SecureNN: 3-Party Secure Computation for Neural Network Training

Sameer Wagh, Divya Gupta, and Nishanth Chandran (PPET'19)

ABSTRACT

本文构建了新的基于三方安全计算协议NN计算模块，包括矩阵乘法、卷积、ReLU、Maxpool、正则化等。本文的工作在以下三个方面优于其它的隐私保护NN方法：

1. Scalability：是第一个用于安全计算卷积神经网络训练的研究
2. Performance：在安全预测中效率优于（SecureML、MiniONN、Chameleon、Gazelle等）$6\times - 113\times$。总的时间快$2-4\times$。在安全训练任务中，分别比2-Party、3-Party的SecureML快$79\times, 3\times$。
3. Security：前人的相关工作只提供了半诚实的安全性保证，本文是第一个在NN的训练和预测任务中考虑恶意敌手存在的安全的工作

TECHNICAL OVERVIEW

神经网络算法中包含着线性运算（如，矩阵乘法、卷积...）和布尔运算（如，ReLU、Maxpool和它的导数...），为了使这些计算兼容因此会引入一些转换协议，但基于GC的布尔运算其通信开销特别大，是安全参数长度的许多倍。本文整个方案只使用了算术加法秘密分享，没有涉及到布尔电路计算，因此效率得到了有效的提升。

Protocols Structure

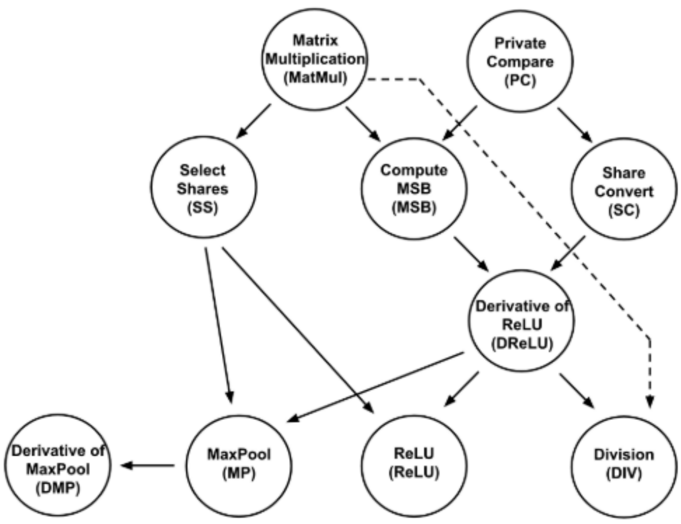

Fig. 3. Functionality dependence of protocols in SecureNN

Supporting Protocols

Matrix Multiplication

乘法采用Beaver三元组的方式，由P2辅助生成三元组，以秘密分享的形式分发给P_0和P_1。（为减少通信，P_2采用分发随机数种子的方式，而不是直接传随机数，最后计算好$[c_1]_i$后给P_1）

Algorithm 1 Mat. Mul. $\Pi_{\mathsf{MatMul}}(\{P_0, P_1\}, P_2)$:

Input: P_0 & P_1 hold $(\langle X\rangle_0^L, \langle Y\rangle_0^L)$ & $(\langle X\rangle_1^L, \langle Y\rangle_1^L)$ resp.

Output: P_0 gets $\langle X \cdot Y\rangle_0^L$ and P_1 gets $\langle X \cdot Y\rangle_1^L$.

Common Randomness: P_0 and P_1 hold shares of zero matrices over $\mathbb{Z}_L^{m \times v}$ resp.; i.e., P_0 holds $\langle 0^{m \times v}\rangle_0^L = U_0$ & P_1 holds $\langle 0^{m \times v}\rangle_1^L = U_1$

1. P_2 picks random matrices $A \xleftarrow{\$} \mathbb{Z}_L^{m \times n}$ and $B \xleftarrow{\$} \mathbb{Z}_L^{n \times v}$ and generates for $j \in \{0,1\}$, $\langle A\rangle_j^L, \langle B\rangle_j^L, \langle C\rangle_j^L$ and sends to P_j, where $C = A \cdot B$.
2. For $j \in \{0,1\}$, P_j computes $\langle E\rangle_j^L = \langle X\rangle_j^L - \langle A\rangle_j^L$ and $\langle F\rangle_j^L = \langle Y\rangle_j^L - \langle B\rangle_j^L$.
3. P_0 & P_1 reconstruct E & F by exchanging shares.
4. For $j \in \{0,1\}$, P_j outputs $-jE \cdot F + \langle X\rangle_j^L \cdot F + E \cdot \langle Y\rangle_j^L + \langle C\rangle_j^L + U_j$.

Select Share

Select Share协议的基本要求是，用一个比特a来选择分享份额，即假设P_0, P_1持有x, y的秘密分享份额，当$a = 0$时选择x，反之y。

Algorithm 2 SelectShare $\Pi_{\mathsf{SS}}(\{P_0, P_1\}, P_2)$:

Input: P_0, P_1 hold $(\langle \alpha\rangle_0^L, \langle x\rangle_0^L, \langle y\rangle_0^L)$ and $(\langle \alpha\rangle_1^L, \langle x\rangle_1^L, \langle y\rangle_1^L)$, resp.

Output: P_0, P_1 get $\langle z\rangle_0^L$ and $\langle z\rangle_1^L$, resp., where $z = (1-\alpha)x + \alpha y$.

Common Randomness: P_0 and P_1 hold shares of 0 over $\mathbb{Z}_L$ denoted by u_0 and u_1.

1. For $j \in \{0,1\}$, P_j compute $\langle w\rangle_j^L = \langle y\rangle_j^L - \langle x\rangle_j^L$
2. P_0, P_1, P_2 invoke $\Pi_{\mathsf{MatMul}}(\{P_0, P_1\}, P_2)$ with $P_j, j \in \{0,1\}$ having input $(\langle \alpha\rangle_j^L, \langle w\rangle_j^L)$ and P_0, P_1 learn $\langle c\rangle_0^L$ and $\langle c\rangle_1^L$, resp.
3. For $j \in \{0,1\}$, P_j outputs $\langle z\rangle_j^L = \langle x\rangle_j^L + \langle c\rangle_j^L + u_j$.

主要思路是：x, y中的选择可以表示为$(1-a)\cdot x + a \cdot y = x + a \cdot (y - x)$，那么协议可以计算$\langle z\rangle_j^L = \langle x\rangle_j^L + \langle a\rangle_j^L \cdot (\langle y\rangle_j^L - \langle x\rangle_j^L)$，先本地计算减法然后调用一 次乘法协议即可，最后增加盲化值u_j。

Private Compare

整个方案的核心在非线性算子的计算。以激活函数为突破口。$\mathrm{ReLU}(x)$的计算关键在于判断x的正负，也就是看$\mathrm{MSB}(x)$。而$\mathrm{MSB}(x)$需要做比特提取（bit extraction），它没有求$\mathrm{LSB}(x)$效率高，因此需要先将问题做一个转化。

原始问题是$\mathrm{MSB}(x)$一般的做法是Boolean加法然后做一次比较。现在希望不在Boolean 电路上做加法和比较。其方法是先观察到在奇数环里有$\mathrm{MSB}(a) \leftrightarrow \mathrm{LSB}(2a)$，然后最低有效位根据环的定义也需要看原数有没有wrap around，即$\langle y\rangle_0^{L-1} + \langle y\rangle_1^{L-1} \geq L-1$。这样问题又回去了。为避免做bit extraction，通过设定$r = y + x \bmod L-1$，把原问题转化为$y[0] = x[0] \oplus r[0] \oplus (x > r)$，其核心在$x > r$的判定，现在可以在算术电路上做比较了，相比于$y > L-1$，$r$可以公开而不会泄露$y$的信息（因为它$r$是由$x$给$y$加掩码计算得到的，如果不这样做的话，直接利用算术电路进行比较会泄露y的信息）

- 对于奇数环$\mathbb{Z}_{L-1}$里，有$\mathrm{MSB}(a) \leftrightarrow \mathrm{LSB}(2a)$

- 在模n的环里，$\mathrm{MSB}(a)=1 \leftrightarrow a>n/2 \leftrightarrow n > 2a-n>0$。因为$n$是奇数，那么$2a-n$是奇数。因此$\mathrm{LSB}(2a-n)=1$。又$2a-n=2a \bmod n$，因此 $\mathrm{LSB}(2a)=1$。同理，当$\mathrm{MSB}(a)=0$，有$\mathrm{LSB}(2a)=0$。
- 假设现在的计算都已经在奇数环$\mathbb{Z}_{L-1}$内，计算得到$y=2a \bmod L-1$。那问题的关键在求$y[0]$。

$$y[0]=\langle y\rangle_0^{L-1}[0] \oplus \langle y\rangle_1^{L-1}[0] \oplus \underbrace{(\langle y\rangle_0^{L-1}+\langle y\rangle_1^{L-1} \geq L-1) \cdot 1}_{\text{根据环的定义，如果wrap around后减}L-1}$$

  - 现在需要计算$y \geq L-1$，为避免回到基于GSW或者GC的思路，因此继续设计
  - 对于$r=y+x \bmod L-1$，如果$x>r$那么必然有$r=y+x-(L-1)$，说明wrap around了。因此 $y[0]=x[0] \oplus r[0] \oplus (x>r)$。 我们可以利用$P_2$生成$x$并以秘密分享的形式给$P_0, P_1$，把$r$（由$P_0, P_1$交换信息算出来的）公开，那么计算$x>r$将容易许多。——**这里也可以由$P_0, P_1$生成各自的$\langle x\rangle_j^L$，再通过交换信息计算出$r$（好像可以提高效率，也没有泄露信息）。**

因此，首先给出计算$x>r$的协议

Algorithm 3 PrivateCompare $\Pi_{\mathsf{PC}}(\{P_0, P_1\}, P_2)$:

Input: P_0, P_1 hold $\{\langle x[i]\rangle_0^p\}_{i\in[\ell]}$ and $\{\langle x[i]\rangle_1^p\}_{i\in[\ell]}$, respectively, a common input r (an ℓ bit integer) and a common random bit β.

Output: P_2 gets a bit $\beta \oplus (x>r)$.

Common Randomness: P_0, P_1 hold ℓ common random values $s_i \in \mathbb{Z}_p^*$ for all $i \in [\ell]$ and a random permutation π for ℓ elements. P_0 and P_1 additionally hold ℓ common random values $u_i \in \mathbb{Z}_p^*$.

1: Let $t=r+1 \mod 2^\ell$.
2: For each $j \in \{0,1\}$, P_j executes Steps 3–14:
3: **for** $i=\{\ell, \ell-1, \ldots, 1\}$ **do**
4:     **if** $\beta=0$ **then**
5:         $\langle w_i\rangle_j^p=\langle x[i]\rangle_j^p+jr[i]-2r[i]\langle x[i]\rangle_j^p$
6:         $\langle c_i\rangle_j^p=jr[i]-\langle x[i]\rangle_j^p+j+\sum_{k=i+1}^{\ell}\langle w_k\rangle_j^p$
7:     **else if** $\beta=1$ AND $r \neq 2^\ell-1$ **then**
8:         $\langle w_i\rangle_j^p=\langle x[i]\rangle_j^p+jt[i]-2t[i]\langle x[i]\rangle_j^p$
9:         $\langle c_i\rangle_j^p=-jt[i]+\langle x[i]\rangle_j^p+j+\sum_{k=i+1}^{\ell}\langle w_k\rangle_j^p$
10:     **else**
11:         If $i \neq 1$, $\langle c_i\rangle_j^p=(1-j)(u_i+1)-ju_i$, else $\langle c_i\rangle_j^p=(-1)^j \cdot u_i$.
12:     **end if**
13: **end for**
14: Send $\{\langle d_i\rangle_j^p\}_i=\pi\left(\left\{s_i\langle c_i\rangle_j^p\right\}_i\right)$ to P_2
15: For all $i \in [\ell]$, P_2 computes $d_i=\mathsf{Reconst}^p(\langle d_i\rangle_0^p, \langle d_i\rangle_1^p)$ and sets $\beta'=1$ iff $\exists i \in [\ell]$ such that $d_i=0$.
16: P_2 outputs β'.

这个方法适用于三方的情况，其主要的思路如下：

1. P_0, P_1分别持有$\{\langle x\rangle_0^p\}_{i\in[l]}, \{\langle x\rangle_1^p\}_{i\in[l]}$。以及一个公共的$l$-bit的$r$和1-bit的$\beta$。$P_2$计算得到$\beta^{'}=\beta \oplus (x>r)$（$\beta=0$计算的是$\beta^{'}=(x>r)$，反之$\beta^{'}=(x \leq r)$）。
2. 对r可以分为$r=2^l-1$和$r \neq 2^l-1$两种情况讨论，因为当$r=2^l-1$时，总有$x \leq r$成立，进而能简化计算。看代码11行表示的是$\beta=1$ AND $r=2^l-1$ 的情况，P_2中$\mathrm{Reconst}(\langle \mathrm{d_i}\rangle_0^{\mathrm{p}}, \langle \mathrm{d_i}\rangle_1^{\mathrm{p}})$值为$111\ldots0$，故$\beta^{'}=1$，它表示$1=1 \oplus (x>r) \rightarrow (x>r)=0 \rightarrow (x \leq r)$。

3. 对于$\beta=0$的情况，$\beta^{'}=1$ 当且仅当$x>r$成立。换句话说，最高非相等的位上有$x[i]\neq r[i]$且$x[i]=1$，代码5-6行表述的就是这个逻辑：
   - $w_i=x[i]\oplus r[i]=x[i]+r[i]-2x[i]r[i]$，$c_i=r[i]-x[x]+1+\sum_{k=i+1}^{l}w_k$
     - 当$w_{i+1}=0,\ldots,w_l=0$表示x,r的前$l-i$比特相等，如果$x[i]=r[i]$可以推出$w_i=0\rightarrow c_i=1$，此时$c_i=1,\ldots,c_l=1$
     - 当$w_{i+1}=0,\ldots,w_l=0$即前$l-i$比特相等，如果$x[i]\neq r[i]$此时$w_i=1$，对于c_i分两种情况，1）$x[i]=1$，可以推出$c_i=0$；2）$x[i]=0$，可以推出$c_i=2$
     - 那么对于后面的$l-i$比特的比较而言，均有$c_i\geq 1$

   最后可以通过查询c_i是否等于0来判定是否有$x>r$。不过出于安全性考虑，P_0,P_1将$\langle c_i\rangle_j^p$传给P_2的时候需要通过非零的s_i和一个随机排列π处理后再发送。

4. 对于$\beta=1$的情况逻辑同3

Share Convert

能够方便的在奇数环里进行比较运算了，现在就要设计一个转换协议$\mathbb{Z}_L\rightarrow\mathbb{Z}_{L-1}$，目标是$\mathrm{Reconst}^{L-1}(\langle y\rangle_0^{L-1}+\langle y\rangle_1^{L-1})=\mathrm{Reconst}^{L}(\langle a\rangle_0^{L}+\langle a\rangle_1^{L})=a$。想法是：

- 当$a<L\rightarrow\ a \bmod L=a \bmod L-1$
- 当$a\geq L\rightarrow\ a \bmod L=a \bmod (L-1+1)\rightarrow= a-(L-1)-1$
- 统一形式，即$\theta=(a>L): a \bmod L-1\ =a-L-1-\underbrace{(a>L)}_{\theta}$

令$\theta=\mathrm{wrap}(\langle a\rangle_0^L,\langle a\rangle_1^L,L)$。现在需要利用前面的比较方法，来进行设计：

1. $r=\langle r\rangle_0^L+\langle r\rangle_1^L-\alpha L$
2. $\langle\tilde{a}\rangle_j^L=\langle a\rangle_j^L+\langle r\rangle_j^L-\beta_j L$
3. $x=\langle\tilde{a}\rangle_0^L+\langle\tilde{a}\rangle_1^L-\delta L$
4. $x=a+r-(1-\eta)L$
5. 令θ满足$a=\langle a\rangle_0^L+\langle a\rangle_1^L-\theta L$

(1)-(2)-(3)+(4)+(5)可以解出来$\theta=\beta_0+\beta_1-\alpha+\delta+\eta-1$

Algorithm 4 ShareConvert $\Pi_{\mathsf{SC}}(\{P_0, P_1\}, P_2)$:

Input: P_0, P_1 hold $\langle a \rangle_0^L$ and $\langle a \rangle_1^L$, respectively such that $\mathsf{Reconst}^L(\langle a \rangle_0^L, \langle a \rangle_1^L) \neq L-1$.

Output: P_0, P_1 get $\langle a \rangle_0^{L-1}$ and $\langle a \rangle_1^{L-1}$.

Common Randomness: P_0, P_1 hold a random bit η'', a random $r \in \mathbb{Z}_L$, shares $\langle r \rangle_0^L, \langle r \rangle_1^L, \alpha = \mathsf{wrap}(\langle r \rangle_0^L, \langle r \rangle_1^L, L)$ and shares of 0 over $\mathbb{Z}_{L-1}$ denoted by u_0 and u_1.

1: For each $j \in \{0,1\}$, P_j executes Steps 2–3
2: $\langle \tilde{a} \rangle_j^L = \langle a \rangle_j^L + \langle r \rangle_j^L$ and $\beta_j = \mathsf{wrap}(\langle a \rangle_j^L, \langle r \rangle_j^L, L)$.
3: Send $\langle \tilde{a} \rangle_j^L$ to P_2.

4: P_2 computes $x = \mathsf{Reconst}^L(\langle \tilde{a} \rangle_0^L, \langle \tilde{a} \rangle_1^L)$ and $\delta = \mathsf{wrap}(\langle \tilde{a} \rangle_0^L, \langle \tilde{a} \rangle_1^L, L)$.
5: P_2 generates shares $\{\langle x[i] \rangle_j^p\}_{i \in [\ell]}$ and $\langle \delta \rangle_j^{L-1}$ for $j \in \{0,1\}$ and sends to P_j.

6: P_0, P_1, P_2 invoke[7] $\Pi_{\mathsf{PC}}(\{P_0, P_1\}, P_2)$ with $P_j, j \in \{0,1\}$ having input $\left(\{\langle x[i] \rangle_j^p\}_{i \in [\ell]}, r-1, \eta''\right)$ and P_2 learns η'.
7: For $j \in \{0,1\}$, P_2 generates $\langle \eta' \rangle_j^{L-1}$ and sends to P_j.

8: For each $j \in \{0,1\}$, P_j executes Steps 9–11
9: $\langle \eta \rangle_j^{L-1} = \langle \eta' \rangle_j^{L-1} + (1-j)\eta'' - 2\eta'' \langle \eta' \rangle_j^{L-1}$
10: $\langle \theta \rangle_j^{L-1} = \beta_j + (1-j) \cdot (-\alpha - 1) + \langle \delta \rangle_j^{L-1} + \langle \eta \rangle_j^{L-1}$
11: Output $\langle y \rangle_j^{L-1} = \langle a \rangle_j^L - \langle \theta \rangle_j^{L-1} + u_j$ (over $L-1$)

Compute MSB

首先在odd环上，$\mathrm{MSB}(a) = \mathrm{LSB}(y)$，$where\ y = 2a$，然后$P_2$生成$x$，$P_0, P_1$计算得到$r = x + y \bmod L - 1$，再通过$x > r$的判定得到$\mathrm{wrap}(y, x, L-1)$，这样就得到了最终结果

Algorithm 5 ComputeMSB $\Pi_{\mathsf{MSB}}(\{P_0, P_1\}, P_2)$:

Input: P_0, P_1 hold $\langle a \rangle_0^{L-1}$ and $\langle a \rangle_1^{L-1}$, respectively.

Output: P_0, P_1 get $\langle \mathsf{MSB}(a) \rangle_0^L$ and $\langle \mathsf{MSB}(a) \rangle_1^L$.

Common Randomness: P_0, P_1 hold a random bit β and random shares of 0 over L, denoted by u_0 and u_1 resp.

1: P_2 picks $x \xleftarrow{\$} \mathbb{Z}_{L-1}$. Next, P_2 generates $\langle x \rangle_j^{L-1}$, $\{\langle x[i] \rangle_j^p\}_i$, $\langle x[0] \rangle_j^L$ for $j \in \{0,1\}$ and sends to P_j.
2: For $j \in \{0,1\}$, P_j computes $\langle y \rangle_j^{L-1} = 2\langle a \rangle_j^{L-1}$ and $\langle r \rangle_j^{L-1} = \langle y \rangle_j^{L-1} + \langle x \rangle_j^{L-1}$.
3: P_0, P_1 reconstruct r by exchanging shares.
4: P_0, P_1, P_2 call $\Pi_{\mathsf{PC}}(\{P_0, P_1\}, P_2)$ with $P_j, j \in \{0,1\}$ having input $\left(\{\langle x[i] \rangle_j^p\}_{i \in [\ell]}, r, \beta\right)$ and P_2 learns β'.
5: P_2 generates $\langle \beta' \rangle_j^L$ and sends to P_j for $j \in \{0,1\}$.
6: For $j \in \{0,1\}$, P_j executes Steps 7–8
7: $\langle \gamma \rangle_j^L = \langle \beta' \rangle_j^L + j\beta - 2\beta\langle \beta' \rangle_j^L$
8: $\langle \delta \rangle_j^L = \langle x[0] \rangle_j^L + jr[0] - 2r[0]\langle x[0] \rangle_j^L$
9: P_0, P_1, P_2 call $\Pi_{\mathsf{MatMul}}(\{P_0, P_1\}, P_2)$ with $P_j, j \in \{0,1\}$ having input $(\langle \gamma \rangle_j^L, \langle \delta \rangle_j^L)$ and P_j learns $\langle \theta \rangle_j^L$.
10: For $j \in \{0,1\}$, P_j outputs $\langle \alpha \rangle_j^L = \langle \gamma \rangle_j^L + \langle \delta \rangle_j^L - 2\langle \theta \rangle_j^L + u_j$.

Overheads of supporting protocols

Protocol	Rounds	Communication
$\mathsf{MatMul}_{m,n,v}$	2	$2(2mn + 2nv + mv)\ell$
$\mathsf{MatMul}_{m,n,v}$ (with PRF)	2	$(2mn + 2nv + mv)\ell$
SelectShare	2	5ℓ
PrivateCompare	1	$2\ell \log p$
ShareConvert	4	$4\ell \log p + 6\ell$
Compute MSB	5	$4\ell \log p + 13\ell$

Table 1. Round & communication complexity of *building blocks*.

Main Protocols

Linear and Convolutional Layer

线性运算实质是一个矩阵乘法计算

Derivative of ReLU

ReLU(x)