Private Machine Learning by Random Transformations

Anonymous Author(s)

ABSTRACT

With the increasing demands for privacy protection, many privacy-preserving machine learning systems were proposed in recent years. However, most of them cannot be put into production due to their slow training and inference speed caused by the heavy cost of homomorphic encryption and secure multiparty computation(MPC) methods. To circumvent this, we proposed a privacy definition which is suitable for large amount of data in machine learning tasks. Based on that, we showed that random transformations like linear transformation and random permutation can well protect privacy. Merging random transformations and arithmetic sharing together, we designed a framework for private machine learning with high efficiency and low computation cost.

CCS CONCEPTS

• **Security and privacy** → *Data anonymization and sanitization*; • **Computing methodologies** → *Artificial intelligence*; *Machine learning*.

KEYWORDS

Privacy, Machine Learning, Secure Multiparty Computing

ACM Reference Format:

Anonymous Author(s). 2021. Private Machine Learning by Random Transformations. In *WWW '21: The Web Conference 2021, April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Machine learning has been widely used in many real-life scenarios in recent years. In most cases, training machine learning models requires a large amount of data. For example, training a neural network to determine whether two pictures belong to the same person may needs at least tens of thousands photos, and training a model to predict the possibility of credit default of someone needs tens of thousands credit records from different people. Those data are always distributed among different facilities. On one hand, governments across the world have published the laws against abuses of data in order to protect people's privacy and prevent those data from being stolen for evil uses. On the other hand, companies do not want their data being exposed to others. When they want to share data with others, it's always difficult to ensure that the other party will not store their data secretly for usages violates the contract. So to make different data owners to share their data and hence to build better machine learning models, privacy-preserving machine learning technologies must be adopted. To achieve this, researchers have worked out many solutions, which can be generalized to following major methods.

- Cryptography-based Methods. There are two major kinds of cryptography-based methods:
  1. Homomorphic Encryptions(HE). The homomorphic encryption methods allow arithmetic operations on the ciphertext. For example, the Paillier cryptosystem[19] supports additions on ciphertext, and the Gentry cryptosystem supports both addition and multiplication, which is the first fully homomorphic encryption scheme. The level of security is based on the security parameter, which is usually in proportion with the key length.
  2. Secure Multiparty Computation(MPC). MPC methods provide ways to calculate a function while keep the inputs private. The most famous scheme is Yao's garbled circuit[13]. Secret-sharing based methods are more efficient on arithmetic calculations, so they are always combined with garbled circuit methods. E.g. SecureML[17].
- Differential Privacy(DP)[6]. DP is a tool to analyze privacy leakage. In order to reduce privacy leakage, noise can be added to the data.
- Federated Learning(FL)[15]. FL methods protect user data privacy by sending model to user devices, e.g. mobile phones, and send the model updates back to server.

However, those methods all have some shortcomings. HE and MPC requires lots of computation or communication, also hard to implement. DP is easy to implement, but adding noises certainly has a huge negative impact on model performance. FL methods need user devices to do computations, and can only be applied to horizontally split data(i.e. different samples with same features).

1.1 Our contributions

Existing methods mostly focus on designing a method or protocol that will leak no information about the raw data in the cryptographic sense. Like using homomorphic encryption, no attackers can gain any information about the data in polynomial time w.r.t. security parameter. However, this definition is not suitable for the data in machine learning setting. What is necessary is that the data cannot be reused. So I proposed a metric to quantify the information leakage during computation, and a practical method to leverage between privacy preserving and efficiency. In this paper, I made the following contributions:

- A privacy definition that evaluates the possibility of reconstructing raw data from transformed data.
- We proved random transformations, i.e. random linear transformation and random permutation can well protect privacy.
- We designed a private machine learning framework combining random transformation and arithmetic sharing together, achieves very high efficiency in machine learning tasks.

WWW '21, April 19–23, 2021, Ljubljana, Slovenia

ACM ISBN 978-1-4503-XXXX-X/18/06.
https://doi.org/10.1145/1122445.1122456

2 RELATED WORK

In this section, we will introduce earlier researches on privacy preserving machine learning, including traditional privacy notions, cryptography-based private preserving machine learning methods, differential privacy and federated learning.

2.1 Privacy Preserving Methods

Privacy preserving during the data analysis process has long been concerned. The k-anonymity[24] method is to perturb or hide some of the attributes which can be used to identify individual records, so-called the quasi identifiers. Beyond it, there are l-diversity[14] aiming for adding diversity in a group of 'close' records, and t-closeness[12] aiming for make the distribution similar for different group of records. However, as the era of big data and machine learning comes, the amount of data become enormous and the structure of data is fairly complicated, which is kind of incompatible with those previous privacy notions. For example, machine learning tasks usually do not need quasi-identifiers i.e. phone numbers, postcodes. But with huge amount of data, even without any quasi identifiers, privacy can be leaked. [18] shows that even a few records exposed, the attacker may be able to locate a specific person in the database. More recently, [7] shows that even trained models can leak some information about the training data. And DeepLeakage[27] shows the possibility to reconstruct training samples from gradients.

2.2 Cryptography-based Methods

There are two major cryptographic methods for privacy preserving machine learning. One is homomorphic encryption, which allows arithemtic operations to be performed on the encrypted data. Another is secure multiparty computing, which enables multiple parties to jointly evaluate a function while keeping their inputs private.

Cryptonets[8] first applied the somewhat fully homomorphic encryption to deep neural network. All computations are done on the encrypted data. The authors tested this model on the MNIST dataset, and achieved 99% accuracy with a throughput about 59000 prediction per hour and a latency for about 250 seconds. Gazelle[10] avoided expensive fully homomorphic encryption and used packed additive homomorphic encryption to improve efficiency, and used garbled circuit to calculate non-linear activations. It reduces single image classification latency to around 30 microseconds. GELU-Net[26] let clients to calculate the activations while server calculate the linear transformation using additive homomorphic encryption.

Alongside the homomorphic encryption methods, secure multiparty computing methods are also widely used. Beaver[2] used precomputed triples to accelerate online multiplication. ABY[5] mixed arithmetic, boolean and Yao sharing together provided a efficient two-party computation protocol that supports various kinds of computations covered common machine learning functions. ABY3[16] hugely improved its efficiency under 3PC setting. SecureML[17] combined arithmetic sharing and garbled circuit together to perform linear regression, logistic regression and neural network. Chameleon[22] improved ABY by third-party aided multiplication triple generation and adopting GMW protocols[9]. SecureNN[25] uses four sharing schemes and customized protocols for private comparison, outperforms previous works on four convolutional networks.

Those methods all have their advantages: Homomorphic encryption requires only one-round communication, and the security is proven by cryptography; Multiparty computing is faster in most cases, and can be more efficient with semi-honest third party introduced; However, these methods still requires heavy computation and massive communication, and are often hard to implement since they requires modern cryptographic methods, which can be difficult for machine learning developers.

2.3 Differential Privacy

Differential privacy was proposed by [6]. It protects privacy by limiting the change of function when one record in the data changed. Differential privacy is always achieved by adding noises somewhere in the data analysis process. For example, [1] adds the noise in the gradients of training. PATE[20] applied differential privacy on the label-generation phase of teacher models. And the ESA architecture[3] uses local differential privacy to ensure the worst-case privacy when all other parties are colluding together. Differential privacy provided a strong tool to evaluate privacy, but it will certainly affects the model performance in machine learning since noises are added. And in our view, it's focused on the maximum effect of one record, but not the actual sensitive data. In other words, maybe we do not need such a strict standard in order to protect data privacy.

2.4 Federated learning

Federated learning was first proposed by [15]. It trains the model in a decentralized manner by sending model parameters to user devices, then the model is trained locally on those devices. After that, server gathers the model updates and calculates the final update. In order to prevent privacy leakage by uploading model updates, secure aggregation[4] and homomorphic encryption[21] methods were proposed. However, federated learning methods are designed for horizontally split data, i.e. different data owners have different samples, but the features are overlapped. But sometimes, data can be vertically split, i.e. same samples, different features. And let user devices to train the model will certainly affects the efficiency and performance.

3 PRIVACY DEFINITION

In this section, we proposed a new privacy definition named reconstructive privacy. Reconstructive privacy evaluated the possibility to reconstruct raw data from transformed data. It is a powerful tool to analyze information leakage during different transformations. Based on this, we proved random permutations and linear transformations leaks very little information of raw data.

3.1 Reconstructive Privacy

In most machine learning scenarios, the data used is in a tabular form. Every row is a training sample and every column is a feature. The metadata, i.e. the ID of each row and the attribute name of each column are very easy to be hide. The only information exposed is the entries of the data table.

How can an attacker use leaked data for his own malicious purpose? There should be two ways for an attacker to use the leaked data:

(1) Linking: The attacker have access to some other data, with overlapped samples. And then he can link the leaked data with the other data, and then gained some extra information.
(2) Reusing: Although the attacker cannot find other data to link with leaked data, he can still store the leaked data and illegally reuse them some time later.

Thanks to cryptography, there are plenty of ways to hide sample's ID while several parties wants to jointly train some model. E.g. Private Set Intersection(PSI) methods. So the attacker cannot directly reuses leaked data. The only way is to link the leaked data to some other data, then 're-identify' the leaked data. Linking requires common columns, so the attacker must obtain the original columns of the data. In order to measure attacker's ability to recover the raw data columns from transformed data, we defined reconstructive privacy as follows:

Definition 3.1 (ϵ-reconstructive privacy). A data transformation T is said to have ϵ-reconstructive privacy under auxiliary information a if an adversary $\mathcal{A}$ with input $T(x)$ and auxiliary information a has a chance at most ϵ to recover the raw data x. In other words, for any function A, $E_x(p[A(T(x); a) = x]) < \epsilon$. If there are no auxiliary ,

For example, consider shuffle on an array of length 5. Without any other information, the adversary can only guess randomly. So he has a $\frac{1}{5!}$ to get the correct raw data. That is, the transformation shuffle has a $\frac{1}{5!}$-reconstructive privacy with no auxiliary information.

Definition 3.2 (ϵ, δ-reconstructive privacy). A data transformation T is said to have ϵ-reconstructive privacy under auxiliary information a if an adversary $\mathcal{A}$ with input $T(x)$ and auxiliary information a has chance ϵ to recover the raw data x with error δ. The error definition can be specifically chosen according to scenario. In other words, for any function A, $E_x(p[|A(T(x); a) - x| < \delta]) < \epsilon$.

For example, consider adding noises $e \sim \mathcal{N}(0, 1)$ to a value x. The adversary get the value $x + e$, with auxiliary input that the variation of noise is 1. so he can guess the real value is in $[x + e - 3, x + e + 3]$ with a $\Psi(3) - \Psi(-3) = 0.997$ confidence. That is, the transformation $T(x) = x + e$ has a 0.997, 3-reconstructive privacy.

3.2 Common Transformations

Linear Transformation: Let raw data be a vector $\mathbf{x}$ of length n. A linear transformation turns x into Ax where $A \in \mathbb{R}^{m\times n}$ is a matrix. Calculating the ϵ and δ in linear transformation's reconstructive privacy is not trivial. In the following theorem, we assume that the raw data x and the elements in matrix A are all random variables drawn from a standard normal distribution.

Theorem 3.3 (Linear transformation's reconstructive privacy). *Let raw data $\mathbf{x} \in \mathbb{R}^n$ be a vector with each element drawn from the standard normal distribution independently, the same as matrix $A \in \mathbb{R}^{m\times n}$. And let the auxiliary information for adversary is the $\mathbf{x}$ and A, both are drawn from standard normal distribution. The linear transformation $T : x \to Ax$ has a ϵ, δ-reconstructive privacy, where $\epsilon < p(|\mathbf{y}| < \delta)$ with $y \in \mathbb{R}^{n-1}, y_1, ..., y_{n-1} \sim \mathcal{N}(0, 1)$ and p is the density function. In other words, it's like the adversary has no information in $n - 1$ dimensions of the raw data x.*

Proof. First, from the reconstructive privacy's definition, we have

$$E_{\mathbf{x}}(p[|C(A\mathbf{x}) - \mathbf{x}| < \delta]) < \epsilon \tag{1}$$

Here we use $C()$ to denote adveray's function to avoid the confusion with transformation matrix A. Since A is also a random variable, we can change (1) into $E_{\mathbf{x},A} I(|C(A\mathbf{x}) - \mathbf{x}| < \delta)$, where I is a indicator function when the condition satisfies is 1, otherwise is 0.

$$\epsilon = \frac{\int_{A,\mathbf{x}} p(T = A)p(X = \mathbf{x})I(|C(A\mathbf{x}) - \mathbf{x}| < \delta)d\mathbf{x}dA}{\int_{A,\mathbf{x}} p(T = A)p(X = \mathbf{x})d\mathbf{x}dA} \tag{2}$$

Where we uses $d\mathbf{x}$ to denote $dx_1dx_2...dx_n$, and uses dA to denote $dA_{1,1}dA_{1,2}...dA_{m,n}$. In order to eliminate the annoying term $C(Ax)$, we have to do a rotation on $\mathbf{x}$'s coordinates and extract $y = A\mathbf{x}$. That produces:

$$\frac{\int_y \int_A \int_{\mathbf{x}, A\mathbf{x}=y} p(T = A)p(X = \mathbf{x})I(|C(y) - \mathbf{x}| < \delta)dvdAdy}{\int_y \int_A \int_{\mathbf{x}, A\mathbf{x}=y} p(T = A)p(X = \mathbf{x})dvdAdy} \tag{3}$$

Then how to find the upper bound of ϵ? The intuition comes from the simple inequality $\frac{\int f(x)dx}{\int g(x)dx} \le \max \frac{f(x)}{g(x)}$ for $f(x), g(x) > 0$. Considering the hyperplane $A\mathbf{x} = y$, the formula $\frac{\int_{A\mathbf{x}=y} p(X = x)I(|C(y) - x| < \delta)dv}{\int_{A\mathbf{x}=y} p(X = x)dv}$ is actually the probability of x lies in the ball $\mathcal{B}(y, \delta)$ on the hyperplane $A\mathbf{x} = y$. Since the marginal distribution on that hyperplane is still a standard normal distribution, Which can be expressed by $p(z) = \frac{1}{\sqrt{(2\pi)^{n-1}}} e^{-|\mathbf{z}|^2}$. So the upperbound of ϵ is lower than the probability that the random vector $\mathbf{x} \in \mathbb{R}^{n-1}$ has a length shorter than δ. □

The above theorem shows the linear transformation will reveal no more information than one dimension of the raw data. Actually, since we used very strong conditions to proof the upperbound of ϵ, the information leakage can be far less than theory, that is, the adversary can only get a little information on one dimension of the raw data. Notice that one dimension does not mean one element in the vector.

Random permutation: Random permutation is a basic method to hide data. Since a random permutation on a sequence of length n can produce $n!$ possible outcomes, we can calculate the reconstructive privacy for it:

Theorem 3.4 (Random permutation's reconstructive privacy). *Random permutation on an vector of length n has a $\frac{1}{n!}$-reconstructive privacy.*

3.3 Attacker models

Assume the transformed data is send to a curious third party, who tries to reconstruct the raw data with some of his own knowledge.

Here we shows that the attacker can hardly gain any other information with some part of raw data or some knowledge of the transformation.

3.3.1 Attacker with some part of raw data. Assume raw data are a collection of samples $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$(can be also represented by X) with dimension d. And the attacker have some part of raw data $\{\mathbf{x}'_1, \mathbf{x}'_2 ...., \mathbf{x}'_m\}$ with dimension $d' < d$. In this case, the attacker's purpose is to join the table and get more attributes for his samples. The attacker have to guess the mapping from his sample to columns of transformed data $Y \in \mathbb{R}^{f \times n}$, where there are $\binom{n}{m}m!$ possibilities.

For linear transformations, assume the transformation matrix $A \in \mathbb{R}^{d \times f}$ and the transformed data are $Y = XA$. And within each possibility, according to (3.3), the attacker still have no more knowledge about $n-m-1$ dimensions of the raw data. Since $\binom{n}{m}m!$ is already big enough, and usually $n-m-1$ should be much larger than 1, so we can say linear transformation is resilient to this sort of attack.

As for random permutation, the possibilities increases exponentially with the data size. Supposed the raw data size is n, and the attacker obtains m raw data values, there are still $(n-m)!$ possibilities for the remaining $n-m$ values. Hence, with large enough data size, the attacker can hardly gain any knowledge on rest of the raw data.

3.3.2 Attacker with knowledge of the transformation. Although it's hard for attackers to know anything about the random transformation, since the data holder decides it, but it is still worth discussion.

For random permutations, if the attacker knows some part of the permutation, say, m out of n entries in a permutation are revealed, then he still has no knowledge about the remaining $(n-m)$'s elements positions. Thus, he knows nothing besides what he already knows.

And for random linear transformations $f : \mathbb{R}^m \to \mathbb{R}^n$, when $n < m$(which is in most of the cases), even if the attacker knows f, he cannot find f^{-1} since it's not a bijective function. One $f(x)$ corresponds to infinite possible x. Only if the attacker also know the distribution of sample space $\{x\}$ and the samples are actually lying on some subspace of $\mathbb{R}^{m'}$ and $m' < n$, could he have a chance to fully reconstruct the original data. However, unlike the data, where attacker may get some of the raw data from other sources, it's impossible for attacker to know all about the transformation since it's performed locally by data owner. So this kinds of attack can be very rare. The transformation can be considered as a 'private key' of data owner, and shall be secure.

3.4 Extending Raw Data's Definition

In the discussion above, we refer 'raw data' as the raw data value, i.e. a vector, a table. But some times, the numeric values are not the essence of the data. For example, a online store gathered millions of people's purchase records. It uses different integers to refer different items and consumers, denoted by ID. For example, a book is represented by 1, and a T-shirt is represented by 100. So the whole dataset can be a matrix where if consumer i bought item j, the entry (i, j) is 1, otherwise 0. Randomly swaps the ID of two items or consumers for multiple times, it's impossible to reconstruct the original matrix. So does it achieves a 'exponentially small ϵ'-reconstructive privacy? Of course not. Since the real information of the dataset is not the interaction matrix. It is the user-item graph. A graph can have exponentially large number of adjacent matrices. Although two matrices may look not like each other at all, but they can be the adjacent matrices of the same graph. An attacker can get the graph, and with some background knowledge, i.e. by examining the degrees of each user node and item node, he can guess which item node corresponding to which real item. Hence he can use those data for his own benefit, i.e. training a recommender system. So are the images. Since convolutional models do the same linear transformation on different parts of the image, thus, the relations of different parts of the image is reserved, and attackers can easily guess the content of the raw image. The above two cases show that the raw data is not always the numeric values, but some structure lies inside the numbers. So in order to achieve privacy preserving, the raw data's definition must be carefully chosen by domain experts.

4 FRAMEWORK DESIGN

In the last section, we showed that the random transformations can preserve privacy by disabling adversaries to recover raw data from the transformed data. So it is safe for data holders to give out its raw data to some third party to perform computation and the get back the results. To take advantage of this, we designed a framework merging random transformations and MPC operations together, in order to perform private machine learning tasks more efficiently.

4.1 Arithmetic Sharing

To our knowledge, arithmetic sharing was first formally proposed on the ABY[5] framework. It's based on shamir's secret sharing scheme[23]. In this paper, we use the 2-party setting for simplicity.

Sharing: A value x is shared among parties P_0, P_1 means that P_0 holds a value $\langle x \rangle_0$ while P_1 holds a value $\langle x \rangle_1$ with the constraint $\langle x \rangle_0 + \langle x \rangle_1 = x$. To share a value $x \in \mathbb{R}^n$, party P_i just simply pick $r \in \mathbb{R}^n$ and send r and $x - r$ to P_0 and P_1 respectively.

Note that here we do not convert float values into fixed point integers. But it can cause problems, i.e., when x is large and r is small, $x - r \approx x$. In order to prevent this, we uses a r whose variation is greater than x, so its safe to share r and $x - r$.

Reconstruction: P_0, P_1 both send their value some party P_i, could be one of them or a third-party. The raw value is reconstructed immediately by P_i summing two values.

Addition: When adding a public value a to a shared value x, the two parties just add $a/2$ to their shares of x. When adding a shared value a to a shared value x, the two parties just add their shares of a and x respectively.

Multiplication: When multiplying a public value a, the two parties just multiply their shares by a. Multiplication with a shared value is kind of tricky. We adopted the beaver triple in this framework. Suppose multiply shared value x with shared value y. This requires a precomputed triple $uv = w$. In the sense of matrix, the u should have the same shape with x and v should have the same shape with y. So $xy = (x-u)(y-v) + (x-u)v + u(y-v) + uv$. And $x-u$ and $y-v$ can be public since u and v are shared private values, this formula can be evaluated in a shared manner. Both parties then shares the product xy.

Convolutional Layers: Convolutional layers are like dense layer. The filters are the weights of a dense layer, and the input image have to be transformed to a new matrix where different rows are different areas of the raw image. Let the input image be X and the filters be F, it's easy to see the convolution operation $X * F$ satisfies the distribution law, i.e. $X * (F_1 + F_2) = X * F_1 + X * F_2$ and $(X_1 + X_2) * F = X_1 * F + X_2 * F$. So does the gradients (actually, the jacobian matrix) of convolution $\frac{\partial(X*F)}{\partial X}$ and $\frac{\partial(X*F)}{\partial F}$. Hence, the calculation of Convolutional layers and its gradients can be applied just like ordinary scalar or matrix multiplications.

4.2 Adding Random Transformation to Arithmetic Sharing

4.2.1 Nonlinear Functions. In previous MPC systems, the most difficult and costly part is the computation of nonlinear functions, including neural network's activation functions and logical functions, i.e. comparisons. Existing works mostly uses garbled circuit or polynomial approximation to calculate them. These methods result in heavy computation and communication costs.

By adopting reconstructive privacy, this can be quite easy. our framework contains several semi-honest third parties who can perform computation. While P_0, P_1 contains a shared vector x, when they wants to compute $f(x)$, where f is some element-wise nonlinear function, they first get a random permutation of x, denoted by x'. Then they send the x' to a third party P_2 who computes $f(x')$ and shares it to P_0 and P_1. Permuting it back, P_1 and P_2 then get the shares of $f(x)$.

Element-wise Functions: 1. Either P_0 or P_1 generates a random permutation P and send it to the other. 2. Each party calculates $\langle x'_i \rangle = P(\langle x_i \rangle)$ to get the permuted shared values. 2. Then they reconstruct value x to a third party P_3. P_3 computes $f(x)$ then share it to P_1 and P_2. P_1 and P_2 then reconstruct the shared value using the inverse permutation P^{-1} which can be easily calculated.

4.2.2 Distribute Works to Third Party. After appropriate random transformations, the data can be securely revealed to third party, then the third party can fit a model. But only fitting the model on transformed data, the performance will certainly dropped since the data are transformed and may lose some information. But this can be overcome by 'fitting' the transformation. Assume data X is shared between two parties P_1 and P_2, so is the parameter W_0. Using arithmetic sharing, they can compute the matrix product XW_0 securely. Then they can send their shares to a third-party P_3 with label Y and its local model F with trainable parameters W_1. P_3 can then compute gradients $\frac{\partial L(F(XW_0),Y)}{\partial W_1}$ and $\frac{\partial L(F(XW_0),Y)}{\partial (XW_0)}$. .The former gradients are directly used by P_3 to update its local parameters. And the latter gradients are shared to P_1 and P_2. Using the chain rule, P_1 and P_2 are able to calculate their gradients $\frac{\partial L(F(XW_0),Y)}{\partial (W_0)}$ and then update their parameters.

In the third-party's perspective, since he knows nothing about the P_1 and P_2's shared parameters W_0, XW_0 can be considered as random transformations. And with the gradients sent back, the random transformations are actually learning themselves. So the whole 'shared' model can achieve same performance just like a local model.

4.3 Put it All Together

Combining arithmetic sharing and random transformations together, the framework mainly provides following functionalities:

- Arithmetic operations, e.g. addition, subtraction, multiplication, multiplication-like operations like convolution.
- Element-wise functions, e.g. sigmoid, relu and their gradients.
- Distribute further computation to a semi-honest third party after proper transformation. E.g. for neural networks, the first layer (without activation) is performed by arithmetic sharing, then the layer outcome is reconstructed by a third party, who uses his local model to perform further computation.

This framework requires at least 1 semi-honest party to generate multiplication triples and doing element-wise function computations. Another semi-honest third party can be chosen to do further computations, or the party who holds the label data.

And the actual implementation depends on different tasks. For example, logistic regression in federated learning setting, the data holders first share their data on two semi-honest servers, with other computation service providers as helpers of matrix multiplication and performing element-wise function calculation. For deep convolutional networks, first a few layers can be performed in a secure way. After that, the output can be considered as 'random transformed', so a third party with strong computation power can do afterwards computation.

So we proposed two methods under our framework:

(1) RTAS: Only use random transformations for non-linear functions. This method is fit for the tasks where label holder do not have computation power, and the label(raw data, already without any ID) is sensitive. Or all parties do not want any third party to perform further training, and the feature holders do not want to reveal the transformed feature data to label holder(since the label holder have the IDs of samples, he may reuse the transformed feature data without warranty).
(2) RTAS-fast: Distrubute further computation to label holder or a third-party. This method is fit for the tasks where label is considered non-sensitive, or label holder have good computation power and the feature holders are not concerned about the reuse of transformed feature data, for example, the transformation reduced many dimensions, hence the transformed data is useless except in the current task.

5 EXPERIMENTS

In this section, we performed experiments on MNIST dataset using RTAS and RTAS-fast. The speed and the network traffic are measured. First experiment is logistic regression on MNIST[11] dataset predicting whether the given digit is 0 or not. The commercial private computing library Rosetta[1] is used for comparation. Rosetta uses polynomial approximation for sigmoid functions, and its basic functionalities are based on SecureNN. The second experiment is

[1] https://github.com/LatticeX-Foundation/Rosetta

a CNN for image classification on MNIST. For comparison, CryptoNets and SecureNN are used. Both training and inference speed are evaluated.

5.1 Implementation

To realize the framework, we uses tensorflow 2.x as the backend to perform the computations, and all computations is performed in the eager mode. We uses the GRPC library for making RPC calls across different parties. As for random permutation generation and inversion, we uses numpy's random generator. We creates a party to deliver all rpc calls according to the protocol, named the coordinator. When a computation needs to be performed on a party, i.e. loading a data file, doing addition, subtraction or matrix multiplication, the coordinator will generate a string representing the expression. The receiver party parses the string and do computation according to it. When the computation is finished, the receiver party saves the result tensor in its container, and then returns a unique key to the coordinator. For coordinator, the key is representing a 'remote tensor'. When one party needs the value of some other party's tensor, it also makes a rpc call. The tensor is serialized by first converting to numpy array and then uses pickle. Parallel rpc calls is made wherever it is possible. The computation can be composition of basic computations in order to reduce number of rpc calls.

5.2 Dataset and System Settings

We used the MNIST dataset for experiments. The MNIST dataset contains 55000 images of handwriten digits from number 0-9, equally numbered. Each image is of 8-bits gray scale and size 28 × 28. The label is a one hot vector of length 10 indicating the image belong to which number. We used 50000 images for training and 5000 for validating. The image pixel values are is scaled to [0,1) by multiply 1/256 for consistency.

The experiment is executed on a cloud server which has 16 processor cores of frequency 2.5GHz and 64GB memory, and a Tesla T4 GPU. All the parties are simulated by individual python processes. For CryptoNets, since it only supports windows system, we implemented it on my laptop with 16Gb RAM and a 4-core intel i7-7700hq processor with base frequency 2.8GHz. The experiments are conducted in both LAN setting and WAN setting. In LAN setting, all parties' addresses are 127.0.0.1. And in WAN setting, all parties' addresses are the public IP address of the cloud server.

5.3 Logistic Regression

We conducted logistic regression on MNIST dataset. The feature's dimension is 784, and the label is 'whether the digit is 0'. The internet traffic is measured by tsharks program. The formula for logistic regression is

$$\mathbf{y} = sigmoid(\mathbf{x}W + \mathbf{b}) \text{ where } sigmoid(z) = \frac{1}{1 + e^{-z}} \tag{4}$$

Since it contains only matrix addition, multiplication and element-wise function(sigmoid), it can be securely calculated by RTAS and RTAS-fast. So is its gradients.

In the experiment, the batch size is set to 32 and the learning rate is set to 0.1. Mean squared loss and SGD optimization is used.

Table 1: One-batch training/inference time for logistic regression

Network	Model	Rosetta	RTAS	RTAS-fast
LAN	Train			
	Infer			
WAN	Train			
	Infer			

Table 2: One-batch training/inference network traffic(Mb) for logistic regression

Model	Rosetta	RTAS	RTAS-fast
Train			
Infer			

5.4 Convolutional Neural Network

We also tested a convolutional neural network(CNN) using our framework and compared the result with CryptoNets[8] and SecureNN[25]. The structure of the CNN is as follows:

(1) Conv layer: filters=4, kernel size=4×4, stride=2, activation=relu
(2) Dense layer: input size=13*13*5, output size=100, activation=sigmoid
(3) Dense layer: input size=100, output size=10, activation=sigmoid

For inference, we used a batch size 8192, since CryptoNets have a minimum batch size 8192. And since CryptoNets do not supports training we used batch size 32 for training.

Table 3: One-batch training/inference time(s) for CNN

Net	Model	CryptoNets	SecureNN	RTAS	RTAS-fast
LAN	Infer	55.21	122.61±1.01	10.61±0.12	
	Train	-	0.38±0.01	0.54±0.00	
WAN	Infer	55.21+100	6981±231	212.15±0.89	
	Train	-	31.64±0.78	3.25±0.09	

Table 4: One-batch training/inference network traffic(Mb) for CNN

Model	Crypto-Nets	Secure-NN	RTAS	RTAS-fast
Train	-	22632	333.6	
Infer	294	1052.33	10.23	

6 CONCLUSIONS

This paper proposed a new privacy notion called reconstructive privacy. Unlike differential privacy, this privacy notion focuses on the probability of reconstructing useful information from the transformed data. And using this definition, we proved that simple

random permutations and linear transformations are very hard to invert.

Based on this, those random transformations can be applied for private machine learning. Non-linear functions can then be computed easily without garbled circuits or other MPC protocols. Moreover, after the data transformed, third party computation servicers can perform computations on the transformed data. By comparing with Rosetta, CryptoNets and SecureNN, we showed that this method hugely reduces the computation and communication costs.

7 ACKNOWLEDGEMENT

REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep Learning with Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 308–318.

[2] Donald Beaver. 1991. Efficient Multiparty Protocols Using Circuit Randomization. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings (Lecture Notes in Computer Science, Vol. 576)*, Joan Feigenbaum (Ed.). Springer, 420–432. https://doi.org/10.1007/3-540-46766-1_34

[3] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnés, and Bernhard Seefeld. 2017. Prochlo: Strong Privacy for Analytics in the Crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*. ACM, 441–459. https://doi.org/10.1145/3132747.3132769

[4] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 1175–1191. https://doi.org/10.1145/3133956.3133982

[5] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society. https://www.ndss-symposium.org/ndss2015/aby---framework-efficient-mixed-protocol-secure-two-party-computation

[6] Cynthia Dwork and Aaron Roth. 2014. *The Algorithmic Foundations of Differential Privacy*.

[7] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, Indrajit Ray, Ninghui Li, and Christopher Kruegel (Eds.). ACM, 1322–1333. https://doi.org/10.1145/2810103.2813677

[8] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016 (JMLR Workshop and Conference Proceedings, Vol. 48)*, Maria-Florina Balcan and Kilian Q. Weinberger (Eds.). JMLR.org, 201–210. http://proceedings.mlr.press/v48/gilad-bachrach16.html

[9] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, Alfred V. Aho (Ed.). ACM, 218–229. https://doi.org/10.1145/28395.28420

[10] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *27th USENIX Security Symposium (USENIX Security 18)*. 1651–1669.

[11] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[12] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. 2007. t-Closeness: Privacy Beyond k-Anonymity and l-Diversity. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, Rada Chirkova, Asuman Dogac, M. Tamer Özsu, and Timos K. Sellis (Eds.). IEEE Computer Society, 106–115. https://doi.org/10.1109/ICDE.2007.367856

[13] Yehuda Lindell and Benny Pinkas. 2009. A Proof of Security of Yao's Protocol for Two-Party Computation. *Journal of Cryptology* 22, 2 (2009), 161–188.

[14] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. 2007. *L*-diversity: Privacy beyond *k*-anonymity. *ACM Trans. Knowl. Discov. Data* 1, 1 (2007), 3. https://doi.org/10.1145/1217299.1217302

[15] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA (Proceedings of Machine Learning Research, Vol. 54)*, Aarti Singh and Xiaojin (Jerry) Zhu (Eds.). PMLR, 1273–1282. http://proceedings.mlr.press/v54/mcmahan17a.html

[16] Payman Mohassel and Peter Rindal. 2018. ABY^3: A Mixed Protocol Framework for Machine Learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM, 35–52. https://doi.org/10.1145/3243734.3243760

[17] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 19–38. https://doi.org/10.1109/SP.2017.12

[18] Arvind Narayanan and Vitaly Shmatikov. 2008. Robust De-anonymization of Large Sparse Datasets. In *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*. IEEE Computer Society, 111–125. https://doi.org/10.1109/SP.2008.33

[19] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*. Springer, 223–238.

[20] Nicolas Papernot, Martín Abadi, Úlfar Erlingsson, Ian J. Goodfellow, and Kunal Talwar. 2017. Semi-supervised Knowledge Transfer for Deep Learning from Private Training Data. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=HkwoSDPgg

[21] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. 2018. Privacy-Preserving Deep Learning via Additively Homomorphic Encryption. *IEEE Trans. Inf. Forensics Secur.* 13, 5 (2018), 1333–1345. https://doi.org/10.1109/TIFS.2017.2787987

[22] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. 2018. Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, June 04-08, 2018*, Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López, and Taesoo Kim (Eds.). ACM, 707–721. https://doi.org/10.1145/3196494.3196522

[23] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.

[24] Latanya Sweeney. 2002. k-Anonymity: A Model for Protecting Privacy. *Int. J. Uncertain. Fuzziness Knowl. Based Syst.* 10, 5 (2002), 557–570. https://doi.org/10.1142/S0218488502001648

[25] Sameer Wagh, Divya Gupta, and Nishanth Chandran. 2019. SecureNN: 3-Party Secure Computation for Neural Network Training. *Proc. Priv. Enhancing Technol.* 2019, 3 (2019), 26–49. https://doi.org/10.2478/popets-2019-0035

[26] Qiao Zhang, Cong Wang, Hongyi Wu, Chunsheng Xin, and Tran V. Phuong. 2018. GELU-Net: A Globally Encrypted, Locally Unencrypted Deep Neural Network for Privacy-Preserved Learning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, Jérôme Lang (Ed.). ijcai.org, 3933–3939. https://doi.org/10.24963/ijcai.2018/547

[27] Ligeng Zhu, Zhijian Liu, and Song Han. 2019. Deep Leakage from Gradients. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 14747–14756. http://papers.nips.cc/paper/9617-deep-leakage-from-gradients