2PC-NNP: Two-Party Computation Framework for Neural Network Predictions Via a Novel Secret Comparison protocol

Written by AAAI Press Staff[1]*
AAAI Style Contributions by Pater Patel Schneider, Sunil Issar, J. Scott Penberthy, George Ferguson, Hans Guesgen, Francisco Cruz†**, Marc Pujol-Gonzalez**†

[1]Association for the Advancement of Artificial Intelligence
2275 East Bayshore Road, Suite 160
Palo Alto, California 94303
publications22@aaai.org

Abstract

Deep learning technology has been widely used in image classification, medical diagnosis and more. Many AI technology companies provide cloud-based prediction services with pre-trained neural network model . However, data owners will inevitably upload their data when requesting such services, resulting in privacy leakage.

For the privacy of the input data and the model, secure multi-party computation is utilized to correctly evaluate the neural network while protecting privacy. The methods currently used all involve asymmetric public key encryption system and have low efficiency.

We propose 2PC-NNP, a two-party computation (2PC) framework for efficient neural network predictions.Compared with the previous work, 2PC-NNP is based on a novel 2PC comparison protocol, which does not require the oblivious transfer (OT) protocol or garbled circuit and only needs three rounds of interactions to calculate the sign of the secret input. Based on this, we design the schemes for evaluating layers of the neural network. Furthermore, we implement the two-party computation for resnet50 network, which costs 23.86 seconds of computation time and 100.13 seconds of communication time.

Introduction

Deep learning, a popular research direction in the field of machine learning, has been widely used in image, voice and text recognition scenes. The application of neural network consists of two stages: (1) training stage: the model is trained with training data, (2) prediction stage: the pre-trained model is used to classify the input data. This work mainly focus on the prediction stage.

Machine learning as a service (MLaaS) is a range of machine learning tools offered by cloud service providers (Hunt et al. 2018). Some tasks that cannot be easily accomplished by data owner can be outsourced to the companies who provide cloud-based pre-trained deep learning model. For example, the CT images of lungs taken by hospitals can be transferred to the cloud computing center supported by major technical companies for medical diagnosis. However, during this period, the data owner inevitably reveals his input images, leading to a privacy leakage.

To address this issue, privacy preserving machine learning based on secure multi-party computation (MPC) is proposed. MPC computation protocols enable players to evaluate a public function with multiple private secret inputs from different players. MPC protocols provide the value of the public function or shares of the value held by each player without disclosing more information than what can be inferred from the result. Via performing MPC protocols, multiple parties can evaluate the prediction without disclosing the input image. The two parties computation (2PC) setting which only involves two parties is a particular case of MPC and the 2PC setting is used throughout the paper.

The biggest difficulty of 2PC framework is that it costs too much to evaluate the comparison operation. The previous works usually utilize the asymmetric encryption operations like the OT protocol and

Our contributions and overview

Here we summarize our contributions on a very high level.

- **2PC comparison protocol**. We begin with the building block of our work, a brand new 2PC-`comparison` protocol, which does not require the asymmetric cryptography operations or the bit-level operations and only needs three rounds of interactions to calculate the sign of the secret input. The speed of it is $40\times$ faster than the sharemind (Sharemind 2022). Equiped with the novel 2PC-`comparison` protocol, we design schemes for more complicated functions like maximum function and piecewise functions.
- **2PC protocols for neural network layers**. Based on the novel 2PC-`comparison` protocol and the exsiting 2PC-`multiplication` protocol, we design 2PC protocols for convolution layer, pooling layer and activation function.
- **2PC-NNP**. Based on the techniques mentioned above, we present 2PC-NNP, a two-party computation framework for evaluating neural network predictions by two

*With help from the AAAI Publications Committee.
†These authors contributed equally.

$\mathcal{DO}$	Data owner
$\mathcal{MP}$	Model provider
$\mathcal{TTP}$	The trusted setup
$\mathcal{P}_0, \mathcal{P}_1$	Two parties who run 2PC-NNP
$\mathcal{P}_i$	The i-th party
$\langle x \rangle$	Additively shared value x
$\langle x \rangle^i$	Additive share of value x held by $\mathcal{P}_i$
$[\![x]\!]$	Multiplicatively shared value x
$[\![x]\!]^i$	Multiplicative share of value x held by $\mathcal{P}_i$
$\otimes$	Multiplication operation performed on shares
$(\langle a \rangle, \langle b \rangle, \langle c \rangle)$	Multiplication triple
$(\langle A \rangle, \langle B \rangle, \langle C \rangle)$	Matrix multiplication triple
$(\langle u \rangle, [\![u]\!])$	Comparison two-tuple

Table 1: Notations

servers efficiently and secretly. To demonstrate the outstanding performance of our work, we conduct 2PC-NNP on the classical neural network model Resnet50 and it costs 23s for computation and 100s for communication.

Related Work

Many ideas have been proposed to achieve privacy preserving neural network predictions. The most common idea is to utilize homomorphic encryption. The data owner sends the encrypted image to the model provider for applying a pretrained model for classification. However, the overhead of homomorphic encryption is so huge that though the associated technology evolves over many years(Liu et al. 2017), the cost of it seems still unacceptable.

To reduce the overhead of complicated homomorphic encryption operations, some more efficient schemes for MPC operations are presented. On a very high level, MPC multiplication is easy to implement by the Beaver triple (Beaver 1991) but MPC comparison is very hard to realize. To avoid the expensive cost of comparison, a modified deep learning model (Gilad-Bachrach et al. 2016) using square activation function and the mean pooling so that no comparison operation is needed in MPC neural network. Obviously, this architecture has a narrow scope of application.

In order to realize practical comparison protocol, ABY framework (Demmler, Schneider, and Zohner 2015) is proposed and then be extended to ABY3 (Mohassel and Rindal 2018) which can be performed among 3 parties. Both ABY and ABY3 involves the asymmetric cryptography operations including the oblivious transfer protocol and the garbled circuit. Sharemind provides a new idea that convert a comparison operation to multi-rounds bit-level multiplication. Though it lowers the computational difficulty, it costs too much rounds for communication.

Preliminaries

Notations

Notations frequently used in this paper is shown in Table 1. Moreover, we use capital letter like A to denote matrix and $\overrightarrow{v}$ to denote vector $\overrightarrow{v} = (v_0, v_1, ..., v_{n-1})$.

Function $sign(x)$ is defined to represent the sign of value x. Notably, $sign(x \cdot y) = sign(x) \cdot sign(y)$

$$sign(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Secure two party computation

A secure two party computation protocol for function $F()$ is that with confidential input x_i of player $i \in \{0, 1\}$, it provides the value $F(x_0, x_1)$ to each players without disclosing more information than what can be inferred from the result. We use '2PC' to stand for 'secure two party computation'. The two parties running 2PC protocols are denoted as $\mathcal{P}_0$ and $\mathcal{P}_1$. In this work, we consider both party as a semi-honest party who will not deviate the protocol but keeps tring to get more information about the secret input of another party from information arise from execution of protocol.

2PC secret sharing

Secret sharing is a method that a secret is split in some ways and each share is held by each participant and 2PC Secret sharing means the secret value is shared between two parties. Our framework involves two types of 2PC secret sharing methods: additive sharing and multiplicative sharing.

- We denote an additively shared secret value x by $\langle x \rangle$. When two parties additively share value x, each party $\mathcal{P}_i$ holds the secret share as $\langle x \rangle^i$, s.t. $x = \langle x \rangle^0 + \langle x \rangle^1$.

  Similarly, we use $\langle A \rangle$ to indicate an additively shared secret matrix and $A = \langle A \rangle^0 + \langle A \rangle^1$.
- We denote a multiplicatively shared secret value x by $[\![x]\!]$. When parties multiplicatively share the secret value x, each party $\mathcal{P}_i$ holds the secret share as $[\![x]\!]^i$, s.t. $x = [\![x]\!]^0 \times [\![x]\!]^1$.

All protocols we present operate on shares, which means the input and the output of protocols is hidden for all parties. For instance, we use $\otimes$ to represent the multiplication between two additively shared value. When performing multiplication operations on shares, we write $\langle z \rangle = \langle x \rangle \otimes \langle y \rangle$, which means computing the product of two additively shared value $\langle x \rangle, \langle y \rangle$ and returning the the additive share of the product. $\mathcal{P}_i$ can only get the value of $\langle x \rangle^i$, $\langle y \rangle^i$ and $\langle z \rangle^i$. This enable two servers to run multiple rounds of different protocols while keeping secret state input.

Auxiliary tuples

Here we introduce some auxiliary tuples of specific structures. Shares of these tuples will be distributed to each party to assit computation for multiplication and comparison.

- Multiplication triple: $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$, where $c = a \cdot b$.
- Matrix multiplication triple: $(\langle A \rangle, \langle B \rangle, \langle C \rangle)$, where $C = A \cdot B$.
- Comparison two-tuple: $(\langle u \rangle, [\![u]\!])$, this tuple constructs of the additive share and multiplicative share of the same value u ($u \neq 0$).

Layers of neural network

Here we briefly introduce the layers of neural network.

- **Convolution layer:** Convolution is an operation between the imput image and the filter. It computes the dot product of the filter and the corresponding submatrix in the input image. This step is repeated by sliding the filter to get the output matrix. To improve the efficiency, convolution is usually converted to matrix multiplication and addition.
- **Activation function:** Activation function is a function acting on the neuron of the artificial neural network, mapping the input of the neuron to the output. Common used activation functions are listed as follow:

$$Maxout(n\ \ pieces) : f(x) = max(\overrightarrow{x})$$
$$Sigmoid : \sigma(x) = \frac{1}{1+e^{-x}}$$
$$ReLU(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leqslant 0 \end{cases}$$

  In the paper, we mainly introduce method to compute $ReLU$ function. Other functions can be computed by similar approach.
- **Pooling layer:** Pooling layer is calculated by $n \times n$ size matrix sliding window. There are two types of pooling layer: average pooling and maximum pooling, which outputs the average or the maximum of $n \times n$ size matrices to construct the input image of the next layer.

Two-party Secure Computation for Basic Operations

In this section we introduce 2PC protocols for basic operations, including addition, comparison, maximum and piecewise function. Our protocols all operate on shares, which means the input and the output is in the form of additive share. This enable two servers to run multiple rounds of different protocols while keeping secret state input and facilitate further processing to implement a complex system.

Formalized protocols are presented in the appendix.

Addition

Computing $\langle z\rangle = \langle x\rangle + \langle y\rangle$. This work can be done locally because $\mathcal{P}_i$ only needs to add his share of x and y together by setting $\langle z\rangle^i = \langle x\rangle^i + \langle y\rangle^i$.

Multiplication

Computing $\langle z\rangle = \langle x\rangle \otimes \langle y\rangle$. `Multiplication` protocol is used to compute the product of two shared values and is performed using a prepared multiplication triple $(\langle a\rangle, \langle b\rangle, \langle c\rangle)$, which can be distributed from trusted third party or generated by using the OT protocol.

Protocol procedure:

- **1.** $\mathcal{P}_i$ gets a multiplication triple $(\langle a\rangle^i, \langle b\rangle^i, \langle c\rangle^i)$ from the trusted third party.
- **2.** $\mathcal{P}_i$ sets $\langle e\rangle^i = \langle x\rangle^i - \langle a\rangle^i$ and $\langle f\rangle^i = \langle y\rangle^i - \langle b\rangle^i$ and sends $(\langle e\rangle^i, \langle f\rangle^i)$ to $\mathcal{P}_{1-i}$.
- **3.** $\mathcal{P}_i$ reconstructs $e = \langle e\rangle^0 + \langle e\rangle^1$ and $f = \langle f\rangle^0 + \langle f\rangle^1$. Then, $\mathcal{P}_i$ sets $\langle z\rangle^i = i \cdot e \cdot f + f \cdot \langle a\rangle^i + e \cdot \langle b\rangle^i + \langle c\rangle^i$.

Moreover, if the secret input data is not in the share state, such as $\mathcal{P}_0$ holding secret value x and $\mathcal{P}_1$ holding secret value y. In this situation, the value x and y can be seen as a special share state, where $\langle x\rangle^0 = x, \langle x\rangle^1 = 0$ and $\langle y\rangle^0 = 0, \langle x\rangle^1 = y$.

They can compute $\langle z\rangle = x \otimes y$ secretly by running `multiplication` protocol with special input of $\mathcal{P}_0 : (x, 0), \mathcal{P}_1 : (0, y)$.

Computing $\langle Z\rangle = \langle X\rangle \otimes \langle Y\rangle$. Similarly, there is a protocol for the 2PC matrix multiplication.

Protocol procedure:

- **1.** $\mathcal{P}_i$ gets a multiplication triple $(\langle A\rangle^i, \langle B\rangle^i, \langle C\rangle^i)$ from the trusted third party.
- **2.** $\mathcal{P}_i$ sets $\langle E\rangle^i = \langle X\rangle^i - \langle A\rangle^i$ and $\langle F\rangle^i = \langle Y\rangle^i - \langle B\rangle^i$ and transmits $(\langle E\rangle^i, \langle F\rangle^i)$ to $\mathcal{P}_{1-i}$.
- **3.** $\mathcal{P}_i$ reconstructs $E = \langle E\rangle^0 + \langle E\rangle^1$ and $F = \langle F\rangle^0 + \langle F\rangle^1$. Then, $\mathcal{P}_i$ sets $\langle Z\rangle^i = i \cdot E \cdot F + \langle A\rangle^i \cdot F + E \cdot \langle B\rangle^i + \langle C\rangle^i$.

Comparison

The 2PC `comparison` protocol is the most important building block of our work. This section is divided into two parts and we will introduce solutions to these problems respectively: how to compute $sign(x)$ and $max(x, y)$ via 2PC protocols.

Computing $\langle s\rangle = \langle sign(\langle x\rangle)\rangle$. The `comparison` protocol for computing the sign of the shared value utilizes the comparison two-tuple $(\langle u\rangle, [\![u]\!])$.

In our scheme, firstly, the two parties compute $\langle z\rangle = \langle x\rangle \otimes \langle u\rangle$ via `multiplication` protocol to blind the value of x. Then, $\mathcal{P}_1$ sends $\langle z\rangle^1$ to $\mathcal{P}_0$ and $\mathcal{P}_0$ reconstructs the true value of z. Note that

$$x = \frac{x \cdot u}{u} = \frac{z \cdot 1}{[\![u]\!]^0 \cdot [\![u]\!]^1} = \frac{z}{[\![u]\!]^0} \cdot \frac{1}{[\![u]\!]^1}$$

Recall that the $sign()$ function is a multiplicative homomorphic function, so

$$sign(x) = \frac{sign(x \cdot u)}{sign(u)} = sign(\frac{z}{[\![u]\!]^0}) \cdot sign(\frac{1}{[\![u]\!]^1})$$

Because the $[\![u]\!]^0$ and $[\![u]\!]^1$ is held by $\mathcal{P}_0$ and $\mathcal{P}_1$ respectively. Therefore, the multiplicative share of $sign(x)$ can be obtained by $\mathcal{P}_0$ setting $[\![s]\!]^0 = sign(\frac{z}{[\![u]\!]^0})$ and $\mathcal{P}_1$ setting $[\![s]\!]^1 = sign(\frac{1}{[\![u]\!]^1})$ locally so that $sign(x) = [\![s]\!]^0 \cdot [\![s]\!]^1$.

Since the additive share is more general to use, the multiplicative share $[\![s]\!]$ should be converted into the additive share $\langle s\rangle$ by calculating $\langle s\rangle = [\![s]\!]^0 \otimes [\![s]\!]^1$. More precisely, two parties run `multiplication` protocol with the input of $\mathcal{P}_0 : ([\![s]\!]^0, 0), \mathcal{P}_1 : (0, [\![s]\!]^1)$, and

$$\begin{aligned} \langle s\rangle^0 + \langle s\rangle^1 &= ([\![s]\!]^0 + 0) \otimes (0 + [\![s]\!]^1) \\ &= [\![s]\!]^0 \cdot [\![s]\!]^1 \\ &= sign(x) \end{aligned}$$

The complete protocol is run as follow.

Protocol procedure:

- **1. Setup**: $\mathcal{P}_i$ gets a comparison two-tuple $(\langle u\rangle^i, [\![u]\!]^i)$ from the trusted third party.
- **2. Blinding**: $\mathcal{P}_i$ computes $\langle z\rangle = \langle u\rangle \otimes \langle x\rangle$ via `multiplication` protocol.
- **3. Reconstructing**: $\mathcal{P}_1$ sends $\langle z\rangle^1$ to $\mathcal{P}_0$ and $\mathcal{P}_0$ reconstructs $z = \langle z\rangle^0 + \langle z\rangle^1$.
- **4. Setting** $[\![s]\!]$: $\mathcal{P}_0$ sets $[\![s]\!]^0 = sign(\frac{z}{[\![u]\!]^0})$ and $\mathcal{P}_1$ sets $[\![s]\!]^1 = sign(\frac{1}{[\![u]\!]^1})$ locally.
- **5. Computing** $\langle s\rangle$: By running `multiplication` protocol with input of $\mathcal{P}_0$: $([\![s]\!]^0, 0), \mathcal{P}_1$: $(0, [\![s]\!]^1)$, two parties calculate $\langle s\rangle = [\![s]\!]^0 \otimes [\![s]\!]^1$.

Computing $\langle maximum\rangle = \langle max(\langle x\rangle, \langle y\rangle)\rangle$. This part aims at providing a method to compute the maximum of two shared values $\langle x\rangle$ and $\langle y\rangle$.

It is easy to find that

$$max(x,y) = \begin{cases} x & x > y \\ \frac{x+y}{2} & x = y \\ y & x < y \end{cases}$$

let $s = sign(x - y)$, which means

$$s = \begin{cases} 1 & x > y \\ 0 & x = y \\ -1 & x < y \end{cases}$$

We can combine two expressions into one:

$$max(x,y) = \frac{(s+1)\cdot x + (-s+1)\cdot y}{2}$$

Following this idea, the whole process of computing $\langle maximum\rangle = \langle max(\langle x\rangle, \langle y\rangle)\rangle$ is given below.

Protocol procedure:

- **1.** $\mathcal{P}_i$ sets $\langle\Delta\rangle^i = \langle x\rangle^i - \langle y\rangle^i$.
- **2.** Two parties run `comparison` protocol to get $\langle s\rangle = \langle sign(\langle\Delta\rangle)\rangle$.
- **3.** Two parties calculate this expression via `multiplication` protocol:

$$\langle maximum\rangle = \frac{(\langle s\rangle + 1)\otimes\langle x\rangle + (-\langle s\rangle + 1)\otimes\langle y\rangle}{2}$$

Furthermore, we explore variations of $max(x, y)$. Firstly, we talk about $max(\overrightarrow{x}) = max(x_0, x_1, ...x_n)$. Oberving that

$$\begin{aligned} max(x_0, x_1, ...x_n) &= max(x_0, max(x_1, x_2, ...x_n)) \\ &= max(x_0, max(x_1, ...max(x_{n-1}, x_n))) \end{aligned}$$

So the maximum of vector $\overrightarrow{x}$ can be computed by recursively running `comparison` protocol.

Secondly, because of $min(x, y) = -max(-x, -y)$, $min(x, y)$ can also be computed. Combined with the above methods, any composite function constructed of $max()$ and $min()$ can be computed following rules metioned above.

Continuous piecewise function

Computing $\langle value\rangle = \langle F(\langle x\rangle)\rangle$. $F()$ is a public continuous piecewise function constructed by a series of sub-functions $f_j(x)$ with dividing point $-\infty = dp_0 < dp_1 < ... < dp_j < ... < dp_n = +\infty$. When $dp_{j-1} < x \leq dp_j$, $F(x) = f_j(x)$. Since $F()$ is a continuous function, $F(dp_j) = f_j(dp_j) = f_{j+1}(dp_j) = \frac{f_j(dp_j)+f_{j+1}(dp_j)}{2}$.

To solve this problem, we need to compute a vector of $\overrightarrow{v} = (v_0, v_1, ..., v_{n-1})$ to represent which span is x in. Let $v_j = \frac{1-sign(dp_{j-1}-x)\cdot sign(dp_j-x)}{2}$. In view of $-\infty = dp_0$ and $dp_n = +\infty$, we set $sign(dp_0 - x) = -1$ and $sign(dp_n - x) = 1$. It is easy to find that

$$v_j = \begin{cases} 0 & \text{if } x \notin (dp_{j-1}, dp_j] \\ \frac{1}{2} & \text{if } x = dp_{j-1} \text{ or } dp_j \\ 1 & \text{if } x \in (dp_{j-1}, dp_j] \end{cases}$$

If $x \in (dp_{j-1}, dp_j)$, then, $v_j = 1$ and $v_k = 0, k \neq j$,$F(x) = v_j \cdot f_j(x) = \sum_{j=1}^{n} v_j \cdot f_j(x)$.

If $x = dp_j$, $dp_j \in dp_1, ..., dp_{n-1}$, then, $v_j = v_{j+1} = \frac{1}{2}$ while $v_k = 0$, if $k \notin \{j, j+1\}$, Since $F(x)$ is a continuous function, $F(x) = f_j(x) = f_{j+1}(x) = \frac{f_j(x)}{2} + \frac{f_{j+1}(x)}{2} = \sum_{j=1}^{n} v_j \cdot f_j(x)$.

So we can conclude two circumstances as $F(x) = \sum_{j=1}^{n} v_j \cdot f_j(x)$.

With `multiplication` protocol and `comparison` protocol, we give the whole process as follow.

Protocol procedure:

- **1.** for each $dp_j, j \in \{1, 2, ..., n-1\}$: two parties computes $\langle s_j\rangle = \langle sign(dp_j - \langle x\rangle)\rangle$.
- **2.** for each $v_j, j \in \{1, 2, ..., n-1\}$: two parties computes $\langle v_j\rangle = \frac{1-\langle s_j\rangle\otimes\langle s_{j-1}\rangle}{2}$.
- **3.** two parties computes $\langle value\rangle = \langle F(x)\rangle = \sum_{j=1}^{n}\langle v_j\rangle \otimes \langle f_j(x)\rangle$.

2PC Neural Network Prediction

In this section, we will introduce the system framework and concrete implementation of our schemes for 2PC neural network prediction.

System framework

The basic setting we used is the classical client-server ($\mathcal{C} - \mathcal{S}$) setting, where the data owner (client) and the model provider (server) directly take the role of $\mathcal{P}_0$ and $\mathcal{P}_1$ to run 2PC-NNP.

In this section, we will introduce schemes of 2PC protocols for each layer in neural network and finally, we present an implementation for a toy neural network.

2PC-neural network layers

neural network has various architectures and most of these architectures are constructed of convolution layer, pooling layer, activation function layer and other variant layers. We design the privacy preserving calculation method to these basic layers.

Convolution layer When convolution is operated in practice, the input image X and weight tensor W are rearranged into 2-dimension matrix X' and W'. We denote this unrolling processing by

$$(X', W') = Unroll(X, W)$$

Convolution can be converted into matrix multiplication and addition: $Y = X' \cdot W' + B'$, where B' is a matrix rearranged from $\overrightarrow{b}$, the bias vector . Y is the output of convolution. Since the addition is easy to implement, we ignore the bias vector in this section by setting $\overrightarrow{b} = \overrightarrow{0}$. So the convolution can be simplified to $Y = X' \cdot W'$.

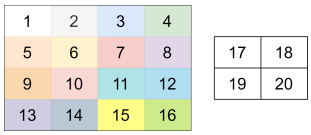


Figure 1: Input image and weight

Figure 1 shows a mini-convolution, where exists a $1 \times 4 \times 4$ input image X and a $1 \times 2 \times 2$ weight tensor W. For the sake of observation, each element of image is marked with different color and number.

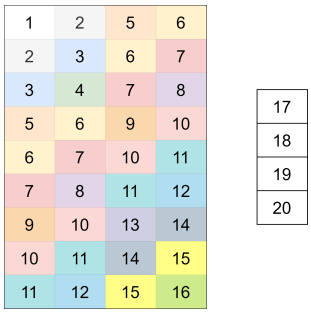


Figure 2: Rearranged image and weight

After rearrangement, we get matrix $(X', W') = Unroll(X, W)$ as shown in Figure 2. When it comes to two-party secure computation, $\mathcal{P}_0$ and $\mathcal{P}_1$ first rearrange the shares of input image $\langle X \rangle$, weight tensor $\langle W \rangle$ to get $\langle X' \rangle$ and $\langle W' \rangle$. Then, using `multiplication` protocol, two servers calculate $\langle Y \rangle = \langle X' \rangle \otimes \langle W' \rangle$ and $\langle Y \rangle$ is the output of the convolution. This is the intuition idea but not the best.

We propose an optimization of 2PC-convolution, that we reduce the communication consumption by not transmitting the duplicated elements. As shown in Figure 2, elements that are not at the edge of the matrix repeated four times and some other elements were also duplicated. If the size of the input image gets larger, elements which repeating four times will be more.

In the original multiplication protocol, to multiply $\langle X' \rangle$ of size 9×4 and $\langle W' \rangle$ of size 4×1, each server totally needs to send $9 \times 4 + 4 \times 1 = 40$ elements. However, we actually do not need so many elements to blind $\langle X' \rangle$ and $\langle W' \rangle$ since there exist lots of duplicated elements in them. We can use the same random number to blind the duplicated element so that

We design a more efficient method to address this problem. Instead of generating matrix multiplication triple directly, TTP randomly generates (A, B) of the same size of the original feature X and weight W (4×4 and 2×2). Then, TTP evaluates $(A', B') = Unroll(A, B)$ and $C' = A' \cdot B'$. Finally, TTP sends shares of (A, B, C') to servers.

In allusion to 2PC-convolution, we propose a modified matrix `multiplication` protocol:

Protocol procedure:

- **1.** $\mathcal{P}_i$ gets multiplication triple $(\langle A \rangle^i, \langle B \rangle^i, \langle C' \rangle^i)$ from trusted third party.
- **2.** $\mathcal{P}_i$ sets $\langle E \rangle^i = \langle X \rangle^i - \langle A \rangle^i$ and $\langle F \rangle^i = \langle W \rangle^i - \langle B \rangle^i$ and transmits $(\langle E \rangle^i, \langle F \rangle^i)$ to $\mathcal{P}_{1-i}$.
- **3.** $\mathcal{P}_i$ reconstruct $E = \langle E \rangle^0 + \langle E \rangle^1$ and $F = \langle F \rangle^0 + \langle F \rangle^1$ and evaluates $(E', F') = Unroll(E, F)$ and $(A', B') = Unroll(A, B)$.
- **4.** $\mathcal{P}_i$ sets $\langle Z \rangle^i = i \cdot E' \cdot F' + \langle A' \rangle^i \cdot F' + E' \cdot \langle B' \rangle^i + \langle C' \rangle^i$.

Essentially, By this way, communication overhead can be observably reduced to almost $\frac{1}{width^2}$ of the original, where $width$ is the width of filter.

Pooling layer There are two types of pooling layer: average pooling and maximum pooling, which output the average or the maximum of $n \times n$ size submatrices respectively to construct the input image of next layer. We take 2×2 window as an example to show how $\mathcal{P}_0$ and $\mathcal{P}_1$ compute the output of a submatrix. For each 2×2 submatrix X, average pooling outputs $y_{avg} = \frac{X_{00}+X_{01}+X_{10}+X_{11}}{4}$ and max pooling outputs $y_{max} = max(X_{00}, X_{01}, X_{10}, X_{11})$

$$\langle X \rangle^0 = \begin{bmatrix} \langle X_{00} \rangle^0 & \langle X_{01} \rangle^0 \\ \langle X_{10} \rangle^0 & \langle X_{11} \rangle^0 \end{bmatrix} \langle X \rangle^1 = \begin{bmatrix} \langle X_{00} \rangle^1 & \langle X_{01} \rangle^1 \\ \langle X_{10} \rangle^1 & \langle X_{11} \rangle^1 \end{bmatrix}$$

2PC-average pooling is easy to implement by locally division. To get the average of elements in submatrix X, $\mathcal{P}_0$ sets $\langle y_{avg} \rangle^0 = \frac{\langle X_{00} \rangle^0 + \langle X_{01} \rangle^0 + \langle X_{10} \rangle^0 + \langle X_{11} \rangle^0}{4}$ and $\mathcal{P}_1$ sets $\langle y_{avg} \rangle^1 = \frac{\langle X_{00} \rangle^1 + \langle X_{01} \rangle^1 + \langle X_{10} \rangle^1 + \langle X_{11} \rangle^1}{4}$. This process can be done without interaction.

2PC-maximum pooling layer can be computed via `comparison` protocol for $y_{max} = max(X_{00}, X_{01}, X_{10}, X_{11})$. The approach to compute this has been briefly discussed before and now we give it in detail. Consider that

$$\begin{aligned} & max(X_{00}, X_{01}, X_{10}, X_{11}) \\ = & max(X_{00}, max(X_{01}, X_{10}, X_{11})) \\ = & max(X_{00}, max(X_{01}, max(X_{10}, X_{11}))) \end{aligned}$$

So the maximum of each submatrix can be obtained by running maximum protocol for three times. After repeating this process to each submatrix, servers can get the output matrix of the pooling layer.

Activation function Here we explore one activation functions: $ReLU()$, a typical continuous piecewise function.

$$ReLU(x) = \begin{cases} 0 & \text{if } x \leqslant 0 \\ x & \text{if } x > 0 \end{cases}$$

$ReLU$ function has three dividing points: $dp_0 = -\infty, dp_1 = 0, dp_n = +\infty$ and two sub-functions: $f_1(x) = 0, f_2(x) = x$. With $v_j = (1 - sign(dp_{j-1} - x) \cdot sign(dp_j - x))/2$, this function can be represented as

$$\begin{aligned} ReLU(x) &= \sum_{j=1}^{2} v_j \cdot f_j(x) = v_1 \cdot f_1(x) + v_2 \cdot f_2(x) \\ &= v_1 \cdot 0 + v_2 \cdot x \\ &= v_2 \cdot x \\ &= \frac{(1 - sign(dp_1 - x) \cdot sign(dp_2 - x)) \cdot x}{2} \\ &= \frac{(1 - sign(0 - x) \cdot sign(+\infty - x)) \cdot x}{2} \\ &= \frac{(1 + sign(x)) \cdot x}{2} \end{aligned}$$

Servers can follow `multiplication` protocol and `comparison` protocol to compute $\langle y \rangle = \langle ReLU(\langle x \rangle) \rangle = \frac{(1+\langle sign(\langle x \rangle)) \otimes \langle x \rangle}{2}$.

Theoretical analysis

The biggest advantage of 2PC-NNP is that we design a novel `comparison` protocol which has a huge improvement over the previous work. We list the state-of-art 2PC-`comparison` protocol that are now used in practice and compare them to our work.

	computation(asymmetric)	communication(bits)	rounds
ABY	$18l$	$7l \cdot \kappa + (l^2 + l)/2$	4
Sharemind	0	$3l$	l
2PC-NNP	0	3	3

Table 2: Experiment II-Layers consumption

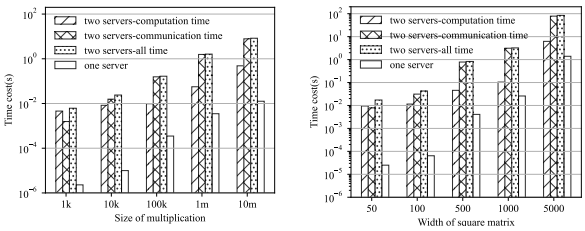
Experiment

To evaluate the performance of our work, we conduct simulation experiment on a desktop (CPU: AMD Ryzen 7 5800H with 3.20GHz). We use two independent folders placed on different physical hard drives to simulate two servers for measuring the time cost of computation and we calculate the communication consumption by counting all of floating point numbers needed to be passed. We use Python 3.9.7 to implement our system and used some functions in the SPDZ library.All operations involved in the experiments are floating point operations. Files are written in form of npy format supported by Python, where a floating point number occupies about 0.0078Kb. For accurate and stable measurement, file transfer rate between servers is fixed at 10M/s. Each experiment was repeated 10 times.

The experiment is divided into three parts: 1.basic operations 2.neural network layers 3.neural network

Operations

In Experiment I, time consumption of basic operations including multiplication, comparison and matrix multiplication are measured. In addition, we give the time consumption of one server operating directly as a reference. The data obtained is shown in Fig.3 and Fig.4.



(a) multiplication (b) matrix multiplication

Figure 3: Experiment I-Overhead of 2PC-multiplication and matrix multiplication

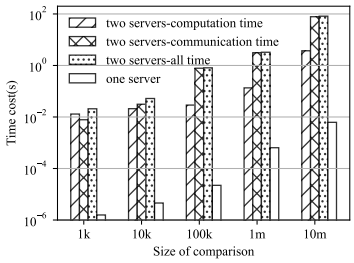


Figure 4: Experiment I-Overhead of 2PC-comparison

The time cost of secure two-party calculation is divided into two parts: local calculation and online communication. the time consumption of online communication increases linearly with the increase of the number of elements, while local computation time is more complicated. Due to the large number of writing and reading operations involved, there will be a lot of fixed time consumption for calling system file functions.

The three operations show the same trend that When the number of elements achieves 10k and more, the communication time becomes longer than the calculation time which accounts for a substantial part of total time when the number grows up to $100k$ and more. Take matrix multiplication as an example, as the width of the matrix increases more, the ratio of the time consumed by one server operation to the computational time cost of two servers is stable at about four times and total time is mainly spent on communication. For instance, when the width is 1000, the communication costs 3.12s, accounting for 96% of total time.

Neural network layers

In Experiment II, overhead of neural network layers including convolution layer, pooling layer and ReLU function are measured.

For different neural network layers, we select several representative shapes of features and filters to test the time cost. The result is shown in 3. In general, the overhead of neural network layers has been reduced to an acceptable range and the communication cost still account for the vast majority.

Convolution Layer					
feature size	size of filter	cost time(s)			communication size(Mb)
		computation	communication	all	
(64,56,56)	(64,64,3,3)	0.64	0.18	0.82	0.28
(256,28,28)	(512,256,1,1)	0.48	0.26	0.74	0.16
(256,14,14)	(256,256,3,3)	0.20	0.50	0.70	0.096
(1024,7,7)	(2048,1024,1,1)	0.21	1.68	1.89	0.086
Pooling Layer					
feature size		cost time(s)			communication size(Mb)
		computation	communication	all	
(256,14,14)		0.088	0.26	0.35	2.6
(256,28,28)		0.107	1.05	1.16	10.5
(256,56,56)		0.240	4.23	4.47	42.3
(256,112,112)		0.746	16.93	17.68	169.3
ReLU					
feature size		cost time(s)			communication size(Mb)
		computation	communication	all	
(256,14,14)		0.025	0.27	0.29	2.7
(256,28,28)		0.054	1.09	1.15	10.9
(256,56,56)		0.160	4.39	4.55	43.9
(256,112,112)		0.631	17.56	18.19	175.6

Table 3: Experiment II-Layers consumption

Neural Networks

To demonstrate the performance of our method in practice, we conduct time-consuming tests on a small network Lenet-5 and a large network Resnet50. The MNIST dataset, consisting of about 70000 images of size $1 \times 28 \times 28$ with 10 classes for handwriting recognition is used for training neural networks. Since the size of input image of Resnet50 is $3 \times 224 \times 224$, we resize each image of MNIST dataset to $3 \times 224 \times 224$ for training Resnet50 model. Notably, we remove the batch normalization layers of Resnet50 because 2PC-NNP cannot implement the division of which the divisor is a secret variable.

model	time(s)			communication size(Mb)
	computation	communication	all	
Lenet-5	0.375	0.32	0.69	3.2
Resnet50	23.85	100.13	123.98	1001.3

Table 4: Experiment III-Neural networks consumption

As shown in Table 4, 2PC-NNP shows an excellent performance both in medium and large neural networks. To evaluate the Resnet50, it costs 23.85s for computation and 100.13s for communication.

Conclusion and Future Work

In this work, we presented 2PC-NNP, a two-party computation framework for neural network predictions. This framework based on a novel 2PC-comparison protocol which does not involve asymmetric encryption operations. The 2PC-comparison protocol enables us to build a efficient framework neural network predictions. Furthermore, we design methods to evaluate each layer in neural network and implement an experiment on Resnet50. This framework has great potential since we only consider the serial implementation of protocols. However, some computation can be excuted in parallel to reduce the rounds of interaction. This task can be considered in future work.

Two-party secure computation protocol for operations

`Multiplication`:

Target: compute $\langle z \rangle = \langle x \rangle \otimes \langle y \rangle$

Input:

$\mathcal{P}_0$	$\mathcal{P}_1$
$\{\langle x\rangle^0, \langle y\rangle^0)$, mul-triple:$(\langle a\rangle^0, \langle b\rangle^0, \langle c\rangle^0)\}$	$\{\langle x\rangle^1, \langle y\rangle^1)$, mul-triple:$(\langle a\rangle^1, \langle b\rangle^1, \langle c\rangle^1)\}$

Procedure:

$\mathcal{P}_0$		$\mathcal{P}_1$
$\langle e\rangle^0 \leftarrow \langle x\rangle^0 - \langle a\rangle^0$ $\langle f\rangle^0 \leftarrow \langle y\rangle^0 - \langle b\rangle^0$		$\langle e\rangle^1 \leftarrow \langle x\rangle^1 - \langle a\rangle^1$ $\langle f\rangle^1 \leftarrow \langle y\rangle^1 - \langle b\rangle^1$
	$\xrightarrow{(\langle e\rangle^0, \langle f\rangle^0)}$	
	$\xleftarrow{(\langle e\rangle^1, \langle f\rangle^1)}$	
$e \leftarrow \langle e\rangle^0 + \langle e\rangle^1$ $f \leftarrow \langle f\rangle^0 + \langle f\rangle^1$ $\langle z\rangle^0 \leftarrow f \cdot \langle a\rangle^0$ $+e \cdot \langle b\rangle^0 + \langle c\rangle^0$		$e \leftarrow \langle e\rangle^0 + \langle e\rangle^1$ $f \leftarrow \langle f\rangle^0 + \langle f\rangle^1$ $\langle z\rangle^1 \leftarrow e \cdot f + f \cdot \langle a\rangle^1$ $+e \cdot \langle b\rangle^1 + \langle c\rangle^1$

`Comparison`:

Computation for $sign(x)$:

Target: compute $\langle s \rangle = \langle sign(\langle x \rangle)\rangle$

Input:

$\mathcal{P}_0$	$\mathcal{P}_1$
$\{\langle x\rangle^0$, comp-triple:$(\langle u\rangle^0, [\![u]\!]^0)\}$	$\{\langle x\rangle^1$, comp-triple:$(\langle u\rangle^1, [\![u]\!]^1)\}$

Procedure:

$\mathcal{P}_0$		$\mathcal{P}_1$
	run `multiplication` protocol to compute $\langle z \rangle = \langle u \rangle \otimes \langle x \rangle$	
gets $\langle z\rangle^0$		gets $\langle z\rangle^1$
	$\xleftarrow{\langle z\rangle^1}$	
$z \leftarrow \langle z\rangle^0 + \langle z\rangle^1$ $[\![s]\!]^0 \leftarrow sign(\frac{1}{[\![u]\!]^0})$		$[\![s]\!]^1 \leftarrow sign(\frac{z}{[\![u]\!]^1})$
	run `multiplication` protocol to compute $\langle s \rangle = [\![s]\!]^0 \otimes [\![s]\!]^1$	
gets $\langle s\rangle^0$		gets $\langle s\rangle^1$

Computation for $max(x)$:

Target: compute $\langle maximum \rangle = \langle max(\langle x \rangle, \langle y \rangle)\rangle$

Input:

$\mathcal{P}_0$	$\mathcal{P}_1$
$\{\langle x\rangle^0, \langle y\rangle^0\}$	$\{\langle x\rangle^1, \langle y\rangle^1\}$

Procedure:

$\mathcal{P}_0$		$\mathcal{P}_1$
$\langle \Delta\rangle^0 \leftarrow \langle x\rangle^0 - \langle y\rangle^0$		$\langle \Delta\rangle^1 \leftarrow \langle x\rangle^1 - \langle y\rangle^1$
	run `comparison` protocol to compute $\langle s \rangle = sign(\langle \Delta \rangle)$	
gets $\langle s\rangle^0$		gets $\langle s\rangle^1$
	run `multiplication` protocol to compute $\langle maximum \rangle = \frac{(\langle s\rangle+1)\otimes\langle x\rangle+(-\langle s\rangle+1)\otimes\langle y\rangle}{2}$	
gets $\langle maximum\rangle^0$		gets $\langle maximum\rangle^1$

Computation for piecewise function $F(x)$:
Target: compute $\langle value \rangle = \langle F(\langle x \rangle) \rangle$
Input:

$\mathcal{P}_0$	$\mathcal{P}_1$
$\{\langle x \rangle^0, F(), (dp_0, ..., dp_n),$ $(f_0(), ..., f_n())\}$	$\{\langle x \rangle^1, F(), (dp_0, ..., dp_n),$ $(f_0(), ..., f_n())\}$

Procedure:

$\mathcal{P}_0$	$\mathcal{P}_1$
For each $dp_j, j \in \{1, 2, ..., n-1\}$: run `comparison` protocol to compute $\langle s_j \rangle = \langle sign(dp_j - \langle x \rangle) \rangle$	
gets $\langle s_j \rangle^0$	gets $\langle s_j \rangle^1$
For each $v_j, j \in \{1, 2, ..., n-1\}$: run `multiplication` protocol to compute $\langle v_j \rangle = \frac{1 - \langle s_j \rangle \otimes \langle s_{j-1} \rangle}{2}$	
gets $\langle v_j \rangle^0$	gets $\langle v_j \rangle^1$
run `multiplication` protocol to compute $\langle value \rangle = \langle F(\langle x \rangle) \rangle = \sum_{j=1}^{n} \langle v_j \rangle \otimes \langle f_j(\langle x \rangle) \rangle$	
gets $\langle value \rangle^0$	gets $\langle value \rangle^1$

Appendix

References

Beaver, D. 1991. Efficient Multiparty Protocols Using Circuit Randomization. In Feigenbaum, J., ed., *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, 420–432. Springer.

Chellapilla, K.; Puri, S.; and Simard, P. 2006. High performance convolutional neural networks for document processing. In *Tenth international workshop on frontiers in handwriting recognition*. Suvisoft.

Chen, H.; Kim, M.; Razenshteyn, I. P.; Rotaru, D.; Song, Y.; and Wagh, S. 2020. Maliciously Secure Matrix Multiplication with Applications to Private Deep Learning. In Moriai, S.; and Wang, H., eds., *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part III*, volume 12493 of *Lecture Notes in Computer Science*, 31–59. Springer.

Demmler, D.; Schneider, T.; and Zohner, M. 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.

Gilad-Bachrach, R.; Dowlin, N.; Laine, K.; Lauter, K. E.; Naehrig, M.; and Wernsing, J. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In Balcan, M.; and Weinberger, K. Q., eds., *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, 201–210. JMLR.org.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 770–778. IEEE Computer Society.

Hunt, T.; Song, C.; Shokri, R.; Shmatikov, V.; and Witchel, E. 2018. Chiron: Privacy-preserving Machine Learning as a Service. *CoRR*, abs/1803.05961.

Liu, J.; Juuti, M.; Lu, Y.; and Asokan, N. 2017. Oblivious Neural Network Predictions via MiniONN Transformations. In Thuraisingham, B.; Evans, D.; Malkin, T.; and Xu, D., eds., *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, 619–631. ACM.

Mohassel, P.; and Rindal, P. 2018. ABY^3: A Mixed Protocol Framework for Machine Learning. In Lie, D.; Mannan, M.; Backes, M.; and Wang, X., eds., *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, 35–52. ACM.

Sharemind. 2022. SharemindMPC. https://sharemind.cyber.ee/.