L.	0	2/255	4/255	8/255	Avg.
STD	94.92	0	0	0	23.73
AT4	88.97	78.52	64.78	34.83	66.78
AT8*	85.31	77.47	67.78	47.52	69.52
AT12	77.83	72.19	65.92	51.39	66.83
FAT4	**90.44**	79.00	62.66	30.92	65.75
FAT8*	88.15	79.11	66.83	40.37	68.61
FAT16	86.89	77.31	65.44	40.15	67.45
GAI4	88.2	79.24	66.13	36.39	67.49
GAI8*	80.96	75.04	67.93	49.83	68.44
GAI16	77.96	72.77	66.17	50.50	66.85
TR-b1-8*	87.42	79.09	**69.65**	44.91	70.27
TR-b6-8*	84.28	76.07	69.54	**51.51**	70.35
TR-b6-16	73.22	68.19	62.56	51.13	63.78
TR-b6-24	72.23	66.93	61.40	49.57	62.53
MMA12*	88.59	**79.43**	68.18	44.31	70.12
MMA20*	86.56	77.36	67.32	49.95	70.30
MMA32*	84.36	74.42	65.63	51.21	68.90
SMD	87.98	79.07	68.05	46.36	**70.37**

(b) CE-CW-PGD

N. L.	0	2/255	4/255	8/255	Avg.
STD	94.92	0	0	0	23.73
AT4	88.97	78.31	64.66	34.48	66.61
AT8*	85.31	76.77	66.70	46.48	68.81
AT12	77.83	71.11	63.76	48.21	65.23
FAT4	**90.44**	78.93	62.61	30.48	65.62
FAT8*	88.15	78.42	65.97	39.67	68.05
FAT16	86.89	76.73	64.80	39.70	67.03
GAI4	88.20	75.73	58.52	27.76	62.55
GAI8*	80.96	68.58	54.20	27.94	57.92
GAI16	77.96	65.75	52.33	28.95	56.25
TR-b1-8*	87.42	**79.55**	68.72	43.46	69.79
TR-b6-8*	84.28	75.91	**69.43**	**51.04**	70.17
TR-b6-16	73.22	66.87	59.85	46.65	61.65
TR-b6-24	72.23	65.67	58.91	45.20	60.50
MMA12*	88.59	79.39	67.99	43.62	69.90
MMA20*	86.56	77.28	67.07	48.47	69.85
MMA32*	84.36	74.43	65.25	50.20	68.56
SMD	87.98	78.95	68.02	45.93	**70.22**

(c) AutoAttack

N. L.	0	2/255	4/255	8/255	Avg.
STD	94.92	0	0	0	23.73
AT4	88.97	78.26	64.48	33.43	66.29
AT8*	85.31	76.62	66.34	43.42	67.92
AT12	77.83	51.39	63.33	46.98	59.88
FAT4	**90.44**	78.91	62.28	28.36	65.00
FAT8*	88.15	78.33	65.63	38.11	67.55
FAT16	86.89	76.59	64.53	38.50	66.63
GAI4	88.20	75.52	57.81	25.46	61.75
GAI8*	80.96	68.36	53.40	25.97	57.17
GAI16	77.96	65.53	51.48	27.26	55.56
TR-b1-8*	87.42	**79.52**	**68.45**	43.23	69.66
TR-b6-8*	84.28	75.85	67.60	**47.25**	68.75
TR-b6-16	73.22	66.77	59.66	45.92	61.39
TR-b6-24	72.23	65.63	58.60	44.41	60.22
MMA12*	88.59	79.30	67.47	41.13	69.12
MMA20*	86.56	77.14	65.70	42.58	67.99
MMA32*	84.36	74.12	63.38	41.49	65.84
SMD	87.98	78.90	67.87	44.35	**69.78**

5.1 AutoAttack v.s. CE-PGD and CE-CW-PGD

Among the three attacks on CIFAR10 in Table 1a, 1b and 1c, CE-PGD and CE-CW-PGD are insufficient and thus give a wrong impression of robustness(Croce & Hein, 2020). AutoAttack is the strongest and considered the major evaluation in this paper.

Table 2: Results of SVHN and Fashion-MNIST Evaluation:

(a) AutoAttack on SVHN

N. L.	0	2/255	4/255	8/255	Avg.
STD	92.57	35.95	8.16	0.12	34.20
AT8*	88.77	80.01	**71.60**	45.31	71.42
AT4	91.44	80.95	66.28	33.41	68.02
FAT8*	91.26	80.88	67.88	39.92	69.99
FAT4	**91.86**	80.01	63.77	31.92	66.89
GAI8*	81.28	67.17	51.63	26.03	56.53
GAI4	89.31	76.37	59.60	28.18	63.37
TR-b6-8*	86.21	80.20	70.28	**49.57**	71.57
TR-b6-4	91.63	80.51	70.59	43.29	71.51
MMA12*	90.55	**81.28**	67.77	39.96	69.89
MMA20*	89.99	80.64	67.84	42.30	70.19
SMD	90.10	80.23	69.15	47.54	**71.76**

(b) AutoAttack on Fashion-MNIST

N. L.	0	2/255	4/255	8/255	Avg.
STD	92.04	78.47	60.58	29.71	65.20
AT8*	91.76	89.01	85.73	77.69	86.05
AT4	**91.79**	89.01	85.72	77.63	86.03
FAT8*	91.63	88.56	84.95	75.83	85.24
FAT4	91.82	87.73	82.56	68.61	82.68
GAI8*	90.68	88.01	84.44	75.27	84.60
GAI4	91.57	87.66	83.13	71.32	83.42
TR-b6-8*	90.45	89.13	**87.35**	**83.51**	87.61
TR-b6-4	**91.79**	88.93	85.26	76.42	85.60
MMA12*	91.77	88.93	85.79	78.08	86.14
MMA20*	90.74	88.83	86.61	81.07	86.81
SMD	90.86	**89.14**	87.30	83.38	**87.67**

(1) For GAI8 and GAI16, the differences between the CE-PGD result and the CE-CW-PGD result (at noise level 8/255) are more than 20% (much larger than other methods). This suggests the GAIRAT-trained models are particularly robust to CE-PGD attack but are very vulnerable to CW attack. Such a significant difference ($>$ 2%) between CE-PGD and CE-CW-PGD also occurs in GAI4 (9%), AT12 (3%), TR-b6-16 (5%), and TR-b6-24 (5%).

(2) For MMA20 and MMA32, the differences between the CE-CW-PGD result and the AutoAttack result (at noise level 8/255) are significant (9% for MMA32 and 6% for MMA20, much larger than other methods). This suggests the MMA-trained models are very vulnerable to AutoAttack. Such a significant difference ($>$2%) between CE-CW-PGD and AutoAttack also occurs in AT8 (3%), FAT4 (2.2%), GAI4 (2.3%), TR-b6-8 (4%), and MMA12 (2.5%).

(3) AutoAttack consists of one white-box untargeted attack, two white-box targeted attacks, and one black-box attack. The major differences between the CE-CW-PGD result and the AutoAttack result are caused by these two white-box targeted attacks (8% for MMA32). This suggests that a method robust to white-box untargeted attacks cannot guarantee to be robust to white-box targeted attacks.

5.2 Performance

(1) SMD has the best overall performance for all evaluations in Table 1a, 1b, 1c, 2a and 2b. Other methods' performances are sensitive to hyperparameters (e.g., ϵ_{train}) on these datasets: when ϵ_{train} is large, clean accuracy degradation is large; when ϵ_{train} is small, noisy accuracy is low.

(2) For strong AutoAttack in Table 1c, 2a and 2b: The clean accuracy of SMD is lower than those of FAT4, FAT8, AT4, GAI4, TR-b6-4 and MMA12, all of which have much lower noisy accuracy at 8/255 than SMD. The noisy accuracy of SMD at 8/255 is lower than those of AT12, TR-b6-16, and TR-b6-8, which have large clean accuracy degradation ($>$ 10% in CIFAR10 and TR-b6-8 has the largest accuracy degradation on SVHN and Fashion-MNIST). SMD seldom has the best result in one single column, but always has the best overall performance. It is the feature of SMD to keep a balance between robustness and accuracy, not to mention it is parameter-free for the user.

6 Conclusion

In this paper, based on the Gradual Expansion Mechanism and the theory of the optimal state, we designed a new adversarial training method, SMD, parameter-free for the user. Most defense methods need to fine-tune ϵ_{train}, but SMD does not have such a problem, which is good for advocating adversarial robustness techniques to application domains. Experiments show SMD outperforms other competing methods on three public datasets. We hope our work may pave the way for developing robust applications, e.g., in the medical field which usually has large datasets and therefore prefers parameter-free methods.

REFERENCES

Yogesh Balaji, Tom Goldstein, et al. Instance adaptive adversarial training: Improved accuracy tradeoffs in neural nets. *preprint arXiv:1910.08051*, 2019.

Qi-Zhi Cai, Chang Liu, et al. Curriculum adversarial training. In *IJCAI*, 2018.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *IEEE SP*, 2017.

Minhao Cheng, Qi Lei, et al. Cat: Customized adversarial training for improved robustness. *preprint arXiv:2002.06789*, 2020.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 1995.

Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *ICML*, pp. 2206–2216. PMLR, 2020.

Gavin Weiguang Ding, Yash Sharma, et al. Mma training: Direct input space margin maximization through adversarial training. In *ICLR*, 2019a.

Gavin Weiguang Ding, Luyu Wang, et al. Advertorch v0. 1: An adversarial robustness toolbox based on pytorch. *preprint arXiv:1902.07623*, 2019b.

Ian Goodfellow, Jonathon Shlens, et al. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

Xiao Han, Rasul Kashif, and Vollgraf Roland. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Alexey Kurakin, Ian Goodfellow, et al. Adversarial examples in the physical world. In *Artificial intelligence safety and security*. 2018.

Aleksander Madry, Aleksandar Makelov, et al. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018.

Yuval Netzer, Tao Wang, et al. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

Adam Paszke, Sam Gross, et al. Automatic differentiation in pytorch. 2017.

Leslie Rice, Eric Wong, and Zico Kolter. Overfitting in adversarially robust deep learning. In *ICML*, pp. 8093–8104. PMLR, 2020.

Chawin Sitawarin, Supriyo Chakraborty, et al. Improving adversarial robustness through progressive hardening. 2020.

Christian Szegedy, Wojciech Zaremba, et al. Intriguing properties of neural networks. In *ICLR*, 2014.

Yisen Wang, Xingjun Ma, et al. On the convergence and robustness of adversarial training. In *ICML*, 2019a.

Yisen Wang, Difan Zou, et al. Improving adversarial robustness requires revisiting misclassified examples. In *ICLR*, 2019b.

Chaojian Yu, Bo Han, Li Shen, Jun Yu, Chen Gong, Mingming Gong, and Tongliang Liu. Understanding robust overfitting of adversarial training and beyond. In *ICML*, pp. 25595–25610. PMLR, 2022.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Hongyang Zhang, Yaodong Yu, and othersl. Theoretically principled trade-off between robustness and accuracy. In *ICML*, 2019.

Jingfeng Zhang, Xilie Xu, et al. Attacks which do not kill training make adversarial learning stronger. In *ICML*, 2020a.

Jingfeng Zhang, Jianing Zhu, et al. Geometry-aware instance-reweighted adversarial training. In *ICLR*, 2020b.

A ALGORITHMS DETAILED EXPLANATIONS

Algorithm 3 SMD Training in One Epoch

Input: the training set S; the DNN model $f(.)$; $g(.)$ is the function that transforms the output of $f(.)$ to a predicted class label, e.g., $argmax$; $Loss$ is the loss function for training the model f; $\mathcal{E}$ is the array of the estimated sample margins: $\mathcal{E}(i)$ is the margin of the sample indexed by the unique ID i, every $\mathcal{E}(i)$ is initialized to be Δ_ε.
Parameters: Δ_ε is the expansion step size given by Eq. (12) and adjusted by Eq. (13) and Eq. (14)
Output: Updated model f after this training epoch
Process:

for each batch of training samples (X, Y) with ID Ids in S **do**
  Run the model f on clean sample: $Z \leftarrow f(X)$
  $L_0 \leftarrow Loss(Z,\ Y)$
  $Flag1 \leftarrow [g(Z) = Y]$ $Flag1$ records those indexes of correctly classified cleans samples
  Generate a noisy samples using the **Algorithm 2**: $\tilde{X}, Flag2 \leftarrow EPGD(X[Flag1], Y[Flag1], \mathcal{E}[Flag1], f, Loss)$
  Use $\tilde{X}$, $f(.)$, $\mathcal{E}$ and Y to compute γ_1 and γ_2 (Eq. (13) and Eq. (14))
  Run the model f on noisy sample: $\tilde{Z} \leftarrow f(\tilde{X})$
  $L_1 \leftarrow Loss(\tilde{Z},\ Y[Flag1])$
  $L \leftarrow (L_0 + L_1)/2$
  Back-propagate from L and update the model $f(.)$
  $\mathcal{E}[Flag1\ \ \&\ \ Flag2] \leftarrow \mathcal{E}[Flag1\ \ \&\ \ Flag2] + \Delta_\varepsilon \cdot (\gamma_1 \cdot \gamma_2)[Flag1\ \ \&\ \ Flag2]$
  $\mathcal{E}[\sim Flag1$ **or** $\sim Flag2] \leftarrow ||X[\sim Flag1$ **or** $\sim Flag2] - \tilde{X}[\sim Flag2]||_p$
end for

Note: This algorithm is implemented in mini-batches. $||.||_p$ denotes vector Lp norm. $\sim$ is "not".

Algorithm 4 (Exploration-PGD): generate noisy samples

Input: training sample batch (X, Y); the estimated margins $\mathcal{E}$, currently; the DNN model $f(.)$; $g(.)$ is the function that transforms the output of $f(.)$ to a predicted class label, e.g., $argmax$; the Loss function $L(.)$.
Parameters: maximum PGD iteration number $K \leftarrow 20$; PGD step size $\alpha \leftarrow (4 \times \varepsilon)/K$.
Output: the generated noisy sample batch $\tilde{X}$, the $Flag2$ which records those indexes of correctly classified noisy/adversarial samples;
Function EPGD(X, Y, $\mathcal{E}$, f, $L(.)$):

$X_{out} \leftarrow X, Counter \leftarrow [0, ..., 0]$ (length is the batch size)
while $K > 0$ **do**
  $\tilde{X} \leftarrow \Pi_{\mathcal{E}}(\alpha \cdot h(\nabla_X L(f(X), Y)) + X)$
  $X \leftarrow \tilde{X}$
  $Counter[g(f(X)) \neq Y] \leftarrow 1$
  $K \leftarrow K - 1$
end while
$N \leftarrow 10$
$\tilde{X} \leftarrow (\tilde{X} + X_{out})/2$
while $N > 0$ **do**
  $\tilde{Y} = f(\tilde{X})$
  $X_{out}[\tilde{Y} = Y] \leftarrow \tilde{X}[\tilde{Y} = Y]$
  $\tilde{X} \leftarrow (\tilde{X} + X_{out})/2$
  $N \leftarrow N - 1$
end while
return X_{out}, $Flag2 = [Counter == 0]$

Note: This algorithm is implemented in mini-batches. $h(.)$ is the normalization function. $||.||_p$ denotes vector Lp norm. N is always set to 10, which is constant. So, this binary search will not enlarge the time complexity of the whole algorithm. $\Pi_{\mathcal{E}}(.)$ ensures that $||\tilde{X} - X||_p \leq \varepsilon$.

For the convenience of readers, we copy Algorithm 1 and Algorithm 2 here.

A detailed description of Algorithm 1: X is a clean training sample batch with true label batch Y, and Ids are the unique IDs for the samples (Line 1). After the clean samples are processed by the DNN model $f(.)$, the loss L_0 on the clean samples is obtained (Line 2-3). The indexes of these

correctly classified samples in the batch X are "recorded" by the binary array $Flag1$ (Line 4). For those correctly classified samples, Algorithm 2 will be used to generate adversarial/noisy samples $\tilde{X}$ (Line 5). γ_1 and γ_2 can be computed via Eq. (13) and Eq. (14). After the model processes the noisy samples, the loss L_1 on the noisy samples (Line 7-8) is obtained. Then, the model $f(.)$ is updated by backpropagation from the combined loss (Line 10). If both the clean samples and the corresponding noisy samples are correctly classified, which means the samples' margins are not large enough to reach the decision boundary, the margins $\mathcal{E}[Flag1 \ \ \& \ \ Flag2]$ for these training samples will be expanded (Line 11). Otherwise, the samples' margins $\mathcal{E}[\sim Flag1$ **OR** $\sim Flag2]$ are too large, and the adversarial training noises with these magnitudes have already pushed $\tilde{X}$ across the decision boundary, and therefore, $\mathcal{E}[\sim Flag1$ **OR** $\sim Flag2]$ should be refined to smaller margins (Line 12).

A detailed description of Algorithm 2 (EPGD): In each iteration (Line 2), X is modified to get $\tilde{X}$ (Line 3-4, see Eq. (3)). The indexes of those samples misclassified by the current model $f(.)$ are "record" by the array $Counter$ (Line 5). So, a binary search is applied to find the new $\tilde{x}$ in Fig. 5 (Line 4), which is just about to cross the decision boundary (Line 5). Then the algorithm runs to the next iteration (Line 6). Line 8-15 explain the binary search. X_{out} stores the noisy samples that are closer to the decision boundary but have not crossed it. This binary search runs for fixed 10 iterations, which is to refine margin estimation by the resolution of $2^{10} = 1024$.

B Experiment Settings

B.1 Basic Configurations

Pytorch1.9.0 (Paszke et al., 2017) is used for model implementation and evaluation. Nvidia V100 GPUs are used for model training and testing. L-inf norm are the most widely used measure for adversarial noise. In this paper, all of the adversarial training perturbations and adversarial noises are measured in the L-inf norm. For all experiments, three popular noise levels 2/255, 4/255, and 8/255 are used.

To describe the DNNs, we use "COV (a, b, c, d, e)" to represent a convolution layer with "a" input channels, "b" output channels, kernel size of "c", stride of "d" and padding of "e"; "Linear (f, g)" to denote a linear layer with input size of "f" and output size of "g"; "IN" to denote instance normalization; "LN" to denote layer normalization, and "LR" to denote leaky ReLU.

B.2 Settings for CIFAR 10 evaluation

In this experiment, we use CIFAR 10 dataset (Krizhevsky et al., 2009) to evaluate our method. CIFAR 10 contains 60000 32×32×3 colorful images. It has 10 classes. There are 6000 images per class with 5000 training (including 500 for validating) and 1000 testing images per class. We apply all the methods to WideResNet-28-4 (WRN-28-4) that is used in the MMA paper (Ding et al., 2019a).

For SMD training: We trained the SMD for 150 epochs. The optimizer is stochastic gradient descent (SGD) with settings the same as those in the MMA paper (Ding et al., 2019a).

B.3 Settings for SVHN Evaluation

Only AutoAttack is used.

The SVHN dataset (Netzer et al., 2011) contains $32 \times 32 \times 3$ color images of $0 \sim 9$ digits, 73257 images for training (including 7326 images for validating) and 26032 images for testing.

The network structure is COV(3, 32, 3, 1, 1)-LR-COV(32, 32, 3, 2, 1)-IN-LR-COV(32, 64, 3, 1, 1)-IN-LR-Conv2d(64, 64, 3, 2, 1)-IN-LR-COV(64, 128, 3, 1, 1)-IN-LR-Conv2d(128, 128, 3, 2, 1)-IN-LR-COV(128, 256, 4, 1, 0)-Flatten-LN-LR-Linear(256, 10).

Settings for SMD training: All settings are exactly the same as those for CIFAR10. **Settings for other methods:** All settings are exactly the same as those for CIFAR10.

B.4 SETTINGS FOR FASHION-MNIST EVALUATION

Only AutoAttack is used.

The fashion-MNIST dataset (Han et al., 2017) contains 28×28 grayscale images in 10 classes (shirts, shoes, etc.). 54000 samples for training, 500 for validation and 10000 for testing.

The network structure is COV(1, 64, 5, 2, 2)-LR-COV(64, 128, 5, 2, 2)-IN-LR-COV (128, 256, 5, 2, 2)-IN-LR-COV (256, 512, 4, 1, 0)-Flatten-LN-LR-Linear(512, 10).

For SMD training: All settings are exactly the same as those for CIFAR10. **For other methods:** All settings are exactly the same as those for CIFAR10.

C ADDITIONAL SUPPORTIVE RESULTS FOR GEM

C.1 MARGIN OVERESTIMATION IN MMA

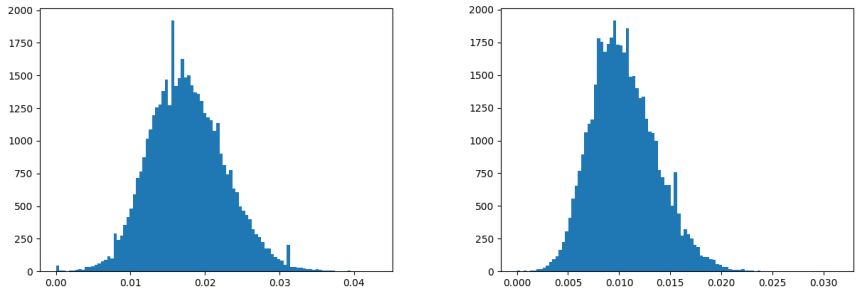


Figure 5: MMA uses the PGD with a large noise level to estimate sample-wise margins. Here, we use the PGD to estimate sample margins of a clean model (WRN-28-4) trained for 199 epochs on CIFAR10. The x-axis is the length of the margins. The y-axis is the number of samples. The left plot shows estimated margins from the PGD with the noise level of 16/255. The right plot shows estimated margins from the PGD with the noise level of 8/255. Estimated margins using the noise level of 16/255 are much larger than those using the noise level of 8/255. Thus, MMA's estimation of margins is very sensitive to the noise level used in the PGD.

C.2 MARGIN DISTRIBUTIONS OF ADVERSARIALLY-TRAINED MODELS

Fig. 6 shows the distribution of the estimated margins from our SMD method. Fig. 7 shows the distribution of the estimated margins from the MMA method. The distributions are from the CIFAR10 dataset. The x-axis is the length of the margins. The y-axis is the number of samples.

(1) Fig. 6 shows a Gaussian-like distribution, which supports the existence of the optimal state led by the Gradual Expansion Mechanism.

(2) Fig. 7 shows the significant margin overestimation problem of the MMA method, which has been analyzed in Section 2.5 and 3.3. The estimated margins follow a long-tail distribution, and roughly half of the SMD estimated margins Fig. 7 are larger than 8/255 (0.031), which means about half of the samples should be resistant to adversarial noise lower than 8/255, which is following the experimental evaluation result. But for MMA, most of the estimated margins are larger than 8/255 (0.031), which is in conflict with the results (see Table 1c).

C.3 TRAINING PROCESS

Fig. 8 shows the trend of SMD model clean accuracy and noisy accuracy in the training process: Clean accuracy and noisy accuracy at noise levels 4/255 and 8/255 keep increasing and reach a stable state after 120 epochs. No obvious robust overfitting exists (Rice et al., 2020)(Yu et al., 2022). This also supports the existence of the optimal state led by the Gradual Expansion Mechanism.

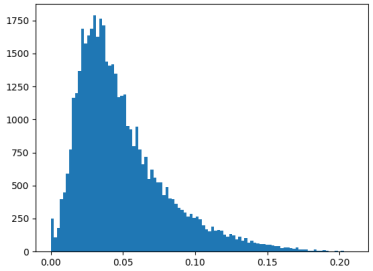


Figure 6: Estimated margin distribution of the SMD-trained model

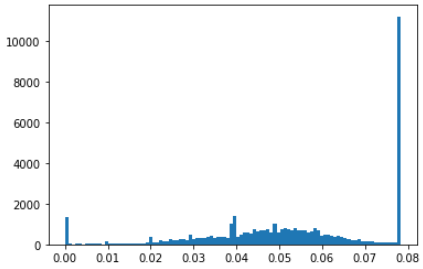


Figure 7: Estimated margin distribution of the MMA-trained model

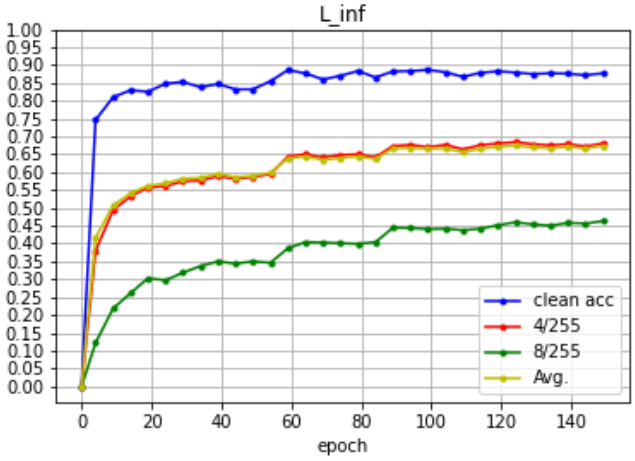


Figure 8: Trend of SMD-trained model in the training process

D THE VALUE OF ρ

Let $\bar{\mathcal{E}}$ be a rough estimation of the mean margin, and $\mathcal{E}_{max}$ be the maximum possible noise level (set to be 128/255, half of the max pixel value). After a number of margin expansion steps, we have the rough estimation:

$$\mathcal{E}_{max} = \Delta_\epsilon + \rho^{\bar{\mathcal{E}}}\Delta_\epsilon + (\rho^{\bar{\mathcal{E}}})^2\Delta_\epsilon + (\rho^{\bar{\mathcal{E}}})^3\Delta_\epsilon + ... \approx \frac{\Delta_\epsilon}{1-\rho^{\bar{\mathcal{E}}}} \tag{15}$$

$$\begin{aligned} &\rho^{\bar{\mathcal{E}}} = 1 - \frac{\Delta_\epsilon}{\mathcal{E}_{max}} \\ &\implies log(\rho) = (1/\bar{\mathcal{E}})log(1 - \frac{\Delta_\epsilon}{\mathcal{E}_{max}}) \\ &\implies \rho = e^{(1/\bar{\mathcal{E}})log(1-\frac{\Delta_\epsilon}{\mathcal{E}_{max}})} \end{aligned} \tag{16}$$

$$assume\ \bar{\mathcal{E}} = \frac{1}{2}\mathcal{E}_{max} \implies \rho \approx 0.9 \tag{17}$$

$$if\ \bar{\mathcal{E}} < \frac{1}{2}\mathcal{E}_{max},\ then\ \rho < 0.9 \tag{18}$$

Thus, we obtain a rough upper bound of ρ, and we use this value (0.9) in this paper.