Hindawi
Security and Communication Networks
Volume 2021, Article ID 5404061, 11 pages
https://doi.org/10.1155/2021/5404061

Research Article

SSGD: A Safe and Efficient Method of Gradient Descent

Jinhuan Duan,[1] Xianxian Li,[1,2] Shiqi Gao,[2] Zili Zhong,[2] and Jinyan Wang[1,2]

[1]*Guangxi Key Lab of Multi-Source Information Mining and Security, Guangxi Normal University, Guilin, China*
[2]*College of Computer Science and Engineering, Guangxi Normal University, Guilin, China*

Correspondence should be addressed to Xianxian Li; lixx@gxnu.edu.cn and Jinyan Wang; wangjy612@gxnu.edu.cn

Received 30 May 2021; Revised 13 July 2021; Accepted 27 July 2021; Published 10 August 2021

Academic Editor: Lu Liu

With the vigorous development of artificial intelligence technology, various engineering technology applications have been implemented one after another. The gradient descent method plays an important role in solving various optimization problems, due to its simple structure, good stability, and easy implementation. However, in multinode machine learning system, the gradients usually need to be shared, which will cause privacy leakage, because attackers can infer training data with the gradient information. In this paper, to prevent gradient leakage while keeping the accuracy of the model, we propose the super stochastic gradient descent approach to update parameters by concealing the modulus length of gradient vectors and converting it or them into a unit vector. Furthermore, we analyze the security of super stochastic gradient descent approach and demonstrate that our algorithm can defend against the attacks on the gradient. Experiment results show that our approach is obviously superior to prevalent gradient descent approaches in terms of accuracy, robustness, and adaptability to large-scale batches. Interestingly, our algorithm can also resist model poisoning attacks to a certain extent.

1. Introduction

Gradient descent (GD) is a technique to minimize an objective function, which is parameterized by the parameters of a model, by updating the parameters with the opposite direction of the gradient of the objective function about the parameters [1]. It has widely been applied in solving various optimization problems because of its simplicity and impressive generalization ability [2], but it is born with a heart of revealing privacy. Mathematically, the gradient is the parametric derivative of the loss function, which is explicitly calculated from the given training data and its true label. Therefore, the attacker may extract the sensitive information of the original training data from the captured gradients. Recently, researches have shown that the attacker, which captures the gradient of a training sample, can successfully infer its attributes [3], label [4], class representation [5, 6], or the data input itself [4, 7–9], with high accuracy. In the actual deep learning system, the gradient of multiple samples is widely used to improve efficiency and performance, which can also be viewed as the per-coordinate average of the single-sample gradients. Is multisample gradient safer for the privacy of training data? Unfortunately, Pan et al. [9] gave the theoretical analysis to indicate that multisample gradient still leaks samples and labels under certain circumstances. Since the work of Zhu et al. [7] was proposed, there is a branch of research [4, 7–9] to explore a violent but universal method for successful data reconstruction attacks, and some meaningful empirical results are given on CIFAR-10 and ImageNet. These works are based on the same learning-based framework. First, a batch of unknown training samples are used as variables, and then the optimal training samples are searched by minimizing the distance between the ground-truth gradient and the gradient calculated by the variables. The main difference between them is the choice of minimizing distance function. L2 and cosine distances are used in [4, 7, 8], respectively. Although Zhao et al. [4] used the properties of neural networks to recover the label of a single sample before the learning-based attack, this technique is only suitable to single-point gradient. It is the same as [7] in the multisample case. Pan et al. [9] gave a theoretical explanation for information leakage of single sample in a fully connected neural network with ReLu activation function. Furthermore, they used the internal

information between neurons to show that in some cases there is sample and label leakage in multiple samples and extended the model to ResNet-18 [10], VGG-11 [11], DenseNet-121 [12], AlexNet [13], ShuffleNet v2-x0-5 [14], InceptionV3 [15], GoogLeNet [16], and MobileNet-V2 [17].

To solve the gradient safety problem, Bonawitz et al. [18] designed a secure aggregation protocol, which is a four-round interactive protocol. Xu et al. proposed VerifyNet [19] and VeriFL [20] by adding verifiability to [18] for ensuring the correctness of aggregation. Bell et al. [21, 22] introduced a secure aggregation protocol with multilogarithmic communication and computational complexity, which reduces one round of interaction compared with [18]. Fereidooni et al. [23] showed that only two rounds of communication can be safely aggregated. All of the above works use encryption algorithms to encrypt the entire data set or intermediate values during the training process. Different from them, Ma et al. [24] used secure verifiable computing delegation to privately label a public data set from locally trained model aggregation and then utilized public data sets to train local models. Phong et al. [25] used homomorphic encryption technology to encrypt the gradient before sending it. Abadi et al. [26] employed differential privacy to protect gradients. Yadav et al. [27] applied differential privacy to federated machine learning by directly adding noise to the gradient. In PrivateDL [28], it is allowed to effectively transfer relational knowledge from sensitive data to public data in a way of privacy protection and enables participants to jointly learn local models based on public data with noise protection labels. However, these methods also have their limitations. The main problem of the secure aggregation protocol is communication overhead and computational efficiency. For differential privacy technology, it needs to consider the tradeoff between privacy and utility. More noise will lead to poor performance, and less noise will not be enough to protect the gradient. PrivateDL [28] requires a public data set and reduces the performance of the algorithm.

Therefore, this paper proposes a new gradient descent method, super stochastic gradient descent (SSGD), for achieving neuron-level security while maintaining the accuracy of model. Moreover, SSGD has stronger robustness. Phong et al. [25] analyzed the leakage of single-sample single-neuron input data in the single-layer perceptron by using the sigmoid activation function. Pan et al. [9] used the ReLu activation function to analyze the sample data leakage from the multilayer fully connected neural network gradient and indicated that multiple samples also reveal privacy. There are two neurons in the last layer which are only activated by the same single sample. Essentially, the leakage is caused by attacking the single-sample gradient. SSGD converts the neuron gradient into a unit vector, which makes that the gradient aggregation of neurons has super-randomness. Superrandomness may significantly worsen the performance of the algorithm and make it difficult to converge. We select multiple-sample gradient composition updates to increase stability. At the same time, the super-randomness also brings strong robustness because the attacker cannot know the true gradient. SSGD invalidates these attacks on the gradient model, including the attack by searching for the optimal training sample [4, 7, 8] based on minimizing the distance between the ground-truth gradient and the gradient calculated by the variable, and the attack by solving the equation system [9] to obtain the training data. Our contributions are summarized as follows.

(1) We propose a gradient descent algorithm, called super stochastic gradient descent. The main idea is to update the parameters by using the unit gradient vector. In neural networks, neuron parameters are updated by using the unit gradient vector of neurons.

(2) We analyze theoretically that SSGD can realize neuron-level security and defend against attacks on the gradient.

(3) Experimental results show our approach has better accuracy and robustness than prevalent gradient descent approaches. And it can resist model poisoning attacks to a certain extent.

The rest of this paper is organized as follows. In Section 2, we review the basic gradient descent methods and the data leakage by gradients. In Section 3, we describe the super stochastic gradient descent and analyze the safety of our approach. The experimental results are shown in Section 4. Finally, we conclude this paper and give the further work.

2. Preliminaries

In this section, we review some basic gradient descent algorithms [1], including batch gradient descent (BGD), stochastic gradient descent (SGD), and mini-batch gradient descent (MBGD). The difference among them is that how much data is used to calculate the gradient of the objective function. Then, we describe the information leakage caused by gradients [19].

2.1. Basic Gradient Descent Algorithms. The BGD is an ordinary form of gradient descent, which takes the entire training samples into account to calculate the gradient of the cost function $\ell(\theta)$ about the parameters θ and then update the parameters by

$$\theta = \theta - \eta \cdot \nabla_\theta \ell(\theta), \tag{1}$$

where η is the learning rate and $\nabla_\theta \ell(\theta)$ represents the gradient of function $\ell(\theta)$ with respect to the parameters θ. The BGD uses the entire training set in each iteration. Therefore, the update is proceeded in the right direction, and finally BGD is guaranteed to converge to the extreme point. On the contrary, the SGD considers a training sample x_i and label y_i randomly selected from the training set in each iteration to perform the update of parameters by

$$\theta = \theta - \eta \cdot \nabla_\theta \ell(\theta; x_i; y_i). \tag{2}$$

The BGD and SGD are two extremes: one uses all training samples and the other uses one sample for gradient descent. Naturally, their advantages and disadvantages are very prominent. For the training speed, the SGD is very fast,

and the BGD cannot be satisfactory when the size of training sample set is large. For accuracy, the SGD determines the direction of the gradient with only one sample, resulting in a solution which may not be optimal. For the convergence rate, because the SGD considers one sample in each iteration and the gradient direction changes greatly, it cannot quickly converge to the local optimal solution.

The MBGD is a compromise between BGD and SGD, which performs an update with a randomly sampled mini-batch of N training samples by

$$\theta = \theta - \eta \cdot \nabla_\theta \ell\left(\theta; x_{(i;i+N)}; y_{(i;i+N)}\right), \tag{3}$$

where N is the number of batches. MBGD decreases the variance of the updates for parameter, so it has more stable convergence. Moreover, the computing of gradient about a mini-batch is very efficient by using highly optimized matrix optimizations that existed in advanced deep learning libraries.

2.2. Analysis of Gradient Information Leakage. Phong et al. [25] illustrated that how gradients leak the data information based on a single neuron shown in Figure 1. Assume that $\hat{x} \in R^d$ represents data input with a label value $y \in R$. $w \in R^d$ is the weight parameter and $b \in R$ is the bias, represented uniformly by $\theta = (w, b) \in R^{(d+1)}$. $g \in R^{(d+1)}$ is the gradient vector of the parameter θ, f is an activation function, and the loss function is $\ell(f(\hat{x}, w, b), y) = (h_{w,b}(\hat{x}) - y)^2$, where $h_{w,b}(\hat{x}) = f(\sum_{i=1}^{d} w_i \hat{x}_i + b)$. Let $g = (\sigma_1, \ldots, \sigma_k, \ldots, \sigma_d, \sigma)$ and $k \in \{1, \ldots, d\}$. We have

$$\sigma_k = \frac{\partial \ell(f(\hat{x}, w, b), y)}{\partial w_k} = 2\left(h_{w,b}(\hat{x}) - y\right) f'\left(\sum_{i=1}^{d} w_i \hat{x}_i + b\right) \cdot \hat{x}_k,$$

$$\sigma = \frac{\partial \ell(f(\hat{x}, w, b), y)}{\partial b} = 2\left(h_{w,b}(\hat{x}) - y\right) f'\left(\sum_{i=1}^{d} w_i \hat{x}_i + b\right). \tag{4}$$

Therefore, we obtain $\sigma_k = \sigma \cdot \hat{x}_k$. By solving the system of equations, we can easily get $\hat{x}$ and y. Also, we know that g is determined by $(\hat{x}, y)$. Therefore, g and $(\hat{x}, y)$ are bijective. In distributed training, w and b usually are the parameters that need to be updated and known. Then, it can infer $(\hat{x}, y)$ from g.

Based on [9], the single-sample analysis of multilayer neural networks by using ReLu activation function, there is also data leakage problem. Although there is no such simple and intuitive leakage of data in a multilayer neural network, we can still know $\hat{x}$ and y by analyzing the internal relationship of the neural network and find that $(\hat{x}, y)$ and g are still bijective.

3. Super Stochastic Gradient Descent

In this section, we propose our super stochastic gradient descent approach for preventing gradient leakage while keeping the accuracy and then analyze in detail the safety of our approach.

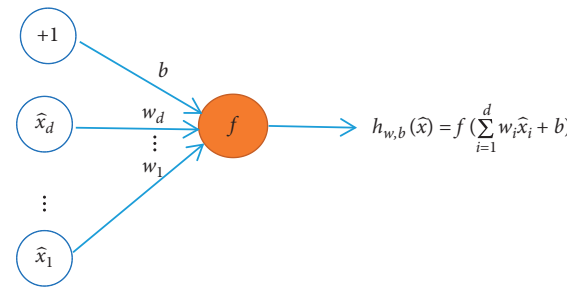


Figure 1: Single neuron structure.

3.1. Approach. It was confirmed that the gradient leaks privacy [7, 25]. For solving the security problem caused by the exchange gradient in stochastic gradient descent or mini-batch gradient descent, we propose the super stochastic gradient descent approach, which can protect the gradient information without losing accuracy by hiding part of the gradient information. The gradient is the first-order partial derivative of the objective function, so it is a vector with both magnitude and direction. We seek the gradient of the objective function to find the fastest descent direction. But it is a little related to the modulus length of the gradient vector. Therefore, we hide the modulus length of the gradient vector and convert the gradient vector into a unit vector.

The superrandomness, caused by the aggregation of multiple unit gradient vectors, may lead to poor results. To guarantee that this kind of randomness is friendly, we utilize the following approaches to reduce the uncertainty caused by superrandomness.

For single-sample training sample x_i and label y_i, we use unit gradient vector to update parameter θ:

$$\theta = \theta - \eta \cdot \frac{\nabla_\theta \ell(\theta; x_i; y_i)}{\left\|\nabla_\theta \ell(\theta; x_i; y_i)\right\|}. \tag{5}$$

For multiple samples, the parameter is updated to

$$\theta = \theta - \frac{\eta}{m} \cdot \sum_{j=1}^{m} \frac{\nabla_\theta \ell\left(\theta; x_{(i;i+n)}; y_{(i;i+n)}\right)_j}{\left\|\nabla_\theta \ell\left(\theta; x_{(i;i+n)}; y_{(i;i+n)}\right)_j\right\|}, \tag{6}$$

where $x_{(i+n)}$ represents n samples and $y_{(i+n)}$ denotes their labels. The gradient $\nabla \ell(\theta; x_{(i;i+n)}; y_{(i;i+n)})$ of n samples is considered as a basic gradient, and m is the number of basic gradients. Aggregating the unit gradient vectors of m basic gradients on average is to further enhance the stability of the algorithm. The algorithm has higher performance with strong randomness. It is secure to share this unit basic gradient in a distributed environment.

Neuron is the smallest information carrier in the neural network structure. In the neural network, we choose to convert each neuron parameter gradient vector into a unit vector. Therefore, the single-layer neural network parameter is updated to

$$\theta_r = \theta_r - \frac{\eta}{m} \cdot \sum_{j=1}^{m} \frac{\nabla_\theta \ell\left(\theta; x_{(i;i+n)}; y_{(i;i+n)}\right)_{rj}}{\left\|\nabla_\theta \ell\left(\theta; x_{(i;i+n)}; y_{(i;i+n)}\right)_{rj}\right\|}, \tag{7}$$

where θ_r represents the rth column or rth row of the parameter matrix in the fully connected layer or convolutional layer (the convolution kernel is regarded as a neuron). In the fully connected layer, $\nabla_\theta \ell(\theta; x_{(i;i+n)}; y_{(i;i+n)})_r$ is expressed as the rth column of the gradient matrix. And in the convolutional layer, it represents the rth row of the gradient matrix of the convolution kernel. Therefore, each row or column of the gradient matrix is a unit vector. Then, we obtain an average gradient matrix by using m such gradient matrices to update the parameters.

3.2. The Safety of SSGD. By analyzing the multilayer neural network with ReLu activation function on a training sample, the following relationship is obtained in [9]:

$$\overline{G}^i = \sum_c \overline{g}_c \left(D^i W^{i-1} \cdots W^0 X\right) \left(\left[W^H\right]_c^T D^H \cdots W^{(i+1)} D^{(i+1)} \right), \tag{8}$$

in which $X = \{x_1, x_2, \ldots, x_n\}$ is the input data, where $x_i \in R^d$ and $X \in R^{d \times n}$. $\overline{g}_c$ represents the cth dimension of the loss vector $\overline{g}$, T is the number of layers of neural network, D^i is the activation pattern of the ith layer of neural network, and $\overline{G}^i$ and W^i denote the gradients and parameters of the ith layer of neural network, respectively. In fact, the attack gradient models are all solutions to the above equations. In the distributed training model that needs to share the gradient, the participants know $\overline{G}^i$, W^i, and D^i. For data reconstruction attacks, it can infer $\overline{g}_c$ and solve X by equation (8).

The left side of equation (8) is the ith layer gradient matrix:

$$\overline{G}^i = \begin{bmatrix} \sigma^i_{1,1} & \sigma^i_{1,2} & \cdots & \sigma^i_{1,w^i} \\ \sigma^i_{2,1} & \sigma^i_{2,2} & \cdots & \sigma^i_{2,w^i} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma^i_{w^{i-1},1} & \sigma^i_{w^{i-1},2} & \cdots & \sigma^i_{w^{i-1},w^i} \end{bmatrix}_{w^{i-1} \times w^i}, \tag{9}$$

where w^i is the number of neurons in the ith layer. The gradient matrix of our SSGD is

$$\widehat{G}^i = \begin{bmatrix} \sigma^i_{1,1} & \sigma^i_{1,2} & \cdots & \sigma^i_{1,w^i} \\ \sigma^i_{2,1} & \sigma^i_{2,2} & \cdots & \sigma^i_{2,w^i} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma^i_{w^{i-1},1} & \sigma^i_{w^{i-1},2} & \cdots & \sigma^i_{w^{i-1},w^i} \end{bmatrix}_{w^{i-1} \times w^i} \begin{bmatrix} \frac{1}{\mu^i_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{\mu^i_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\mu^i_{w^i}} \end{bmatrix}_{w^i \times w^i}. \tag{10}$$

Each column of $\widehat{G}^i$ is a unit vector, and μ^i_1 is the modulus length of the 1st column vector of the ith layer gradient matrix, i.e., the modulus length of the 1st neuron gradient of the ith layer neural network. Essentially, the parameter matrix of a layer of neural network is multiplied by a diagonal matrix U^i on the right, and the value of the diagonal matrix is the reciprocal of the modulus length of the gradient vector of each neuron. By using our SSGD, equation (8) is represented as

$$\widehat{G}^i = \sum_c \overline{g}_c \left(D^i W^{i-1} \cdots W^0 X\right) \left(\left[W^H\right]_c^T D^H \cdots W^{i+1} D^{i+1} \right) U^i, \tag{11}$$

where U^i is unknown and is not uniquely determined when the loss functions are nonconvex and nonconcave functions. According to [29], we know that the loss function of multilayer neural networks are nonconvex and nonconcave functions. Due to the dynamicity of U^i, even if $\overline{g}_c$, $\widehat{G}^i$, W^i, and D^i are known, X is not obtained.

Our method hides the correlation between the gradient and the sample, eliminates the information between neurons, and achieves neuron-level security. SSGD is a multi-sample training; there is no information leakage problem like in [19], which is a single-sample leakage of privacy. SSGD can defend against attacks on the gradient.

Since training a model requires rounds of iterations, is it safe to use multiple rounds of iterations? We previously analyzed that the gradient g and the training data $(\hat{x}, y)$ are bijective in terms of parameter θ, i.e., $g = \nabla_\theta f(\theta | (\hat{x}, y))$, where f is a functional relationship. We use θ^i and θ^{i+1} to denote the training parameters of the ith and $i+1$st rounds, respectively. Then, we have $\theta^{(i+1)} = \theta^i - \eta \cdot g^i$. The ith gradient $g^i = \nabla_\theta f(\theta^i | (\hat{x}, y))$. Therefore, we have $\theta^{(i+1)} = \theta^i - \eta \cdot \nabla_\theta f(\theta^i | (\hat{x}, y))$. Furthermore, we obtain $g^{(i+1)} = \nabla_\theta f(\theta^i - \eta \cdot \nabla f(\theta^i | (\hat{x}, y)) | (\hat{x}, y))$. By comparing $g^{(i+1)}$ with g^i, we can see that there is not additional information in $g^{(i+1)}$. The information of the model is only related to the training samples, initial parameters, and learning rate. Therefore, the iteration operation does not cause the information leakage.

4. Experiments

Data. We use MNIST (https://yann.lecun.com/exdb/mnist) and Fashion-MNIST (https://fashion-mnist.s3-website.eu-central-1.amazonaws.com) datasets to assess the performance of our algorithm. The MNIST contains 60000 training images and 10000 test images, where every image is a 28×28 grayscale image, and each pixel is an octet. The Fashion-MNIST [30] is composed of 28×28 grayscale images of 70,000 fashion products from 10 categories, with 7,000 images per category. The training set and test set contain 60,000 images and 10,000 images, respectively.

Model. The lenet-5 [31] contains two convolutional layers, two pooling layers, and three fully connected layers. The activation function is ReLu. The input dimensions are 784, and output dimensions are 10.

Evaluation Index (The Test Accuracy). We use 60000 training images to train model. The test accuracy is the average value of ten experimental results, and every experiment obtains

the average test accuracy of randomly selecting 1000 samples from the test set. The number of iterations is 10,000. The highest test accuracy of these compared algorithms in the same experimental environment is shown in bold.

4.1. Accuracy and Efficiency. We compare SSGD with SGD, SGDm [32], and Adam [33], which are widely used gradient descent algorithms. The batch size ($N = m \times n$) is set to 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, and 8192, where n is set to 1, 4, 8, 16, 32, 64, and 128 and m is set to 4, 8, 16, 32, and 64. When $m = 1$, it is the MBGD. There is not set same learning rate as a good experimental result, because SGD and SGDm have poor adaptability in large batches.

For MNIST data set, the momentum of SGDm is set to 0.999. For the experimental parameters of Adam, the learning rate is set to 5×10^{-4}, and β_1 and β_2 are set to 0.9 and 0.999, respectively. For SSGD, the learning rate in this experiment is set to 10^{-1}. For Fashion-MNIST data set, the momentum of SGDm is set to 0.99. For the experimental parameters of Adam, the learning rate is set to 10^{-3}, and β_1 and β_2 are set to 0.9 and 0.999, respectively. For SSGD, the learning rate in this experiment is set to $10^{-2}/1.5^{0.002j}$, where j is the number of iterations.

The comparative experimental results of SGD, SGDm, Adam, and SSGD are shown in Tables 1 and 2, where the numbers in bracket in the second and third columns denote the learning rates of SGD and SGDm, respectively, and the number in bracket in the fifth column is the value of m. From Tables 1 and 2, we can see that the performance of our algorithm is better than that of SGD, SGDm, and Adam for large batches of data. In this case, SGD and SGDm need to reduce the learning rate to adapt to it. And Adam also has obvious overfitting in large batches of data. SSGD has always maintained high precision. On the whole, our algorithm on test accuracy is better and more stable than SGD, SGDm, and Adam.

Tables 3 and 4 show the running results of our SSGD approach in different numbers of training batches. We can see that the larger the number of training batches ($N = m \times n$) is, the better the test accuracy is. When the number of training batches is too small, the effect of n on performance is greater than that of m. The distribution of m values in Tables 1 and 2 also shows this point.

The convergence rate graphs on MNIST and Fashion-MNIST are shown in Figures 2(a) and 2(b), respectively. The value in longitudinal axis is the average accuracy of every 10 iterations. The SSGDm is SSGD with momentum. We choose the intermediate value 256 as the batch number in the convergence experiment, where $n = 16$ and $m = 16$ for SSGD and SSGDm. In Figure 2(a), the learning rates of SGD and SGDm are 10^{-4} and 5×10^{-4}, respectively. The momentum of SGDm is set to 0.999. The learning rate of SSGDm is $10/1.0002^j$, where j is the number of iterations, and its momentum is 0.99. The other parameters are consistent with the above experiment on MNIST. In Figure 2(b), we choose the larger batch number 1024 as the batch number in the convergence experiment, where $n = 64$ and $m = 16$ for SSGD and SSGDm. The learning rates of SGD and SGDm are 10^{-5} and 10^{-3}, respectively. The momentum of SGDm is set to 0.99. The other parameters are consistent with the above experiment on Fashion-MNIST. The learning rate of SSGDm is $1/1.5^{0.002j}$, where j is the number of iterations, and its momentum is 0.9. From Figure 2, we can see

Table 1: The test accuracy of compared algorithms on MNIST.

$N = m \times n$	SGD (η)	SGDm (η)	Adam	SSGD (m)
16	0.9781 (5×10^{-4})	0.9778 (5×10^{-4})	0.9767	**0.9832** (4)
32	0.9702 (5×10^{-4})	0.9768 (5×10^{-4})	0.9850	**0.9876** (8)
64	0.9794 (5×10^{-4})	0.9792 (5×10^{-4})	0.9842	**0.9900** (8)
128	0.9676 (5×10^{-4})	0.9780 (5×10^{-4})	0.9896	**0.9901** (16)
256	0.9755 (10^{-4})	0.9804 (5×10^{-4})	0.9855	**0.9877** (16)
512	0.9665 (10^{-4})	0.9789 (5×10^{-4})	0.9814	**0.9861** (16)
1024	0.9738 (10^{-5})	0.9749 (10^{-4})	0.9778	**0.9869** (16)
2048	0.9763 (10^{-5})	0.9717 (10^{-4})	**0.9894**	0.9886 (64)
4096	0.9703 (10^{-5})	0.9806 (10^{-4})	0.9788	**0.9878** (64)
8192	0.9785 (2×10^{-6})	0.8994 (10^{-4})	0.9753	**0.9855** (64)

Table 2: The test accuracy of compared algorithms on fashion-MNIST.

$N = m \times n$	SGD (η)	SGDm (η)	Adam	SSGD (m)
16	**0.8253** (10^{-4})	0.7795 (10^{-3})	0.7175	0.8035 (4)
32	**0.8241** (10^{-4})	0.8031 (10^{-3})	0.7513	0.8046 (4)
64	**0.8468** (10^{-4})	0.8163 (10^{-3})	0.7674	0.8344 (4)
128	**0.8629** (10^{-4})	0.8331 (10^{-3})	0.7835	0.8437 (4)
256	0.8527 (10^{-4})	0.8511 (10^{-3})	0.8252	**0.8602** (4)
512	**0.8682** (10^{-4})	0.8569 (10^{-3})	0.8566	0.8590 (4)
1024	0.8569 (10^{-5})	0.8612 (10^{-3})	0.8547	**0.8668** (16)
2048	0.8457 (10^{-5})	0.8351 (10^{-4})	0.8511	**0.8652** (32)
4096	0.8629 (10^{-6})	0.8518 (10^{-4})	0.8321	**0.8704** (32)
8192	0.8325 (10^{-6})	0.8278 (10^{-4})	0.8144	**0.8648** (64)

Table 3: Test accuracy of SSGD on MNIST.

	$n = 1$	$n = 4$	$n = 8$	$n = 16$	$n = 32$	$n = 64$	$n = 128$
$m = 4$	0.9717	0.9832	0.9842	0.9846	0.9895	0.9886	0.9888
$m = 8$	0.9815	0.9876	0.9900	0.9884	0.9855	0.9861	0.9849
$m = 16$	0.9801	0.9854	0.9901	0.9877	0.9861	0.9869	0.9832
$m = 32$	0.9851	0.9804	0.9901	0.9833	0.9819	0.9867	0.9828
$m = 64$	0.9830	0.9849	0.9831	0.9833	0.9886	0.9878	0.9855

Table 4: Test accuracy of SSGD on Fashion-MNIST.

	$n = 1$	$n = 4$	$n = 8$	$n = 16$	$n = 32$	$n = 64$	$n = 128$
$m = 4$	0.7739	0.8035	0.8046	0.8344	0.8437	0.8602	0.8590
$m = 8$	0.8152	0.8136	0.8150	0.8271	0.8401	0.8564	0.8655
$m = 16$	0.8137	0.8176	0.8246	0.8446	0.8446	0.8668	0.8663
$m = 32$	0.8189	0.8125	0.8198	0.8353	0.8574	0.8652	0.8704
$m = 64$	0.8215	0.8196	0.8238	0.8289	0.8577	0.8655	0.8648

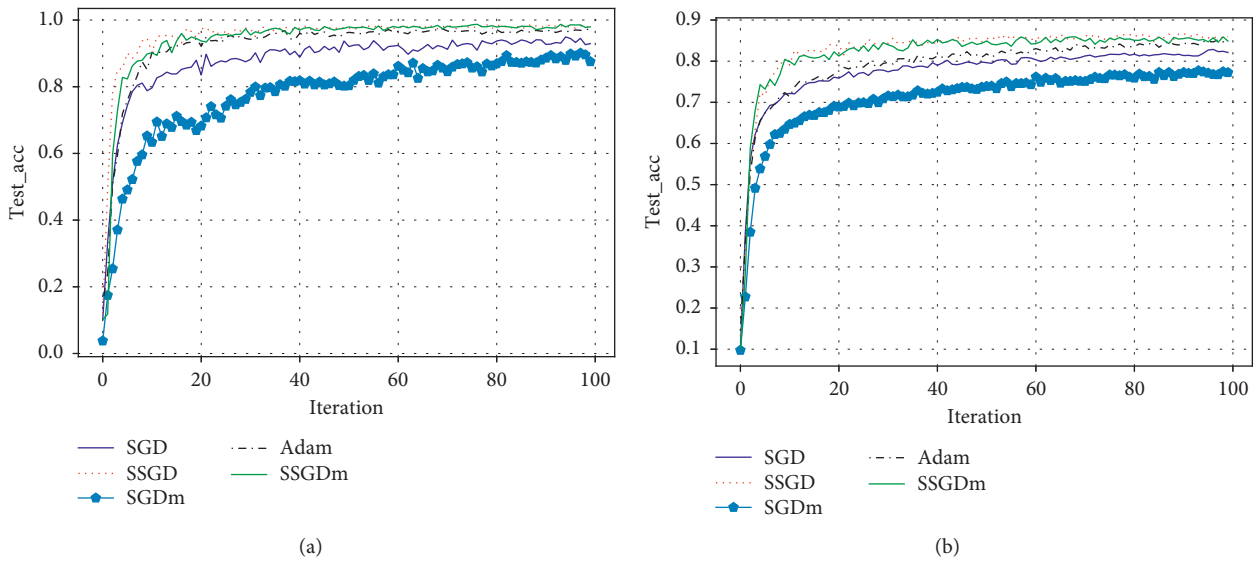


Figure 2: The convergence speed of test accuracy (a) on MNIST and (b) on Fashion-MNIST.

that the convergence speed of our algorithm is faster and more stable than SGD, SGDm, and Adam.

4.2. Robustness. Robustness is the robustness of the system, which refers to the characteristic that the system maintains a certain performance under certain parameter perturbations. To check the robustness of our algorithm, we add random noise to the gradient. At the same time, we noticed that differential privacy is a way to protect gradient information by adding random noise that meets a certain distribution. To compare the performances of our algorithm and the model with differential privacy, we choose the model in the robustness experiment to add noise that satisfies differential privacy. In this section, we compare the performances of the traditional gradient descent algorithm and SSGD with noises. In [20], the large gradient does not participate in the update, which will seriously affect the gradient descent performance. However, the large gradient participating in the update will cause the noise scale to be too large, which makes the algorithm effect extremely poor or even unable to converge. Different from cutting gradient value in [20], we strictly define sensitivity as the maximum value minus the minimum value in the gradient matrix. We add Laplacian noises of the same scale on comparing algorithms and set privacy budget $\varepsilon = 4$ and $\varepsilon = 2$ on MNIST and Fashion-MNIST, respectively.

We use SGDm and Adam as the compared algorithms. Also, we have tested SGD algorithm. When noise or the number of batches is large, the gradient explosion will occur and the SGD cannot converge on MNIST. SGDm and Adam algorithms have better robustness. Because both SGDm and Adam have momentum, SSGDm is chosen as our comparison algorithm. We adjust hyper parameters to get more performance for SGDm and Adam with noises. To make SGDm, Adam, and SSGDm experiments in the same environment, the batch number is $N = n \times m$, where n is set to 4, 8, 16, 32, and 64, and m is set to 4, 8, 16, 32, and 64. For each iteration, after the n vectors are added, the Laplace noises of $\varepsilon = 4$ or $\varepsilon = 2$ that strictly meet the differential privacy are added. The sensitivity is set to the maximum value minus the minimum value of the gradient matrix of the same batch. Then, we use SGDm, Adam, and SSGDm algorithms to update their parameters, respectively. For SGDm, the momentum is 0.99. The learning rate is 10^{-2} and 10^{-3} on MNIST and Fashion-MNIST, respectively. For Adam, the learning rate is 10^{-3} on MNIST and Fashion-MNIST, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. For SSGDm, we use the average of multiple-unit gradient vectors to update the gradient. Therefore, the module length of the update gradient vector decreases very slowly, and dynamic learning rates need to be set. The learning rate of SSGDm is set to $10/1.0002^j$ and $1/1.5^{0.002j}$on MNIST and Fashion-MNIST, respectively. The momentum = 0.9.

From Tables 5 and 6, all three algorithms comply with the law of acquaintance; that is, the larger the batch size is, the better the accuracy is. We can see that SSGDm is more robust than the SGDm and Adam algorithms when the noises of the same scale are added in gradients on test accuracy. On MNIST, compared with SGDm and Adam, the average test accuracy of SSGDm is increased by 4.12% and 1.60%, respectively. On Fashion-MNIST, compared with SGDm and Adam, the average test accuracy of SSGDm is increased by 5.24% and 1.64%, respectively.

Where is the limit of the robustness of our algorithm? On MINST, we try to increase the scale of noises and make ε be 0.2, 0.5, 1, 2, and 4. The experimental environment is the same as the robustness experiment above, and the parameter settings are also the same. The batch number is set to $n = 16$ and $m = 16$. On Fashion-MNIST, we make ε be 0.5, 1, 2, and 4. The batch number is set to $n = 64$ and $m = 16$. From Tables 7 and 8, it is clear that our SSGDm has obvious advantages in robustness. The greater the scale of noises is, the more obvious the advantage of our algorithm is.

Table 5: Test accuracy with $\varepsilon = 4$ on MINST.

SGDm\Adam\SSGDm	$n = 4$	$n = 8$	$n = 16$	$n = 32$	$n = 64$
$m = 4$	0.8714\0.9321**0.9730**	0.9299\0.9515**0.9816**	0.9508\0.9592**0.9822**	0.9559\0.9567**0.9774**	0.9518\0.9715**0.9851**
$m = 8$	0.9242\0.9570**0.9785**	0.9337\0.9657**0.9829**	0.9582\0.9737**0.9790**	0.9569\0.9569**0.9832**	0.9625\0.9797**0.9802**
$m = 16$	0.9471\0.9607**0.9684**	0.9390\0.9606**0.9839**	0.9662\0.9723**0.9833**	0.9674\0.9726**0.9844**	0.9699\0.9796**0.9860**
$m = 32$	0.9203\0.9580**0.9780**	0.9333\0.9693**0.9805**	0.9594\0.9658**0.9859**	0.9647\0.9763**0.9838**	0.9758\0.9806**0.9837**
$m = 64$	0.9163\0.9545**0.9772**	0.9514\0.9771**0.9861**	0.9560\0.9715**0.987**	0.9710\0.9702**0.9853**	0.9678\0.9833**0.9885**

Table 6: Test accuracy with $\varepsilon = 2$ on Fashion-MNIST.

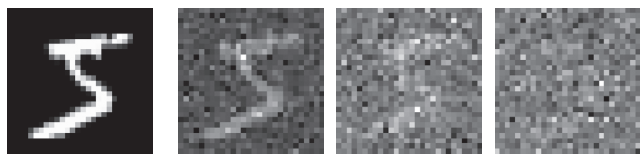
SGDm\Adam\SSGDm	$n = 4$	$n = 8$	$n = 16$	$n = 32$	$n = 64$
$m = 4$	0.7089\0.7177**0.7732**	0.7623\0.7729**0.7875**	0.7894\0.7813**0.7947**	0.7941\0.8180**0.8233**	0.8108\0.8192**0.8343**
$m = 8$	0.7123\0.7402**0.7885**	0.7658\0.7548**0.8040**	0.7806\0.8022**0.8175**	0.8044\0.8242**0.8412**	0.8159\0.8373**0.8459**
$m = 16$	0.7127\0.7723**0.7903**	0.7763\0.8052**0.8139**	0.7724\0.8173**0.8389**	0.7993\0.8291**0.8474**	0.8199\0.8386**0.8455**
$m = 32$	0.7159\0.8013**0.8177**	0.7669\0.8066**0.8333**	0.7735\0.8339**0.8467**	0.8103\0.8540**0.8611**	0.8299\0.8553**0.8597**
$m = 64$	0.7158\0.8140**0.8235**	0.7432\0.8309**0.8412**	0.7853\0.8497**0.8524**	0.8064\0.8586**0.8647**	0.8283**0.8663**\0.8655

Table 7: The test accuracy by varying ε on MINST.

	$\varepsilon = 0.2$	$\varepsilon = 0.5$	$\varepsilon = 1$	$\varepsilon = 2$	$\varepsilon = 4$
SGDm	0.0970	0.3745	0.8166	0.9344	0.9662
Adam	0.8074	0.8813	0.9140	0.9416	0.9723
SSGDm	**0.8481**	**0.9395**	**0.9671**	**0.9719**	**0.9833**

Table 8: The test accuracy by varying ε on fashion-MNIST.

	$\varepsilon = 0.5$	$\varepsilon = 1$	$\varepsilon = 2$	$\varepsilon = 4$
SGDm	0.0995	0.5921	0.8199	0.8363
Adam	0.6474	0.7934	0.8386	0.8524
SSGDm	**0.7587**	**0.8106**	**0.8455**	**0.8618**



(a) (b)

Figure 3: Training sample image of MNIST. (a) The original image. (b) From left to right are the poisoned images with $\varepsilon = 5$, 2, and 1, respectively.

Table 9: Test accuracy by varying ε on MNIST.

	$\varepsilon = 1$	$\varepsilon = 2$	$\varepsilon = 5$
SGD	0.1095	0.9171	0.9679
Adam	0.4859	0.8160	0.8890
SSGD	**0.9446**	**0.9621**	**0.9827**

Table 10: Test accuracy by varying ε on Fashion-MNIST.

	$\varepsilon = 1$	$\varepsilon = 2$	$\varepsilon = 5$
SGD	0.1012	0.5813	0.7969
Adam	0.2888	0.3452	0.6296
SSGD	**0.4638**	**0.7651**	**0.8290**

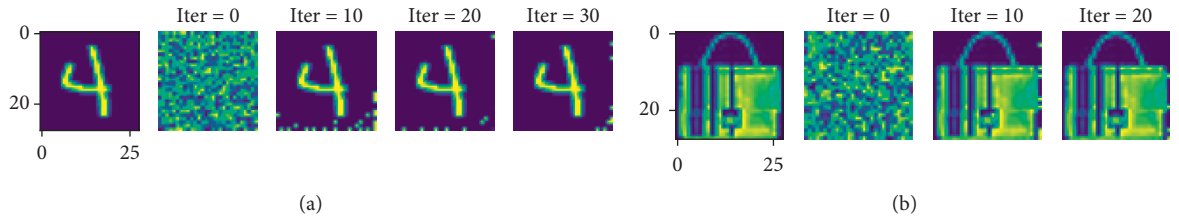


Figure 4: DLG attacks SGD (a) on MINST dataset and (b) on Fashion-MNIST dataset.

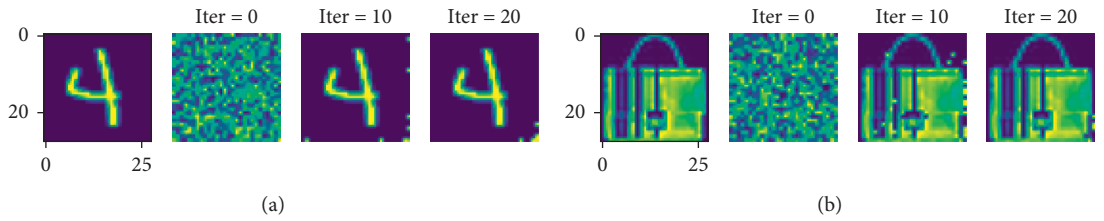


Figure 5: iDLG attacks SGD (a) on MINST dataset and (b) on Fashion-MNIST dataset.

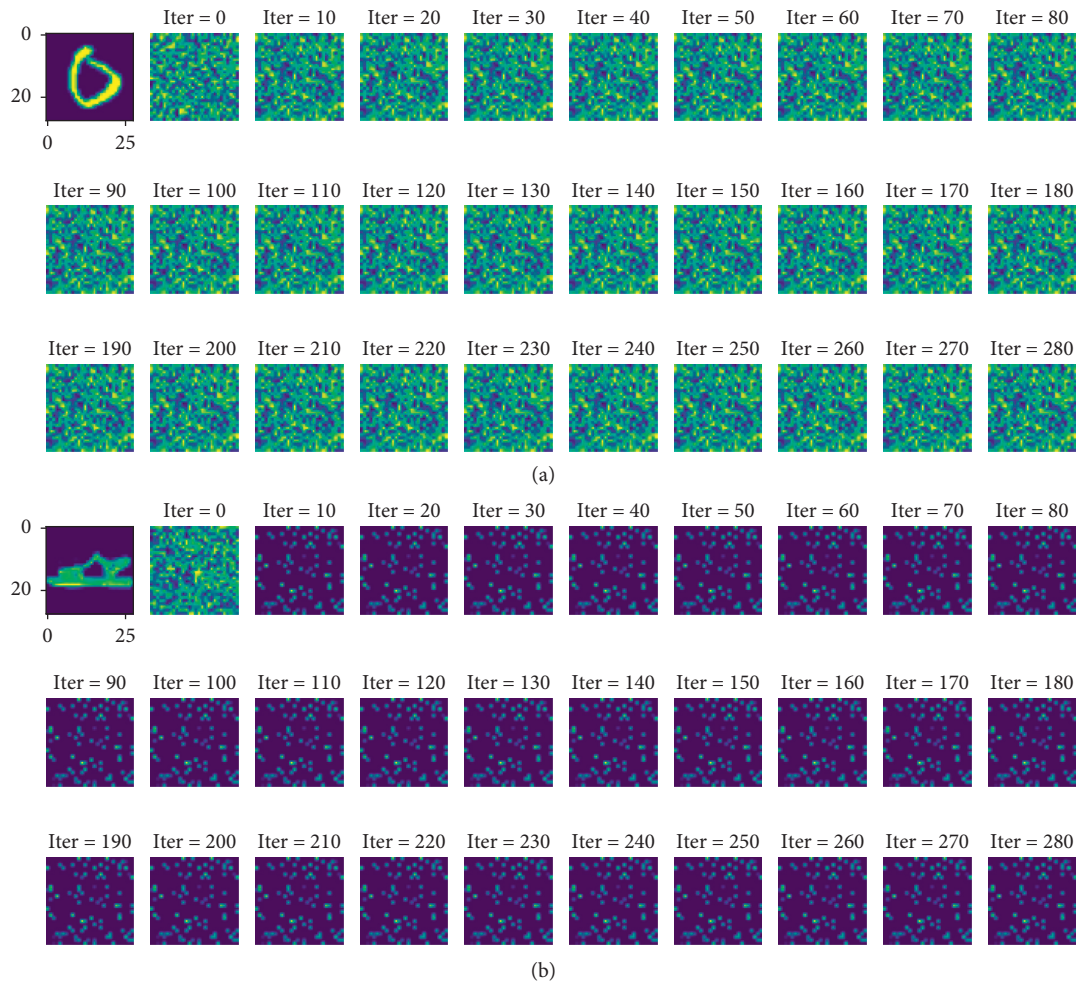


Figure 6: DLG attacks SSGD (a) on MINST dataset and (b) on Fashion-MNIST dataset.

4.3. Poisoning Attack. The goal of poisoning attack is to destroy the integrity and availability of data. The robustness experiment results show that our algorithm can resist the poisoning attack added to the gradient to a certain extent. According to the previous analysis, the gradient is a kind of mapping of the training data. Then, our algorithm should be

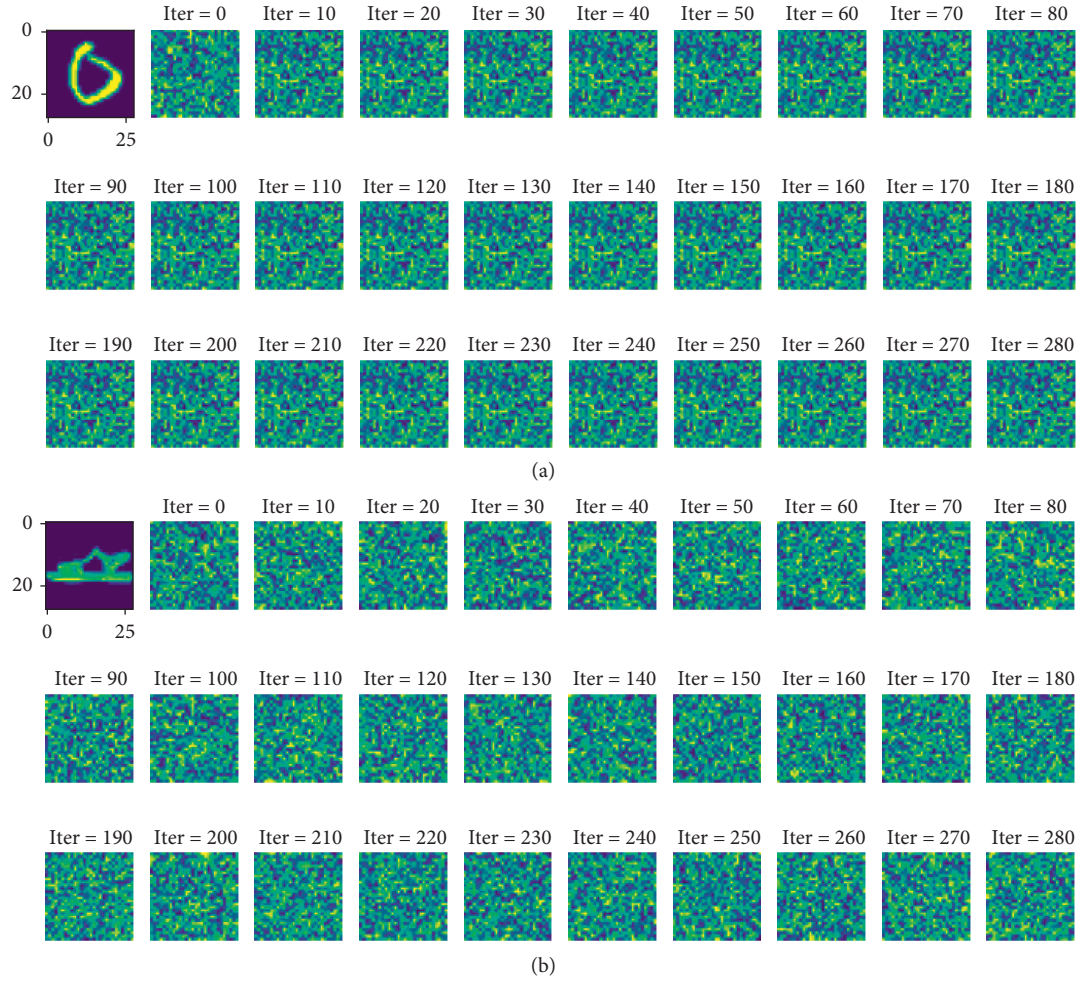


Figure 7: iDLG attacks SSGD (a) on MINST dataset and (b) on Fashion-MNIST dataset.

effective against data poisoning attacks. This part of the experiment is to verify the performance of our algorithm in data poisoning attacks.

SGD is a more basic gradient descent method. In this experiment, we chose SGD as compared algorithm. This experiment compares the performance of SGD, Adam, and SSGD on the same data set with noises. To determine the scale of added noises, the differential privacy mechanisms still are used to add noises with the same methods as the robustness experiment. We add Laplacian noises of different scales to 60,000 training samples. The evaluation method of the experiment result is the same as the above experiment. On MNIST, the batch number is set to $n = 64$ and $m = 4$. The learning rate of SGD, Adam, and SSGD is 10^{-4}, 10^{-4}, and 10^{-2}, respectively. On Fashion-MNIST, the batch number is set to $n = 64$ and $m = 16$. The learning rate of SGD, Adam, and SSGD is 10^{-4}, 10^{-4}, and $10^{-2}/1.5^{0.002j}$, respectively. The other settings are the same as the above experiment.

Figure 3 is the effect picture after adding different noise scales. From Tables 9 and 10, we can see that SSGD is significantly better than SGD and Adam in test accuracy. Also, our algorithm still maintains a higher test accuracy while continuously increasing the scale of noises. Therefore, SSGD can resist gradient poisoning attacks and parametric poisoning attacks to a certain extent.

4.4. Data Reconstruction Attack. Zhu et al. [7] presented an approach which shows the possibility of obtaining private training data from the publicly shared gradients. In their deep leakage from gradient (DLG) method, they synthesized the dummy data and corresponding labels with the supervision of shared gradients. Specifically, they start with random initialization of pseudodata and labels. Virtual gradients are computed on the current shared model in the distributed setup. By minimizing the difference between the virtual gradient and the shared real gradient, they iteratively update the virtual data and labels simultaneously. iDLG [4] is an improvement based on DLG. The following experimental diagrams include the experimental results of DLG [7] and iDLG [4] attacking SGD and SSGD algorithms on MINST datasets and Fashion-MNIST datasets. Figures 4 and 5 are the experimental results of DLG and iDLG attacking SGD. Figures 6 and 7 are about the experimental results of DLG and iDLG attacking SSGD. The number of iterations is 300, and the iteration is stopped if the

predetermined accuracy is reached. We can see that our algorithm can defend against DLG and iDLG.

5. Conclusions

In this paper, we propose a new gradient descent approach, called super stochastic gradient descent. The SSGD enhances the randomness of gradients to protect against gradient-based attacks. Simultaneously, we use multisample aggregation to enhance stability and eliminate the uncertainty brought about by superrandomness. Our approach achieves neuron-level security and can defend against attacks on the gradient. Experimental results demonstrate that SSGD has good accuracy and strong robustness because its stability and randomness are enhanced. SSGD can also resist model poisoning attacks to a certain extent. But for attacks with the same degree of poisoning, data poisoning has a greater impact on performance. In the future, we will continue to find a more suitable method for resisting data poisoning attacks.

Data Availability

All the experimental data used to support the findings of this study are included within the article.

Disclosure

An earlier version of this study's preprint is given in the following link: https://arxiv.org/abs/2012.02076.

Conflicts of Interest

The authors declare that they have no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (nos. 61672176 and 61763003), Research Fund of Guangxi Key Lab of Multi-Source Information Mining and Security (no. 19-A-02-01), Guangxi 1000-Plan of Training Middle-Aged/Young Teachers in Higher Education Institutions, Guangxi "Bagui Scholar" Teams for Innovation and Research Project, Guangxi Talent Highland Project of Big Data Intelligence and Application, and Guangxi Collaborative Innovation Center of Multi-source Information Integration and Intelligent Processing.

References

[1] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, https://arxiv.org/abs/1609.04747.

[2] L. Yang and D. Cai, "AdaDB: an adaptive gradient method with data-dependent bound," *Neurocomputing*, vol. 419, pp. 183–189, 2021.

[3] L. Melis, C. Song, E. D. Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pp. 691–706, San Francisco, CA, USA, May 2019.

[4] B. Zhao, K. R. Mopuri, and H. Bilen, "iDLG: improved deep leakage from gradients," 2020, https://arxiv.org/abs/2001.02610.

[5] B. Hitaj, G. Ateniese, and F. Pérez-Cruz, "Deep models under the GAN: Information leakage from collaborative deep learning," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 603–618, Dallas, TX, USA, November 2017.

[6] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, "Beyond inferring class representatives: user-level privacy leakage from federated learning," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, pp. 2512–2520, Paris, France, April 2019.

[7] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," in *Proceedings of the Annual Conference on Neural Information Processing Systems 2019 (NeurIPS)*, pp. 14747–14756, Vancouver, Canada, December 2019.

[8] J. Geiping, H. Bauermeister, H. Droge, and M. Moeller, "Inverting gradients-how easy is it to break privacy in federated learning?" in *Proceedings of the Annual Conference on Neural Information Processing Systems 2020 (NeurIPS) Virtual Event*, December 2020.

[9] X. Pan, M. Zhang, Y. Yan, J. Zhu, and M. Yang, "Theory-oriented deep leakage from gradients via linear equation solver," 2020, https://arxiv.org/abs/2010.13356.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, Las Vegas, NV, USA, June 2016.

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proceedings of the International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, May 2015.

[12] G. Huang, Z. Liu, L. V. D. Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269, Honolulu, HI, USA, July 2017.

[13] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," 2014, https://arxiv.org/abs/1404.5997.

[14] N. Ma, X. Zhang, H. Zheng, and J. Sun, "Shufflenet V2: practical guidelines for efficient CNN architecture design," in *Proceedings of the 15th European Conference on Computer Vision (ECCV)*, pp. 122–138, Munich, Germany, September 2018.

[15] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826, Las Vegas, NV, USA, June 2016.

[16] C. Szegedy, W. Liu, Y. Jia et al., "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, Boston, MA, USA, June 2015.

[17] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Mobilenetv2: inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4510–4520, Salt Lake City, UT, USA, June 2018.

[18] K. Bonawitz, V. Ivanov, B. Kreuter et al., "Practical secure aggregation for privacy-preserving machine learning," in *Proceedinegs of the ACM SIGSAC Conference on Computer*

and Communications Security (CCS), pp. 1175–1191, Dallas, TX, USA, October 2017.

[19] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, "VerifyNet: secure and verifiable federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 911–926, 2020.

[20] X. Guo, Z. Liu, J. Li et al., "VeriFL: communication-efficient and fast verifiable aggregation for federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 1736–1751, 2021.

[21] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, "Secure single-server aggregation with (Poly) logarithmic overhead," in *Proceedings of the ACM SIGSAC Conference on Computer and communications security (CCS)*, pp. 1253–1269, Virtual Event, USA, November 2020.

[22] B. Choi, J.-y. Sohn, D.-J. Han, and J. Moon, "Communication computation efficient secure aggregation for federated learning," 2020, https://arxiv.org/abs/2012.05433.

[23] H. Fereidooni, S. Marchal, M. Miettinen et al., "SAFELearn: Secure aggregation for private FEderated learning," in *Proceedings of the 2021 IEEE Security and Privacy Workshops (SPW)*, pp. 56–62, Virtual Event, May 2021.

[24] X. Ma, C. Ji, X. Zhang et al., "Secure multiparty learning from the aggregation of locally trained models," *Journal of Network and Computer Applications*, vol. 167, Article ID 102754, 2020.

[25] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-Preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2018.

[26] M. Abadi, A. Chu, I. J. Goodfellow et al., "Deep Learning with Differential Privacy," in *Proceedings Of the ACM SIGSAC Conference on Computer and communications security (CCS)*, pp. 308–318, Vienna, Austria, October 2016.

[27] K. Yadav, B. B. Gupta, K. T. Chui, and K. E. Psannis, "Differential privacy approach to solve gradient leakage attack in a federated machine learning environment," in *Proceedings of the International Conference on Computational Data and Social Network (CSoNet)*, pp. 378–385, Dallas, TX, USA, December 2020.

[28] Q. Zhao, C. Zhao, S. Cui, S. Jing, and Z. Chen, "PrivateDL: privacy-preserving collaborative deep learning against leakage from gradient sharing," *International Journal of Intelligent Systems*, vol. 35, no. 8, pp. 1262–1279, 2020.

[29] K. Kawaguchi, "Deep learning without poor local minima," in *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, pp. 586–594, Barcelona, Spain, May 2016.

[30] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms," 2017, https://arxiv.org/abs/1708.07747.

[31] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[32] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, no. 1, pp. 145–151, 1999.

[33] D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization," in *Proceedings of the International Conference On Learning Representations (ICLR)*, San Diego, CA, USA, May 2015.