IWA: Integrated gradient-based white-box attacks for fooling deep neural networks

Yixiang Wang1 | Jiqiang Liu1 | Xiaolin Chang1 | Jelena Mišić2 | Vojislav B. Mišić2

1Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, Beijing, China
2Department of Computer Science, Ryerson University, Toronto, Ontario, Canada

Correspondence
Jiqiang Liu, Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, 100044 Beijing, China. Email: jqliu@bjtu.edu.cn

Abstract
The widespread application of deep neural network (DNN) techniques is being challenged by adversarial examples—the legitimate input added with imperceptible and well-designed perturbation that can fool DNNs easily in the DNN testing/deploying stage. Previous white-box adversarial example generation algorithms used the Jacobian gradient information to add the perturbation. This imprecise and inexplicit information can cause unnecessary perturbation when generating adversarial examples. This paper aims to address this issue. We first propose to apply the more informative and distilled gradient information, namely, integrated gradient, to generate adversarial examples. To further make the perturbation more imperceptible, we propose to employ the restriction combination of $L_0$ and $L_1/L_2$ second, which can restrict the total perturbation and the perturbation points simultaneously. Meanwhile, to address the nondifferentiable problem of $L_1$, we explore a proximal operation of $L_1$ third. On the basis of these three works, we propose two Integrated gradient-based White-box Adversarial example generation algorithms (IWA): Integrated gradient-based Finite Point Attack (IFPA) and Integrated gradient-based Universe Attack (IUA). IFPA is suitable for situations where there are a determined number of points to be perturbed. IUA is suitable for situations...
where no perturbation point number is preset to obtain more adversarial examples. We verify the effectiveness of the proposed algorithms on both structured and unstructured data sets, and compare them with five baseline generation algorithms. The results show that our proposed algorithms craft adversarial examples with more imperceptible perturbation and satisfactory crafting rate. 

$L_2$ restriction is suitable for unstructured data sets and $L_1$ restriction performs better in the structured data set.

**KEYWORDS**
adversarial example, deep learning, integrated gradient, perturbation, white-box attack

## 1 INTRODUCTION

The fast rise of artificial intelligence in the past few years stems from the advancement of deep learning (DL) techniques, and DL has made fantastic success and state-of-the-art performance in various fields. However, adversarial examples challenge the security and reliability of DL techniques and affect the widespread application of DL in security-critical settings. Adversarial examples are the legitimate input added with tiny and well-designed perturbation, which can cause deep neural networks (DNNs) misclassification and the sharp degradation of their performance. Usually, the perturbation is hard, if not impossible, to be identified by human eyes. Since then, considerable researchers have devoted efforts to the study of adversarial examples mainly from two aspects: attacks and defenses. The past years witnessed diverse adversarial example generation algorithms with more imperceptible perturbation and more aggressive skills, which can be used to conduct white-box and/or black-box attacks.

In white-box settings, the adversary has access to the deployed DNN model, and in black-box settings, the adversary has no access to the deployed model. So, the white-box attacks are more powerful than the black-box attacks, and are investigated in various fields, such as sentiment analysis, malware detection, and automatic speech recognition. Meanwhile, defensive techniques to detect and/or resist adversarial examples have been proposed, such as FOCS method and TRADES adversarial training.

In this paper, we focus on the white-box adversarial example generation attacks. There are two primary problems with previous white-box adversarial works:

1. They used the Jacobian gradient information as the basis for the addition of perturbation, which will cause unnecessary perturbation. Concretely, in the white-box attacks, the information provided by the Jacobian gradient is not clear and does not reflect what a DNN model learns. This can cause redundant and undependable perturbation when this information is used to generate adversarial examples.

2. They were designed for a specific type of data sets, and their capabilities of generating high-quality adversarial examples were only verified on the image unstructured data sets. For instance, Carlini and Wanger (CW) attack verified their algorithms only on
image unstructured data sets, such as MNIST, CIFAR-10, and ImageNet. Grosse et al.\textsuperscript{18} verified their attack only on a structured malware data set. Neither of them considered the applicability of the proposed algorithms on the other type data sets. It is not comprehensive to verify the algorithm on the single type data sets. Here, the structured data resembles one table in a database. Unstructured data are like video, image, and so forth.\textsuperscript{19}

In this paper, we propose two Integrated gradient-based White-box Adversarial example generation algorithms (IWA): Integrated gradient-based Finite Point Attack algorithm (IFPA) and Integrated gradient-based Universe Attack algorithm (IUA), to address these concerns. IFPA is suitable for situations where there are a determined number of points to be perturbed. IUA is suitable for situations where no perturbation point number is preset to obtain more adversarial examples. Both algorithms have the following three major features:

1. Use the integrated gradient as the guideline to generate adversarial examples. Compared with the Jacobian gradient, the integrated gradient contains more informative and distilled gradient information and can better reflect each input point's contribution to the DNN output. It can guide to selecting points to be perturbed. In the structured data set, one point represents one actual value in an item. In the image unstructured data set, different from one pixel whose definition is three-channel values in a certain position of an RGB image, one point is only one channel value.

2. Employ $L_p$ norm restriction to limit the perturbation added on each point. Previous work has proven the effectiveness of $L_p$ norm restriction.\textsuperscript{9} Both two algorithms achieve the combination $L_0$ with $L_1/L_2$ restriction. That is, when generating adversarial examples, the proposed algorithms restrict not only the perturbation points but also the total perturbation.

3. Employ a proximal operation of $L_1$ to address the nondifferentiable problem of $L_1$. $L_1$ restriction rarely appears in adversarial generation algorithms since it is not differentiable.\textsuperscript{20} In this paper, we use the proximal operation\textsuperscript{21} to solve it mathematically.

To the best of our knowledge, we are the first to propose the above three features in the adversarial example generation algorithms. We verify the effectiveness of the two proposed algorithms on the structured data set, NSL-KDD, and unstructured data sets, MNIST and CIFAR-10. The experimental results verify that our algorithms can generate adversarial examples with imperceptible and low perturbation and satisfactory crafting rate, better than the baseline methods. Also, extensive results indicate that $L_2$ restriction is more suitable for unstructured data sets and $L_1$ restriction performs better in the structured data set and worse in unstructured data sets. Figures 1 and 2 are the adversarial examples generated by our proposed algorithm. We hope that our proposed attack can provide guidelines to researches on adversarial example defense.

The rest of the paper is organized as follows. In Section 2, we present the notations and describe the background and development of adversarial examples. Problem definition and the two proposed algorithms are given in Section 3. Section 4 is the evaluation of extensive experimental results on baselines and ours. Conclusions and future work are investigated in Section 5.

## 2 | BACKGROUND AND RELATED WORK

This section first presents the notations and background knowledge of adversarial examples. Then, we present the related work and five comparable baseline algorithms.
FIGURE 1  The adversarial examples generated by our proposed IFPA algorithm with specific points on MNIST data set. Legitimate denotes the legitimate input and adversarial means the corresponding adversarial example. The number under each image represents the category the model predicts. IFPA, Integrated gradient-based Finite Point Attack

FIGURE 2  Adversarial examples generated by the proposed IFPA algorithm with 1-, 2-, and 3-point attacks on the CIFAR-10 data set. IFPA, Integrated gradient-based Finite Point Attack [Color figure can be viewed at wileyonlinelibrary.com]
2.1 | Background

2.1.1 | Neural network and notations

Given the legitimate input $x \in \mathbb{R}^n$, DNNs can output $y \in \mathbb{R}^m$, and can be formalized as a sophisticated function with multiple layers:

$$y = f_k(f_{k-1}(...f_2(f_1(x); \theta_1); \theta_2)...); \theta_k) = F(x; \theta),$$

(2.1)

where $k$ is the $k$th layer, and $f_k(x; \theta_k)$ denotes the function learned in the $k$th layer. Each layer function $f$ is unified as

$$f(x; \theta) = \sigma(\theta_{\text{weight}} \cdot x) + \theta_{\text{bias}},$$

(2.2)

where $\sigma(\cdot)$ is a nonlinear activation function, such as ReLU, LeakyReLU, and softmax. $\theta_{\text{weight}}$ and $\theta_{\text{bias}}$ in each layer combine the model parameters $\theta$. In this paper, we consider the model as an $m$-class classifier. The model output is transformed to the probability distribution of $m$ classes via the softmax function in the last layer. That is, we refer $F(x)$ to the probability of the $i$th class and every component of $F(x)$ satisfies: (i) $\sum_{i=1}^{m} F(x)_i = 1$ and (ii) $\forall F(x)_i, F(x)_i \in [0, 1]$. Then, $\text{argmax}(F(x))$ is the predicted label of $x$, $C(x)$, and the ground-truth label of $x$ is $C^*(x)$. So, another formalization of $F$ is

$$F(x; \theta) = \text{softmax}(Z(x; \theta)) = y.$$  

(2.3)

$F(x; \theta)$ is the complete function a DNN model learns, and $Z(x; \theta)$ is the output without the softmax function and we call this the logit output. In this paper, we focus on generating adversarial examples on the logit output.

2.1.2 | Adversarial example and thread model

Szegedy et al.\textsuperscript{4} were the first to uncover the adversarial examples: given the legitimate input $x$ and its ground-truth label $C^*(x)$, the adversary can find $x_{\text{adv}}$ very similar to $x$ but $C(x_{\text{adv}}) \neq C^*(x)$. And $C(x_{\text{adv}})$ produces different results according to the adversary’s knowledge and goals.

When it comes to attacks, it is necessary to quantify the enemy’s capabilities. Papernot et al.\textsuperscript{24} first decomposed the space of adversaries in the DL systems, according to the adversarial goals and knowledge. In this paper, we follow their work\textsuperscript{5,24} to classify all adversarial attacks. That is, according to the adversary’s knowledge, we divide the attacks into the black-box attacks and the white-box attacks:

- White-box attacks assume that the adversary has access to a neural network, which means that the adversary knows the model topology, model parameters, and the data training for the model.
- Black-box attacks assume that the adversary has no access to a neural network, which means the adversary has no idea of the model topology and parameters but knows the model output.
According to the adversary’s goals, we can categorize the attacks into targeted attacks and nontarget attacks:

- Nontarget attacks are attacks in which the adversarial examples generated by adversaries cause the model to output the arbitrary class except the ground-truth class \( C^*(x) \), that is, \( C(x_{\text{adv}}) \neq C^*(x) \).
- Targeted attacks mean that the adversarial example \( x_{\text{adv}} \) generated by the adversary can cause the model to output class \( t \) that is specified by the adversary, where this class differs from the ground-truth class \( C^*(x) \), that is, \( t = C(x_{\text{adv}}) \neq C^*(x) \).

In this paper, we assume the adversary has the ability to carry out white-box attacks, and its goal is to cause a nontarget attack. It is reasonable since the previous work has shown that the nature of adversarial examples, namely, transferability, causes model \( B \) to be vulnerable to adversarial examples generated by model \( A \). So, if the adversary implements a black-box attack on model \( B \), it can attack the self-defined model \( A \) in a white-box manner. This is equivalent to a white-box attack to some extent.

### 2.1.3 Integrated gradient

Sundararajan et al. proposed a method to solve the problem of attributing the prediction of a deep network to its input features. That is, the proposed method, integrated gradient, can calculate the contribution of each input component to the specific class. The formula of the integrated gradient is as follows.

\[
\text{IntegratedGradient}(x) = \left( x_i - x_i^{\text{baseline}} \right) \times \sum_{j=1}^{S} \frac{\partial Z_i}{\partial x_i} \left( x_i^{\text{baseline}} + \frac{1}{S} \times (x - x_i^{\text{baseline}}) \right) \times \frac{1}{S},
\]

where \( x_i^{\text{baseline}} \) is the baseline example in their settings, for instance, the pure black image in the image data set and the zero-embedding vector in text data set. \( S \) is the iteration steps. As shown in Figure 3, we give an example to examine the difference between the Jacobian gradients of \( Z(x) \) w.r.t. \( x \) and the integrated gradients. We can see that the integrated gradient is more precise than the Jacobian gradient and has more detailed textures at the object’s edge. Thus, we can conclude that using the integrated gradient can help generate adversarial examples better than the Jacobian gradient.

### 2.1.4 \( L_p \) norm distance

\( L_p \) norm distance is an evaluation metric to assess the gap between two vectors in a vector space. We use it to evaluate the difference between adversarial examples and legitimate examples. The definition of \( L_p \) norm of \( x \) and \( x' \) is shown below.

\[
L_p(x - x') = \left( \sum_{i=1}^{n} |x_i - x'_i|^p \right)^{1/p}.
\]
Specifically:

- $L_0$ measures the number of different points in $x$ and $x'$, that is, $\sum_{i=1}^{n} I(x_i \neq x'_i)$, where $I(\cdot)$ is the indicator function when the inside condition is true, and $I(\cdot) = 1$.
- $L_1$ measures the sum of the absolute values of differences at all points, that is, $\sum_{i=1}^{n} |x_i - x'_i|$.
- $L_2$ is the well-known Euclidean distance, whose formula is $\sqrt{\sum_{i=1}^{n} (x_i - x'_i)^2}$.

In our experiments, $L_0$ is combined with $L_1$ or $L_2$ to restrict the perturbation between $x$ and $x'$, which will be described in Section 3, and $L_p$ is also used as the evaluation metrics.

### 2.2 Related work

As mentioned above, Szegedy et al.\textsuperscript{4} applied L-BFGS algorithm to uncover the existence of adversarial examples. Then, fast gradient sign method (FGSM)\textsuperscript{15} was proposed to use the gradient of the model to generate adversarial examples. Many follow-up algorithms were affected by it to generate adversarial examples via gradient information. Instead of using the Jacobian gradient information to generate adversarial examples, we apply a sophisticated gradient, the integrated gradient, to obtain the informative gradient.

Certainly, some algorithms do not use gradient information to generate adversarial examples in the black-box manner, such as one-pixel,\textsuperscript{17} DeepFool,\textsuperscript{26} NATTACK,\textsuperscript{27} and zeroth-order optimization (ZOO) attack.\textsuperscript{28} Concretely, ZOO\textsuperscript{28} attack uses a finite difference method to approximate gradients. This means that the adversary can generate adversarial examples without obtaining the model information to get gradients. In this paper, we only consider white-box attacks on the model gradients.

As for the perturbation restriction, adversarial generation algorithms use $L_p$ to constrain the perturbation. For instance, DeepFool\textsuperscript{26} places $L_2$ restriction in the divisor part. CW\textsuperscript{9} uses $L_0$, $L_2$, $L_\infty$ to restrict the perturbation, respectively. But there is no research on how to utilize $L_1$ fully since $L_1$ is not differentiable in the vector space. Previously, there was only
one work to deal with $L_1$ restriction. But they used the Reluplex algorithm to achieve similar $L_0$ restriction. Different from their work, we introduce proximal $L_1$ to solve the $L_1$ nondifferentiable problem mathematically. Also, how to combine $L_0$ and $L_1/L_2$ is necessary. One-pixel attack achieves the goal of restricting $L_0$, but it fails to use $L_1$ or $L_2$ to restrict the total perturbation so that the perturbation points can be found clearly. In this paper, our proposed IFPA algorithm solves the problem mentioned above.

Mechanisms to defend against adversarial examples were also proposed by researchers. Distillation was proposed by Papernot et al. to defend DNN against adversarial examples. Concretely, they divided the logit output by the temperature $T$ to control the degree of distillation. But the emergence of the CW attack proved that distillation could not resist CW attack. Afterward, other work proved some defensive algorithms were not as robust as they declared. In this paper, we do not pay attention to how to break through the defensive algorithms, but focus on generating high-quality adversarial examples.

## 2.3 Adversarial example generation algorithms

Thus far, several studies have proposed classical white-box adversarial example generation algorithms. Here, to facilitate the comparison between our algorithms and these algorithms, we briefly present each algorithm.

### 2.3.1 FGSM

FGSM algorithm was proposed to generate adversarial examples by

$$x_{\text{adv}} = x + \varepsilon \cdot \text{sign} (\nabla_x J(x, C^*(x))),$$

where $\text{sign}(v)$ denotes the sign of components in vector $v$, $\nabla_x J(\cdot)$ denotes the derivate of the loss function $J(\cdot)$ w.r.t. $x$, and $\varepsilon$ is the one-step perturbation ratio. This is a simple way to craft $x_{\text{adv}}$. FGSM inspires a lot of subsequent works, including building information modeling (BIM).

### 2.3.2 BIM

BIM is an iterative multistep variant of FGSM. The iterative function is defined as

$$x_{\text{adv}}^{N+1} = \text{clip}(x_{\text{adv}}^N + \varepsilon \cdot \text{sign}(\nabla_x J(x_{\text{adv}}^N, C^*(x))), x - \varepsilon, x + \varepsilon),$$

where function $\text{clip}(v, lb, ub)$ restricts components in $v$ between the lower bound $lb$ and the upper bound $ub$. $N$ is the iterations. $\varepsilon$ is the total perturbation. BIM solves the problem of lower adversarial example crafting rate in FGSM.
2.3.3  CW attack

CW attack was proposed to verify that the defensive distillation method is invalid for some attacks. CW attack can easily break the distilled model. The objective function of CW attack is that

$$\minimize L_p(\epsilon) + c \cdot g(x + \epsilon)$$

s.t. $x + \epsilon \in [0, 1]^n$,  \hspace{1cm} (2.8)

where $g(\cdot)$ is the self-designed function:

$$g(x + \epsilon) = \max(\max(Z(x + \epsilon)_{C^r(x)}) - Z(x + \epsilon), -x). \hspace{1cm} (2.9)$$

The objective function differs from ours described in Equation (3.3). CW attack optimizes the $L_p$ perturbation directly, but we use the $L_p$ distance between $x$ and $x'$. These two algorithms are different when implemented. Also, the penalty term is different. CW attack penalizes their function $g(\cdot)$, but we penalize the $L_p$ distance. Our objective function is mainly to minimize $Z(x_{adv})_{C^r(x)}$, but theirs is to minimize the perturbation.

2.3.4  Projected Gradient Descent (PGD)

PGD attack is another multistep variant of FGSM. The formula is as follows:

$$x_{adv}^{N+1} = \prod \left( x_{adv}^N + \epsilon \cdot \text{sign} \left( \nabla_x J \left( x_{adv}^N, C^r(x) \right) \right) \right). \hspace{1cm} (2.10)$$

More commonly, adversarial examples generated by PGD are used to do adversarial training, a method to make the model more robust. But in essence, it is still an algorithm of adversarial example generation.

2.3.5  AutoAttack

AutoAttack was proposed to solve two problems of PGD attack: (1) the step size is suboptimal; (2) the objective function is not proper. Croce and Hein proposed a new objective function Difference of Logits Ratio Loss (DLR), which is the shift and rescaling invariant. And the formula is

$$DLR(x, y) = \frac{Z(x)_y - \max_{i \neq y} Z(x)_i}{Z(x)_{\pi} - Z(x)_{\pi_0}}, \hspace{1cm} (2.11)$$

where $\pi_i$ denotes the $i$th component of $Z(x)$ in the descending order. Considering the complexity of the AutoAttack, we do not introduce it as detailed and nuanced. Readers who are interested can read it here.6
Table 1 is the comparison between previous works and our work. ‘Algo.’ means the comparable algorithms. ‘Data set’ indicates the type of experimental data set when the algorithm is proposed. The adversarial goal has been explained in Section 2.1.2. ‘Perturbation restriction’ means that the algorithm uses the $L_p$ norm to restrict the perturbation. ‘Attack frequency’ represents the algorithm to generate adversarial examples in an iterative way or one-step way. These five adversarial generation baselines are used to verify the effectiveness of our proposed algorithm in Section 4.

### 3 METHODOLOGY

In this section, we introduce the definition of the problem of adversarial examples in Section 3.1, which will be the fundamental basis to compose generation algorithms. Then, we propose two generation algorithms and explain them in detail. Finally, we outline the evaluation metrics used in Section 3.4.

#### 3.1 Problem definition

Previously, there were two common optimization ways to generate adversarial examples. One is maximizing the loss value of $x_{\text{adv}}$ in the direction of $C^*(x)$. At the same time, $x_{\text{adv}}$ should satisfy: (1) each component of $x_{\text{adv}}$ should be legitimate, that is, $x_{\text{adv}} \in [0, 1]^n$; and (2) the difference between $x_{\text{adv}}$ and legitimate $x$ should be smaller than $\varepsilon$, where $\varepsilon$ is as small as possible. In this way, the objective function with constraints can be formalized in Equation (3.1).

$$\begin{align*}
\text{maximize} & \quad J(F(x_{\text{adv}}), C^*(x)) \\
\text{s.t.} & \quad x_{\text{adv}} \in [0, 1]^n, \\
& \quad |x - x_{\text{adv}}| < \varepsilon.
\end{align*} \tag{3.1}$$

Maximizing the loss value is effective in generating adversarial examples because a large loss value implies misclassification. The other is minimizing the distance between $x_{\text{adv}}$ and

| Algo. | Data set         | Perturbation restriction | Attack frequency |
|-------|------------------|--------------------------|------------------|
|       |                  | $L_0$ | $L_1$ | $L_2$ |              |
| FGSM  | Unstructured     | –    | –    | –    | One-step     |
| BIM   | Unstructured     | –    | –    | –    | Iterative-step |
| PGD   | Unstructured     | –    | –    | ✓    | Iterative-step |
| CW    | Unstructured     | ✓    | –    | ✓    | Iterative-step |
| AutoAttack | Unstructured | –    | –    | ✓    | Iterative-step |
| Ours  | Structured and Unstructured | ✓    | ✓    | ✓    | Iterative-step |

Abbreviations: BIM, building information modeling; CW, Carlini and Wanger; FGSM, fast gradient sign method; PGD, Projected Gradient Descent.
legitimate $x$ while satisfying the misclassification and $x_{\text{adv}} \in [0, 1]^n$. The formula is shown in Equation (3.2).

\[
\begin{align*}
\text{minimize} & \quad \|x - x_{\text{adv}}\|_p \\
\text{s.t.} & \quad x_{\text{adv}} \in [0, 1]^n, \\
& \quad F(x) \neq F(x_{\text{adv}}).
\end{align*}
\]  

(3.2)

However, Carlini and Wagner\textsuperscript{9} demonstrated that the form in Equation (3.2) is hard to optimize. So, in this paper, we combine objective functions and constraints in Equations (3.1) and (3.2) and transform them to form a new objective function shown in Equation (3.3).

\[
\begin{align*}
\text{minimize} & \quad Z(x_{\text{adv}}) + c \cdot L_p(x, x_{\text{adv}}) \\
\text{s.t.} & \quad x_{\text{adv}} \in [0, 1]^n.
\end{align*}
\]  

(3.3)

That is, we aim to minimize two terms simultaneously. The first term is to minimize the logit output $Z(x_{\text{adv}})$ of $x_{\text{adv}}$. The second term is to minimize the $L_p$ distance between $x$ and $x_{\text{adv}}$. We believe that when $Z(x_{\text{adv}})$ is not the maximum in the logit output, misclassification also occurs. Here, $c$ is a positive constant adopted to control the proportion of the distance part in the objective function. However, when we specify $L_p$ as $L_1$, there is one problem: $L_1$ is not differentiable in the whole vector space.\textsuperscript{20} To solve this, we propose to use the proximal operation of $L_1$:\textsuperscript{21}:

\[
\text{Prox}_{L_1}(v) = \text{sign}(v) \cdot \max(|v| - \lambda, 0),
\]  

(3.4)

where $\lambda$ is the threshold parameter, and $L_1$ is denoted as $\text{Prox}_{L_1}$ in the rest of the paper. Readers interested in the theory of proximal operations can refer to the book\textsuperscript{21} therein for details. We use Equation (3.3) to construct the following two algorithms, IFPA and IUA, to generate adversarial examples with low perturbation.

### 3.2 | IFPA algorithm

As mentioned in Section 2.1.3, the integrated gradient method can reflect each input component’s contribution to the logit output. Here, we use the integrated gradient to help guide the selection of input points to be perturbated. It is intuitive if we perturb the point contributing the most to the ground-truth $C^*(x)$, the logit output $Z(x)_{C^*(x)}$ will be affected more easily. Furthermore, Equation (3.3) helps decrease the logit output $Z(x)_{C^*(x)}$, and misclassification is very likely to occur.

We follow the idea mentioned above and employ Equation (3.3) to design our first adversarial example algorithm, IFPA, and the flow is shown in Algorithm 1. In IFPA, we can preset the threshold of perturbed points according to integrated gradients. That is, the bigger the value of integrated gradients, the more the priority we have to perturbate the corresponding input points.

Concretely, we first get the copy of the legitimate input $x$ in line 1. In line 2, by employing the integrated gradient algorithm, we can obtain the integrated gradients of
Z(x)_{C^*(x)}. According to the input threshold $\mathcal{N}$, we get the index positions of $\mathcal{N}$ maximum values in the integrated gradients via function $\text{TopIndex}(\cdot)$. Subsequently, a for iteration in line 4 is implemented to generate adversarial examples since BIM has proven that the iterative way is more efficient. Lines 5 and 6 are the pseudocode of Equation (3.3) and $L_p$ can be $L_1$ or $L_2$ in line 6. To optimize the objective function in line 6, Adam optimizer is employed since it is proven the best choice among other optimizers. The output of $\text{Adam}(\cdot)$ is the gradient of the objective w.r.t. $x_{\text{adv}}$. The product of the gradient and $\varepsilon$ is the perturbation. To restrict the perturbation in the specified points, $\text{mask}(\cdot)$ function is further introduced. $\text{mask}(\cdot)$ function generates a '0' matrix with the same shape of $\text{ig}$, and sets the index positions, $\text{idx}$, to '1'. The benefit is that we can perturb the specified points while the other points are not affected. The Hadamard product $\odot$ of the perturbation and $\text{mask}(\cdot)$ is the final perturbation we add to the legitimate $x$. In line 9, $\text{clip}(\cdot)$ function restrict $x_{\text{adv}}$ in the input domain and in this paper, we set $\text{clip min}$ to 0 and $\text{clip max}$ to 1 in all data sets.

**Algorithm 1. IFPA**

$x$ is the legitimate input, $C^*(x)$ is the label of $x$, iters is iterations, $Z(\cdot)$ is the function model learned, $\varepsilon$ is the perturbation rate, $\mathcal{N}$ is the threshold, $c$ is the constraint parameter

**Input:** $x$, $C^*(x)$, $Z(\cdot)$, $\varepsilon$, $\mathcal{N}$

1. $x_{\text{adv}} = x$
2. $\text{ig} = \text{IntegratedGradient}(x, Z(\cdot), C^*(x))$
3. $\text{idx} = \text{TopIndex}(\text{ig}, \mathcal{N})$
4. for $i = 1$ to iters:
   5. $\text{logits} = Z(x_{\text{adv}})$
   6. $\text{obj} = \text{logits}[C^*(x)] + c \cdot L_p(x_{\text{adv}}, x)$
   7. $x_{\text{adv}} = x_{\text{adv}} - \varepsilon \cdot \text{Adam}(\text{obj}, x_{\text{adv}}) \odot \text{mask}(\text{idx})$
8. end for
9. $x_{\text{adv}} = \text{clip}(x_{\text{adv}}, \text{clip min}, \text{clip max})$
10. return $x_{\text{adv}}$

**How do we combine $L_0$ with $L_1$ or $L_2$?** At the beginning, we preset the number of perturbation points. Meanwhile, $L_0$ is the number of different points between $x$ and $x_{\text{adv}}$. Different points between $x$ and $x_{\text{adv}}$ are actually the embodiment of the perturbation points. Also, when generating adversarial examples, we use $L_1$ or $L_2$ to restrict the perturbation as much as possible. So, this attack is a combination of $L_0$ and $L_1$ or $L_2$. We call this attack IFPA, and IFPA has variants according to $L_p$. That is, when Equation (3.3) uses $L_1$, we call $\text{IFPA-R(\text{restrict})}_1$ attack, and the like.

### 3.3 IUA algorithm

In this section, we also present a universal iterative attack: IUA. The algorithm flow is shown in Algorithm 2. Algorithm 2 is similar to Algorithm 1 from the perspective of the objective function and the way to perturb the points. But the way to set the number of perturbation points is different. Specifically, it can be said that Algorithm 2 is an extension of Algorithm 1.
In IFPA, we preset \( N \) before Algorithm 1 runs. But there are two potential problems: (1) \( N \) is not enough to generate an adversarial example; (2) \( N \) is too large when generating an adversarial example. To solve these problems, a heuristic search of the perturbation points is implemented in Algorithm 2 to let the algorithm find the most appropriate number of perturbation points.

**Algorithm 2. IUA**

\( x \) is the legitimate input, \( C^*(x) \) is the label of \( x \), \( \text{iters} \) is iterations, \( Z(\cdot) \) is the function model learned, \( \varepsilon \) is the perturbation rate, and \( c \) is the constraint parameter.

**Input:** \( x, C^*(x), Z(\cdot), \varepsilon \)

1. \( x_{\text{adv}} = x \)
2. \( \text{ig} = \text{IntegratedGradient}(x, Z(\cdot), C^*(x)) \)
3. \( \text{sorted_idx} = \text{sort}(\text{ig}) \)
4. \( \text{idx} = 1 \)
5. **while** True:
   6.     **for** \( i = 1 \) to \( \text{iters} \):
   7.         \( \text{logits} = Z(x_{\text{adv}}) \)
   8.         \( \text{obj} = \text{logits}[y_{\text{true}}] + c \cdot L_p(x_{\text{adv}}, x) \)
   9.         \( x_{\text{adv}} = x - \text{Adam}(\text{obj}, \varepsilon) \odot \text{masked} (\text{sorted_idx}[\text{idx}]) \)
   10.        \( x_{\text{adv}} = \text{clip}(x_{\text{adv}}, \text{clip}\_\text{min}, \text{clip}\_\text{max}) \)
   11.        **if** \( \text{argmax}(Z(x_{\text{adv}})) \neq C^*(x) \): **return** \( x_{\text{adv}} \)
   12.        \( \text{idx} + = 1 \)
   13.        **if** \( \text{idx} \geq \text{len} (\text{sorted_idx}) \):
   14.            **break**
   15.    **end** for
   16. **end** while
17. **return** \( x_{\text{adv}} \)

Concretely, in line 3, \( \text{sort}(\cdot) \) function sorts the descending order of all points according to the integrated gradients. Then, in the **while** loop (line 5), we perturb one point every time in the way of Algorithm 1 (lines 7–10). The process is repeated until an adversarial example is generated (line 11), or we perturb all input points (line 13).

IUA is another pattern to combine \( L_0 \) with \( L_1/L_2 \) since we do not perturb all the points at the beginning but only perturbing the points one by one. In this case, the perturbation will not be added to the impractical point when generating adversarial examples, and we call this algorithm **IUA**.

### 3.4 Evaluation metrics

The following metrics will be used to evaluate the two proposed algorithms, that is, crafting rate, the mean and standard deviation (std) of \( L_0, L_1, \) and \( L_2 \). The definition of \( L_p \) has been given in Section 2.1.4. Here, \( L_1 \) is not the proximal operation but the formulation (2.5) when \( p = 1 \). The definition of crafting rate is
\[ \text{CraftingRate} = \frac{\# \text{ of } x_{\text{adv}}}{\# \text{ of } x}, \]  

(3.5)

where \# denotes numbers. The crafting rate demonstrates the efficiency of the algorithm in generating adversarial examples. \(L_0\) metric denotes how many points are perturbed. \(L_1\) metric measures how much perturbation is added to the legitimate \(x\). \(L_2\) metric reflects Euclidean distance between \(x_{\text{adv}}\) and \(x\). These metrics comprehensively evaluate the quality of the generated adversarial examples. Mean and standard deviation of \(L_p\) are used to quantify the distribution of perturbation under \(L_p\) metrics.

4 | EVALUATION

This section first presents the data sets and models used in the experiments. Then, we discuss the results of IFPA under the metrics in Section 4.2. Also, comparison between IUA and other algorithms is made in Section 4.3. Summary is given in Section 4.4.

4.1 | Description and setup

We do experiments on different types of data sets to verify the effectiveness of our algorithms. Two types are considered: structured data set (NSL-KDD\(^{36}\)) and unstructured data set (MNIST\(^{37}\) and CIFAR-10\(^{38}\)). We choose two unstructured data sets since they are commonly used for evaluation.\(^6\) The details of the three data sets are shown in Table 2, where the ‘Size’ column means the shape of one sample from the data set, and ‘Class’ indicates the number of classes in each data set.

There is no baseline model to train the NSL-KDD data set, so we stack a DNN to train it. The model structure is fc512–fc512–fc256–fc64–fc5, where fc512 means the fully connected layer with 512 nodes. Each layer uses the \(\text{LeakyReLU}\) activation function. We use Adam optimizer to train the KDD model. Other hyperparameters are as follows: learning rate is \(10^{-4}\), epoch number is 20, and batch size is 32. The performance of the NSL-KDD model on the test set reaches an acceptable accuracy score of 78.96%.

In the MNIST data set, we train an MNIST model described in Table 3, where each convolution layer has a \(\text{ReLU}\) activation function. \(3 \times 3 \times 32\) in the 1st line denotes this layer has 32 convolution kernels, whose shape is \(3 \times 3\). We use the SGD optimizer to train the MNIST model, where the learning rate is 0.01, momentum is 0.5, the batch size is 32, and the epoch is 10. Under these hyperparameters, the MNIST model achieves an accuracy score of 99.54% on the test set.

In the CIFAR-10 data set, we adopt the resnet-18\(^{39}\) to train the CIFAR-10. Resnet is an epoch-making DL model. Its emergence solves the problem that the model is too deep to do

| Data set   | Samples | | | | |
|------------|---------|---|---|---|
|            | Train   | Test | Size           | Class |
| NSL-KDD    | 126,003 | 22,544 | 1 \(\times\) 122 | 5     |
| MNIST      | 60,000  | 10,000 | 1 \(\times\) 28 \(\times\) 28 | 10    |
| CIFAR-10   | 50,000  | 10,000 | 3 \(\times\) 32 \(\times\) 32 | 10    |
backpropagation. The structure of resnet-18 is shown in Table 3. \([3 \times 3 \times 64; 3 \times 3 \times 64] \times 2\) means that the insider matrixes form a block, and this block is stacked twice, as shown in Figure 4. Then, the block with different convolutional kernel sizes will be stacked to form the resnet-18. We train the resnet-18 with the AdaBelief optimizer\(^{40}\) and LookAhead method,\(^{41}\) whose learning rate is 0.1. The epoch number and batch size we set are 300 and 128, respectively. The reset-18 model achieves a 94.78% accuracy on the CIFAR-10 test set.

One point to declare is that all data sets need to be preprocessed before training the models. In the image unstructured data set, we rescale the pixel to \([0, 1]\). In the NSL-KDD data set, the preprocessing process is a little more complicated because there are continuous data and

| Layer          | MNIST model | CIFAR-10 model |
|----------------|-------------|----------------|
| ReLU(Convolution) | \(3 \times 3 \times 32\) | \(7 \times 7 \times 64\) |
| ReLU(Convolution) | \(3 \times 3 \times 32\) | – |
| MaxPool         | \(2 \times 2\)  | \(3 \times 3\)  |
| ReLU(Convolution) | \(3 \times 3 \times 64\) | \([3 \times 3 \times 64; 3 \times 3 \times 64] \times 2\) |
| ReLU(Convolution) | \(3 \times 3 \times 64\) | \([3 \times 3 \times 128; 3 \times 3 \times 128] \times 2\) |
| ReLU(Convolution) | – | \([3 \times 3 \times 256; 3 \times 3 \times 256] \times 2\) |
| ReLU(Convolution) | – | \([3 \times 3 \times 512; 3 \times 3 \times 512] \times 2\) |
| MaxPool         | \(2 \times 2\)  | – |
| AvgPool         | – | \(7 \times 7\)  |
| ReLU(FullyConnected) | 1024         | 1000           |
| ReLU(FullyConnected) | 512          | –              |
| Output          | 10           | 10             |

**FIGURE 4** The block forms the resnet-18

---

TABLE 3  The structure of MNIST and CIFAR-10 models
discrete data. We can use Equation (4.1) to process the continuous data. \( x_{\text{mean}} \) and \( x_{\text{std}} \) are the mean and standard deviation of \( x \). As for discrete data, we use the one-hot encoder to transform them. The input domain of NSL-KDD is also in the range \([0, 1]\).

\[
x = (x - x_{\text{mean}})/x_{\text{std}}.
\]  

(4.1)

All models are implemented by the PyTorch library\(^{42}\) and the baseline methods except the AutoAttack are implemented by the Advertorch Library.\(^{43}\) The code of the AutoAttack is published in authors’ paper.\(^{6}\) All the hyperparameters of baselines are the default settings in the libraries. We use an Intel Xeon E5-2650v4 CPU with a single NVIDIA GTX 1080Ti GPU for experiments on three data sets.

### 4.2 | IFPA results and analysis

This section analyzes our proposed IFPA algorithm (Algorithm 1) under the metrics described in Section 3.4, as shown in Figures 5–7. Considering a large number of repeated experiments, we randomly select 1000 samples in MNIST and CIFAR-10 data sets and 10,000 samples in the...
NSL-KDD data set to evaluate our IFPA algorithms. The horizontal axis represents the number of perturbed points and we denote it as $L_0$. The reason has been given in Section 3.2. The vertical axis represents the corresponding evaluation values.

Figure 5A displays the crafting rate curves of the IFPA-$R_p$ algorithm under the NSL-KDD data set. There is no significant difference when using the IFPA-$R_1$ attack or the IFPA-$R_2$ attack. Both curves show that the crafting rate increases gradually with the perturbation points. A remarkable outcome is that 10 perturbed points will produce a 90 + % crafting rate. However, it is still slightly different since the crafting rate of $L_1$ is slightly higher than that of $L_2$. We then analyze the perturbation generated by the IFPA-$R_1$ attack and the IFPA-$R_2$ attack. Figure 5B,C provides the statistics for $L_p$, $p \in \{1, 2\}$ mean and standard deviation curves under IFPA-$R_p$, $p \in \{1, 2\}$ attacks. The solid curves show the $L_p$ mean curves and the light-colored area around the mean curves is the standard deviation. As can be seen in Figure 5B,C, the $L_p$, $p \in \{1, 2\}$ mean curves and std areas are almost coincident with each other. No significant differences between mean curves in the IFPA-$R_1$ and IFPA-$R_2$ attacks are revealed in Figure 5B,C. The same is true for std areas. Overall, in all metrics, the performance of IFPA-$R_1$ attack is slightly better than or equal to that of IFPA-$R_2$ attack.

In the MNIST data set, as regards the crafting rate shown in Figure 6A, it is interesting that the crafting rate curve of IFPA-$R_1$ attack is far higher than that of IFPA-$R_2$ attack, where 55 perturbed points will produce a 90 + % crafting rate under IFPA-$R_1$ attack. In Figure 6B,C, every component in $L_p$ mean curves of $R_1$ attack is higher than that of $R_2$ attack. And the $L_1$ std areas are smaller than the $L_2$ std areas, indicating that the perturbation generated by IFPA-$R_1$ attack is more stable than that by IFPA-$R_2$ attack. From the three subfigures, we can get an intuitive inference: IFPA-$R_1$ attack is more effective than IFPA-$R_2$ attack in the MNIST data set. But the perturbation of IFPA-$R_1$ is also larger than that of IFPA-$R_2$ attack, so we cannot judge which attack is strong since high crafting rate and low perturbation should be the criterion to a strong adversarial example attack.

In terms of the CIFAR-10 data set, as shown in Figure 7A, the crafting rate curves of IFPA-$R_p$ attacks are completely contrasting from those attacks on the MNIST data set. That is, the crafting rate curves of IFPA-$R_2$ attack are much higher than those of IFPA-$R_1$ attack. Especially, when the number of perturbation points is greater than 190, the crafting rate exceeds 90%. Figure 7B,C shows the performance of IFPA-$R_p$ attacks under $L_p$ metrics. We can see that the $L_p$ mean curves of IFPA-$R_1$ attack are larger than those of IFPA-$R_2$ attack, which indicates that the perturbation generated by IFPA-$R_1$ attack is much larger than that of
IFPA-R₂. Considering the high crafting rate of IFPA-R₂ attack, we conclude that IFPA-R₂ attack can generate the adversarial examples with a high crafting rate and low perturbation, and thus IFPA-R₂ is a strong adversarial example attack in the CIFAR-10 data set.

In summary, IFPA-R₁ attack achieves equal results with IFPA-R₂ attack on the structured NSL-KDD data set. But on the unstructured data set, the conclusion does not have uniformity. Specifically, IFPA-R₁ attack is totally inferior to IFPA-R₂ attack in the CIFAR-10 data set and IFPA-R₂ attack can generate adversarial examples with low perturbation and high crafting rate. But IFPA-R₁ attack has a higher crafting rate with more perturbation than IFPA-R₂ attack on the MNIST data set. We will explore the performance of the IUA attack on these data sets to further compare Lₚ in Section 4.3.

### 4.3 IUA results and analysis

In this section, we evaluate the proposed algorithm, IUA. As mentioned in Section 2.3, we choose five adversarial generation algorithms to compare with our IUA algorithm. Here, we use L₁ and L₂ restriction to construct our IUA algorithms, respectively, and we denote them as IUA-L₁ and IUA-L₂. As mentioned before, an adversarial example attack should be stronger when the crafting rate is higher and the perturbation is lower. It is acceptable to generate adversarial examples with the lowest perturbation without losing the crafting rate or with a lightly reduced crafting rate. The comparison results are shown in Tables 4–6. The first three rows of each table represent the ablation experiment of our IUA methods, where Lₚ denotes the IUA only has the individual Lₚ constraint. Ablation experiments are denoted as Abl-Lₚ, p ∈ {0, 1, 2}. Specifically, Abl-L₀ is the IUA algorithm described in Algorithm 2 without the Lₚ constraint (line 8). Abl-L₁ and Abl-L₂ are the IUA algorithms without the while loop (line 5). In this way, the integrated gradient method is useless. We implement the three ablation

| NSL-KDD | Crafting rate (%) | \( L₀ \) | \( L₁ \) | \( L₂ \) |
|---------|-------------------|---------|---------|---------|
|         |                   | Mean    | Std     | Mean    | Std     | Mean    | Std     |
| Abl-L₀  | 92.05             | 7.22    | 12.35   | 1.69    | 2.72    | 0.56    | 0.49    |
| Abl-L₁  | 91.49             | 118.72  | 1.35    | 8.50    | 1.51    | 0.87    | 0.12    |
| Abl-L₂  | 91.63             | 118.62  | 1.36    | 8.59    | 1.45    | 0.87    | 0.12    |
| IUA-L₁  | 93.49             | 4.63    | 8.68    | 1.25    | 2.25    | 0.50    | 0.48    |
| IUA-L₂  | 93.38             | 4.84    | 9.88    | 1.24    | 2.15    | 0.49    | 0.47    |
| FGSM    | 86.67             | 60.75   | 12.55   | 5.89    | 1.26    | 0.76    | 0.08    |
| BIM     | 89.55             | 75.16   | 15.83   | 5.24    | 0.96    | 0.66    | 0.07    |
| PGD     | 98.62             | 120.02  | 14.45   | 19.29   | 1.94    | 3.02    | 0.23    |
| CW      | 90.10             | 121.78  | 1.33    | 0.63    | 0.66    | 0.16    | 0.16    |
| AutoAttack | 10.81            | 115.42  | 5.10    | 31.87   | 11.51   | 9.87    | 8.88    |

*Note: Bold and italic numbers indicate the smallest and second smallest value in a column respectively.*

*Abbreviations: BIM, building information modeling; CW, Carlini and Wanger; FGSM, fast gradient sign method; IUA, Integrated gradient-based Universe Attack; PGD, Projected Gradient Descent.*
experiments to verify whether the integrated gradient is useful and whether $L_p$ constraints can restrict the perturbation.

Table 4 provides the summary statistics of five algorithms and ours under all metrics on the NSL-KDD structured data set. We first analyze the ablation results. Without the integrated gradients, the $L_0$ norm distance is large in the Abl-$L_1$ and Abl-$L_2$. And the performance of

| Table 5 | Comparison between five algorithms and our IUA algorithm on the MNIST data set |
|---------|-------------------------------------------|
| **MNIST** | Crafting rate (%) | $L_0$ | $L_1$ | $L_2$ |
| | | Mean | Std | Mean | Std | Mean | Std |
| Abl-$L_0$ | 98.49 | 199.26 | 130.85 | 87.81 | 74.80 | 6.18 | 2.97 |
| Abl-$L_1$ | 96.98 | 166.50 | 38.59 | 91.87 | 26.81 | 8.13 | 1.33 |
| Abl-$L_2$ | 96.48 | 488.57 | 29.63 | 69.19 | 6.67 | 3.60 | 0.22 |
| IUA-$L_1$ | 98.49 | 190.19 | 127.50 | 72.58 | 56.14 | 5.53 | 2.40 |
| IUA-$L_2$ | **100.00** | **179.95** | **118.78** | **64.65** | **48.12** | **5.18** | **2.13** |
| FGSM | 83.94 | 474.71 | 31.76 | 132 | 9.01 | 6.25 | 0.22 |
| BIM | 93.09 | 479.08 | 24.25 | 132.08 | 5.23 | 6.25 | 0.13 |
| PGD | 93.23 | 479.44 | 24.1 | 132.2 | 5.11 | 6.26 | **0.12** |
| CW | 100.00 | 757.24 | 19.7 | **12.18** | **4.06** | **1.13** | 0.33 |
| AutoAttack | 100.00 | 595.29 | **18.74** | 122.38 | 7.51 | 5.72 | 0.22 |

**Note:** Bold and italic numbers indicate the smallest and second smallest value in a column respectively.

Abbreviations: BIM, building information modeling; CW, Carlini and Wanger; FGSM, fast gradient sign method; IUA, Integrated gradient-based Universe Attack; PGD, Projected Gradient Descent.

| Table 6 | Comparison between five algorithms and our IUA algorithm on the CIFAR-10 data set |
|---------|-------------------------------------------|
| **CIFAR-10** | Crafting rate (%) | $L_0$ | $L_1$ | $L_2$ |
| | | Mean | Std | Mean | Std | Mean | Std |
| Abl-$L_0$ | 98.92 | 146.01 | 184.86 | 121.03 | 166.51 | 12.25 | 10.76 |
| Abl-$L_1$ | 91.64 | 3064.88 | 43.07 | 566.80 | 72.25 | 11.15 | 1.06 |
| Abl-$L_2$ | 93.40 | 3055.43 | 59.39 | 215.07 | 24.19 | 4.58 | 0.35 |
| IUA-$L_1$ | 95.83 | **97.89** | 265.48 | 186.18 | 803.98 | 11.45 | 19.28 |
| IUA-$L_2$ | 98.98 | **138.67** | 167.30 | 89.15 | 109.44 | 10.01 | 7.79 |
| FGSM | 83.52 | 3053.85 | 71.5 | 823.48 | 63.04 | 15.347 | 0.74 |
| BIM | 98.44 | 3053.37 | 71.81 | 823.27 | 63.12 | 15.346 | 0.74 |
| PGD | 98.41 | 3053.27 | 72.15 | 823.2 | 63.11 | 15.345 | 0.74 |
| CW | 99.98 | 3071.86 | **0.99** | **4.85** | **4.26** | **0.14** | **0.11** |
| AutoAttack | **99.99** | 3069.77 | 25.07 | 62.15 | 10.48 | 1.23 | 0.16 |

**Note:** Bold and italic numbers indicate the smallest and second smallest value in a column respectively.

Abbreviations: BIM, building information modeling; CW, Carlini and Wanger; FGSM, fast gradient sign method; IUA, Integrated gradient-based Universe Attack; PGD, Projected Gradient Descent.
Abl-$L_1$ and Abl-$L_2$ is almost the same in $L_p$ metrics. Compared with Abl-$L_1$ and Abl-$L_2$, the results of Abl-$L_0$ perform best in the ablation experiments, but its crafting rate is lower than ours and the perturbation is larger than that of IUA-$L_p$ without the $L_1/L_2$ constraint. Moreover, the low $L_0$ values of Abl-$L_0$ imply that the integrated gradients can help select the points to be perturbated. To sum up, the ablation results in the NSL-KDD data set indicate that Abl-$L_p$ methods are limited and inferior compared with IUA-$L_p$.

We compare IUA-$L_p$ with five baselines and the results are shown in the last seven rows of Table 5. We can see that each algorithm has little difference under the evaluation of Euclidean distance except PGD. Our IUA-$L_1$ algorithm achieves the second-highest accuracy, whereas PGD is the most efficient to generate adversarial examples, and its accuracy is 5% higher than ours. But under $L_p$ metrics, PGD performs worse than ours. Concretely, the number of perturbation points ($L_0$ mean value) of PGD is 30 times larger than that of IUA-$L_1$. This indicates that PGD needs more perturbation points to generate adversarial examples. The total perturbation ($L_1$ mean value) of PGD is 15 times larger than that of IUA-$L_1$. It is worthwhile to consider the trade-off between the perturbation and the crafting rate. The low perturbation with an acceptably high crafting rate is supposed sufficient for a strong adversarial attack. All the metrics of IUA-$L_2$ have almost no differences from IUA-$L_1$. Other algorithms, such as FGSM and BIM, do not perform as well as ours, but they confirm that the iterative way is better than one step to generate adversarial examples. The results of CW perform normally. Auto-Attack, however, has the lowest crafting rate, which indicates that some adversarial algorithms are not universal in the structured data set. It is necessary to verify the effectiveness of the structured data set when proposing a new adversarial algorithm.

As for the MNIST image unstructured data sets shown in Table 5, the ablation results reflect that the IUA-$L_p$ algorithm is stronger compared with Abl-$L_p$. Concretely, the result of Abl-$L_0$ is closest to that of IUA-$L_1$, but the perturbation of Abl-$L_0$ is larger without the $L_p$ constraint. Without the integrated gradients, $L_0$ values of Abl-$L_1$ become the largest among all methods. One surprising result of Abl-$L_1$ is that its $L_0$ mean is the lowest among all methods but the $L_1$ and $L_2$ values are very large. This phenomenon might be due to the unstable convergence of the Adam optimizer (line 9 in Algorithm 2). The $L_1$ and $L_2$ metrics of Abl-$L_1$ imply that $L_1$ does not constrain the perturbation in the unstructured data set. Overall, IUA-$L_p$ is more effective in generating adversarial examples than Abl-$L_p$, and $L_p$ constraint and integrated gradients are necessary for the IUA to restrict the $L_0$ and $L_1/L_2$ norm distance.

Compared with baselines, our proposed IUA-$L_2$ achieves the second-best performance. Specifically, the perturbation points are the least and the total perturbation and Euclidean distance is the second-least, whereas CW reaches the least. Also, the highest crafting rate of IUA-$L_2$ also implies that our algorithm is stronger than comparable baselines. Although the IUA-$L_1$ attack gets the second-highest crafting rate, the perturbation is also added relatively more than that of IUA-$L_2$, which is likely caused by the fact that the $L_1$ restriction is not as suitable as $L_2$ restriction for unstructured data sets. Meanwhile, $L_1$ values imply that $L_2$ restriction can restrict the $L_1$ norm distance implicitly. Other algorithms, such as BIM and PGD, perform well with their crafting rates reaching 90+%. The perturbation of PGD is slightly larger than that of BIM, but the perturbation of the two algorithms is unacceptably larger than that of our IUA-$L_2$. CW algorithm performs the best among all methods, while its perturbation is the least and the crafting rate is the highest. AutoAttack also reaches the best crafting rate but its perturbation is relatively larger than that of our IUA-$L_2$. We can conclude based on Table 5 clearly: our proposed IUA-$L_2$ is a strong adversarial example attack method and restricts the perturbation points and the total perturbation simultaneously while keeping a high crafting rate.
The results under all metrics of the CIFAR-10 data set are shown in Table 6. In the ablation results, the performance of Abl-$L_0$ is close to that of IUA-$L_2$ but the perturbation is larger than ours without the $L_p$ constraints, and these results give evidence to the effectiveness of the $L_p$ constraints. High $L_0$ values of Abl-$L_1$ and Abl-$L_2$ demonstrate the effectiveness of the integrated gradients. One notable result is that $L_1$ distance is also implicitly restricted in Abl-$L_2$. Abl-$L_1$ performs badly since it cannot restrict $L_p$ distances, and thus the $L_1$ constraint is not suitable for the unstructured data set.

As for the comparison with baselines, our proposed IUA-$L_2$ achieves the third-best performance. Concretely, the $L_0$ perturbation is the least and the other metrics are the third best. The performance of IUA-$L_1$ proves that the $L_1$ constraint is not suitable for unstructured data sets and the $L_2$ constraint can better restrict the perturbation of unstructured data sets. Moreover, the $L_2$ restriction constrains the $L_1$ distance implicitly in terms of $L_1$ values of IUA. AutoAttack reaches the best crafting rate while generating acceptable perturbation. CW attack reaches the lowest perturbation and second-best crafting rate, where the difference between the crafting rate of CW and AutoAttack can be ignored. Other methods, such as BIM and PGD, generate large perturbation with normal perturbation.

In all data sets, BIM and PGD achieve good crafting rates, but no restriction terms lead to a large perturbation. CW and AutoAttack perform well in the unstructured data sets and poorly in the structured data set. This indicates that it is critical to verify the proposed algorithm on all types of data sets. Meanwhile, our proposed IUA algorithm is able to handle all type data sets and restrict the perturbation based on the $L_p$ constraint, especially in $L_0$ norm distance, and performs better than state-of-the-art attacks in some data sets. We also find that $L_1$ restriction performs badly on unstructured data sets and $L_2$ restriction is suitable for unstructured data sets compared with $L_1$.

### 4.4 Summary

We make a summary of the extensive results and analysis from Sections 4.2 and 4.3. For both IFPA and IUA, $L_2$ restriction is much better than $L_1$ on unstructured data sets, and $L_2$ restriction can constrain the $L_2$ distance explicitly and $L_1$ distance implicitly. $L_1$ restriction performs worse on unstructured data set and has achieved better results than $L_2$ restriction on the structured data set.

### 5 Conclusions and Future Work

In this paper, we propose two integrated-based adversarial generation algorithms, IFPA and IUA, which can conduct white-box attacks on DNNs. The extensive experimental results verify that our algorithms can generate adversarial examples on all types of data sets with lower perturbation and higher crafting rates. Meanwhile, the results indicate that $L_2$ restriction is suitable for unstructured data sets and $L_1$ restriction does not match with unstructured data sets.

In future work, a targeted adversarial attack based on the integrated gradient will be considered. In addition, the interpretation of why $L_2$ restriction performs well on unstructured data sets should be explored from the perspective of mathematical theories. And more realistic experiments will be considered.
ACKNOWLEDGMENTS
This study was supported in part by National Key R&D Program of China, under Grant 2020YFB2103802, major scientific technological innovation projects in Shandong Province 2019JZZY020128, and supported by Chinese National Key Laboratory of Science and Technology on Information System Security.

ORCID
Yixiang Wang https://orcid.org/0000-0002-2393-4127
Jiqiang Liu https://orcid.org/0000-0003-1147-4327
Xiaolin Chang https://orcid.org/0000-0002-2975-8857
Jelena Mišić https://orcid.org/0000-0002-1251-3730
Vojislav B. Mišić https://orcid.org/0000-0001-7760-9920

REFERENCES
1. Wang C, Wu Y, Liu S, Zhou M, Yang Z. Curriculum pre-training for end-to-end speech translation. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics; 2020:3728-3738. doi:10.18653/v1/2020.acl-main.344
2. Onizuka H, Hayirci Z, Thomas D, Sugimoto A, Uchiyama H, Taniguchi R. TetraTSDF: 3D human reconstruction from a single image with a tetrahedral outer shell. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR); 2020:6011-6020.
3. Wang W, Zhao M, Wang J. Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network. J Ambient Intell Hum Comput. 2019;10(8):3035-3043. doi:10.1007/s12652-018-0803-6
4. Szegedy C, Zaremba W, Sutskever I, et al. Intriguing properties of neural networks. In: International Conference on Learning Representations; 2014.
5. Yuan X, He P, Zhu Q, Li X. Adversarial examples: attacks and defenses for deep learning. IEEE Trans Neural Netw Learn Syst. 2019;30(9):2805-2824. doi:10.1109/TNNLS.2018.2886017
6. Croce F, Hein M. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In: International Conference on Machine Learning. PMLR; 2020:2206-2216.
7. Croce F, Hein M. Minimally distorted adversarial examples with a fast adaptive boundary attack. In: International Conference on Machine Learning. PMLR; 2020:2196-2205. Accessed August 26, 2021. https://proceedings.mlr.press/v119/croce20a.html
8. Samanta S, Mehta S. Generating adversarial text samples. In: Pasi G, Piwowarski B, Azzopardi L, Hanbury A, eds. Advances in Information Retrieval. Vol 10772. Lecture Notes in Computer Science. Springer International Publishing; 2018:744-749. doi:10.1007/978-3-319-76941-7_71
9. Carlini N, Wagner D. Towards evaluating the robustness of neural networks. In: 2017 IEEE Symposium on Security and Privacy (SP). IEEE; 2017:39-57. doi:10.1109/SP.2017.49
10. Li H, Zhou S, Yuan W, Li J, Leung H. Adversarial-example attacks toward android malware detection system. IEEE Syst J. 2020;14(1):653-656. doi:10.1109/JSYST.2019.2906120
11. Carlini N, Wagner D. Audio adversarial examples: targeted attacks on speech-to-text. In: 2018 IEEE Security and Privacy Workshops (SPW). IEEE; 2018:1-7. doi:10.1109/SPW.2018.00009
12. Wang Y, Ma X, Bailey J, Yi J, Zhou B, Gu Q. On the convergence and robustness of adversarial training. In: International Conference on Machine Learning. PMLR; 2019:6586-6595.
13. Zhang H, Yu Y, Jiao J, Xing EP, Ghaoui LE, Jordan MI. Theoretically principled trade-off between robustness and accuracy. In: Chaudhuri K, Salakhutdinov R, et al., eds. Proceedings of the 36th International Conference on Machine Learning (ICML 2019), June 9-15, 2019, Long Beach, California, USA. 97. Proceedings of Machine Learning Research (PMLR); 2019:7472-7482.
14. Sundararajan M, Taly A, Yan Q. Axiomatic attribution for deep networks. In: Proceedings of the 34th International Conference on Machine Learning (ICML’17). Vol 70. JMLR.org; 2017:3319-3328.
15. Goodfellow I, Shlens J, Szegedy C. Explaining and harnessing adversarial examples. In: International Conference on Learning Representations; 2015.
39. He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE; 2016:770-778. doi:10.1109/CVPR.2016.90

40. Zhuang J, Tang T, Ding Y, et al. AdaBelief optimizer: adapting stepsizes by the belief in observed gradients. In: *Advances in Neural Information Processing Systems*. Vol 33. Curran Associates Inc.; 2020: 18795-18806. Accessed September 11, 2021.

41. Zhang M, Lucas J, Ba J, Hinton GE. LookAhead Optimizer: k steps forward, 1 step back. In: *Advances in Neural Information Processing Systems*. Vol 32. Curran Associates Inc.; 2019.

42. Paszke A, Gross S, Massa F, et al. PyTorch: an imperative style, high-performance deep learning library. In: Wallach H, Larochelle H, Beygelzimer A, Alché-Buc F, Fox E, Garnett R, eds. *Advances in Neural Information Processing Systems*. Vol 32. Curran Associates Inc.; 2019:8026-8037.

43. Ding GW, Wang L, Jin X. advertorch v0.1: an adversarial robustness toolbox based on PyTorch. ArXiv190207623 Cs Stat. Published online February 20, 2019. Accessed August 22, 2021. http://arxiv.org/abs/1902.07623

**How to cite this article:** Wang Y, Liu J, Chang X, Mišić J, Mišić VB. IWA: Integrated gradient-based white-box attacks for fooling deep neural networks. *Int J Intell Syst*. 2022;37:4253-4276. doi:10.1002/int.22720