An Efficient 3-Party Framework for Privacy-Preserving Neural Network Inference

Liyan Shen[1,2](✉), Xiaojun Chen[1], Jinqiao Shi[3], Ye Dong[1,2], and Binxing Fang[4]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
{shenliyan,chenxiaojun,dongye}@iie.ac.cn

[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

[3] Beijing University of Posts and Telecommunications, Beijing, China
shijinqiao@bupt.edu.cn

[4] Institute of Electronic and Information Engineering of UESTC in Guangdong, Dongguan, China
fangbx@cae.cn

Abstract. In the era of big data, users pay more attention to data privacy issues in many application fields, such as healthcare, finance, and so on. However, in the current application scenarios of machine learning as a service, service providers require users' private inputs to complete neural network inference tasks. Previous works have shown that some cryptographic tools can be used to achieve the secure neural network inference, but the performance gap is still existed to make those techniques practical.

In this paper, we focus on the efficiency problem of privacy-preserving neural network inference and propose novel 3-party secure protocols to implement amounts of nonlinear activation functions such as ReLU and Sigmod, etc. Experiments on five popular neural network models demonstrate that our protocols achieve about 1.2×–11.8× and 1.08×–4.8× performance improvement than the state-of-the-art 3-party protocols (SecureNN [28]) in terms of computation and communication overhead. Furthermore, we are the first to implement the privacy-preserving inference of graph convolutional networks.

Keywords: Privacy-preserving computation · Neural network inference · Secret sharing

1 Introduction

Machine learning (ML) has been widely used in many applications such as medical diagnosis, credit risk assessment [4], facial recognition. With the rapid development of ML, Machine Learning as a Service (MLaaS) has been a prevalent business model. Many technology companies such as Google, Microsoft, and

Amazon are providing MLaaS which helps clients benefit from ML without the cognate cost, time, and so on. The most common scenario of using MLaaS is neural network predictions that customers upload their input data to the service provider for inference or classification tasks. However, the data on which the inference is performed involves medical, genomic, financial and other types of sensitive information. For example, the patients would like to predict the probability of heart disease using Google's pre-trained disease diagnosis model. It will require patients to send personal electronic medical records to service providers for accessing inference services. More seriously, due to the restrictive laws and regulations [1,2], the MLaaS service would be prohibited. One naive solution is to let the consumers acquire the model and perform the inference task on locally trusted platforms. However, this solution is not feasible for two main reasons. First, these models were trained using massive amounts of private data, therefore these models have commercial value and the service providers should want them confidential to preserve companies' competitive advantage. Second, revealing the model may compromise the privacy of the training data containing sensitive information [22].

Recent advances in research provide many cryptographic tools like secure multiparty computation (MPC) and fully homomorphic encryption (FHE), to help us address some of these concerns. They can ensure that during the inference, neither the customers' data nor the service providers' models will be revealed by the others except themselves. The only information available for the customers is the inference or classification label, and for the service providers is nothing. Using these tools, privacy-preserving neural network inference could be achieved [15,18,22–24,27,28].

Concretely, the proposed privacy-preserving neural network computation methods mainly use cryptographic protocols to implement each layer of the neural network. Most of the works use mixed-protocols other than a single protocol for better performance. For the linear layers such as fully connected or convolution layers, they are usually represented as arithmetic circuits and evaluated using homomorphic encryption. And for the non-linear layers such as ReLU or Sigmoid functions, they are usually described as a boolean circuit and evaluated using Yao's garbled circuit [30]. There have been many secure computation protocols [6,9,13] used to solve the private computation problems under different circuit representations.

These protocols for the linear layers are practical enough, due to the use of lightweight cryptographic primitives, such as multiplication triples [8] generated by the three-parties [27,28] and homomorphic encryption with the SIMD batch processing technique [18,22]. However, for the non-linear layers, the protocols based on garbled circuits still have performance bottlenecks. Because garbled circuits incur a multiplicative communication overhead proportional to the security parameter. As noted by prior work [22], 2^{16} invocations of ReLU and Sigmoid could lead communication overhead about 10^3 MB and 10^4 MB respectively. This is not feasible in real applications with complex neural networks.

One solution to alleviate the overhead of non-linear functions is three-party secure protocols. SecureNN [28] constructs new and efficient protocols with the help of a third-party and results in a significant reduction in computation and communication overhead. They implement a secure protocol of derivative computation of ReLU function (DReLU) based on a series of sub-protocols such as private compare(PC), share convert(SC) and multiplication (MUL) etc. And then they use DReLU as a basic building block to implement the computation of non-linear layers such as ReLU and Maxpool. The main problem of DReLU protocol in SecureNN is the complexity and it has a long dependence chain of other sub-protocols.

Our work is motivated by SecureNN and attempt to implement a more efficient secure computation of non-linear functions with less complexity. Concretely, we propose a novel and more efficient three-party protocol for DReLU, which is several times faster than the version of SecureNN and only depends on the multiplication sub-protocol. And based on DReLU, we implement not only the activation functions ReLU and Maxpool that are done in SecureNN, but also the more complex activation function Sigmoid. In known secure protocols, Sigmoid is usually approximated by a set of piecewise continuous polynomials and implemented using garbled circuit [22]. Our Sigmoid implementation with a third-party can alleviate the overheads about $360\times$.

Besides, current works have been applied in the inference of many neural networks, such as linear regression, multi-layer perceptron (MLP), deep neural networks (DNN) and convolutional neural networks (CNN), we implement a privacy inference on graph convolutional networks (GCN) [29], which are popular when processing non-Euclidean data structures such as social network, finance transactions, etc. To the best of our knowledge, we are the first to implement the privacy-preserving inference on GCN.

1.1 Contribution and Roadmap

In this paper, we propose an efficient 3-party framework for privacy-preserving neural network inference. In detail, our contributions are described as follows:

- We propose a novel protocol of the DReLU which is much faster than that in [28]. Based on the protocol, our 3-party framework could efficiently implement the commonly used non-linear activation functions in neural networks.
- We are the first work to implement the privacy-preserving inference scheme of GCN on the MNIST dataset with an accuracy of more than 99%. And the GCN model is more complex than those in [28] in terms of the invocation numbers of linear and non-linear functions.
- We give a detailed proof of security and correctness. And we provide the same security as in SecureNN. Concretely, our protocols are secure against one single semi-honest corruption and against one malicious corruption under the privacy notion in Araki et al. [5].
- Experiments demonstrate that our novel protocol can obtain better performance in computation and communication overhead. Concretely, our proto-

cols outperform prior 3-party work SecureNN by 1.2×–11.8×. And our protocol for the secure computation of sigmoid is about 360× faster than that in MiniONN [22].

The remainder of this paper is organized as follows. In Sect. 2 we give the definition of symbols and primitives related to neural networks and cryptographic building blocks. An introduction of the general framework for the privacy-preserving inference of neural networks is given in Sect. 3. In Sect. 4 we propose our efficient 3-party protocol constructions. Then, we give detailed correctness and security analysis of our protocols in Sect. 5. In Sect. 6 we give the implementation of our protocols. The related work is presented in Sect. 7. Finally, we conclude this paper in Sect. 8.

2 Preliminary

2.1 Definitions

Firstly, we define the symbols used in the article in Table 1.

Table 1. Table for notations.

$\mathcal{S}$	Server
$\mathcal{C}$	Client
$\mathcal{P}$	Semi-honest Third Party
$\boldsymbol{a}$	Lowercase bold letter denotes vector
$\boldsymbol{A}$	Uppercase bold letter denotes matrix
$\boldsymbol{a}[i]$	Denotes the ith elements of $\boldsymbol{a}$
PRG	Pseudo Random Generator
$\in_R$	Uniformly random sampling from a distribution
$\in_R D$	Sampling according to a probability distribution D
s	The statistical security parameter

2.2 Neural Networks

The neural networks usually have similar structures with many layers stacked on top of each other, the output of the previous layer serves as the input of the next layer. And the layers are usually classified into linear layers and non-linear layers.

Linear Layers: The main operations in linear layers are matrix multiplications and additions. Concretely,

- The Fully Connected (FC) layer used in many neural networks can be formulated as $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$, where $\boldsymbol{x}$ is the input vector, $\boldsymbol{y}$ is the output of FC layer, $\boldsymbol{W}$ is the weight matrix and $\boldsymbol{b}$ is the bias vector.
- Convolution layer used in CNN networks can be converted into matrix multiplication and addition as noted in [11] and can be formulated as $\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{X} + \boldsymbol{B}$.
- The graph convolution layer used in GCN networks is essentially two consecutive matrix multiplication operations $\boldsymbol{Y} = \sum_{k=0}^{K-1} T_k(\tilde{\boldsymbol{L}})\boldsymbol{X}\boldsymbol{W}$ as we have described in Appendix A.
- The mean pooling operation used in CNN or GCN networks is the average of a region of the proceeding layer. It can be formulated as $y = mean(\boldsymbol{x})$.

Non-linear Layers: Operations in the non-linear layer are usually comparison, exponent, and so on. We identify three common categories:

- Piecewise linear functions [22]. e.g. Rectified Linear Units (ReLU): $f(\boldsymbol{y}) = [max(0, y_i)]$; the max pooling operation: $y = max(\boldsymbol{x})$.
- Smooth activation functions. e.g. Sigmoid: $f(\boldsymbol{y}) = [\frac{1}{1+e^{-y_i}}]$.
- Softmax: $f(\boldsymbol{y}) = [\frac{e^{-y_i}}{\sum_j e^{-y_j}}]$. It is usually applied to the last layer and can be replaced by argmax operation. As in the prediction phase, it is order-preserving and will not change the prediction result.

2.3 Additive Secret Sharing

In our protocols, all values are secret-shared between the server and the client. Consider a 2-out-of-2 secret sharing protocol, a value is shared between two parties P_0 and P_1 such that the addition of two shares yields the true value. Suppose the value x is shared additively in the ring $\mathbb{Z}_{2^\ell}$ (integers modulo 2^ℓ), the two shares of x are denoted as $\langle x\rangle_0$ and $\langle x\rangle_1$.

- The algorithm `Share`(x) generates the two shares of x over $\mathbb{Z}_{2^\ell}$ by choosing $r \in_R \mathbb{Z}_{2^\ell}$ and sets $\langle x\rangle_0 = r$, $\langle x\rangle_1 = x - r \bmod 2^\ell$.
- The algorithm `Reconst`(x_0, x_1) reconstructs the value $x = x_0 + x_1 \bmod 2^\ell$ using x_0 and x_1 as the two shares.

Suppose in two-party applications, both P_0 and P_1 share their inputs x,y with each other. Then, the two parties run secure computation protocols on the input shares.

- For addition operation, $\langle z\rangle = \langle x\rangle + \langle y\rangle$: P_i locally computes $\langle z\rangle_i = \langle x\rangle_i + \langle y\rangle_i$.
- For multiplication operation, $\langle z\rangle = \langle x\rangle \cdot \langle y\rangle$: it will be performed using pre-computed Multiplication Triples (MTs) [8] of the form $\langle c\rangle = \langle a\rangle \cdot \langle b\rangle$. Based on the shares of a MT, multiplication will be performed as follows:
  (1) P_i computes $\langle e\rangle_i = \langle x\rangle_i - \langle a\rangle_i$ and $\langle f\rangle_i = \langle y\rangle_i - \langle b\rangle_i$
  (2) P_i performs `Reconst`(e_0, e_1) and `Reconst`(f_0, f_1)
  (3) P_i sets its output share of the multiplication as

$$\langle z\rangle_i = -i \times e \times f + f \times \langle x\rangle_i + e \times \langle y\rangle_i + \langle c\rangle_i$$

The secret sharing scheme could also be used for matrix $\boldsymbol{X}$ by sharing the elements of $\boldsymbol{X}$ component-wise.

2.4 Threat Model and Security

The parties involved in our protocols are service provider $\mathcal{S}$, the customer $\mathcal{C}$ and the third server $\mathcal{P}$. In our protocols, we consider an adversary who can corrupt only one of the three parties under semi-honest and malicious security.

Semi-honest Security: A semi-honest (passive) adversary is an adversary who corrupts parties but follows the protocol specification. That is, the corrupt parties run the protocol honestly but try to learn additional information from the received messages.

We follow the security proof method in the real-ideal paradigm [10]. Let π be a protocol running in the real interaction and $\mathcal{F}$ be the ideal functionality completed by a trusted party. The ideal-world adversary is referred to as a simulator *Sim*. We define the two interactions as follows:

- $\text{Real}_{\pi}(k, C; x_1, ..., x_n)$ run the protocol π with security parameter k, where each party P_i's input is x_i and the set of corrupted parties is C.
  Output $\{V_i, i \in C\}, (y_1, ..., y_n)$. The final view and final output of P_i are V_i and y_i respectively.
- $\text{Ideal}_{\mathcal{F},Sim}(k, C; x_1, ..., x_n)$ compute $(y_1, ..., y_n) \leftarrow \mathcal{F}(x_1, ..., x_n)$
  Output $Sim(C, \{(x_i, y_i), i \in C\}), (y_1, ..., y_n)$.

In the semi-honest model, a protocol π is secure, it must be possible that the ideal world adversary's view is indistinguishable from the real world adversary's view.

Privacy Against Malicious Adversary: A malicious (active) adversary may cause the corrupted parties to deviate from the protocol. Araki et al. [5] formalized the notion of privacy against malicious adversaries using an indistinguishability based argument, which is weaker than the full simulation-based malicious security. Nevertheless, it guarantees that privacy is not violated even if one of the parties behaves maliciously. The indistinguishability-based malicious security is that for any two inputs of the honest parties, the view of the adversary in the protocol is indistinguishable.

3 The General Framework for the Secure Inference of Neural Networks

The two-party mixed privacy-preserving inference protocol of neural networks is based on additive secret sharing. $\mathcal{S}$ and $\mathcal{C}$ run the interactive secure computation protocol for each layer on the input shares, during which they do not learn any intermediate information. And the outputs of each layer also be secretly shared among the two parties. Suppose the neural network is defined as: $\boldsymbol{y} = (\boldsymbol{W}^{L-1} \cdot f_{L-2}(...f_0(\boldsymbol{W}^0 \cdot \boldsymbol{X})...))$, the corresponding computation process of the model is presented in Fig. 1. Specifically, for each layer, $\mathcal{S}$ and $\mathcal{C}$ will each hold a share of $\boldsymbol{Y}^i$ such that `Reconst` of the shares are equal to the input/output to that layer which is performed in the version of plaintext computation of neural networks. The output values will be used as inputs for the next layer. Finally, $\mathcal{S}$ sends the output shares $\boldsymbol{y}_0$ to $\mathcal{C}$ who can reconstruct the output predictions.

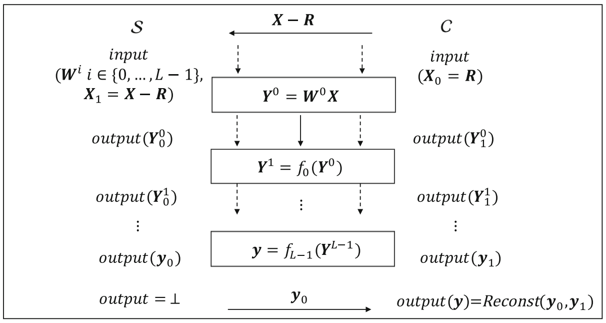
$\mathcal{S}$
$\boldsymbol{X}-\boldsymbol{R}$
$\mathcal{C}$
input
$(\boldsymbol{W}^i\ i \in \{0,\dots,L-1\},$
$\boldsymbol{X}_1 = \boldsymbol{X}-\boldsymbol{R})$
$\boldsymbol{Y}^0 = \boldsymbol{W}^0\boldsymbol{X}$
input
$(\boldsymbol{X}_0 = \boldsymbol{R})$
$output(\boldsymbol{Y}_0^0)$
$output(\boldsymbol{Y}_1^0)$
$\boldsymbol{Y}^1 = f_0(\boldsymbol{Y}^0)$
$output(\boldsymbol{Y}_0^1)$
$output(\boldsymbol{Y}_1^1)$
$output(\boldsymbol{y}_0)$
$\boldsymbol{y} = f_{L-1}(\boldsymbol{Y}^{L-1})$
$output(\boldsymbol{y}_1)$
$output = \perp$
$\boldsymbol{y}_0$
$output(\boldsymbol{y})$=$Reconst(\boldsymbol{y}_0,\boldsymbol{y}_1)$

Fig. 1. The flow chart of the neural network computation process.

Most network weights and input/output features are represented as floating-point numbers, while in cryptographic protocols, they are typically encoded into integer form. In our paper, all values are secretly shared in the ring $\mathbb{Z}_L$, where $L = 2^{64}$. And we follow the work in [28] using fixed-point arithmetic to perform all computations with α and β bits (as in [28], we set $\beta = 13$) for integer and fraction parts respectively. Each value is represented in two's complement format, of which the most significant bit (MSB) is the sign bit. It works straightforwardly for the addition and subtraction of two fixed-point decimal numbers. While for multiplication, it needs to truncate the last β bits of the product.

4 Protocol Constructions

The idea of the 3-party protocol in our design is the same as the 2-party protocol as we described above, except that it needs a third-party $\mathcal{P}$ to assist in the computation for each layer protocol of the neural network. Concretely, for the linear layer, we utilize the protocol [27,28] based on the Beaver MTs which should be generated by the complex cryptographic protocol. We use $\mathcal{P}$ to provide the relevant randomness in the MTs, thereby it can significantly reduce the overhead. The corresponding ideal functionality $\mathcal{F}$ is $\boldsymbol{Z} = \boldsymbol{W}\boldsymbol{X}$, where $\boldsymbol{W} \in \mathbb{Z}_L^{m\times n}$ and $\boldsymbol{X} \in \mathbb{Z}_L^{n\times v}$. The detail of the protocol π_{Mul} is described in Algorithm 1. The parties generate the MTs in step 1. $\mathcal{S}$ and $\mathcal{C}$ compute the matrix multiplication in step 2–4, with each party obtain one share of $\boldsymbol{Z}$.

Algorithm 1. Matrix Multiplication. $\pi_{\text{Mul}}(\{\mathcal{S},\mathcal{C}\},\mathcal{P})$

```
Input:
    $\mathcal{S}$: $\langle \boldsymbol{W} \rangle_0, \langle \boldsymbol{X} \rangle_0$, $\mathcal{C}$: $\langle \boldsymbol{W} \rangle_1, \langle \boldsymbol{X} \rangle_1$
Output:
    $\mathcal{S}$: $\langle \boldsymbol{WX} \rangle_0$, $\mathcal{C}$: $\langle \boldsymbol{WX} \rangle_1$
1: $\mathcal{P}$ generates random matrices $\boldsymbol{A} \in_R \mathbb{Z}_L^{m\times n}$ and $\boldsymbol{B} \in_R \mathbb{Z}_L^{n\times v}$ using PRG, and sends
   $(\langle \boldsymbol{A} \rangle_0, \langle \boldsymbol{B} \rangle_0, \langle \boldsymbol{C} \rangle_0)$, $(\langle \boldsymbol{A} \rangle_1, \langle \boldsymbol{B} \rangle_1, \langle \boldsymbol{C} \rangle_1)$ to $\mathcal{S}$ and $\mathcal{C}$ respectively, where $\boldsymbol{C} = \boldsymbol{AB}$.
2: $\mathcal{S}$ computes $\langle \boldsymbol{E} \rangle_0 = \langle \boldsymbol{W} \rangle_0 - \langle \boldsymbol{A} \rangle_0$ and $\langle \boldsymbol{F} \rangle_0 = \langle \boldsymbol{X} \rangle_0 - \langle \boldsymbol{B} \rangle_0$,
   $\mathcal{C}$ computes $\langle \boldsymbol{E} \rangle_1 = \langle \boldsymbol{W} \rangle_1 - \langle \boldsymbol{A} \rangle_1$ and $\langle \boldsymbol{F} \rangle_1 = \langle \boldsymbol{X} \rangle_1 - \langle \boldsymbol{B} \rangle_1$.
3: $\mathcal{S}$ and $\mathcal{C}$ run algorithm $\boldsymbol{E}$=Reconst$(\langle \boldsymbol{E} \rangle_0, \langle \boldsymbol{E} \rangle_1)$ and $\boldsymbol{F}$=Reconst$(\langle \boldsymbol{F} \rangle_0, \langle \boldsymbol{F} \rangle_1)$ by
   exchanging shares.
4: $\mathcal{S}$ outputs $\langle \boldsymbol{W} \rangle_0 \boldsymbol{F} + \boldsymbol{E} \langle \boldsymbol{X} \rangle_0 + \langle \boldsymbol{C} \rangle_0$,
   $\mathcal{C}$ outputs $\langle \boldsymbol{W} \rangle_1 \boldsymbol{F} + \boldsymbol{E} \langle \boldsymbol{X} \rangle_1 + \langle \boldsymbol{C} \rangle_1 - \boldsymbol{EF}$.
```

For the non-linear layer, we design efficient protocol with the help of $\mathcal{P}$. The detailed description is as follows. We consider the non-linear functions in two categories. The first kind includes ReLU and Max-pooling operation. The second category includes some complex smooth activation functions such as Sigmoid. The secure computation protocols for the non-linear activation functions all are based on the computation of derivatives of ReLU.

Protocol for DReLU: The formula of DReLU is $\text{ReLU}'(x)$. It is 1 if $x \geq 0$ and 0 otherwise. It can be computed by the sign function of x, $\text{ReLU}'(x) = 1 - sign(x)$. Our protocol implements the computation of sign function with the help of $\mathcal{P}$. Concretely, for $\forall x$, $sign(x) = sign(x \cdot r)$ with $r > 0$. $\mathcal{S}$ and $\mathcal{C}$ need to generate random *positive* numbers $r_0 \in_R \mathbb{Z}_L$ and $r_1 \in_R \mathbb{Z}_L$ in advance respectively. It must satisfy that r=`Reconst`(r_0, r_1) be positive number in $\mathbb{Z}_L$. Then $\mathcal{S}$ and $\mathcal{C}$ perform multiplication protocol based on Beaver MTs and the outputs of two parties are $\langle y \rangle_0 = \langle x \cdot r \rangle_0$ and $\langle y \rangle_1 = \langle x \cdot r \rangle_1$ respectively. Since r is randomly generated, the value of $x \cdot r$ is also a random number in $\mathbb{Z}_L$. $\mathcal{S}$ and $\mathcal{C}$ send $\langle y \rangle_0$ and $\langle y \rangle_1$ to $\mathcal{P}$ who performs `Reconst`$(\langle y \rangle_0, \langle y \rangle_1)$. $\mathcal{P}$ computes the sign function of the random number $z = sign(y)$, if $z = 1$, $\mathcal{P}$ generates secret shares of zero and sends each share to $\mathcal{S}$ and $\mathcal{C}$; if $z = 0$, $\mathcal{P}$ generates secret shares of one and sends to $\mathcal{S}$ and $\mathcal{C}$. We could also cut the communication of the last step to half by generating the output shares using a shared PRG key between $\mathcal{P}$ and one of the parties. The protocol is described in Algorithm 2.

It should be noted that after the multiplication protocol, it needs to truncate the last β bits of the product. Because for the multiplication of two fixed-point values, the rightmost 2β bits of the product now corresponds to the fraction part instead of β bits. While in this case, r is a random positive number in $\mathbb{Z}_L$ without scaling, the product of $x \cdot r$ does not need to truncate.

Algorithm 2. $\mathrm{ReLU}'(x)$. $\pi_{\mathrm{DReLU}}(\{\mathcal{S},\mathcal{C}\},\mathcal{P})$

```
Input:
    $\mathcal{S}$: $\langle x\rangle_0$, $\mathcal{C}$: $\langle x\rangle_1$
Output:
    $\mathcal{S}$: $\langle \mathrm{ReLU}'(x)\rangle_0$, $\mathcal{C}$: $\langle \mathrm{ReLU}'(x)\rangle_1$
1: $\mathcal{S}$ and $\mathcal{C}$ generate random positive numbers $r_0 \in_R \mathbb{Z}_L$ and $r_1 \in_R \mathbb{Z}_L$ resp., s.t.
   $r$=Reconst$(r_0, r_1)$ be positive number in $\mathbb{Z}_L$.
   $\mathcal{P}$ generates shares of zero $\langle u\rangle_i = \langle 0\rangle_i$ and shares of one $\langle v\rangle_i = \langle 1\rangle_i$, $i \in \{0,1\}$.
2: $\mathcal{S}$, $\mathcal{C}$ and $\mathcal{P}$ perform $\pi_{\mathrm{Mul}}(\{\mathcal{S},\mathcal{C}\},\mathcal{P})$ protocol with input $(\langle x\rangle_i, r_i)$ and output $\langle y\rangle_i$,
   $i \in \{0,1\}$ of $\mathcal{S},\mathcal{C}$ resp.
3: $\mathcal{S}$ and $\mathcal{C}$ send the shares of $\langle y\rangle_i$ to $\mathcal{P}$ who performs Reconst$(\langle y\rangle_0, \langle y\rangle_1)$.
4: $\mathcal{P}$ computes the sign function of the random number $z = sign(y)$,
   if $z = 1$, $\mathcal{P}$ sends $\langle u\rangle_0$, $\langle u\rangle_1$ to $\mathcal{S}$ and $\mathcal{C}$ resp, $\mathcal{S}$ outputs $\langle u\rangle_0$, $\mathcal{C}$ outputs $\langle u\rangle_1$;
   if $z = 0$, $\mathcal{P}$ sends $\langle v\rangle_0$, $\langle v\rangle_1$ to $\mathcal{S}$ and $\mathcal{C}$ resp, $\mathcal{S}$ outputs $\langle v\rangle_0$, $\mathcal{C}$ outputs $\langle v\rangle_1$.
```

Protocol for ReLU: The formula of ReLU is $\mathrm{ReLU}(x) = x{\cdot}\mathrm{ReLU}'(x)$. It equals the multiplication of x and $\mathrm{ReLU}'(x)$, the protocol is described in Algorithm 3. Similarly, the product of $x{\cdot}\mathrm{ReLU}'(x)$ does not need to truncate.

Algorithm 3. ReLU. $\pi_{\mathrm{ReLU}}(\{\mathcal{S},\mathcal{C}\},\mathcal{P})$

```
Input:
    $\mathcal{S}$: $\langle x\rangle_0$, $\mathcal{C}$: $\langle x\rangle_1$
Output:
    $\mathcal{S}$: $\langle \mathrm{ReLU}(x)\rangle_0$, $\mathcal{C}$: $\langle \mathrm{ReLU}(x)\rangle_1$
1: $\mathcal{S}$, $\mathcal{C}$ and $\mathcal{P}$ perform $\pi_{\mathrm{DReLU}}(\{\mathcal{S},\mathcal{C}\},\mathcal{P})$ protocol with input $\langle x\rangle_i$ and output $\langle y\rangle_i$,
   $i \in \{0,1\}$ of $\mathcal{S},\mathcal{C}$ resp.
2: $\mathcal{S}$, $\mathcal{C}$ and $\mathcal{P}$ perform $\pi_{\mathrm{Mul}}(\{\mathcal{S},\mathcal{C}\},\mathcal{P})$ protocol with input $(\langle x\rangle_i, \langle y\rangle_i)$ and output
   $\langle c\rangle_i$, $i \in \{0,1\}$ of $\mathcal{S},\mathcal{C}$ resp.
```

Protocol for Maxpool: The Maxpool protocol is the same as that in [28] except that we replace the DReLU module with our novel protocol π_{DReLU}. We give a brief description of the protocol in Appendix B due to space limits. And we point the reader to [28] for further details on the Maxpool protocol.

Protocol for Sigmoid: We adapt the method in [22] to approximate the smooth activation functions that can be efficiently computed and incurs negligible accuracy loss. The activation function $f()$ is split into $m+1$ intervals using m knots which are switchover positions for polynomials expressions. For simplicity, the author uses 1-degree polynomial to approximate sigmoid function:

$$\bar{f}(x) = \begin{cases} 0 & x < x_1 \\ a_1 x + b_1, & x_1 \le x < x_2 \\ \ldots & \\ a_{m-1}x + b_{m-1} & x_{m-1} \le x < x_m \\ 1 & x \ge x_m \end{cases} \tag{1}$$

However, the author uses garbled circuit to implement the approximate activation functions which results in a large amount of overhead. Instead, we propose a 3-party protocol for the approximate sigmoid function $\bar{f}(x)$. It is clear that $\bar{f}(x)$ should be public to all parties. According formula 2, $\mathcal{S}$ and $\mathcal{C}$ will be able to determine the range of x by the subtraction of x and knot x_i.

$$\bar{f}(x) = \begin{cases} 0 & \forall i, i \in \{1, ..., m\}, x - x_i < 0 \\ a_i x + b_i & \exists i, i \in \{1, ..., m-1\}, x - x_{i+1} < 0, x - x_i \geq 0 \\ 1 & \forall i, i \in \{1, ..., m\}, x - x_i \geq 0 \end{cases} \tag{2}$$

$\mathcal{S}$, $\mathcal{C}$ and $\mathcal{P}$ can perform a variation of the π_{DReLU} protocol to determine the range of x, which is described in Algorithm 4. $\mathcal{S}$ and $\mathcal{C}$ compute the subtraction of x and knot x_i in step 1. With the help of $\mathcal{P}$, the parties determine the range of x in step 2–5. The outputs of $\mathcal{S}$ and $\mathcal{C}$ are shares of a_i and b_i parameters in $\bar{f}()$. π_{DReLU} can be seen as a special form of π_{VoDReLU}, which only has one knot $x_1 = 0$.

Algorithm 4. VoDReLU. $\pi_{\text{VoDReLU}}(\{\mathcal{S}, \mathcal{C}\}, \mathcal{P})$

```
Input:
    $\mathcal{S}$: $\langle x \rangle_0$, $\mathcal{C}$: $\langle x \rangle_1$
Output:
    $\mathcal{S}$: $\langle a_i \rangle_0$, $\langle b_i \rangle_0$, $\mathcal{C}$: $\langle a_i \rangle_1$, $\langle b_i \rangle_1$
    suppose $x$ in the $(i+1)$th interval for $i \in \{0, ..., m\}$
1: $\mathcal{S}$ and $\mathcal{C}$ compute the subtraction of $x$ and $x_i$ locally, for $i \in \{0, ..., m-1\}$.
    $\mathcal{S}$: $\langle \boldsymbol{y} \rangle_0 = \boldsymbol{x} - \hat{\boldsymbol{x}}$, with $\boldsymbol{x}[i] = \langle x \rangle_0$, $\hat{\boldsymbol{x}}[i] = x_{i+1}$.
    $\mathcal{C}$: $\langle \boldsymbol{y} \rangle_1 = \boldsymbol{x}$, with $\boldsymbol{x}[i] = \langle x \rangle_1$. s.t. $\boldsymbol{y}[i] = x - x_{i+1}$.
2: $\mathcal{S}$ and $\mathcal{C}$ generate random positive numbers $\boldsymbol{r}_0 \in_R \mathbb{Z}_L^m$ and $\boldsymbol{r}_1 \in_R \mathbb{Z}_L^m$ resp, s.t.
    $\boldsymbol{r} = \texttt{Reconst}(\boldsymbol{r}_0, \boldsymbol{r}_1)$, $\boldsymbol{r}[i]$ be positive number in $\mathbb{Z}_L$.
    $\mathcal{P}$ generates shares of $a_i$ and $b_i$ for $i \in \{0, ..., m\}$ and $a_0 = b_0 = a_m = 0$, $b_m = 1$.
3: $\mathcal{S}, \mathcal{C}$ and $\mathcal{P}$ perform $\pi_{\text{Mul}}(\{\mathcal{S}, \mathcal{C}\}, \mathcal{P})$ protocol element-wise with input $(\langle \boldsymbol{y} \rangle_j, \boldsymbol{r}_j)$ and
    output $\langle \boldsymbol{z} \rangle_j$, $j \in \{0, 1\}$ of $\mathcal{S}, \mathcal{C}$ resp.
4: $\mathcal{S}$ sends $\langle \boldsymbol{z} \rangle_0$ and $\mathcal{C}$ sends $\langle \boldsymbol{z} \rangle_1$ to $\mathcal{P}$ who performs $\texttt{Reconst}(\langle \boldsymbol{z} \rangle_0, \langle \boldsymbol{z} \rangle_1)$.
5: $\mathcal{P}$ computes the sign function of the random number $\boldsymbol{c} = sign(\boldsymbol{z})$
    if $\exists i \in \{0, ..., m-2\}, \boldsymbol{c}[i] = 0$ and $\boldsymbol{c}[i+1] = 1$, $\mathcal{P}$ sends $\langle u \rangle_j = \langle a_i \rangle_j$, $\langle v \rangle_j = \langle b_i \rangle_j$ to
    $\mathcal{S}$ and $\mathcal{C}$ resp, for $j \in \{0, 1\}$;
    if $\forall i \in \{0, ..., m-1\}, \boldsymbol{c}[i] = 1$, $\mathcal{P}$ sends $\langle u \rangle_j = \langle a_0 \rangle_j$, $\langle v \rangle_j = \langle b_0 \rangle_j$ to $\mathcal{S}$ and $\mathcal{C}$ resp,
    for $j \in \{0, 1\}$;
    if $\forall i \in \{0, ..., m-1\}, \boldsymbol{c}[i] = 0$, $\mathcal{P}$ sends $\langle u \rangle_j = \langle a_m \rangle_j$, $\langle v \rangle_j = \langle b_m \rangle_j$ to $\mathcal{S}$ and $\mathcal{C}$ resp,
    for $j \in \{0, 1\}$;
    $\mathcal{S}$ outputs $\langle u \rangle_0$, $\langle v \rangle_0$ and $\mathcal{C}$ outputs $\langle u \rangle_1$, $\langle v \rangle_1$;
```

The protocol of Sigmoid is described in Algorithm 5. In order to reduce the overhead, the outputs of π_{VoDReLU} could also be the value of i which indicates x in the $i+1$th interval. Then $\mathcal{S}$ and $\mathcal{C}$ could perform multiplication locally, that is $a_i x = a_i(\langle x \rangle_0 + \langle x \rangle_1)$, since all a_i, b_i are public.

The dependence of these protocols is presented in Fig. 2. It can be seen that the protocol for ReLU only depends on the DReLU and Mul subprotocol. Similarly, the protocol dependency of Sigmoid and Maxpool is simple.

Algorithm 5. Sigmoid. $\pi_{\text{Sigmoid}}(\{\mathcal{S},\mathcal{C}\},\mathcal{P})$

```
Input:
    S: $\langle x\rangle_0$, C: $\langle x\rangle_1$
Output:
    S: $\langle \bar{f}(x)\rangle_0$, C: $\langle \bar{f}(x)\rangle_1$
1: $\mathcal{S}$, $\mathcal{C}$ and $\mathcal{P}$ perform $\pi_{\text{VoDReLU}}(\{\mathcal{S},\mathcal{C}\},\mathcal{P})$ protocol with input $\langle x\rangle_i$ and output $\langle u\rangle_i$, $\langle v\rangle_i$ $i \in \{0,1\}$ of $\mathcal{S},\mathcal{C}$ resp.
2: $\mathcal{S}$, $\mathcal{C}$ and $\mathcal{P}$ perform $\pi_{\text{Mul}}(\{\mathcal{S},\mathcal{C}\},\mathcal{P})$ protocol with input $(\langle x\rangle_i, \langle u\rangle_i)$ and output $\langle c\rangle_i$, $i \in \{0,1\}$ of $\mathcal{S},\mathcal{C}$ resp.
3: $\mathcal{S}$ outputs $\langle c\rangle_0 + \langle v\rangle_0$, $\mathcal{C}$ outputs $\langle c\rangle_1 + \langle v\rangle_1$.
```

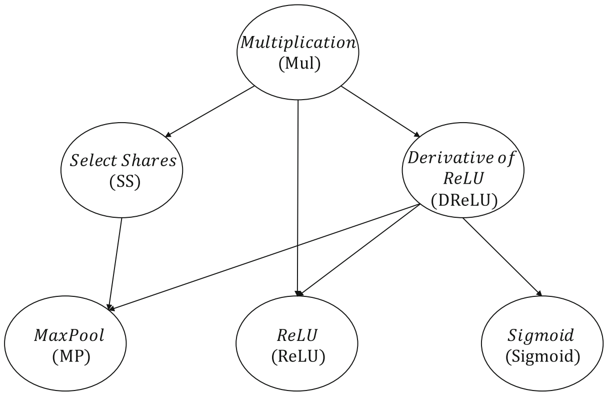

Fig. 2. Protocol dependence of the non-linear activation function.

5 Correctness and Security Analysis

5.1 Correctness Analysis

The protocols are correct as long as the core dependency module of the framework π_{DReLU} is correct. There are two conditions that must be satisfied.

1. As we described above, π_{DReLU} relies on the computation of sign function and it is obvious that $sign(x) = sign(x \cdot r)$ as long as $\forall r > 0$.
2. All the values in the protocol are in the ring $\mathbb{Z}_L$, in order to make sure the correctness of the protocol, all absolute value of any (intermediate) results will not exceed $\lfloor \frac{L}{2} \rfloor$.

Therefore, we let the bit length of random value $r \in_R \mathbb{Z}_L$ be relatively small, such as smaller than 32. Then r will be positive value in the ring and the probability that the result $x \cdot r$ exceeds $\lfloor \frac{L}{2} \rfloor$ will be ignored.

5.2 Security Analysis

Our protocols provide security against one single corrupted party under the semi-honest and malicious model. The universal composability framework [10] guarantees the security of arbitrary composition of different protocols. Therefore, we only need to prove the security of individual protocols. we first proof the security under semi-honest model in the real-ideal paradigm.

Semi-honest Security

Security for π_{DReLU}: The output of the simulator *Sim* which simulates for corrupted $\mathcal{S}$ is one share generated by the algorithm `Share` which is uniformly random chosen from $\mathbb{Z}_L$ and $\mathcal{S}$'s view in the real execution is also random share which is indistinguishable with that in ideal execution. It is the same for the simulation of $\mathcal{C}$. The output of the simulator *Sim* for $\mathcal{P}$ is a random value in $\mathbb{Z}_L$ that is indistinguishable with $x \cdot r$ generated in real-world execution.

Security for π_{ReLU}: $\mathcal{P}$ learns no information from the protocol, for both π_{DReLU} $(\{\mathcal{S},\mathcal{C}\},\mathcal{P})$ and $\pi_{\mathrm{Mul}}(\{\mathcal{S},\mathcal{C}\},\mathcal{P})$ provide outputs only to $\mathcal{S}$ and $\mathcal{C}$. $\mathcal{S}$'s and $\mathcal{C}$'s view in the real execution is random share, which is indistinguishable with that in ideal execution.
Security for π_{Maxpool}: The Maxpool protocol in [28] has been proven secure, the only difference between us is that we have replaced the sub-protocol DReLU. π_{DReLU} has been proven secure as mentioned above. Due to the universal composability, the Maxpool protocol we use is secure under the semi-honest model.

Security for π_{VoDReLU}: The views of $\mathcal{S}$ and $\mathcal{C}$ are random shares in $\mathbb{Z}_L$, and the same as the views in π_{DReLU}. The view of $\mathcal{P}$ is a set of random shares $\boldsymbol{x} \circ \boldsymbol{r}$ ($\circ$ denotes hadamard product) rather than one in π_{DReLU}, and it is indistinguishable with that in ideal execution.

In order to reduce the overhead, we also provide an alternative solution. The outputs of $\mathcal{S}$ and $\mathcal{C}$ are plaintext values i indicating the interval information. Based on i, $\mathcal{S}$ and $\mathcal{C}$ could infer the range of input x and output $\bar{f}(x)$. However, the floating-point number corresponding to x in an interval is infinite. The only case that $\mathcal{S}$ could infer the classification result of $\mathcal{C}$ is that $\mathcal{S}$ knows exactly the output $\bar{f}(x)$ of hidden layer. We will prove that this situation doesn't exist.

Suppose the input dimension of the sigmoid function is p, where the number of input in range (x_1, x_m) is q, and the number of input in range $(-\infty, x_1), (x_m, \infty)$ is $p - q$. Then there will be $\alpha = (\eta \cdot 2^{\beta})^q$ possible values of $\bar{f}(\boldsymbol{x})$, where $\boldsymbol{x} \in \mathbb{Z}_L^p$, η is the interval size between two continuous knots. Our protocol could be secure when the statistical security parameter s is 40 by choosing appropriate value of α. For example, we set $\eta = 2$ (e.g. the sigmoid function is approximated in the set of knots $[-20, -18, ..., 18, 20]$) and $\beta = 13$, then as long as $q \geq 3$, statistical security can be satisfied.

In practical applications, the nodes in a hidden layer are usually high-dimensional vector, so our alternative scheme is secure under the semi-honest model.

Security for π_{Sigmoid}: $\mathcal{P}$ learns no information from the protocol, for both π_{VoDReLU} $(\{\mathcal{S},\mathcal{C}\},\mathcal{P})$ and $\pi_{\mathrm{Mul}}(\{\mathcal{S},\mathcal{C}\},\mathcal{P})$ provide outputs only to $\mathcal{S}$ and $\mathcal{C}$. $\mathcal{S}$'s and $\mathcal{C}$'s view in the real execution is random share, which is indistinguishable with that in ideal execution.

Malicious Security
The same as [28], our protocols provide privacy against a malicious $\mathcal{S}$ or $\mathcal{P}$ in the indistinguishability paradigm [5]. This holds because all the incoming messages to $\mathcal{S}$ or $\mathcal{P}$ are random shares generated in $\mathbb{Z}_L$. For any two inputs of the honest $\mathcal{C}$, the view of the adversary is indistinguishable.

Collusion Analysis
Compared with SecureNN, our protocols for the non-linear functions can resist against collusion attack, as long as the underlying multiplication module is collusion resistant. For example, we can replace the multiplication protocol based on the 3-party with a traditional 2-party protocol based on HE or other techniques. Because the random number r_i owned by $\mathcal{S}$ and $\mathcal{C}$ is generated separately, it can be regarded as one-time-pad, thereby protecting the private information of each party. However, the protocols in SecureNN will reveal computation results at each layer, as long as the third-party collude with $\mathcal{S}$ or $\mathcal{C}$ even if the underlying multiplication protocol is collusion resistant.

6 Experimental Results

6.1 Experimental Enviroment

Our experiments are executed on a server with an Intel Xeon E5-2650 CPU(2.30 GHz) and 126GB RAM in two environments, respectively modeling LAN and WAN settings. The network bandwidth and latency are simulated using Linux Traffic Tools(tc) command. We consider the WAN setting with 40MB/s and 50 ms RTT (round-trip time). All our protocols and that in SecureNN has been implemented and executed more than 10 times on our server (The source code of SecureNN is obtained from Github[1] and we use their code directly in our comparison experiment). We take the average of experimental results for comparison and analysis.

6.2 Neural Networks

We evaluate the experiments in five different neural networks denoted by A, B, C, D and E over the MNIST dataset. More details about those neural networks can be found in Appendix C.
Network A is a 3-layer deep neural network comprising of three FC layers with a ReLU activation function next to every FC layer (Fig. 3).
Network B is a 3-layer convolutional neural network comprising of one convolutional layer and two full connection layers. Every layer is followed by a ReLU activation function (Fig. 4).
Network C is 4-layer convolutional neural networks, the first two layers of which are convolutional layers followed by ReLU and a 2×2 Maxpool. The last two layers are FC layers followed by the ReLU activation function (Fig. 5).

[1] https://github.com/snwagh/securenn-public.

Network D is 4-layer convolutional neural networks similar to **Network C**, and is a larger version with more output channels and more weights (Fig. 5).
Network E is a graph convolutional neural network on MNIST datasets. The network consists of one graph convolutional layer with 32 output feature maps, followed by the ReLU activation layer and one FC layer (Fig. 6).

Network A, B, C and D were also evaluated in SecureNN, and graph convolutional neural network E is implemented for private inference for the first time. For network E, we construct an 8-nearest neighbor graph of the 2D grid with $|\mathcal{V}| = 784$ nodes. Each image x is transformed to a 784 dimension vector and the pixel value of an image x_i serves as the 1-dimensional feature on vertex v_i. We simplify the model in [12]. Network E just have one graph convolutional layer without pooling operation rather than the original model with two graph convolutional layer followed by a pooling operation separately, and the order of filter is set $K = 5$ rather than $K = 25$. The simplified network structure can faster the secure computation and also achieve an accuracy of more than 99% on MINST classification task.

6.3 Inference Results

Compared with previous two-party protocols, such as SecureML [24] and Minionn [22], the performance gains for 3-party protocols SecureNN come from using the third-party to generate Beaver MTs for the linear matrix multiplication and avoiding the use of garbled circuits for the non-linear activation functions. However, SecureNN still depends on a series of subprotocol for the secure computation of non-linear activation functions, such as DReLU protocol, it first needs to convert values that are shared over $\mathbb{Z}_L$ into shares over $\mathbb{Z}_{L-1}$ and then computes the MSB of this value based on private compare and multiplication subprotocol and finally convert results that are shared over $\mathbb{Z}_{L-1}$ into shares over $\mathbb{Z}_L$ for the computation of the next linear layer. Our protocol of the DReLU only depends on the third-party multiplication subprotocol which is obviously faster than SecureNN.

The experimental inference results of the five neural network are described in Table 2. We run the privacy-preserving inference for 1 prediction and batch of 128 predictions in both the LAN and WAN settings. It can be found that our protocol is about 1.2×–4.7× faster than SecureNN for 1 prediction, about 2.5×–11.8× faster for 128 predictions and the communication cost is about 1.08×–4.8× fewer than SecureNN. The neural network E is more complex than C or D, for they have 3211264, 1323520 and 1948160 invocations of ReLU respectively, and the invocations for multiplication module is also larger than that of network C and D. And the result of SecureNN for network E is performed based on their source code.

The experimental microbenchmark results for various protocols are described in Table 3. It is about 2×–40× faster than SecureNN and the communication cost of the DReLU, ReLU and Maxpool protocols are about 9.2×, 6.3× and 4.9× fewer than those in SecureNN respectively.

Table 2. Inference results for batch size 1 vs 128 on Networks A-E over MNIST.

Batch size		LAN (s)		WAN (s)		Comm (MB)	
		1	128	1	128	1	128
A	SecureNN	0.036	0.52	6.16	7.39	2.1	29
	Us	**0.02**	**0.11**	**3.09**	**3.29**	**1.94**	**8.28**
B	SecureNN	0.076	4.24	7.76	25.86	4.05	317.7
	Us	**0.03**	**0.68**	**3.87**	**7.43**	**2.28**	**90.45**
C	SecureNN	0.14	14.204	9.34	64.37	8.86	1066
	Us	**0.03**	**1.2**	**4.86**	**15.35**	**2.73**	**281.65**
D	SecureNN	0.24	19.96	10.3	89.05	18.94	1550
	Us	**0.07**	**2.25**	**5.11**	**19.84**	**9.92**	**395.21**
E	SecureNN	1.917	32.993	7.57	128.48	61.76	2398.83
	Us	**1.605**	**9.071**	**6.06**	**29.25**	**46.91**	**497.76**

Table 3. Microbenchmarks in LAN/WAN settings.

Protocol	Dimension	LAN (s)		WAN (s)		Comm (MB)	
		SecureNN	Us	SecureNN	Us	SecureNN	Us
DReLU	64×16	15.66	**1.61**	464.21	**190.79**	0.68	**0.074**
	128×128	204.54	**7.09**	1058.5	**200.146**	10.88	**1.18**
	576×20	134.05	**3.28**	874.36	**195.9**	7.65	**0.83**
ReLU	64×16	15.71	**1.65**	513.25	**233.98**	0.72	**0.115**
	128×128	210.2	**8.5**	1115.48	**281.543**	11.534	**1.835**
	576×20	135.8	**5.6**	921.76	**266.91**	8.11	**1.29**
Maxpool	$8 \times 8 \times 50, 4 \times 4$	64.9	**13.59**	7641.7	**3812.79**	2.23	**0.46**
	$24 \times 24 \times 16, 2 \times 2$	85.6	**5.79**	1724.7	**802.02**	5.14	**1.05**
	$24 \times 24 \times 20, 2 \times 2$	98.4	**6.82**	1763.9	**804.98**	6.43	**1.31**

Table 4. Overhead of secure computation of the sigmoid function.

	2^{16}		2^{20}	
	LAN (ms)	Comm (MB)	LAN (ms)	Comm (MB)
MiniONN	10^5	10^4	–	–
Us	**278.039**	**88.08**	**5470.12**	**1409.29**

We perform our sigmoid approximation protocol with the ranges as $[x_1, x_m] = [-11, 12]$ and 25 pieces, the experimental results are described in Table 4. Our protocol achieves about 360× improvement, and the communication cost is about 114× fewer than that in MiniONN when there are 2^{16} invocations of the sigmoid function. We also implement 2^{20} invocations of the sigmoid function with acceptable overhead.

7 Related Work

As a very active research field in the literature, privacy-preserving inference of neural networks has been extensively considered in recent years. And existing work mainly adopts different security protocols to realize the secure computation of neural networks. Early work in the area can be traced back to the work by Barni et al. [7,25], which is based on homomorphic encryption. Gilad-Bachrach et al. [15] proposed CryptoNets based on leveled homomorphic encryption (LHE) which supports additions and multiplications and only allows a limited number of two operations. Thus, only low-degree polynomial functions can be computed in a straightforward way. Due to the limitations LHE, the author proposed several alternatives to the activation functions and pooling operations used in the CNN layers, for example, they used square activation function instead of ReLU and mean pooling instead of max pooling. It will affect the accuracy of the model.

Instead of purely relying on HE, most of the work uses multiple secure protocols. SecureML presented protocols for privacy-preserving machine learning for linear regression, logistic regression and neural network training. The data owners distributed their data among two non-colluding servers to train various models using secure two-party computation (2PC). The author performed multiplications based on additive secret sharing and offline-generated multiplication triplets. And the non-linear activation function is approximated using a piecewise linear function, which is processed using garbled circuits.

MiniONN further optimized the matrix multiplication protocols. It performed leveled homomorphic encryption operations with the single instruction multiple data (SIMD) batch processing technique to complete the linear transformation in offline. The author used polynomial splines to approximate widely-used nonlinear functions (e.g. sigmoid and tanh) with negligible loss in accuracy. And the author used secret sharing and garbled circuits to complete the activation functions in online.

Chameleon [27] used the same technique as in [24] to complete the matrix multiplication operations. The difference is that [24] completed the multiplication triplets based on oblivious transfer [17,20] or HE, while Chameleon completed it with a third-party that can generate correlated randomness used in multiplication triplets. Besides, the oblivious transfer used in garbled circuits also completed based on the third-party. Almost all of the heavy cryptographic operations are precomputed in an offline phase which substantially reduces the computation and communication overhead. However, some information could be revealed if the third-party colluded with either party.

Gazelle [18] is state-of-the-art 2PC privacy-preserving inference framework of neural networks. It performed multiplications based on homomorphic encryption and used specialized packing schemes for the Brakerski-Fan-Vercauteren (BFV) [14] scheme. However, in order to achieve circuit privacy, the experiment result is about 3-3.6 times slow down for the HE component of Gazelle [26]. Similarly, it used garbled circuits for non-linear activations.

All prior works [22,24,27] use a secure computation protocol for Yao's garbled circuits to compute the non-linear activation functions. However, it has been noted [5] that garbled circuits incur a multiplicative overhead proportional to the security parameter in communication and are a major computational bottleneck. SecureNN constructed a novel protocol for non-linear functions such as ReLU and maxpool that completely avoid the use of garbled circuits. It is developed with the help of a third-party and is the state-of-the-art protocol in secure training and inference of CNNs.

Besides, there are some other works combine with quantization / binary neural networks [3,26] or some other constructions based on three-party protocols [23] (the three parties both have shares of the private input in the model).

8 Conclusions

In this paper, we propose novel and efficient 3-party protocols for privacy-preserving neural network inference. And we are the first work to implement the privacy-preserving inference scheme of the graph convolutional network. We conducted a detailed analysis of the correctness and security of this scheme. Finally, we give an implementation for the private prediction of five different neural networks, it has better performance than the previous 3-party work.

Acknowledgment. This work was supported by the Strategic Priority Research Program of Chinese Academy of Sciences, Grant (No.XDC02040400), DongGuan Innovative Research Team Program (Grant No.201636000100038).

A Graph Convolutional Network

For spectral-based GCN, the goal of these models is to learn a function of signals/features on a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{A})$, where $\mathcal{V}$ is a finite set of $|\mathcal{V}| = \mathrm{n}$ vertices, $\mathcal{E}$ is a set of edges and $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is a representative description of the graph structure typically in the form of an adjacency matrix. It will be computed based on the training data by the server and the graph structure is identical for all signals [16]. $\boldsymbol{x} \in \mathbb{R}^n$ is a signal, each row x_v is a scalar for node v. An essential operator in spectral graph analysis is the graph Laplacian $\boldsymbol{L}$, and the normalized definition is $\boldsymbol{L} = \boldsymbol{I}_n - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$, where $\boldsymbol{I}_n$ is the identity matrix and $\boldsymbol{D}$ is the diagonal node degree matrix of $\boldsymbol{A}$.

Defferrard et al. [12] proposed to approximate the graph filters by a truncated expansion in terms of Chebyshev polynomials. The Chebyshev polynomials are recursively defined as $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, with $T_0(x) = 1$ and $T_1(x) = x$. And filtering of a signal $\boldsymbol{x}$ with a K-localized filter $g_{\boldsymbol{\theta}}$ can be performed using: $g_{\boldsymbol{\theta}} * \boldsymbol{x} = \sum_{k=0}^{K-1} \boldsymbol{\theta}_k T_k(\tilde{\boldsymbol{L}})\boldsymbol{x}$, with $\tilde{\boldsymbol{L}} = \frac{2}{\lambda_{max}}\boldsymbol{L} - \boldsymbol{I}_n$. λ_{max} is the largest eigenvalue of $\boldsymbol{L}$. $\boldsymbol{\theta} \in \mathbb{R}^K$ is a vector of Chebyshev coefficients.

The definition generalized to a signal matrix $\boldsymbol{X} \in \mathbb{R}^{n \times c}$ with c dimensional feature vector for each node and f feature maps is as follows: $\boldsymbol{Y} = \sum_{k=0}^{K-1} T_k(\tilde{\boldsymbol{L}})\boldsymbol{X}\boldsymbol{\Theta}_k$ where $\boldsymbol{Y} \in \mathbb{R}^{n \times f}$, $\boldsymbol{\Theta}_k \in \mathbb{R}^{c \times f}$ and the total number of trainable parameters per layer is $c \times f \times K$ ($\boldsymbol{\Theta} \in \mathbb{R}^{K \times c \times f}$). Through graph convolution layer, GCN can capture feature vector information of all neighboring nodes. It preserves both network topology structure and node feature information.

However, the privacy-preserving inference of GCN is not suitable for node classification tasks, in which test nodes (without labels) are included in GCN training. It could not quickly generate embeddings and make predictions for unseen nodes [19].

B Maxpool Protocol

Algorithm 6. Maxpool. $\pi_{\text{MP}}(\{\mathcal{S},\mathcal{C}\},\mathcal{P})$

```
Input:
    $\mathcal{S}$: $\{\langle x_i\rangle_0\}_{i\in[n]}$, $\mathcal{C}$: $\{\langle x_i\rangle_1\}_{i\in[n]}$
Output:
    $\mathcal{S}$: $\langle y\rangle_0$, $\mathcal{C}$: $\langle y\rangle_1$, s.t. $y = \mathrm{Max}(\{\mathrm{x_i}\}_{\mathrm{i\in[n]}})$
1: $\mathcal{S}$ sets $\langle max_1\rangle_0 = \langle x_1\rangle_0$ and $\mathcal{C}$ sets $\langle max_1\rangle_1 = \langle x_1\rangle_1$.
2: for $i = \{2, ..., n\}$ do
3:     $\mathcal{S}$ sets $\langle w_i\rangle_0 = \langle x_i\rangle_0 - \langle max_{i-1}\rangle_0$ and $\mathcal{C}$ sets $\langle w_i\rangle_1 = \langle x_i\rangle_1 - \langle max_{i-1}\rangle_1$.
4:     $\mathcal{S}$, $\mathcal{C}$ and $\mathcal{P}$ perform $\pi_{\text{DReLU}}(\{\mathcal{S},\mathcal{C}\},\mathcal{P})$ protocol with input $(\langle w_i\rangle_j)$ and output
       $\langle y_i\rangle_j$, $j \in \{0,1\}$ of $\mathcal{S},\mathcal{C}$ resp.
5:     $\mathcal{S}$, $\mathcal{C}$ and $\mathcal{P}$ perform $\pi_{\text{SS}}(\{\mathcal{S},\mathcal{C}\},\mathcal{P})$ protocol with input $(\langle y_i\rangle_j$, $\langle max_{i-1}\rangle_j$,$\langle x_i\rangle_j$,$)$
       and output $\langle max_i\rangle_j$, $j \in \{0,1\}$ of $\mathcal{S},\mathcal{C}$ resp.
6: end for
7: $\mathcal{S}$ outputs $\langle max_n\rangle_0$ and $\mathcal{C}$ outputs $\langle max_n\rangle_1$.
```

Algorithm 7. SelectShare. $\pi_{\text{SS}}(\{\mathcal{S},\mathcal{C}\},\mathcal{P})$

```
Input:
    $\mathcal{S}$: $(\langle\alpha\rangle_0, \langle x\rangle_0, \langle y\rangle_0)$, $\mathcal{C}$: $(\langle\alpha\rangle_1, \langle x\rangle_1, \langle y\rangle_1)$
Output:
    $\mathcal{S}$: $\langle z\rangle_0$, $\mathcal{C}$: $\langle z\rangle_1$, s.t. $z = (1-\alpha)x + \alpha y$
1: $\mathcal{S}$ sets $\langle w\rangle_0 = \langle y\rangle_0 - \langle x\rangle_0$ and $\mathcal{C}$ sets $\langle w\rangle_1 = \langle y\rangle_1 - \langle x\rangle_1$.
2: $\mathcal{S}$, $\mathcal{C}$ and $\mathcal{P}$ perform $\pi_{\text{Mul}}(\{\mathcal{S},\mathcal{C}\},\mathcal{P})$ protocol with input $(\langle\alpha\rangle_j, \langle w\rangle_j)$ and output
   $\langle c\rangle_j$, $j \in \{0,1\}$ of $\mathcal{S},\mathcal{C}$ resp.
3: $\mathcal{S}$ outputs $\langle z\rangle_0 = \langle x\rangle_0 + \langle c\rangle_0$ and $\mathcal{C}$ outputs $\langle z\rangle_1 = \langle x\rangle_1 + \langle c\rangle_1$.
```

C Neural Network Structure

(1) *FC*: input image 28×28, the output: $\mathbb{R}^{128\times 1} \leftarrow \mathbb{R}^{128\times 784} \cdot \mathbb{R}^{784\times 1}$
(2) *ReLU activation*: calculates ReLU for each input.
(3) *FC*: input size 128, the output: $\mathbb{R}^{128\times 1} \leftarrow \mathbb{R}^{128\times 128} \cdot \mathbb{R}^{128\times 1}$
(4) *ReLU activation*: calculates ReLU for each input.
(5) *FC*: input size 128, the output: $\mathbb{R}^{10\times 1} \leftarrow \mathbb{R}^{10\times 128} \cdot \mathbb{R}^{128\times 1}$
(6) *ReLU activation*: calculates ReLU for each input.

Fig. 3. The neural network A presented in SecureML [24]

(1) *Convolution*: input image 28×28, window size 5×5, stride (2,2), output channels 5: $\mathbb{R}^{5\times 196} \leftarrow \mathbb{R}^{5\times 25} \cdot \mathbb{R}^{25\times 196}$
(2) *ReLU activation*: calculates ReLU for each input.
(3) *FC*: input size 980, the output: $\mathbb{R}^{100\times 1} \leftarrow \mathbb{R}^{100\times 980} \cdot \mathbb{R}^{980\times 1}$
(4) *ReLU activation*: calculates ReLU for each input.
(5) *FC*: input size 100, the output: $\mathbb{R}^{10\times 1} \leftarrow \mathbb{R}^{10\times 100} \cdot \mathbb{R}^{100\times 1}$
(6) *ReLU activation*: calculates ReLU for each input.

Fig. 4. The neural network B presented in Chameleon [27]

(1) *Convolution*: input image 28×28, window size 5×5, stride (1,1), output channels 16: $\mathbb{R}^{16\times 576} \leftarrow \mathbb{R}^{16\times 25}\cdot\mathbb{R}^{25\times 576}$ /output channels 20: $\mathbb{R}^{20\times 576} \leftarrow \mathbb{R}^{20\times 25}\cdot\mathbb{R}^{25\times 576}$
(2) *ReLU activation*: calculates ReLU for each input.
(3) *Max Pooling*: window size 2×2, stride (2,2), the output: $\mathbb{R}^{16\times 12\times 12}/\mathbb{R}^{20\times 12\times 12}$
(4) *Convolution*: window size 5×5, stride (1,1), output channels 16: $\mathbb{R}^{16\times 64} \leftarrow \mathbb{R}^{16\times 400} \cdot \mathbb{R}^{400\times 64}$/output channels 50: $\mathbb{R}^{50\times 64} \leftarrow \mathbb{R}^{50\times 400} \cdot \mathbb{R}^{400\times 64}$
(5) *ReLU activation*: calculates ReLU for each input.
(6) *Max Pooling*: window size 2×2, stride (2,2), the output: $\mathbb{R}^{16\times 4\times 4}/\mathbb{R}^{50\times 4\times 4}$
(7) *FC*: input size 256, the output: $\mathbb{R}^{100\times 1} \leftarrow \mathbb{R}^{100\times 256} \cdot \mathbb{R}^{256\times 1}$
/*FC*: input size 800, the output: $\mathbb{R}^{500\times 1} \leftarrow \mathbb{R}^{500\times 800} \cdot \mathbb{R}^{800\times 1}$
(8) *ReLU activation*: calculates ReLU for each input.
(9) *FC*: input size 100, the output: $\mathbb{R}^{10\times 1} \leftarrow \mathbb{R}^{10\times 100} \cdot \mathbb{R}^{100\times 1}$
/*FC*: input size 500, the output: $\mathbb{R}^{10\times 1} \leftarrow \mathbb{R}^{10\times 500} \cdot \mathbb{R}^{500\times 1}$
(10) *ReLU activation*: calculates ReLU for each input.

Fig. 5. The neural network C/D presented in MiniONN [22] and [21] resp.

(1) *Graph convolution*: input image 28×28, with $T_k(\tilde{\boldsymbol{L}}) \in \mathbb{R}^{784\times784}$, $\boldsymbol{W}_k \in \mathbb{R}^{1\times32}$ output $\sum_{k=0}^{K-1} T_k(\tilde{\boldsymbol{L}})\boldsymbol{x}\boldsymbol{W}_k$, $\mathbb{R}^{784\times32} \leftarrow \mathbb{R}^{784\times784} \cdot \mathbb{R}^{784\times1} \cdot \mathbb{R}^{1\times32}$.
(2) *ReLU activation*: calculates ReLU for each input.
(3) *FC*: $\mathbb{R}^{10\times1} \leftarrow \mathbb{R}^{10\times(784\times32)} \cdot \mathbb{R}^{(784\times32)\times1}$

Fig. 6. Graph convolutional neural network trained from the MNIST dataset

References

1. The health insurance portability and accountability act of 1996 (hipaa). https://www.hhs.gov/hipaa/index.html
2. Regulation (eu) 2016/679 of the European parliament and of the council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (gdpr). https://gdpr-info.eu/
3. Agrawal, N., Shahin Shamsabadi, A., Kusner, M.J., Gascón, A.: Quotient: two-party secure neural network training and prediction. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 1231–1247 (2019)
4. Angelini, E., di Tollo, G., Roli, A.: A neural network approach for credit risk evaluation. Q. Rev. Econ. Finan. **48**(4), 733–755 (2008)
5. Araki, T., Furukawa, J., Lindell, Y., Nof, A., Ohara, K.: High-throughput semi-honest secure three-party computation with an honest majority. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 805–817 (2016)
6. Barni, M., Failla, P., Kolesnikov, V., Lazzeretti, R., Sadeghi, A.-R., Schneider, T.: Secure evaluation of private linear branching programs with medical applications. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 424–439. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04444-1_26
7. Barni, M., Orlandi, C., Piva, A.: A privacy-preserving protocol for neural-network-based computation. In: Proceedings of the 8th workshop on Multimedia and security, pp. 146–151 (2006)
8. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 420–432. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_34
9. Bogdanov, D., Laur, S., Willemson, J.: Sharemind: a framework for fast privacy-preserving computations. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 192–206. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88313-5_13
10. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science, pp. 136–145. IEEE (2001)
11. Chellapilla, K., Puri, S., Simard, P.: High performance convolutional neural networks for document processing (2006)
12. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Advances in Neural Information Processing Systems, pp. 3844–3852 (2016)
13. Demmler, D., Schneider, T., Zohner, M.: Aby-a framework for efficient mixed-protocol secure two-party computation. In: NDSS (2015)

14. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptology ePrint Archive 2012, 144 (2012)
15. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In: International Conference on Machine Learning, pp. 201–210 (2016)
16. Henaff, M., Bruna, J., LeCun, Y.: Deep convolutional networks on graph-structured data. arXiv preprint arXiv:1506.05163 (2015)
17. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_9
18. Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.: {GAZELLE}: a low latency framework for secure neural network inference. In: 27th {USENIX} Security Symposium ({USENIX} Security 18), pp. 1651–1669 (2018)
19. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
20. Kolesnikov, V., Kumaresan, R.: Improved OT extension for transferring short secrets. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 54–70. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_4
21. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
22. Liu, J., Juuti, M., Lu, Y., Asokan, N.: Oblivious neural network predictions via minionn transformations. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 619–631 (2017)
23. Mohassel, P., Rindal, P.: Aby3: a mixed protocol framework for machine learning. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 35–52 (2018)
24. Mohassel, P., Zhang, Y.: Secureml: a system for scalable privacy-preserving machine learning. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 19–38. IEEE (2017)
25. Orlandi, C., Piva, A., Barni, M.: Oblivious neural network computing via homomorphic encryption. EURASIP J. Inf. Secur. **2007**, 1–11 (2007). https://doi.org/10.1155/2007/37343
26. Riazi, M.S., Samragh, M., Chen, H., Laine, K., Lauter, K., Koushanfar, F.: {XONN}: Xnor-based oblivious deep neural network inference. In: 28th {USENIX} Security Symposium ({USENIX} Security 19), pp. 1501–1518 (2019)
27. Riazi, M.S., Weinert, C., Tkachenko, O., Songhori, E.M., Schneider, T., Koushanfar, F.: Chameleon: a hybrid secure computation framework for machine learning applications. In: Proceedings of the 2018 on Asia Conference on Computer and Communications Security, pp. 707–721 (2018)
28. Wagh, S., Gupta, D., Chandran, N.: Securenn: 3-party secure computation for neural network training. Proc. Priv. Enhanc. Technol. **2019**(3), 26–49 (2019)
29. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A comprehensive survey on graph neural networks. arXiv preprint arXiv:1901.00596 (2019)
30. Yao, A.C.C.: How to generate and exchange secrets. In: 27th Annual Symposium on Foundations of Computer Science (sfcs 1986), pp. 162–167. IEEE (1986)