"Virtual user" attack to distributed machine learning*

†

ABSTRACT

Nowadays deep learning method is becoming more and more popular. It shows power on image classification with high precision . Recently, commercial companies such as Google and Amazon have built some deep learning service platforms. Users upload their data to the platform and raise their requirements, then the platform automatically trains a model to help users make decisions.

But this will bring some security issues. The users' sensitive data will be exposed to the company. In order to solve this problem, at CCS'15 Shokri and Shmatikov proposed a distributed training method that allows multiple users to collaboratively train a model. Users only need to upload a part of parameters without revealing their sensitive data to the company, which protects users' privacy.

However, we find that even in the case of distributed training, there is still potential possibility of sensitive data leakage. We propose a "virtual user" attack. When the company is malicious, it can apply this kind of attack acquiring the users' sensitive data, even without being noticed by users. We show that the sensitive data can be stolen including passwords, private photos and even more important information. It may causes great hidden trouble to the data security of users and reputation of commercial companies. Later, we propose a "finetune" training method to perfectly defend against this attack, which only takes a little extra time of users. We hope our article will serve as a warning to the public.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**; **Distributed systems security**;

KEYWORDS

Deep learning; Privacy; Attack; Defense

1 INTRODUCTION

Deep learning is a branch of machine learning technique. It takes advantage of a large number of statistical information and has shown amazing accuracy in the fields of session recognition, image recognition and object detection. Compared with previous statistical learning methods, deep learning technology has a huge amount of parameters and needs a lot of computation. It has extremely high requirements on the computing power of the machine. Usually we need to use GPU to perform calculations, but the high price of GPU is unbearable for common users.

For this reason, commercial companies have launched many profitable platforms using deep learning method, which enable common users to use large companies' GPUs. After gathering the data uploaded by the users, through online calculation, the platform builds an accurate model for users. For example, if a user uploads some animal pictures, the platform will return a classification model to the user to identify various species of animals.

However, the data security has become the focus of attention. Here are some security issues that may arise from the centralized training platform. For example, the leakage of people's ID number may cause property damage.

In this case, Shokri and Shmatikov propose a distributed training approach[10]. In the distributed training scenario, users do not need to upload data to the platform. They only need to train with their own data locally, randomly upload certain parameters to the platform, and download some parameters to update their local neural network. Finally users can get a model with high degree of accuracy. They can use all users' data to train a more powerful and ubiquitous model without exposing their local data to each other. Therefore, the distributed training method has quickly become focus of attention because it has two obvious advantages:

1.It can protect data privacy.

2. multiple users make full use of their data and train a more powerful model.

However, we find that this distributed training method has great security risks. We propose an attack method to obtain sensitive data of users in this case. Our attack takes advantage of GAN. Its main purpose is to generate fake data most like real data. The basic framework of the original GAN model consists of a generative model and a discriminative model. The initial input of Generator is a random noise signal and the purpose of Generator is to produce data as real as possible. The discriminator outputs a probability, indicating the confidence of whether the input is real or fake data. The two models are trained synchronously to improve each other's performance. When finally discriminator cannot distinguish between the real data and the generated data, the generator is considered to be optimal.

In view of the distributed training mentioned above, we propose a "virtual user" attack. We show that if a company who provides a distributed training platform is malicious, it can build a "virtual user" and uses some special initialization to disguise it, which enables it to be added to real users set and train a model together. During that, we use GAN to generate sensitive data of users. We use the obtained model as a discriminator to determine whether the data we generate is the users' sensitive data, and ultimately get some data similar to users'. Here we emphasize that although there are some data encryption measures, such as secure multi-party computing, homomorphic encryption, etc. can make the transmitted information completely invisible. As we know that in practical applications, due to the huge amount of data of deep learning, which puts extremely high requirements on communication technology, any encryption method cannot be applied in practice. Therefore, the attack we have proposed is completely plausible.

This kind of attack is difficult to be noticed by users. First, in the training process, although our attack adds a "virtual user", it neither significantly increases the training time nor obviously causes a rise

*

†

in memory consumption. Furthermore, after the training is completed, the users can still get a usable model. Compared with the model trained when the "virtual user" attack is not implemented, the accuracy and user experience will not significantly change. Therefore, the attack we proposed is covert and not easy to be detected by users. In conclusion, it may cause serious privacy leakage unconsciously.

Then we propose a defense measure, here we take advantage of the "finetune" training method. The "finetune" method is one of the transfer learning methods. It is often the case that a previously trained model (generally, this model is based on a larger dataset) is used for current mission requirements(a smaller dataset). In the field of deep learning, because the process of training is complex, training from scratch usually takes a long time, so training with finetune can save a lot of time.

Our defense is inspired by the "finetune" training method. It can effectively reduce the accuracy of the above "virtual user" attack. The main process is as follows: the users perform certain process on the data in advance (for example, lowering the pixels of the image), train it as real data, and obtain a model from distributed learning. After downloading the model, users perform "finetune" method on it. Because the training data is pre-processed, this will make the "virtual user" model inaccurate, ultimately the malicious company can only obtain fake data. Thus we perfectly protect users' sensitive data from being leaked to the malicious companies.

We emphasize that our "finetune" method does not significantly increase the training time consumption. Since users have already done major training process on the platform, they only need to do a little local fine-tuning with real data. Therefore,the defense we propose are feasible.

Here we make a summary, the main contributions of our article are:

1. For distributed training system, we propose a potential threat that companies can apply a "virtual user" attack to acquire user's sensitive data.

2. We propose a "finetune" defense that allows users to protect their data and avoid the privacy leakage. And this approach does not significantly increase the cost of time.

2 RELATED WORK

2.1 Distributed training

In the research field of machine learning privacy, Reza Shokri proposed a distributed training approach that allows users to train a model together without exposing data to the company[10]. The distributed training approach can get a model with excellent precision, which not only protects the user's privacy, but also allows model to be shared among users. Users even do not have to truly "see" the sensitive data.

Given the scenario of distributed training, Briland Hitaj [6]proposed that although the distributed method well protects the user's sensitive data from the company, there is still a potential threat: If a user is malicious, he can destroy the model by deliberately uploading "fake" data, and obtain sensitive data of other users.

Later, in the case of distributed training, Le Trieu Phong[1] proposed a potential threat, claiming that despite of the distributed training method, malicious platform can still obtain users' sensitive data without being perceived by users. They further proposed a method of homomorphic encryption, which can completely eliminate this malicious attack and protect users' sensitive data. However, we point out again that the current homomorphic encryption method cannot be applied to the actual scene due to excessive time cost.

2.2 Machine learning privacy

In the field of privacy inference of machine learning model, there have been a series of works proposed in recent years. These tasks are mainly divided into two aspects, the black box (the attacker does not know the parameters inside the model and can only access the API) and the white box (the attacker knows the parameters inside the model).

At S&P' 17, Reza Shokri[11] proposed an overfitting model could lead to the disclosure of sensitive data. In addition, At CCS'16 Michael Backes [2]proposed that the model of RNA sequence construction will lead privacy leakage due to certain parameters of the model, they proposed two effective attack methods and gave defense. Also,At CCS'15 Matt Fredrikson[3] demonstrate how the trust information returned by many machine learning models is exploited by "model inversion" attacks and infers the user's personal information. Furthermore, At CCS'17 Congzheng Song [12]proposed several methods for inferring sensitive data through parameters of neural networks. At last, At USENIX Security'14 Matthew Fredrikson [4]proposed that the model used in warfarin therapy would pose a threat to the patient's genomic privacy.

2.3 Generative adversarial network

Ian Goodfellow [5]proposed the generative adversarial network. The GAN contains two models: a generative model and a discriminative model. The generator learns how to generate deceptive pictures through the feedback of the discriminator. It has broad prospects in the field of unsupervised learning. Alec Radford [8]proposed DCGAN that convolutional network is introduced into the generator and some improvements are put forward to ensure the stability of training. Augustus Odena [7]proposed SSGAN which apply labeled data to GAN method for semi-supervised learning. At the same time Salimans [9] proposed some helpful tricks for GAN.

3 BACKGROUND

3.1 Deep learning

Deep learning originates from neural network technology. It consists of multilayer perceptron. which connects the input data to the output. For example, input information is some images and output is categories. Each connection between neurons has a weight. A neural network training algorithm is to adjust the weight to the best value ensuring the prediction of the entire network is optimal.

In a multi-layer perceptron, we use a to represent the input and w to represent the weight. There is also a bias b, which is connected to all neurons of the next layer. The neural network calculates the value of the neurons in the next layer by multiplying the value in the previous layer by the weight w and adding the bias b. Each neuron must be applied with an activation function Leaky RELU. We use the largest value output in the last layer as the highest confidence category.

Then, we measure the accuracy of the model by calculating the *LOSS* function. The *LOSS* function is obtained by calculating the difference between the output in the last layer and the standard output, usually with a cross entropy function. Then we use the back-propagation algorithm to calculate the partial derivative of *LOSS* relative to each neuron, which is the gradient. We apply gradient descent algorithm and use this gradient to update the weights. The above is the training process in an epoch. After that we continue to apply the above algorithm to update the parameters and finally get a classification model.

3.2 Distributed Privacy-Preserving Deep Learning

At CCS'15, Reza Shokri proposed a distributed training approach. There are two advantages of this training method.

1. Users can train a model without uploading data to the platform.

2. it enables each user to jointly train a more powerful and ubiquitous model for everyone, making full use of everyone's data.

First, all users need to agree on one neural network architecture, including the inner parameters and API. In the first training section, n users first train their own neural networks with their own data, calculate the *LOSS* function, then get the partial derivative of each weight, finally obtain the gradient of each neuron. Each user uploads a portion of their gradients to the platform.

After the platform collects the gradients of the neurons sent by each user, the largest one is selected among all the gradients of each neuron and is returned to each user.

Receiving the gradient sent by the platform, each user updates the neural weights with the obtained gradient, thereby obtaining a new neural network after one training epoch.

The above is the details of one training epoch. After that, each epoch is like the first epoch, continuously improves the accuracy of the training model, finally obtains the user's local model.

3.3 GAN attack method

Briland Hitaj proposed that in the distributed training scenario, although a malicious platform could not get the users' sensitive data, if a user was malicious, he could infer the sensitive data of other users through GAN.

GAN is inspired by the two-player game in game theory. The two players in the GAN model is composed of a generative model and a discriminative model. Generative model G captures the distribution of sample data, and generates a fake sample as good as the real sample; the discriminative model D is a classifier, which estimates the probability that a sample is derived from the real data. If the sample is from real training data, D outputs a large probability, otherwise, D outputs a small probability. In the process of training, one model parameter is fixed, the other model's weight is updated. In this process, both models try to optimize their networks to form a competition until the two sides reach a dynamic equilibrium (Nash equilibrium). Generative model G can be thought of as generating exactly the same sample as real data. GAN try to optimize the function as follow:

$$\min_{\theta_G}\max_{\theta_D}\sum_{i=1}^{n_+} log f(x_i;\theta_D) + \sum_{j=1}^{n_-} log(1 - f(g(z_j;\theta_G);\theta_D))$$

where x_i and x_j are relatively real data and fake data. f is Discriminator and g is Generator. θ_D and θ_G are relatively the parameter of two networks.

Briland Hitaj [6] proposed that they can train a generative model to generate sensitive data, and the user model obtained by distributed training can be used as a discriminator to optimize the generator's output. Their GAN method is a real-time attack because the Generator is trained synchronously with the distributed learning.

Their method is similar to the way proposed by Augustus Odena [7]and Salimans [9]. They modify the output layer from n to $n + 1$. The additional output is where the "fake" data label is placed.

4 "VIRTUAL USER" ATTACK

In the situation of distributed deep learning method protecting the users' privacy, we find that although the platform cannot directly obtain either the users' data or the trained neural network, the communication of parameters is processed through the platform. In a specific situation, the malicious platform can still obtain the users' sensitive data, as shown in Figure 1. The attack process is as follow.

As shown in Figure 1, we first assumed that there are n users who jointly train a model on a distributed training platform. In the training process, we take an epoch as an example. If the platform wants to obtain the sensitive data of users, as users calculate the gradient of all the neurons locally and upload a part of Δweights, the platform creates a "virtual user".

Then the platform maliciously gives users some "too big" weights as distributed Δweights (here "too big" refers to more than two orders of magnitude of the initial weights, for convenience here we assume 1 relative to 0.01), then users download the artificial Δweights and the local neural network weights become very closed to 1. At the same time, the platform sends "virtual user" the same "too big" weights to the same neuron, randomly giving some weights to remaining neurons (as long as they are the same orders of magnitude with common neural network). In this case, we have such an intuition that the performance of this "virtual user" neural network should be similar with the local neural network of users. (because the most weights of the two neural networks are similar, and the other weights are so small that their differences can be ignored here). So our "virtual user" gets a neural network similar to the one obtained by users locally.

The above is the description in one training epoch. In the later training process, the malicious platform can be back to normal function, continues the work of calculating and distributing Δweights in the original agreement, and updates the "virtual user" parameter in each epoch. This ensures its performance to be similar to the neural network obtained by real users. Finally our "virtual user" builds a model that behaves like real users' model.

As we mentioned before, there has been a series of work in recent years, dedicated to inferring the data in the training set through a machine learning model, whether it is a black box or a white

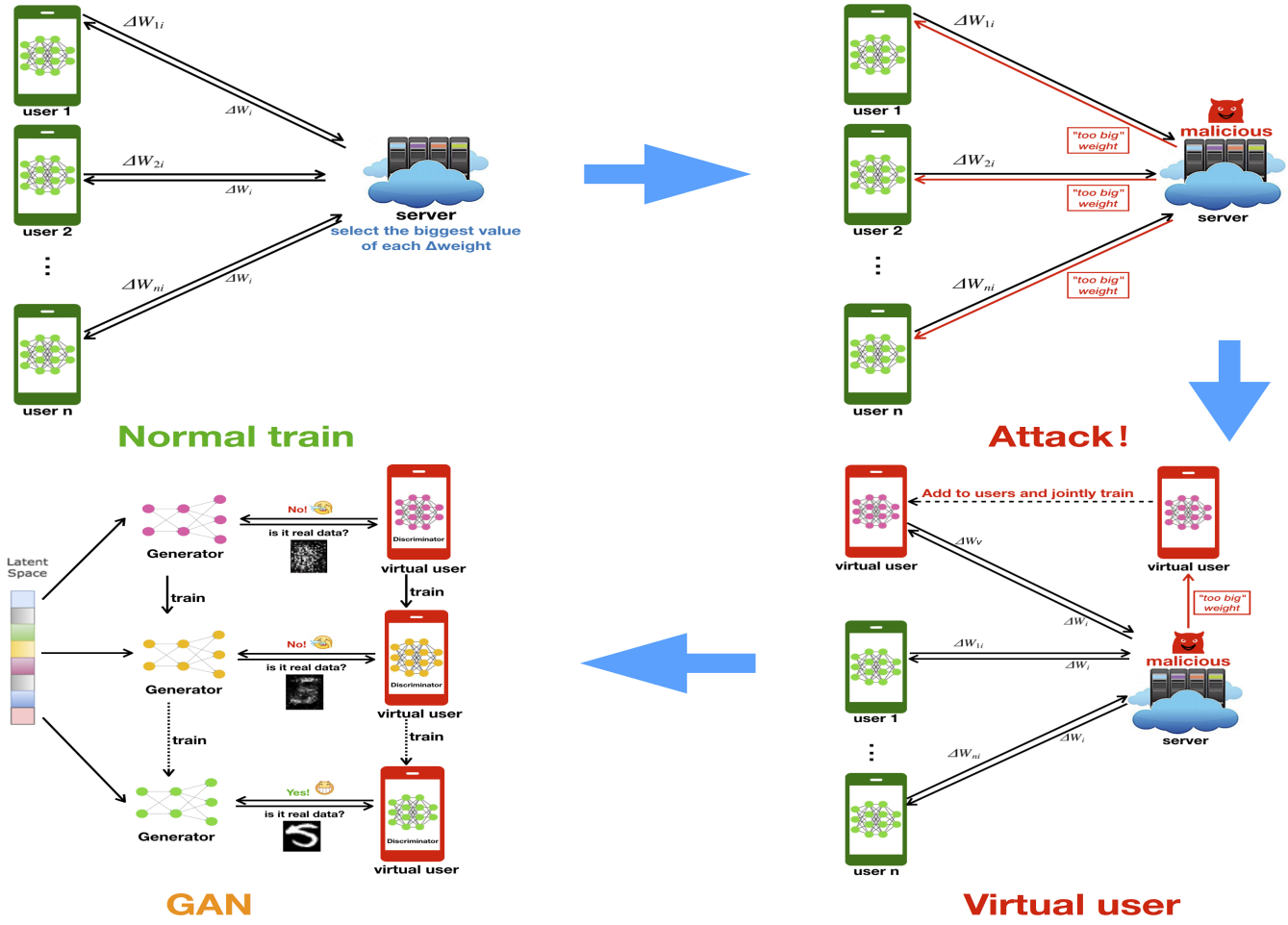


Figure 1: This is "virtual user" attack. n users train a model on a distributed training platform. The platform creates a"virtual user", and maliciously gives users and "virtual user" some "too big" weights (here we assume 1 relative to 0.01). Because the most weights of the two neural networks are similar and bigger, and the other weights are smaller and negligible, we can assert the performance of this "virtual user" neural network should be similar with the real users'.

box (the difference is whether the parameters of neural network can be known). For example, the model inversion method can be used to attack the trained model. In this paper we apply the GAN attack method proposed by Briland Hitaj to the model obtained by our "virtual user" to get the training set data. (This is an obvious intuition. Because the two models behave similarly, their training set data should be similar). Our GAN method is implemented the same as Briland Hitaj proposed. We treat the "virtual user" model as Discriminator, training it synchronously with a Generator which generate fake data. As the "virtual user" model becomes accurate enough, our Generator can generate data similar to the real users.

5 PROOF OF ATTACK FEASIBILITY

Now let's prove the feasibility of our attack which is based on an intuition, namely if most weight values of two neural networks are similar, then the two neural networks will behave similarly. We use $X = (X_1, X_2, ..., X_n)$ to represent the neuron value of each layer, and $X_i = (X_{i_1}, X_{i_2}, ..., X_{i_{m_i}})$. n is the number of layers. $m = (m_1, m_2, ..., m_n)$ is the number of neurons in each layer. W is the weight value, so W_{ij} is the value of weight connecting neuron i and neuron j. We also constructed a "virtual user" network with slight error $E = (E_1, E_2, ..., E_n)$ of weights in each layer, and $E_i = (E_{i_1}, E_{i_2}, ...E_{i_{m_i}})$. $W^{'}$ and $X^{'}$ is the weight value and neuron value of this "virtual user" network. Now we will estimate the output error of the whole network by the error of ith layer weights.

At first we can calculate the jth neuron value of the $i+1$th layer X_{i+1j}:

$$X^{'}_{i+1j} = W^{\top}_{ij} X^{'}_{i} = \sum_{k=1}^{m_i} W_{i_k j} X_{i_k}$$

Then we apply the "leaky RELU" active function to it:

$$\begin{cases} X^{'}_{i+1j} = W^{\top}_{ij} X^{'}_{i}, & X^{'}_{i+1j} \geq 0 \\ X^{'}_{i+1j} = \alpha W^{\top}_{ij} X^{'}_{i}, & X^{'}_{i+1j} < 0, (\alpha < 1) \end{cases}$$

and the error of the jth neuron of $i+1$ th layer is as follow:

(There is an explanation, we can ignore the α of leaky RELU function, the reason we will say in the process of calculating the whole error of this network)

$$\Delta X_{i+1j} = X^{'}_{i+1j} - X_{i+1j} = \sum_{k=1}^{m_i} E_{i_k j} X_{i_k} \tag{1}$$

Thus we get the effect of the ith layer's error on the jth neuron of $i+1$th layer, then we can calculate the effect of the ith layer's error on jth neuron of the last layer:

$$\Delta X_{nj} = X^{'}_{nj} - X_{nj} = \sum_{k=1}^{m_n} \sum_{k=1}^{m_{n-1}} \cdots \sum_{k=1}^{m_i} E_{i_k j} X_{i_k}$$

The effect of the 1th to ith layer's error on jth neuron of the last layer is:

$$\Delta X_{nj} = \sum_{i=1}^{n} \sum_{k=1}^{m_n} \sum_{k=1}^{m_{n-1}} \cdots \sum_{k=1}^{m_i} E_{i_k j} X_{i_k} \tag{2}$$

To calculate the output error, we apply the softmax function to calculate the probability of ith category. To make the equation clearer, we denote the value of last layer neuron is $Z = (X_1, X_2, ...X_{m_n}) = (Z_1, Z_2, ..., Z_k)$:

$$P(Z_i) = \frac{e^{Z_i}}{\sum_{j=1}^{k} e^{Z_j}}$$

$$\Delta P(Z_i) = P(Z_i)^{'} - P(Z_i) = \frac{e^{Z_i + \Delta Z_i}}{\sum_{j=1}^{k} e^{Z_j + \Delta Z_i}} - \frac{e^{Z_i}}{\sum_{j=1}^{k} e^{Z_j}}$$

We notice that since we're dealing with upper bound of the final output error, we want to make the numerator as large as possible and the denominator as small as possible in the first term, while the numerator is as small as possible and the denominator is as large as possible in the second term. So we're going to scale it up. For this reason we can ignore the α of leaky RELU in equation (1). Notice that $\alpha \leq 1$, we want to make the numerator larger and denominator smaller, so we can ignore it to make the equation largest. We denote the minimum of $(Z_i - \Delta Z_i)$ is $(Z_l - \Delta Z_l)$, and the maximum of Z_i is Z_h :

$$\begin{aligned} \Delta P(Z_i) &\leq \frac{e^{Z_i + \Delta Z_i}}{n e^{Z_l - \Delta Z_l}} - \frac{e^{Z_i}}{n e^{Z_h}} \\ &= \frac{1}{n} \left(\frac{e^{Z_i + \Delta Z_i}}{e^{Z_l - \Delta Z_l}} - \frac{e^{Z_i}}{e^{Z_h}} \right) \\ &= \frac{1}{n} \left(e^{\Delta Z_i} e^{\Delta Z_l} e^{Z_i - Z_l} - e^{Z_i - Z_h} \right) \\ &\leq \frac{1}{n} e^{2\Delta Z_i} e^{Z_i - Z_l} - \frac{1}{n} e^{Z_i - Z_h} \end{aligned} \tag{3}$$

It's an upper bound of $\Delta P(Z_i)$. So we finally obtain the relationship between the output error and the error of all the layer in the neural network by substituting equation (2) into equation (3). We know that the normal weight value is 10 to the minus 2 power or less, so the final error caused by these errors is very small. We can also make this conclusion by the following experiments. From this we conclude that the effectiveness of our attacks is confirmed.

Here we ignore the convolution layer of the neural network. Because as we know, the convolution layer is one kind of special full-connected layer where some neurons have value of 0. Our deduction includes this particular case, we can also see the experimental result supports our idea.

Here we also ignore the max-pooling and zero-padding operation in common neural network. Because max-pooling collects the biggest value in the target field of parameter matrix. Its upper bound is the largest value in normal full-connected layers. The zero-padding add zero to the margin of matrix, so the upper bound will not change. Our deduction can perfectly includes the case where max-pooling and zero-padding are applied. Our experiment can confirm this claim.

Algorithm 1 "Virtual user" attack

1: set the initial weight w_0
2: users train model on local dataset and get the gradientΔw_0, uploading to platform
3: platform selects some neurons and sends the "too big" weights Δw_t to the users
4: platform creates a virtual user and updates the gradient Δw_t
5: users update the local weight w_1 with the gradient Δw_t
6: **while** training in n epoch,$i = 1, 2...n$ **do**
7: users get the gradient Δw_i in local training , uploading to platform
8: platform selects the largest Δw_i among all gradients as Δw_j and sends it to the users
9: platform send ΔW_j to the "virtual user"
10: users use the gradient Δw_j issued by the platform to update a new weight w_{i+1}
11: **end while**

6 "FINETUNE" DEFENSE

In order to defend the above attack, we make use of a training method called "finetune". Finetune is a kind of transfer learning method, that is, in order to save training time and make up for the defect of insufficient training data, the user does not directly train the model from the scratch, but uses an already trained model to retrain, which can greatly speed up the training process. Our defense procedure is as follows.

First, each user needs to make an appointment on preprocessing data by lowering a certain number of pixels, then uses the data for distributed training. After downloading the trained model from platform, we only need to finetune the model in original real dataset to make it meet our requirements.

At the same time, we can find that because the dataset used for distributed learning is not the original clearer dataset, but through the pixel-reduce procedure, the "virtue user" attack method can not obtain a sufficiently accurate network model. It will lead the attack be ineffective.

Algorithm 2 "Finetune" defense

users agree on the initial weight w_0
users reduce the pixel of the image
while training in n epoch,$i = 1, 2...n$ **do**
    users train locally to get the gradient Δw_i, upload to platform
    platform selects the largest Δw_j among all gradients and sends it to the users
    users simultaneously uses the gradient issued by the platform to obtain a new weight w_{i+1}
end while
users locally perform a finetune operation on the obtained model with own dataset to retrain the model

7 EXPERIMENT

7.1 Experiment Setup

7.1.1 **Dataset.** We apply distributed training method to the MNIST dataset, the Fashion-MNIST dataset and the notMNIST dataset.

MNIST (Mixed National Institute of Standards and Technology database) is a computer vision dataset that contains 60,000 grayscale images of handwritten Numbers from 0 to 9, each of which contains 28x28 pixels. It also has 10000 test images.

Fashion-MNIST dataset is an advanced version of MNIST dataset. It consists of 10 categories clothes. The purpose of this dataset is to replace MNIST as a good benchmark to evaluate machine learning algorithms. In order to be compatible with MNIST, the format, category, data amount, training and test set of FashionMNIST and MNIST are completely consistent.

The notMNIST dataset is a set of image folders classified by character, consisting of 10 folders A, B, C, D, E, F, G, H, I, J. Each image has a dimension of 28X28. It is a more complicated dataset which can replaces MNIST for more precise evaluation. We apply some method to make its format the same as MNIST.

7.1.2 **Discriminator architecture.** Table 3-5 in appendix show our neural network structure. Here we test multiple network structures, so we can obtain more comprehensive and reliable experimental results. In these experiments, we set the learning rate=0.001, momentum=0.9, active function as RELU function, and the convolution kernel as 5X5.

7.1.3 **Distributed training setting.** We set up the distributed learning platform by ourselves. We construct some users to train their own data locally and upload their gradients during training, as Shokri proposed. The server receives users' gradients and selects the biggest value of each neuron, then give back to users. users update their model by the downloaded gradients. The operation of "virtual user" is the same as real users.

In the "virtual user" attack, as we know the order of magnitude of normal neural network weights is about 0.01, we set the "too big" weight value about 1. because the weight value of neural network had better fit normal distribution to ensure convergence, we sample it from a normal distribution which has mean of 1. Then we impose the "too big" weight to users' model.

7.1.4 **Generator architecture.** Table 6 in appendix shows our Generator structure. It is trained synchronously with the "virtual user" model when users perform distributed learning method.

7.1.5 **GAN attack setting.** We set up the GAN attack similar to Briland Hitaj proposed. We modify the output layer from 10 to 11, the additional output is where the "fake" label is placed. Then we train a Generator by minimizing the probability Discriminator put generated data to fake label.

Table 1: Model Accuracy

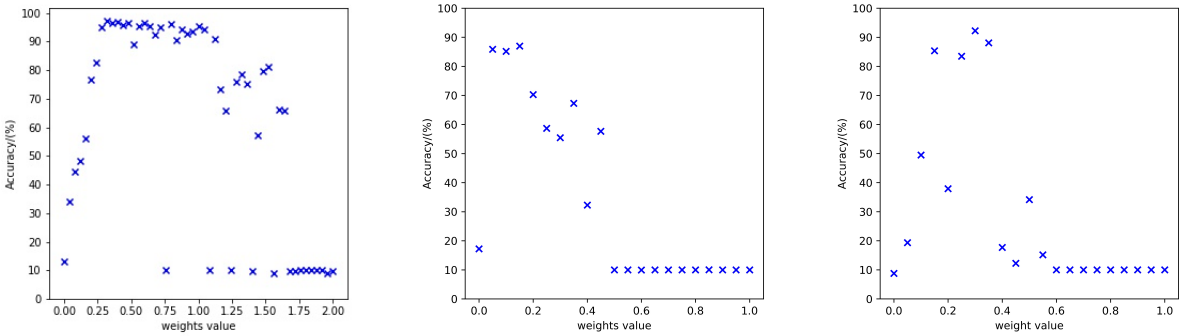
dataset	"virtual user"	real user
MNIST	97.39%	98.01%
Fashion-MNIST	87.21%	89.10%
notMNIST	92.23%	92.31%

7.2 "Virtual user" attack

Table 1 show our "virtual user" attack model accuracy. First, we set up multiple users to train a model together on a distributed training platform. The platform creates a "virtual user" and maliciously gives users some "too large" weights. In the experiment, our malicious weights' order of magnitude is about 1, while the real weights' order of magnitude is about 0.01. At the same time, the platform gives "virtual user" the "too big" weights to the same neurons, randomly giving weights of 0.01 order of magnitude to the remaining neurons. The above is description of the first training epoch. After the first epoch, the malicious platform can work normally, maintain the work of weights distribution in the original protocol, and update the parameters of "virtual user" in each epoch timely to ensure its accuracy. The experimental results here prove that our intuition and proof are correct, that the performance of this "virtual user" neural network should be the same as real users' local model.

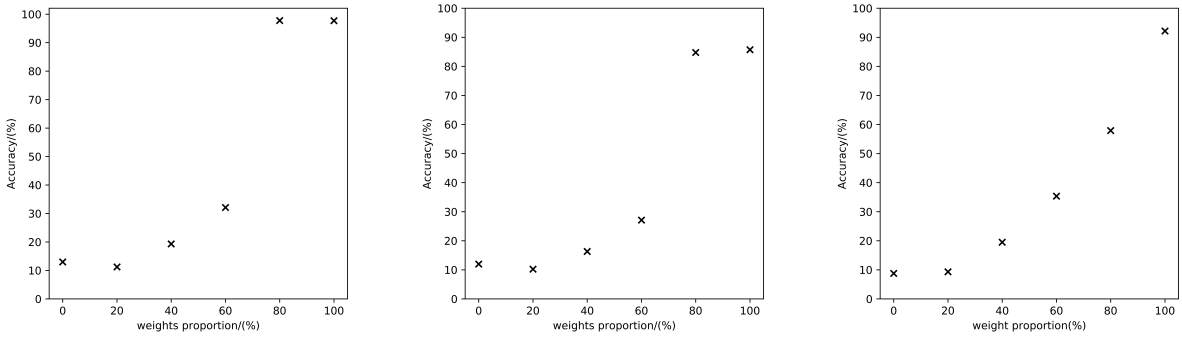
7.2.1 **"Too big" weight value setting.** Figure 2 shows the relation of weights value and performance of the "virtual user" neural network. Here we show the precision of the model when the platform maliciously gives users different values of "too big" weights. Here we can see, as the "too big" weights increase, the "virtual user" model accuracy first increases and then decreases. It is easy to understand, because at the beginning the "too big" weights are very small, the difference of "virtual user" model and real user model is very large. Distributed training cannot get very accurate model. as the "too big" weights increase, the "virtual user" model starts to be similar to the real user model. The accuracy of the distributed training model will be greatly improved. Finally, when the "too big" weights come to a certain level, as we known, weights initialization must meet unique distribution to ensure the training convergence reliable. But obviously now the weights are no longer normal distribution and cause gradients explosion, training process can't converge. "virtual user" attack starts to go wrong.

7.2.2 **"Too big" weight proportion setting.** Then we examine the effect of "too big" weights proportion in all weights on the accuracy of "virtual user" attack. Figure 3 shows the relation of weights proportion and performance of the "virtual user" neural network. Here we can see that as the proportion of "too big" weights



(a) MNIST dataset (b) Fashion-MNIST dataset (c) notMNIST dataset

Figure 2: The relation of "too big" weights value and accuracy. The horizontal ordinate is the order of magnitude of our weights, not the exact value. We sample the data from normal distribution of mean value of it. Because the Fashion-MNIST dataset is more complicated than MNIST, the training process collapse quicker than MNIST. The scales of each graph are different beause we want to display the collapse point of every dataset. We can see as the dataset is more complicated, the collapse will becomes quicker.



(a) MNIST dataset (b) Fashion-MNIST dataset (c) notMNIST dataset

Figure 3: The relation of "too big" weights proportion and accuracy.

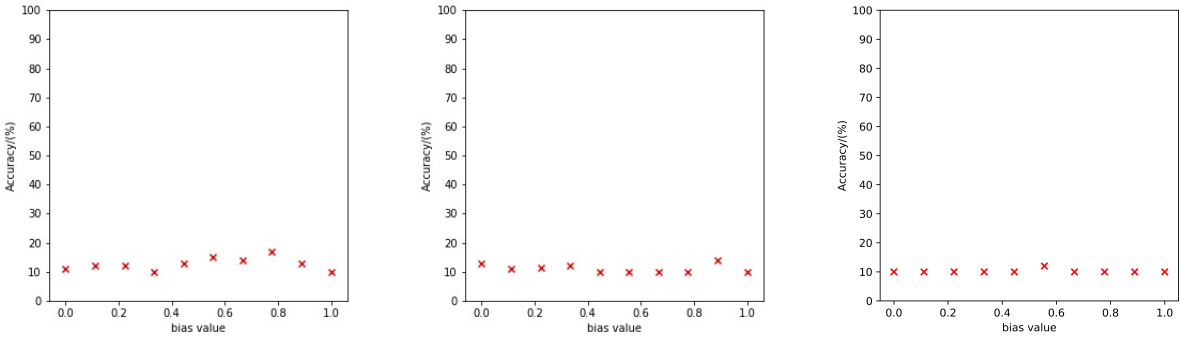
in all weights increases, the model accuracy becomes higher. It's consistent with our reasoning. Because the proportion of "too big" weights was too small at the beginning, "virtual user" model was different from the user's local model, the model accuracy was definitely low. However, as the proportion of too big weight becomes larger, "virtual user" model and the real user's model become more similar.

7.2.3 **Bias setting.** We test the effect of different bias values on the accuracy of "virtual user" attack. Figure 4 shows the experiment result. As we can see, since the proportion of bias in all parameters is too small, changing bias cannot guarantee the "virtual user" model similar to the real user's local neural network, the model is of poor accuracy eventually.

7.2.4 **Architecture setting.** We examine the influence of the neural network depth on the accuracy of "virtual user" attack. Figure 5 shows the experiment result. Here we can see, with the depth of the neural network increasing, the "virtual user" attack accuracy will gradually decrease. It's consistent with our deduction. Because the attack precision and the weights of each layer have a positive correlation.

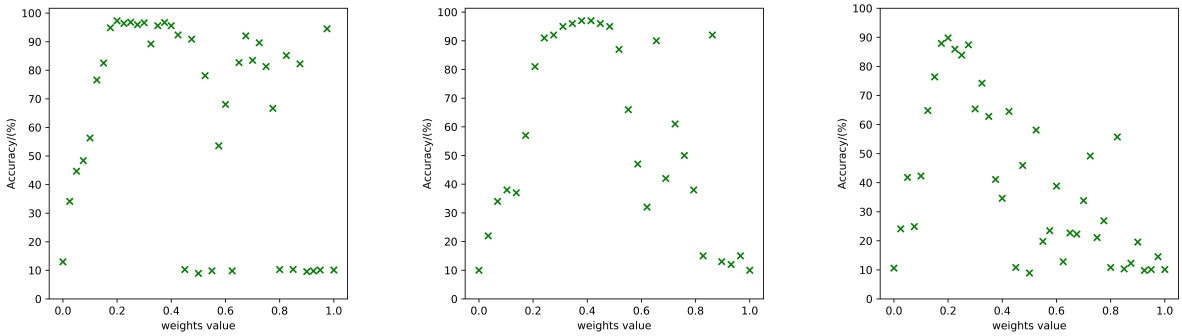
7.2.5 **Effect on real users' model.** We can see our "virtual user" and the imposed "too big" weight have an effect on real users' training. From above experimental results show some questionable collapses, namely sometimes the model accuracy becomes 10% or less. Because the training convergence of neural network has some strict requirements. We believe we can try more appropriate "too big" weight to avoid this collapse as much as possible.

The training time of real users becomes longer due to the participation of "virtual user", as in Figure 7. In our experiment, when we construct 2 real users to perform distributed training on MNIST



(a) MNIST dataset (b) Fashion-MNIST dataset (c) notMNIST dataset

Figure 4: The relation of bias value and accuracy. The horizontal ordinate is the order of magnitude of our bias, not the exact value. We sample the data from normal distribution of mean value of it. Bias has very little influence on our model accuracy.



(a) 2 layers (b) 4 layers (c) 6 layers

Figure 5: The relation of network depth and accuracy.We notice that the 4-layer neural network will crash more slowly with the increase of "too big" weight. Therefore, in order to fully show the experimental results, we set the order of magnitude of 1 on the 2-layer and 6-layer networks and order of magnitude of 2 on the 4-layer network.

dataset, the participation of "virtual user" will increase the time consumption even more than 3 real users collaboratively train. We can easily understand it because our "too big" weight can make the model hard to converge. But we can choose some appropriate value to conceal our attack.

7.3 Deal with deep architecture

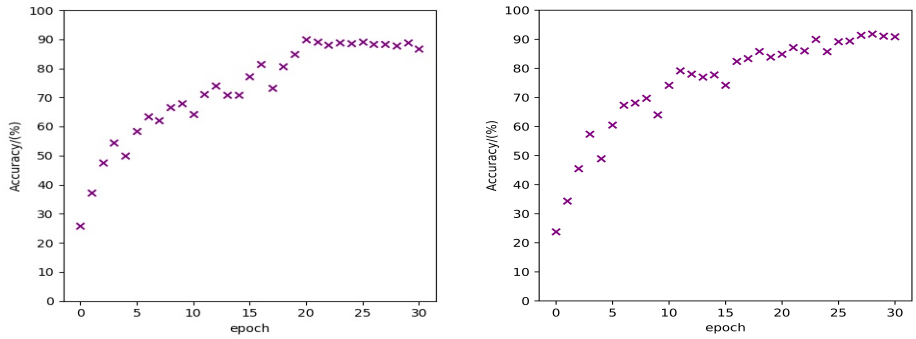
We emphasize that although the attack accuracy will decrease with the increase of network depth, according to our previous proof of feasibility, when weight noise is closer to the network output side, the impact on the accuracy is lower. we can improve the attack accuracy by adjusting "too big" weights related to the network output side. For example, "too big" weights close to the output side are set to a larger number, and which far from the output side are set to a smaller number. This can effectively reduce the impact of network depth and improve the accuracy of attack. Figure 6 shows the experiment verification of our deduction.

7.4 Multi-users attack

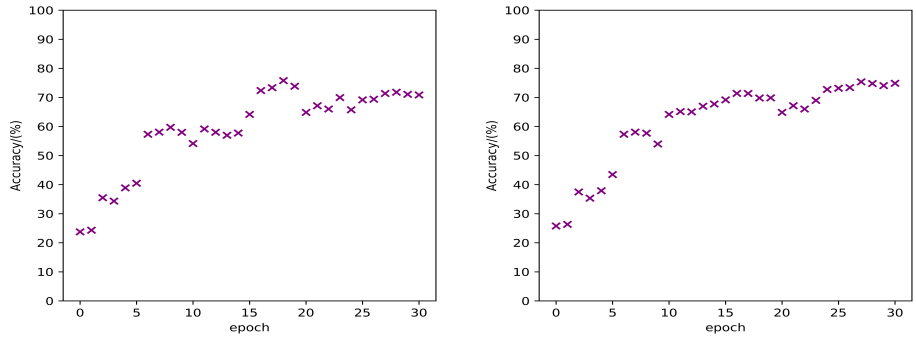
We examined the effect of users number on "virtual user" attack. We find that the number of users does not have much influence on our attack, which indicates that our attack is universal and can be applied to large-scale scenario in practice. Table 2 shows the experiment result of different user numbers.

Table 2: Attack accuracy on MNIST

users number	2	5	10
"virtual user"	97.39%	97.12%	97.54%
real user	98.21%	98.12%	98.03%



(a) MNIST



(b) Fashion-MNIST

Figure 6: The relation of weights value choice and accuracy. The weights that have different distances from the output are set differently. The left figure shows that when "too big" weight is close to output, we set it to a smaller value. The right figure shows that when "too big" weight is close to output, we set it to a larger value. We find that the accuracy of the model on the right was higher than that on the left. Although the improvement of performance is too trivial to see, we believe that we can get higher advance in more smart weights choice.

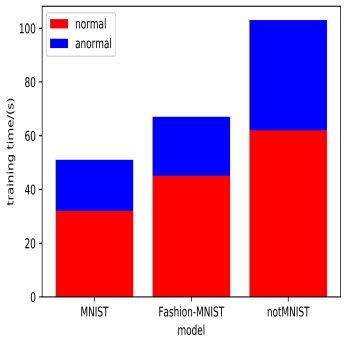


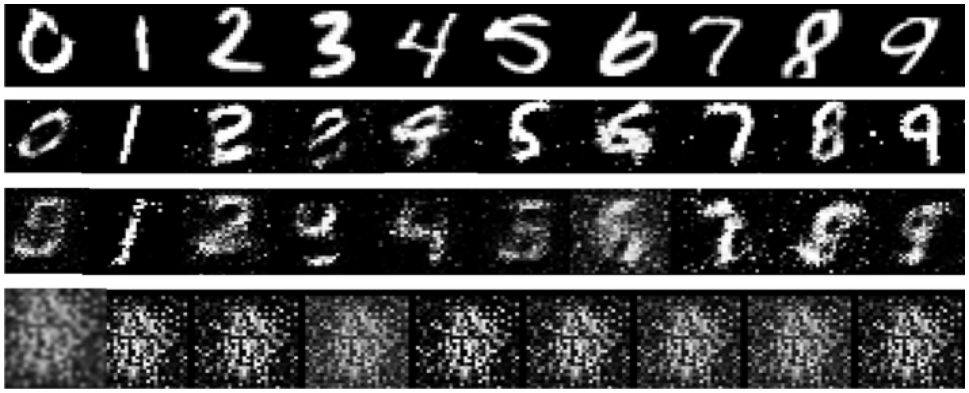
Figure 7: The attack effect on real users' training. The y-axis is the training time, the x-axis is the dataset.

7.5 GAN and "finetune" defense

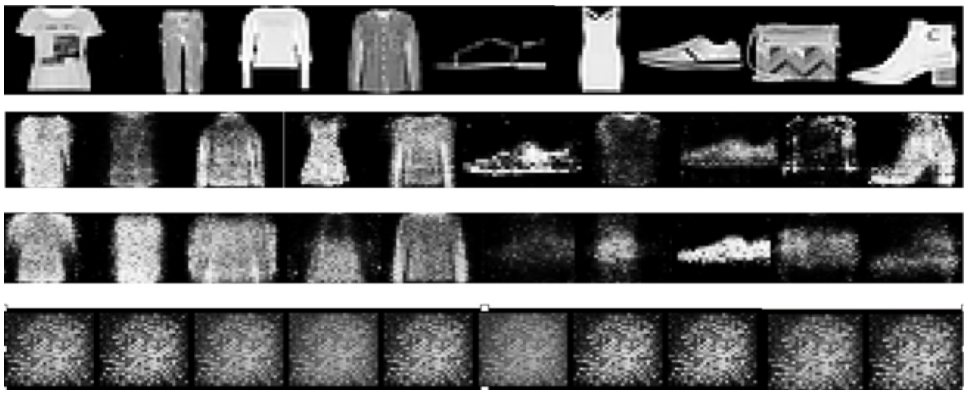
We apply the GAN method to obtain the user's sensitive data. Figure 8 shows the image of the user's dataset, and then we applied finetune defense to defend the attack.

First, we reduce the pixels in user's dataset image, then use the data for distributed training. The users should finetune the model obtained after training on the original dataset to meet their requirements. We find that since the model only requires a small amount of time in the local finetune process, this kind of defense will not significantly increase the time cost. The experimental results prove our previous idea. Because such a training dataset is not the original dataset, but reduces the pixel value, the "virtual user" attack method can not get accurate model. This defense makes GAN not able to produce reliable data, so as to make the attack not effectively implemented.

We can see that the user's sensitive data was clearly obtained with "virtue user" attack at the beginning. As we apply finetune defense, the clarity of images we obtained began to decline. When



(a) MNIST



(b) Fashion-MNIST

(c) not-MNIST

Figure 8: This is GAN attack and "finetune" defense on MNIST, Fashion-MNIST and notMNIST. The first row is real data. We can see that at the second row we used the "virtual user" attack and GAN method to get the user's sensitive data very clearly. But as we applied the "finetune" defense, the clarity of image we obtained began to decline, as the third row shows. When our defense reached a certain intensity, the obtained image cannot be distinguished in the last row.

our defense reachs a certain intensity, the image could not be distinguished. Here we emphasize that although the finetune defense can lead to a certain increase in training time, in our experiment, it only takes a little time to achieve a good defense effect. Therefore, the "finetune" defense is feasible.

7.5.1 **Time consuming.** We also look at the time cost of the finetune defense in Figure 9. We find that it takes only a little time

in all of our training process. Our experimental results show that when the time consumption of finetune defense reachs a certain degree, our defense approach could achieve a satisfactory effect. This shows our defense is feasible.

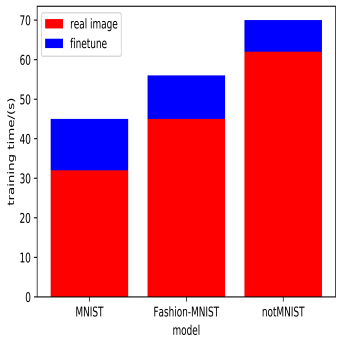


Figure 9: The finetune defense effect on real users' training. The y-axis is the training time, the x-axis is the dataset.

8 DISCUSSION

Our experiments have confirmed that "virtual user" attack is feasible, but there are still some deficiencies. In this paper, we prove that the attack accuracy has certain error, and we deduce the upper bound of error. We find that the error is related to the depth of the neural network. As the depth of the neural network increases, our attack error will also increase. This makes it difficult for us to carry out attack on very deep neural network models. However, we also proved that the error of attack is related to our "too big" weights selection, and the error caused by "too big" weights near the output side is relatively small. Therefore, we emphasize that we can implement more accurate attack with smarter parameter selection method.

In the process of training, we find that the neural network model will bring the problem of convergence instability due to the attack. Because we impose some outer parameters in the attack, which will cause occasional collapse in distributed training, for neural network convergence needs the parameters to meet unique distribution. This situation may cause users to distrust the platform, which is a blow to the company's credibility. However, we can optimize the selection of attack parameters to meet the need of neural network convergence.

In our experiments, although our "too big" weights are somewhat large, users may find anomalies in information transmission and take some preventive measures. For example, setting the threshold of the transmission parameter to truncate the abnormal parameters. But as far as we know, the current commercial platform does not provide this service, so our experiment warns these companies, hoping that these companies can monitor the information transmitted, to be fair to users and achieve privacy protection.

In our attack scenario, we assume that a malicious platform can know the users' local neural network structure, and launch the attack based on this known information to obtain the users' privacy. Although there are ways users can make it impossible for the platform to know the exact network structure, in the current commercial application the platform can obtain the network structure of users. In the research field, the attack methods proposed by some researchers are also based on the assumption of the known network structure to the platform. Therefore, the attack we proposed is of practical significance.

In our experiment, "virtual user" attacks impose a large number of "too big" weights on users. In previous papers, the author proposed a "random selection of parameters" algorithm to achieve privacy protection purpose, which seems to gain some protection against our attack. However, on the one hand, in the process of downloading and uploading parameters, too few parameters and too random selection algorithm will reduce the accuracy of model, so in actual deployment, users will often obtain as many parameters as possible to ensure the stability of training. On the other hand, We prove in the above experiment that our attack does not need to impose all the parameters "too big" weights, but only needs a part to achieve high precision. This result ensures that our attack can be carried out successfully even if the users only select a part of the parameters downloaded.

In recent years some people are working on methods to make neural network parameters completely encrypted during communication, so as to protect users' privacy. However, we emphasize that current encryption methods, such as secure multiparty computation, homomorphic encryption and differential privacy, cannot be applied in real scenario due to the large number of parameters and the need of high communication accuracy during training. Therefore, we emphasize here that the attack we propose is fully achievable.

9 CONCLUSION

In this paper, we show potential data leakage threat in distributed training, and our experiments prove that we can apply this attack to obtain the user's sensitive data. Our work shows that there is still a lot of risks in distributed training, which is a warning to Google, Amazon and other commercial companies to launch the deep learning method service.

In the future work, we believe which can be improved is that people may conveniently encrypt the information through communication, so that the accuracy and efficiency can be balanced in practical application. We can also find out more security risks and propose some methods to prevent them from happening.

REFERENCES

[1] Yoshinori Aono, Takuya Hayashi, Lihua Wang, Shiho Moriai, et al. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13(5):1333–1345, 2018.

[2] Michael Backes, Pascal Berrang, Mathias Humbert, and Praveen Manoharan. Membership privacy in microrna-based studies. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 319–330. ACM, 2016.

[3] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333. ACM, 2015.

[4] Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In *USENIX Security Symposium*, pages 17–32, 2014.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[6] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. Deep models under the gan: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 603–618. ACM, 2017.

[7] Augustus Odena. Semi-supervised learning with generative adversarial networks. *arXiv preprint arXiv:1606.01583*, 2016.

[8] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[9] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.

[10] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1310–1321. ACM, 2015.

[11] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 3–18. IEEE, 2017.

[12] Congzheng Song, Thomas Ristenpart, and Vitaly Shmatikov. Machine learning models that remember too much. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 587–601. ACM, 2017.

A SYSTEM ARCHITECTURE

Table 3: MNIST

input→(1)→ ... →(10)
(1). nn.Conv2d()
(2). nn.ReLU(max_pool2d)
(3). nn.Conv2d()
(4). nn.ReLU(max_pool2d)
(5). nn.Linear()
(6). nn.ReLU
(7). nn.Linear()
(8). nn.ReLU
(9). nn.Linear()
(10). logsoftmax

Table 4: Fashion-MNIST

input→(1)→ ... →(12)
(1). nn.Conv2d()
(2). nn.ReLU(max_pool2d)
(3). nn.Conv2d()
(4). nn.ReLU(max_pool2d)
(5). nn.Linear()
(6). nn.ReLU
(7). nn.Linear()
(8). nn.ReLU
(9). nn.Linear()
(10). nn.ReLU
(11). nn.Linear()
(12). logsoftmax

Table 5: notMNIST

input→(2)→ ... →(14)
(1). nn.Conv2d()
(2). nn.ReLU(max_pool2d)
(3). nn.Conv2d()
(4). nn.ReLU(max_pool2d)
(5). nn.Linear()
(6). nn.ReLU
(7). nn.Linear()
(8). nn.ReLU
(9). nn.Linear()
(10). nn.ReLU
(11). nn.Linear()
(12). nn.ReLU
(13). nn.Linear()
(14). logsoftmax

Table 6: Generator architecture

(1). input(latent sample)
(2). nn.Conv2d()
(3). nn.ReLU()
(4). nn.Conv2d()
(5). nn.ReLU()
(6). nn.Linear()
(7). nn.Tanh
(8). output(image)