ML相关分享

向诗雨　赵焱阳

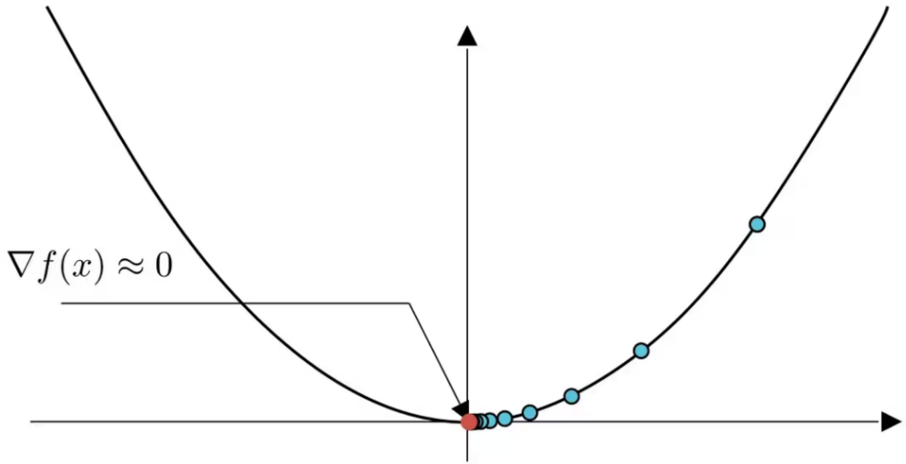
目录

CONTENTS

ML基础知识介绍

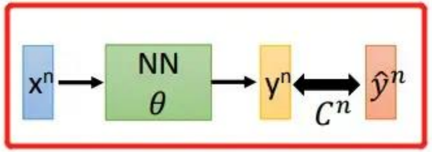
第一单元

梯度下降



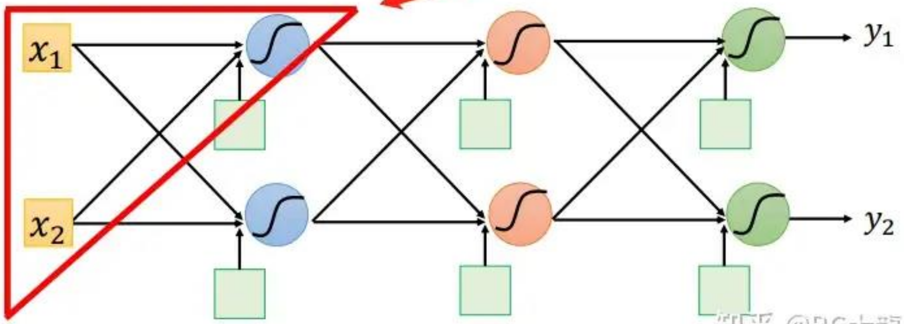
Backpropagation

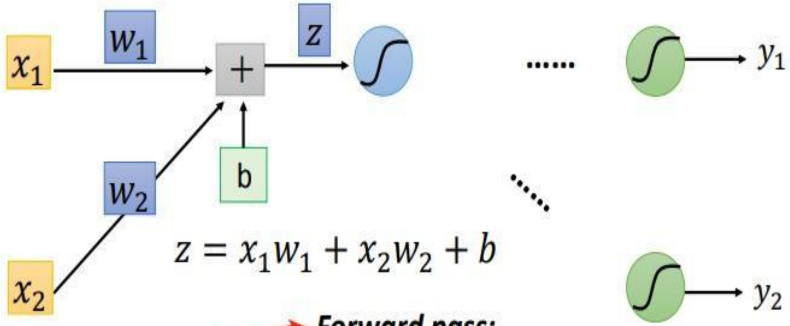
模型



损失函数　　交叉熵

$$\boxed{L(\theta)} = \sum_{n=1}^{N} \boxed{C^n(\theta)} \Rightarrow \frac{\partial L(\theta)}{\partial w} = \sum_{n=1}^{N} \boxed{\frac{\partial C^n(\theta)}{\partial w}}$$





$$z = x_1 w_1 + x_2 w_2 + b$$

$$\frac{\partial C}{\partial w} = ? \quad \boxed{\frac{\partial z}{\partial w}} \boxed{\frac{\partial C}{\partial z}}$$

(Chain rule)

Forward pass:

Compute $\partial z / \partial w$ for all parameters

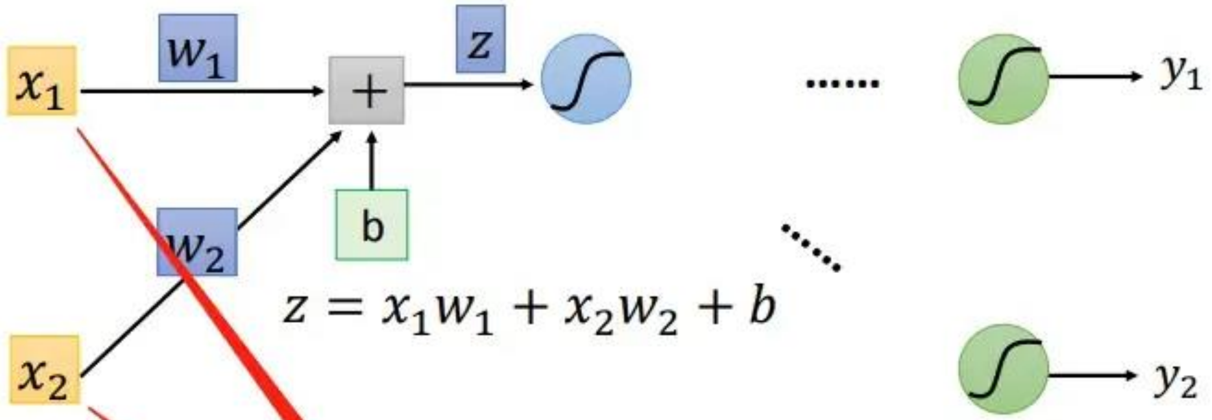
Backward pass:

Compute $\partial C / \partial z$ for all activation function inputs z

知乎 @BG大龍
知乎 @BG大龍

前向传播

Compute $\partial z / \partial w$ for all parameters



$$z = x_1 w_1 + x_2 w_2 + b$$

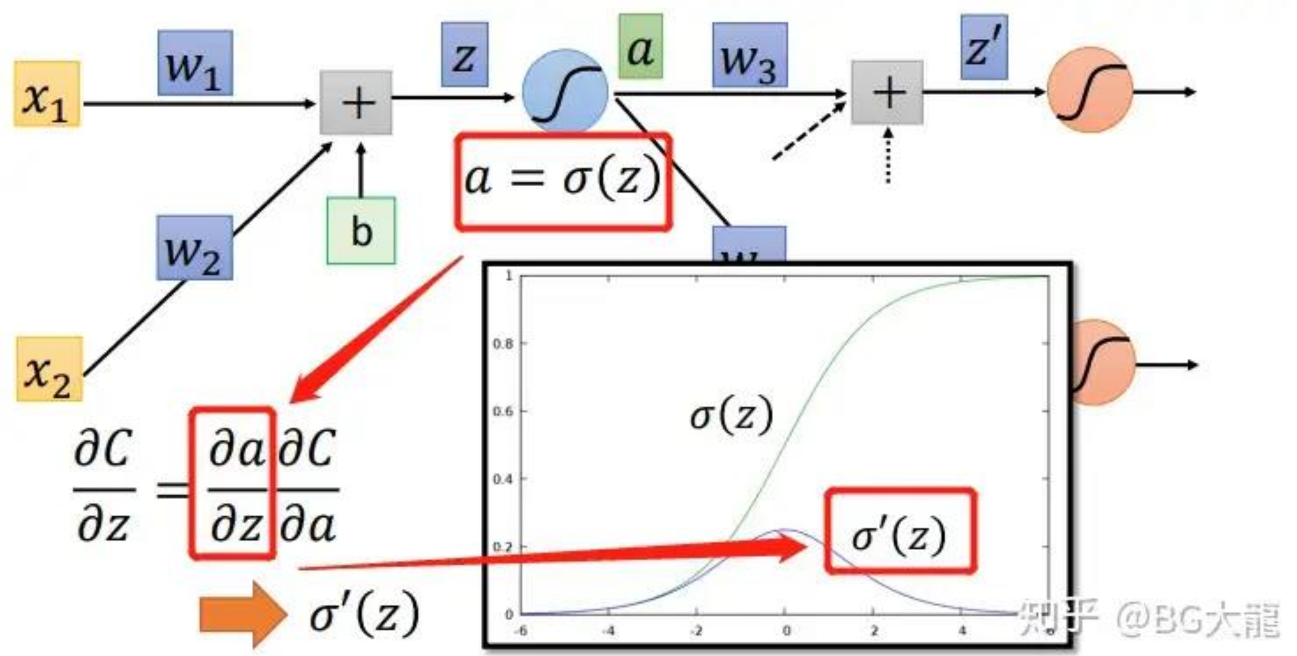
$\partial z / \partial w_1 =?\ x_1$

$\partial z / \partial w_2 =?\ x_2$

The value of the input connected by the weight

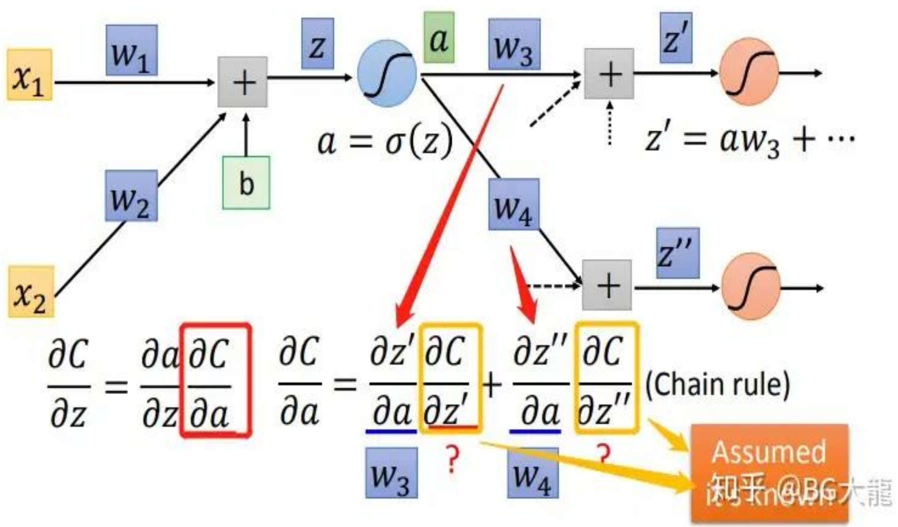
知乎 @BG大龍

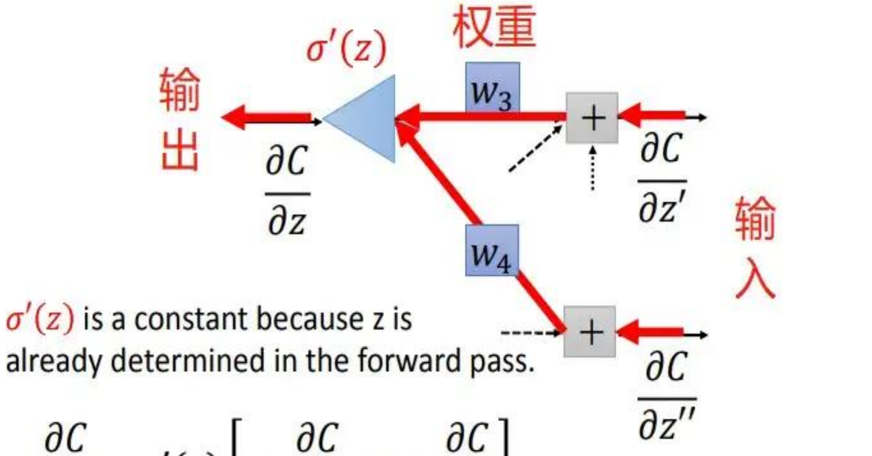
反向传播



x_1
w_1
$+$
z
a
w_3
$+$
z'
$a = \sigma(z)$
b
w_2
x_2
$\frac{\partial C}{\partial z} = \frac{\partial a}{\partial z}\frac{\partial C}{\partial a}$
$\sigma'(z)$
$\sigma(z)$
$\sigma'(z)$
知乎 @BG大龍

反向传播-对 ∂C/∂a 进行处理





$$\frac{\partial C}{\partial z}=\sigma'(z)\left[w_3\frac{\partial C}{\partial z'}+w_4\frac{\partial C}{\partial z''}\right]$$

Linear & Non-Linear Layers

Linear Layers：

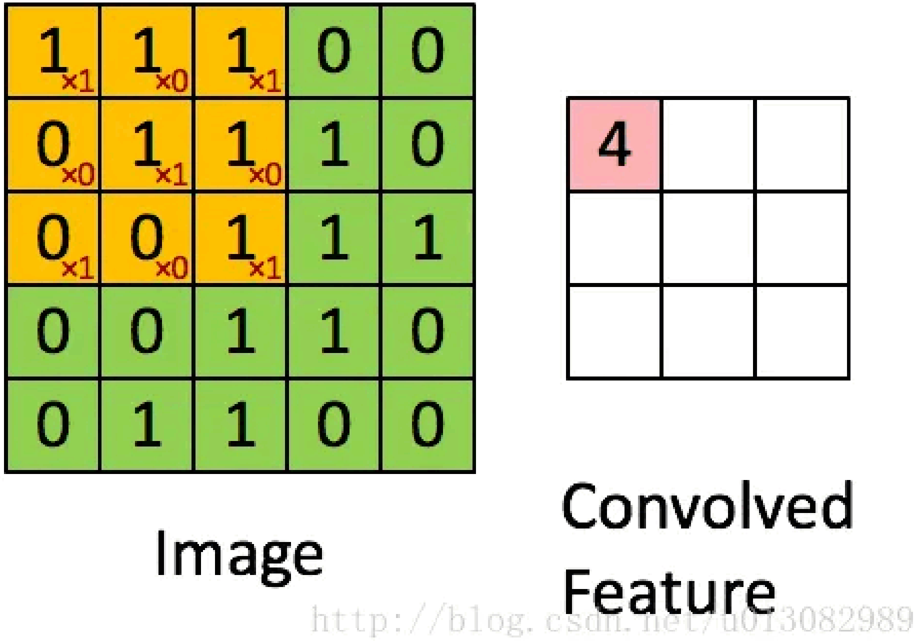
- **dense/fully-connected layers (matrix multiplication)**
- **2D convolution.**

Common Kinds of Non-Linear Layers:

- **Activation layers**
- **Pooling layers**
- **Normalization layers**
- **The output layer computes the inference output based on all its inputs, e.g., softmax or argmax.**

(注：此来自SoK: Cryptographic Neural-Network Computation的分类)

convolution



http://blog.csdn.net/u013082989

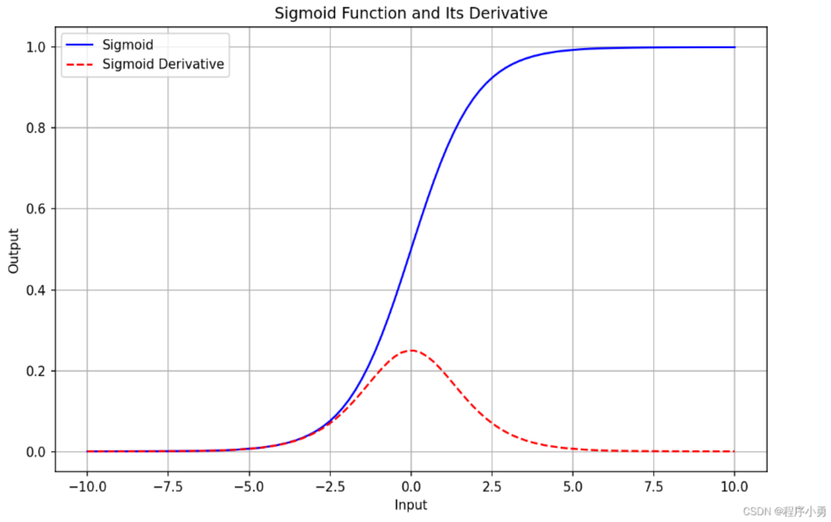
Filter：

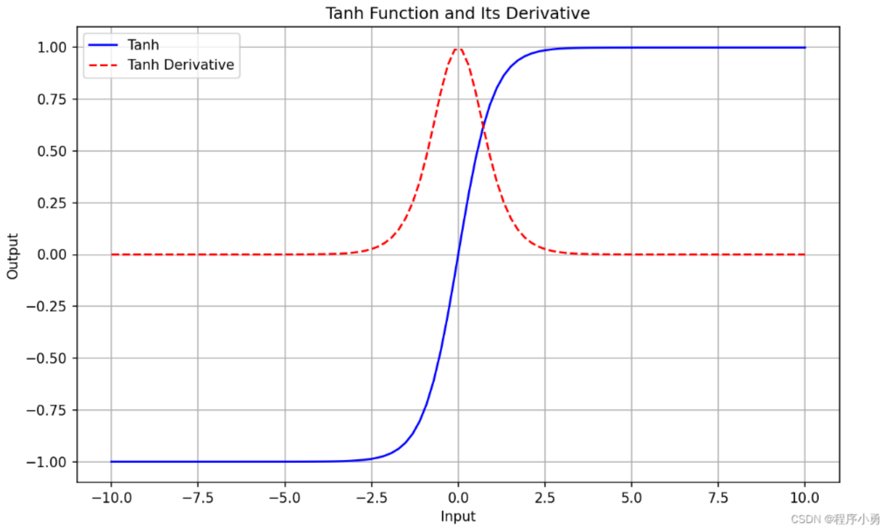
$$
\begin{matrix}
1 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 1
\end{matrix}
$$

Activation layers

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

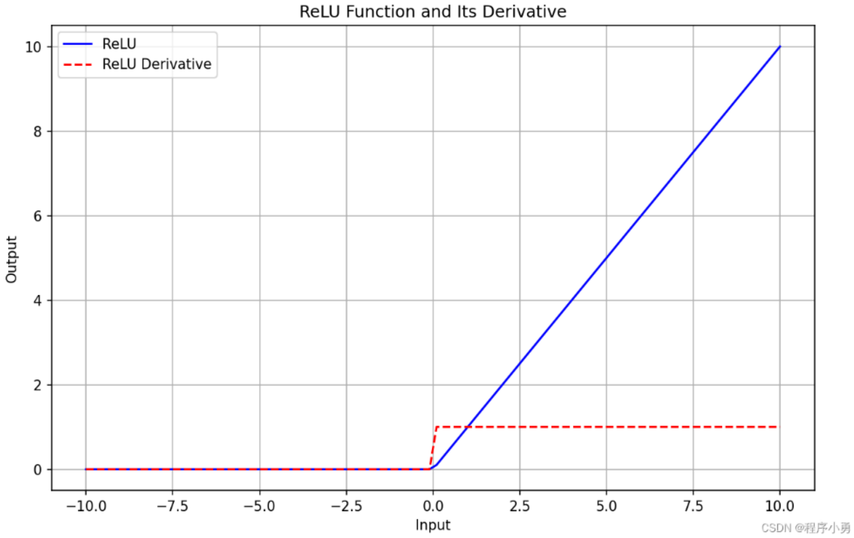


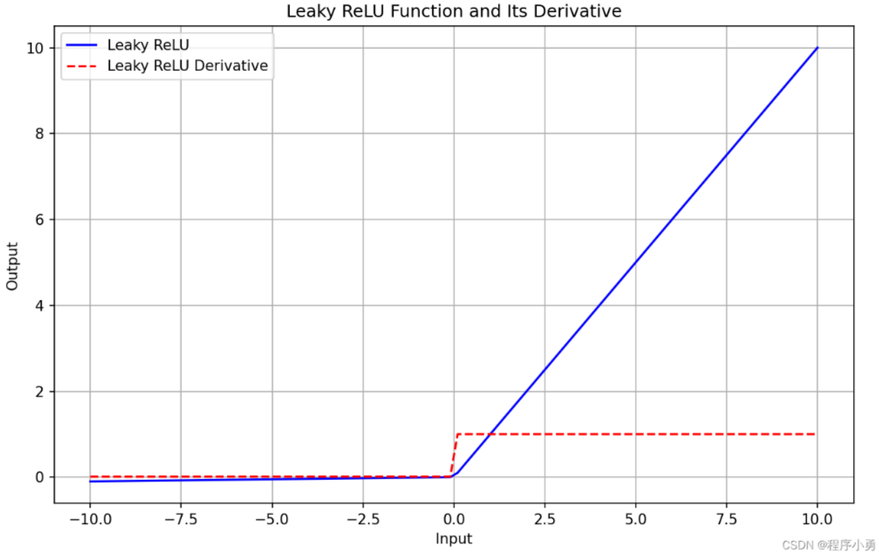


Activation layers

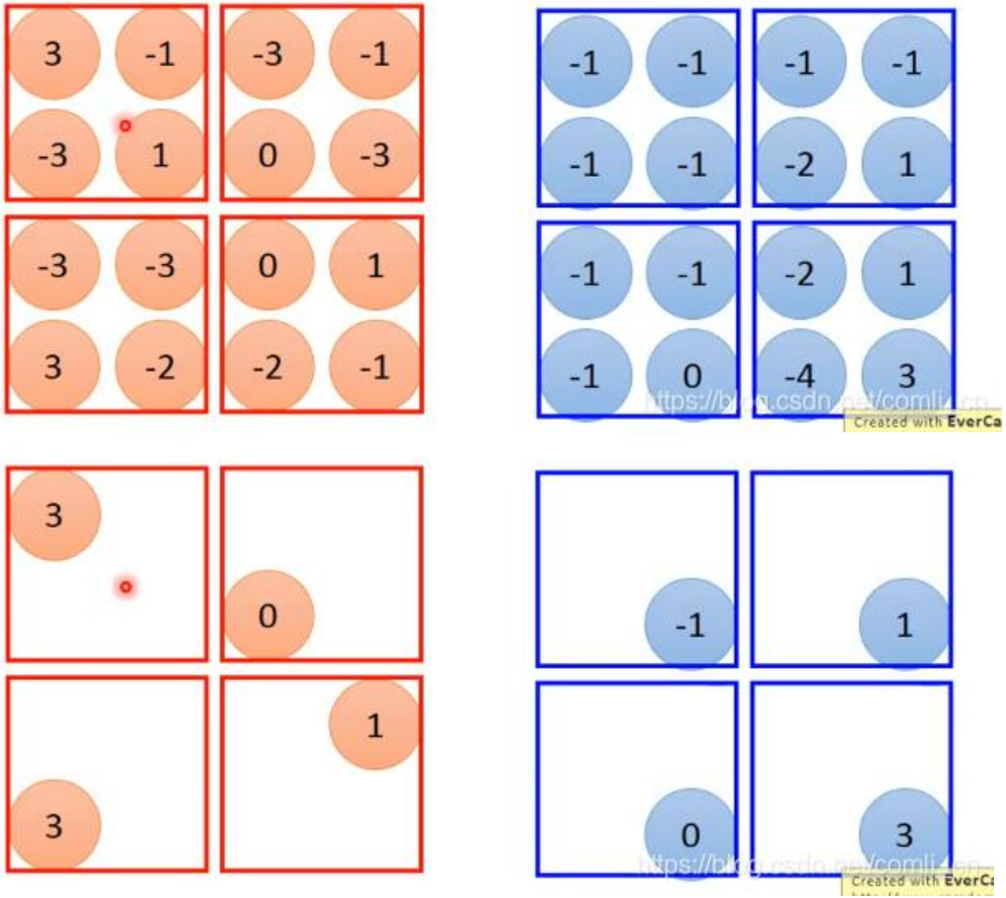
$$\mathrm{ReLU}(x) = \max(0, x)$$

$$\mathrm{LeakyReLU}(x) = \max(\alpha x, x)$$





Pooling layers



Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

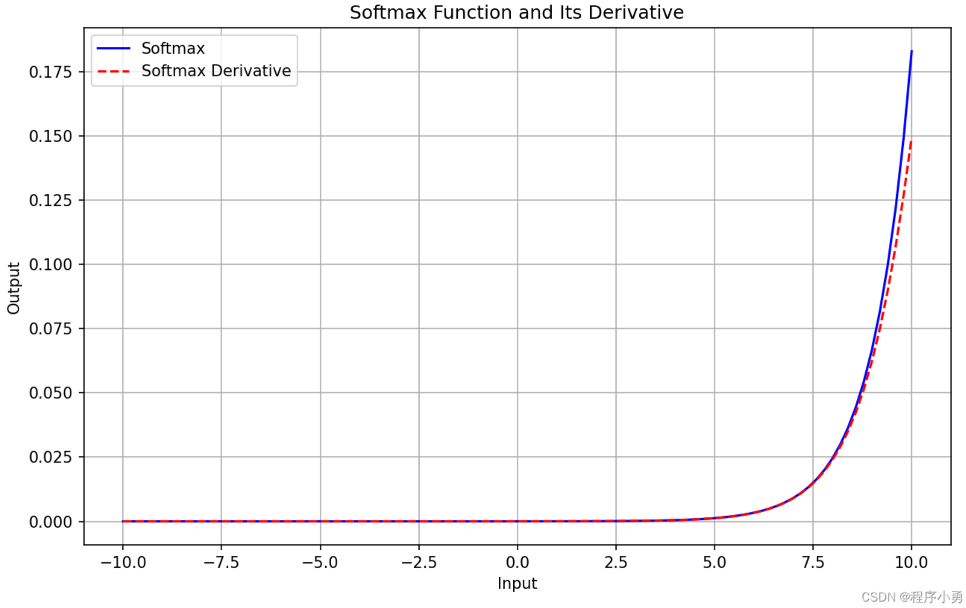
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Softmax

$$\mathrm{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$



For the above nonlinear operations， **BatchNorm needs private division $\frac{1}{y}$, inverse square root $\frac{1}{\sqrt{y}}$; and softmax needs exponential function e^x.**

总结

- 实际上在做Backword Pass的时候，就是建立一个反向的neural network的过程，对损失函数求导 = 前向传播 * 后向传播
- 链式法则将计算∂C / ∂w 拆成前向过程与后向过程。
- 前向过程计算的是∂z / ∂w，这里z是w所指neuron的input，计算结果是与w相连的值。
- 后向过程计算的是∂C / ∂z，这里z仍是w所指neuron的input，计算结果通过从后至前递归得到。

论文阅读1 SecureNN

3-Party Secure Computation for Neural Network Training

第二单元

Main contribution

To construct new and efficient protocols for non-linear functions such as ReLU and Maxpool that completely avoid the use of garbled circuits and give at least 8× improvement in communication complexity.

Protocols Structure

- **Supporting Protocols**：Matrix Multiplication、Select Share、Private Compare、Share Convert、Compute MSB
- **Main Protocols**： Linear and Convolutional Layer、Derivative of ReLU、ReLU、Division、Maxpool、Derivative of Maxpool

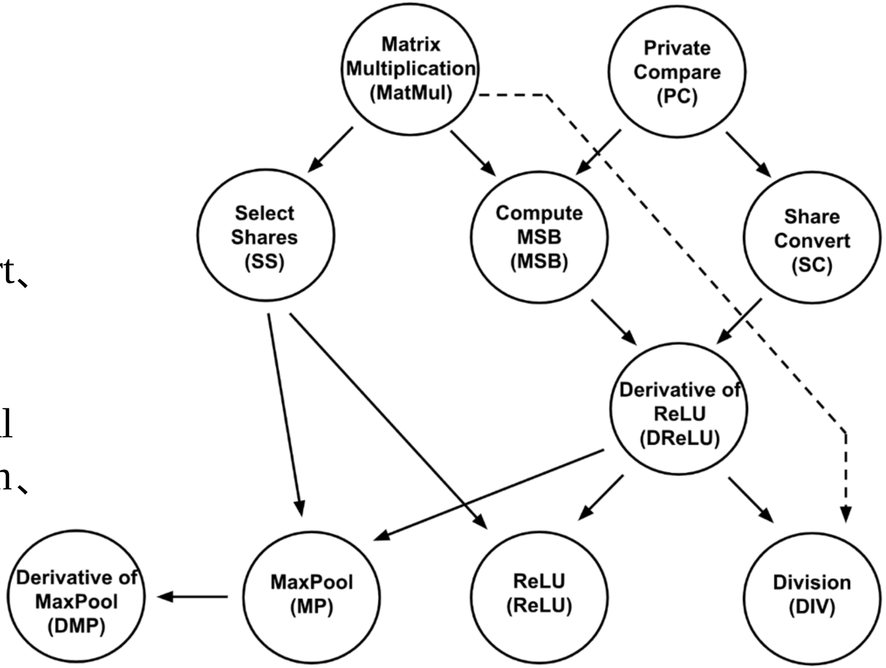


Fig. 3. Functionality dependence of protocols in SecureNN

notation

1. 环的定义:

- Z_L：模 L 的整数环，其中 $L = 2^\ell$。
- Z_{L-1}：模 $L-1$ 的整数环，是一个奇数大小的环。
- Z_p：模 p 的整数环，其中 p 是一个素数，构成一个域。

2.秘密共享类型:

- $\langle x \rangle_0^t$ 和 $\langle x \rangle_1^t$：在 Z_t 上 x 的两个加法份额。
- $Share^t(x)$：算法，用于生成 x 在 Z_t 上的两个份额。
- $\mathrm{Reconst}^t(x_0, x_1)$：算法，用于使用两个份额 x_0 和 x_1 重建 x 的值。

3.份额表示:

- $x[i]$：表示 ℓ 位整数 x 的第 i 位比特。
- $\{\langle x[i] \rangle^t\}_{i \in [\ell]}$：表示 x 的比特份额在 Z_t 上的集合。
- $\langle X \rangle_0^t$ 和 $\langle X \rangle_1^t$：表示通过逐元素秘密共享 $m \times n$ 矩阵 X 的元素创建的矩阵。
- $\mathrm{Reconst}^t(X_0, X_1)$：算法，用于重建秘密共享的矩阵 X，定义为逐元素重建。

下面让我们来想想DReLU的计算...

1. 计算目标：给定整数 a，$f(a) = 1\ if\ a \geq 0\ else\ 0$

2.补码表示下进一步可以转换为求符号位MSB（$f(a) = 1\ if\ MSB(a) = 0\ else\ 0$）

3. 奇数环下有 $MSB(a) = LSB(2a)$

证明：假设$MSB(a) = 1$

$\rightarrow a > n/2$

$\rightarrow n > 2a - n > 0$

$\rightarrow 2a - n$ 为奇数

$\rightarrow LSB(2a - n) = 1$

$\rightarrow LSB(2a) = 1$

4. LSB的计算和环绕问题

- P_0 和 P_1双方本地计算 $y=2a$，则 $MSB(a)=LSB(y)=y[0]$ （此时n为奇数）
- P_2 随机选择整数 x 并将其share(以及 $x[0]$ 的share)发送给 P_0 和 P_1，P_0 和 P_1 本地计算 $r=y+x$
- $y[0]=r[0]\oplus x[0]\oplus \text{wrap}(x,y,n)$

  其中$\text{wrap}(x,y,n)=1 \text{ if } x+y\geq n \text{ else } 0$

  证：$x+y<n$时 $y[0]=r[0]\oplus x[0]$

  $x+y\geq n$时 $y[0]=r[0]\oplus x[0]\oplus 1$

  故定义 $\text{wrap}(x,y,n)=1 \text{ if } x+y\geq n \text{ else}$
- 如何求wrap？

  当$x+y\geq n$ 时 $r=x+y-n \rightarrow n-y=x-r$

  $y<n \rightarrow x>r$

  故$wrap\,(x,y,n) = (x>r)$

SecureNN主要用到了三个环: Z_L、Z_{L-1}、Z_p，其中 $L=2^{\varrho}$，ϱ是个小素数，在实验中设定 $\varrho=64, p=67$。

Matrix Multiplication

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 $-jE \cdot F + \langle X\rangle_j^L \cdot F + E \cdot \langle Y\rangle_j^L + \langle C\rangle_j^L + U_j$.

P_0: $X_0 \cdot F + E \cdot Y_0 + C_0 + U_0$

P_1: $-E \cdot F + X_1 \cdot F + E \cdot Y_1 + C_1 + U_1$

$P_0 + P_1$: $-E \cdot F + X \cdot F + E \cdot Y + C + U$

$$= -(X - A)(Y - B) + \mathrm{X}(\mathrm{Y} - \mathrm{B}) + (\mathrm{X} - \mathrm{A})\mathrm{Y} + \mathrm{AB} + \mathrm{U}$$

$$= XY$$

Select Share

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 $\langle z\rangle_j^L = \langle x\rangle_j^L + \langle c\rangle_j^L + u_j$.

$$(1-\alpha)x + \alpha y = x + \alpha(y - x)$$

Private Compare

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'.

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 $u_i \in \mathbb{Z}_p^*$.

主要思想：比较大小也就是判断从最高有效位往右数的第一个不同比特位谁为1

P_0 和 P_1握有整数 $x(x \in Z_L, L = 2^{\varrho})$ 中每一比特在 Z_p上的share，并持有一个 ϱ 比特的公共整数 r 和公共随机比特 β，执行该算法后，P_2 获得

$$\beta' = \beta \oplus (x > r)(x > r \in \{0,1\})$$

- 如果 $\beta = 0$，则 $\beta' = (x > r)$，
  1. 从左到右遍历每一个比特（step3），计算$w_i = x[i] \oplus r[i] = x[i] + r[i] - 2x[i]r[i]$ (step5)
     （证明：$w_i \in \{0,1\}$，$-2x[i]r[i]$是为了防止$x[i] + r[i]$的情况。也可以使用真值表）
  2. 计算$c[i] = r[i] - x[i] + 1 + \sum_{k=i+1}^{\varrho} w_k$ (step6)
     如果 x 和 r 第 i 个比特不同时，$(x[i], r[i] \in \{0,1\})$
     - $x[i] = 1 \to c[i] = 0$（此时 $r[i] = 0, x[i] = 1, \sum_{k=i+1}^{\varrho} w_k = 0$），
     - $x[i] = 0 \to c[i] = 2$，同时后续的所有 $c[i]$都会大于等于1（因为 $\sum_{k=i+1}^{\varrho} w_k \geq 1$），

     所以最后只需要判断是否存在一个$c[i] = 0$ 即可（step15），为了不泄露敏感信息给 P_2，P_0 和 P_1 在发送 $c[i]$ 的share给 P_2 之前乘了个随机掩码 s_i 并作集合位置随机置换 π（step14）；
- 如果 $\beta = 1$ 且 $r \neq 2^{\varrho} - 1$，则$\beta' = (x \leq r) = (x < r + 1)$，计算逻辑是一样的（step8~9）；
- 如果 $\beta = 1$ 且 $r = 2^{\varrho} - 1$，则 $(x \leq r)$ 必成立，此时随便置个 $c[i] = 0$ 即可，算法里选的是 $i = 1$（step11）

Share Convert

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 $L-1$)

$$\theta = wrap(\langle a\rangle_0^L, \langle a\rangle_1^L, L)$$

$$r = \langle r\rangle_0^L + \langle r\rangle_1^L - \alpha L \quad (1)$$

$$\langle \tilde{a}\rangle_j^L = \langle a\rangle_j^L + \langle r\rangle_j^L - \beta_j L \quad (2)$$

$$x = \langle \tilde{a}\rangle_0^L + \langle \tilde{a}\rangle_1^L - \delta L \quad (3)$$

$$1\{x > r-1\}$$

$$\eta = 1\{x > r-1\} \qquad 1-\eta = 1\{x < r\}$$

$$x = a + r - (1-\eta)L \quad (4)$$

$$a = \langle a\rangle_0^L + \langle a\rangle_1^L - \theta L \quad (5)$$

$$\theta = (1)\text{-}(2)\text{-}(3)+(4)+(5)$$

Compute MSB

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 $\langle \alpha \rangle_j^L = \langle \gamma \rangle_j^L + \langle \delta \rangle_j^L - 2\langle \theta \rangle_j^L + u_j$.

P_2 picks a random number x, P_2 gives secret shares of x as well as shares of x[0] to P_0, P_1

$y = 2a \quad r = y + x$

$1\{x > r\}$

$\beta' = \beta \oplus (x > r)$

$\gamma = \beta' \oplus \beta = (x > r)$

$\delta = x[0] \oplus r[0]$

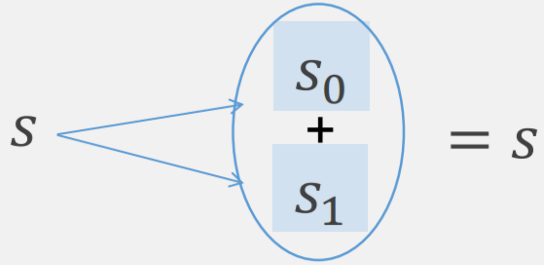
$y[0] = x[0] \oplus r[0] \oplus wrap(x, y, L) = \gamma \oplus \delta$

FSS和DPF的构造

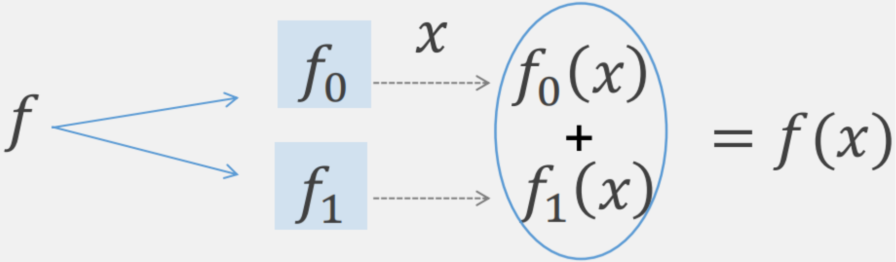
第三单元

FSS(function secret sharing)

- 直觉
  - Additive Secret Sharing
    - Elements in Abelian group G



    - Secrecy: s_i hides s
    - Reconstruction: $s_0 + s_1 = s$ (in G)
  - Additive Secret Sharing of Functions



    - Secrecy: f_i (computationally) hides f
    - Reconstruction: $f_0(x) + f_1(x) = f(x)$
    - Efficiency: $|f_i| \sim f$

DPF和DCF

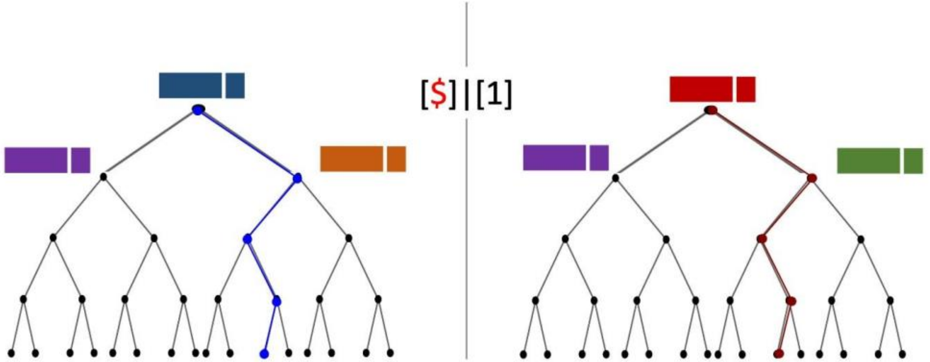
DPF: $f_{(\alpha,\beta)}:\{0,1\}^l \to G$， 当$x=\alpha$时有$f_{\alpha,\beta}(x)=\beta$，
当$x\neq\alpha$时 有$f_{\alpha,\beta}(x)=0$

DCF: $f_{(\alpha,\beta)}:\{0,1\}^l \to G$， 当$x<\alpha$时有$f_{\alpha,\beta}(x)=\beta$，
当$x\geq\alpha$时 有$f_{\alpha,\beta}(x)=0$

1. **密钥生成：**
   a) 构建n位输入树，确定一条特殊路径α。
   b) 路径α上的节点包含随机种子和控制比特，其他节点为0。
   c) 每层添加纠正词，确保左孩子节点和右孩子节点中，一个节点的信息为0，另一个节点的随机种子为随机串与1的组合。
2. **函数评估：**
   a) 双方使用随机种子分片和控制比特通过伪随机生成器（PRG）生成子树。
   b) 根据控制比特决定是否添加纠正词。
   c) 沿着路径α评估函数，保持节点信息为1，偏离路径则节点信息为0。

DPF的构造

DPF Construction from PRGs



Invariant for Eval:

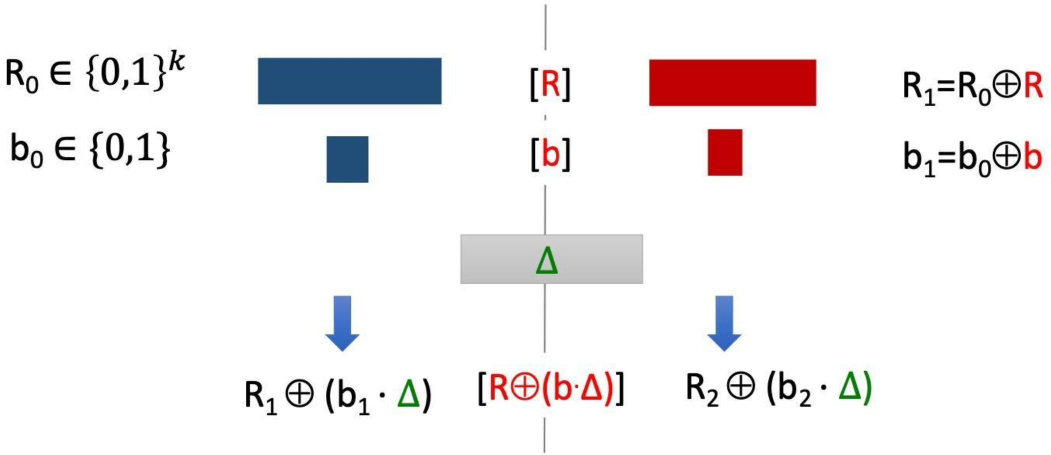
For each node v on evaluation path we have [S]||[b]

- v on special path: S is pseudorandom, b=1
- v off special path: S=0, b=0

知乎 @深夜适合听yoga

DPF的构造

Gadget: Conditional Correction



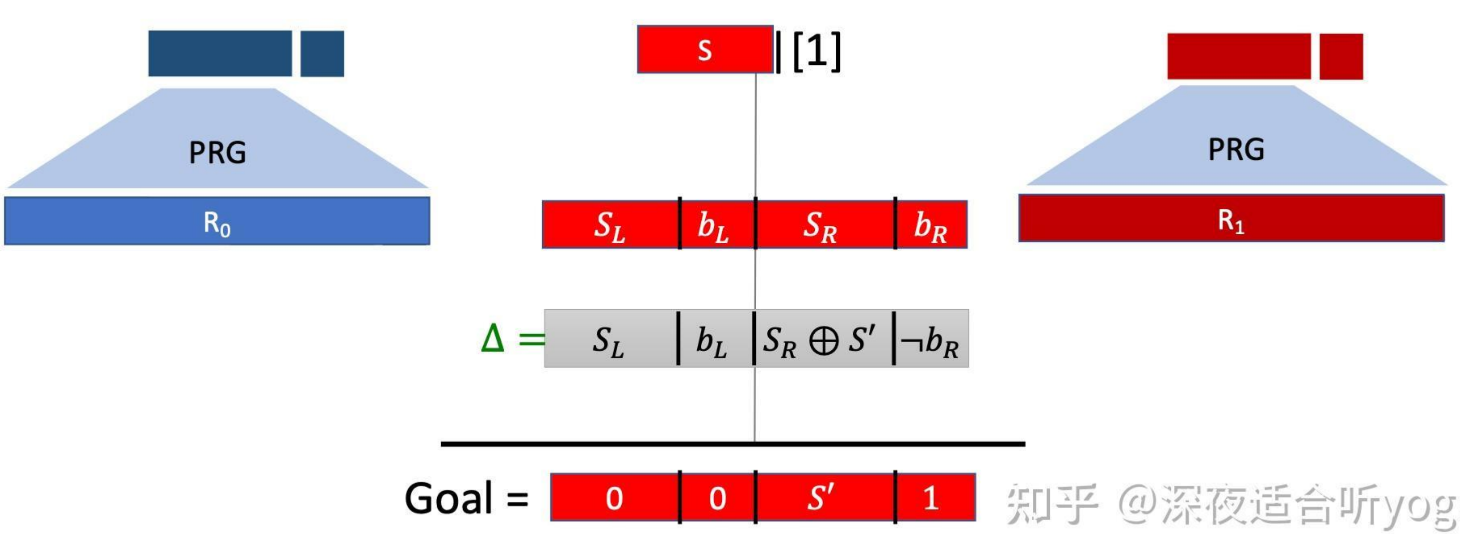
Test yourself:

- R=0, b=0 $\Rightarrow$ generate shares of... 0!
- Δ=R, b=1 $\Rightarrow$ generate shares of... 0!

知乎 @深夜适合听yoga

DPF的构造

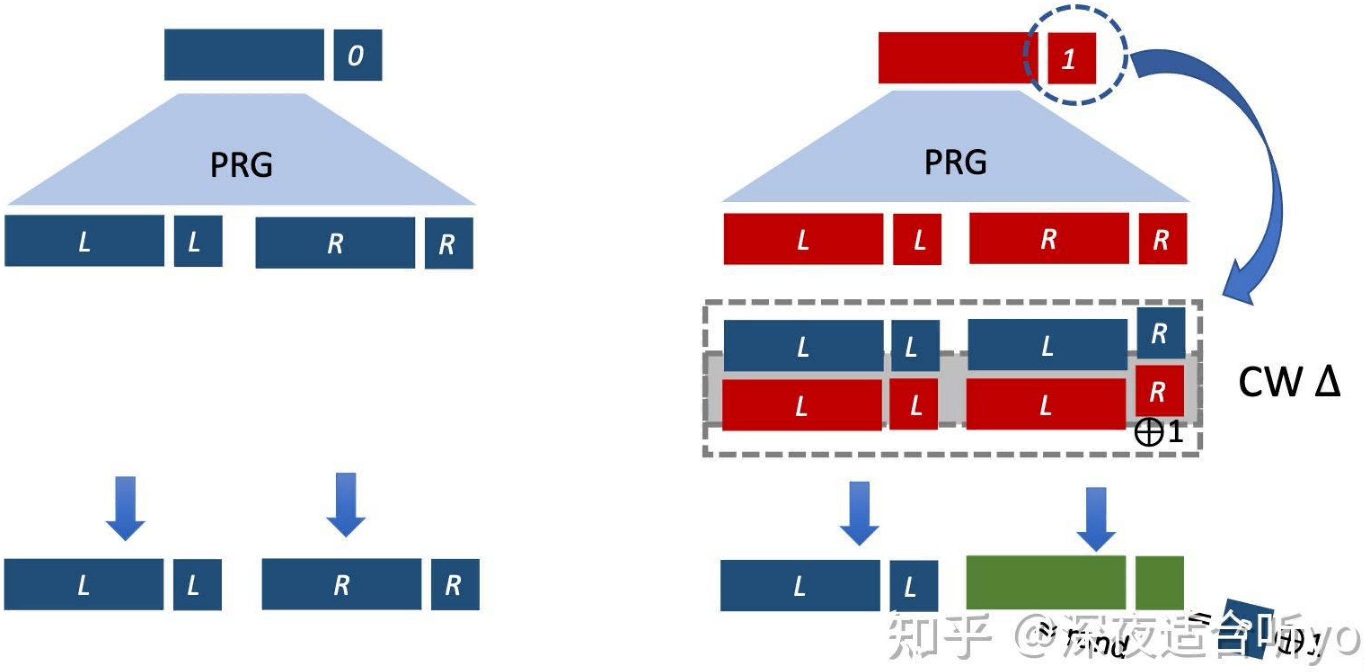
Building the Correction Word Δ



知乎 @深夜适合听yoga

DPF的构造

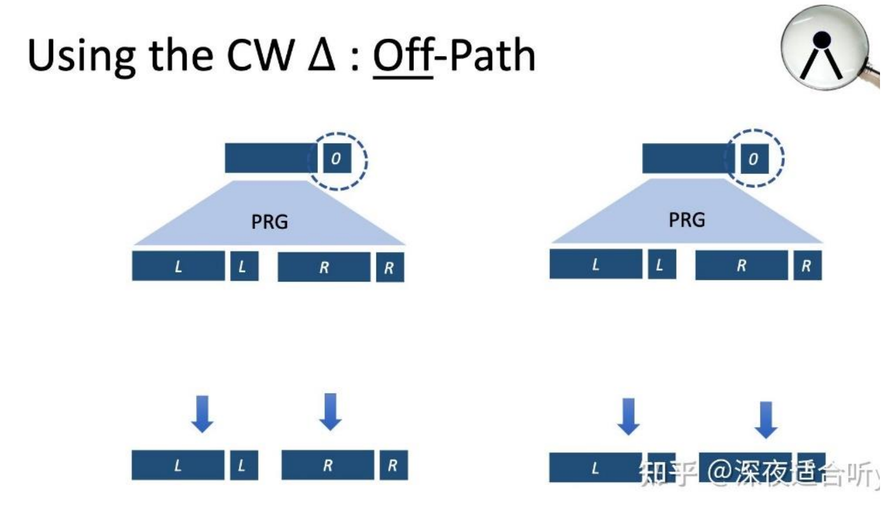
Using the CW Δ : On-Path



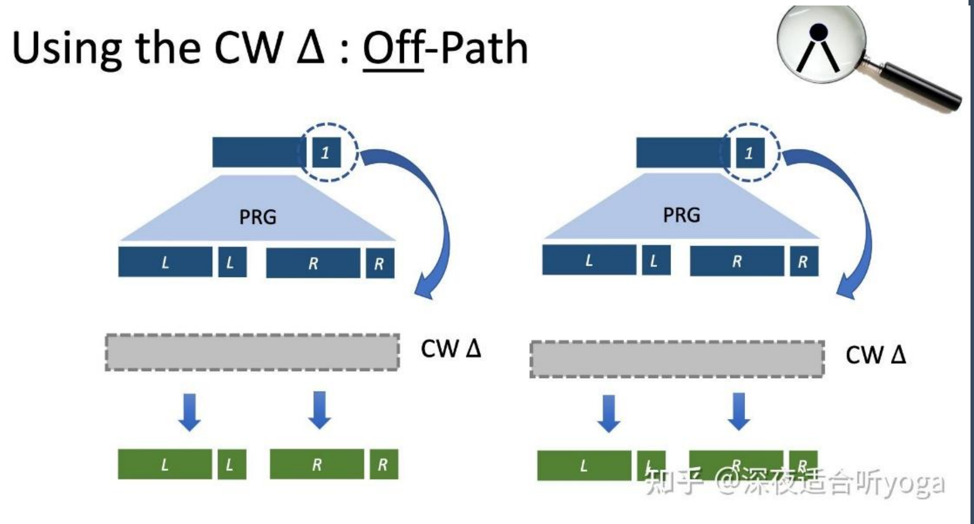
知乎 @深夜适合听yoga

DPF的构造

（1）两边的控制比特都是0



（2）两边的控制比特都是1



DPF的构造（BGI 15，P12/BGI16b，P5）

Optimized Distributed Point Function $(\mathsf{Gen}^\bullet, \mathsf{Eval}^\bullet)$

Let $G : \{0,1\}^\lambda \to \{0,1\}^{2(\lambda+1)}$ a pseudorandom generator.
Let $\mathsf{Convert}_{\mathbb{G}} : \{0,1\}^\lambda \to \mathbb{G}$ be a map converting a random λ-bit string to a pseudorandom group element of $\mathbb{G}$.

$\mathsf{Gen}^\bullet(1^\lambda, \alpha, \beta, \mathbb{G})$:

Let $\alpha = \alpha_1, \ldots, \alpha_n \in \{0,1\}^n$ be the bit decomposition.
Sample random $s_0^{(0)} \leftarrow \{0,1\}^\lambda$ and $s_1^{(0)} \leftarrow \{0,1\}^\lambda$
Sample random $t_0^{(0)} \leftarrow \{0,1\}$ and take $t_1^{(0)} \leftarrow t_0^{(0)} \oplus 1$
for $i = 1$ to n **do**
    $s_0^L || t_0^L \;||\; s_0^R || t_0^R \leftarrow G(s_0^{(i-1)})$ and $s_1^L || t_1^L \;||\; s_1^R || t_1^R \leftarrow G(s_1^{(i-1)})$.
    if $\alpha_i = 0$ **then** $\mathsf{Keep} \leftarrow L$, $\mathsf{Lose} \leftarrow R$
    else $\mathsf{Keep} \leftarrow R$, $\mathsf{Lose} \leftarrow L$
    end if
    $s_{CW} \leftarrow s_0^{\mathsf{Lose}} \oplus s_1^{\mathsf{Lose}}$
    $t_{CW}^L \leftarrow t_0^L \oplus t_1^L \oplus \alpha_i \oplus 1$ and $t_{CW}^R \leftarrow t_0^R \oplus t_1^R \oplus \alpha_i$
    $CW^{(i)} \leftarrow s_{CW} || t_{CW}^L || t_{CW}^R$
    $s_b^{(i)} \leftarrow s_b^{\mathsf{Keep}} \oplus t_b^{(i-1)} \cdot s_{CW}$ for $b = 0, 1$
    $t_b^{(i)} \leftarrow t_b^{\mathsf{Keep}} \oplus t_b^{(i-1)} \cdot t_{CW}^{\mathsf{Keep}}$ for $b = 0, 1$
end for
$CW^{(n+1)} \leftarrow (-1)^{t_1^n} \left[\beta - \mathsf{Convert}(s_0^{(n)}) + \mathsf{Convert}(s_1^{(n)})\right]$, with addition in $\mathbb{G}$
Let $k_b = s_b^{(0)} || t_b^{(0)} || CW^{(1)} || \cdots || CW^{(n+1)}$
return (k_0, k_1)

$\mathsf{Eval}^\bullet(b, k_b, x)$:

Parse $k_b = s^{(0)} || t^{(0)} || CW^{(1)} || \cdots || CW^{(n+1)}$
for $i = 1$ to n **do**
    Parse $CW^{(i)} = s_{CW} || t_{CW}^L || t_{CW}^R$
    $\tau^{(i)} \leftarrow G(s^{(i-1)}) \oplus (t^{(i-1)} \cdot [s_{CW} || t_{CW}^L || s_{CW} || t_{CW}^R])$
    Parse $\tau^{(i)} = s^L || t^L \;||\; s^R || t^R \in \{0,1\}^{2(\lambda+1)}$
    if $x_i = 0$ **then** $s^{(i)} \leftarrow s^L, t^{(i)} \leftarrow t^L$
    else $s^{(i)} \leftarrow s^R$ and $t^{(i)} \leftarrow t^R$
    end if
end for
return $(-1)^b \left[\mathsf{Convert}(s^{(n)}) + t^{(n)} \cdot CW^{(n+1)}\right] \in \mathbb{G}$

Orca: FSS-based Secure Training and Inference with GPUs

2024 S&P

Main idea

- Function secret sharing (FSS) based secure 2-party computation (2PC) protocols in the pre-processing model
- Key feature: **reduce online communication** while increasing compute and storage

Contributions

Accelerate compute: GPU micro-architecture

- use GPU’ s scratchpad memory
- optimize data layout

Reducing time to read FSS keys:

bottleneck: read the FSS keys from the storage (SSD) to GPU’ s memory

- reduce key size

System optimizations

Novel protocols:

- Stochastic truncations
- ReLUs
- maxpools
- floating-point softmax

Cryptographic improvements

Secure 2PC with pre-processing using FSS

secure 2PC protocol using FSS（BGI, TCC 2019　Boyle et al.(2020)）

Offline Phase:

- For each wire w_i in the computation circuit, randomly sample a mask r_i
- For each gate g, with input wire w_i and output wire w_j
- generate FSS keys $\left(k_0^g, k_1^g\right)$ for the **offset function**, $g^{[r_i,r_j]}(x) = g(x - r_i) + r_j$, and provide party b with k_b^g
- For input and output wires w_i owned by party b, party b learns the corresponding mask r_i

Secure 2PC with pre-processing using FSS

Online Phase:

- For each input wire w_i with value x_i owned by party b, the party b calculates masked wire value $\hat{x}_i = x_i + r_i$ and sends it to the other party
- From the input gates, the two parties process gates in topological order to receive masked output wire values
- To process a gate g, with input wire w_i , output wire w_j , and masked input wire value $\hat{x}_i = x_i + r_i$, party b uses **Eval** with k_b^g and $\hat{x}_i$ to obtain a share of the masked output wire value $\hat{x}_j = g^{[r_i,r_j]}(\hat{x}_i) = g(\hat{x}_i - r_i) + r_j = g(x_i) + r_j = x_j + r_j$

Secure 2PC with pre-processing using FSS

FSS Gates :

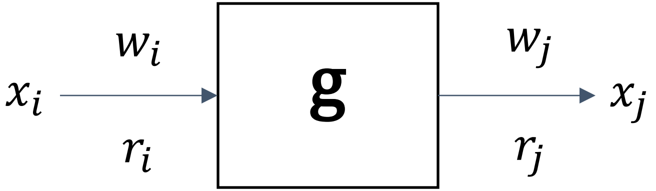
Definition 3 (FSS Gates [3]). *Let* $\mathcal{G} = \{g : \mathbb{G}^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}}\}$ *be a computation gate (parameterized by input and output groups* $\mathbb{G}^{\mathsf{in}}, \mathbb{G}^{\mathsf{out}}$*). The family of offset functions* $\hat{\mathcal{G}}$ *of* $\mathcal{G}$ *is given by*

$$\hat{\mathcal{G}} := \left\{ g^{[\mathsf{r}^{in},\mathsf{r}^{out}]} : \mathbb{G}^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}} \;\middle|\; \begin{array}{c} g : \mathbb{G}^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}} \in \mathcal{G}, \\ \mathsf{r}^{in} \in \mathbb{G}^{\mathsf{in}}, \mathsf{r}^{out} \in \mathbb{G}^{\mathsf{out}} \end{array} \right\}$$

$$\textit{where } g^{[\mathsf{r}^{in},\mathsf{r}^{out}]}(x) := g(x - \mathsf{r}^{in}) + \mathsf{r}^{out},$$

and $g^{[\mathsf{r}^{in},\mathsf{r}^{out}]}$ *contains an explicit description of* $\mathsf{r}^{in}, \mathsf{r}^{out}$*. Finally, we use the term FSS gate for* $\mathcal{G}$ *to denote an FSS scheme for the corresponding offset family* $\hat{\mathcal{G}}$*.*

Secure 2PC with pre-processing using FSS



$$g(x_i) = x_j$$

$$\hat{x}_i = x_i + r_i \qquad \hat{x}_j = g(x_i) + r_j$$

$$\boldsymbol{P_0}: \quad \hat{x}_i \xrightarrow{\quad k_0 \quad} f_0(\hat{x}_i) = g(\hat{x}_i - r_i) + r_j$$

$$\boldsymbol{P_1}: \quad \hat{x}_i \xrightarrow{\quad k_1 \quad} f_1(\hat{x}_i) = g(\hat{x}_i - r_i) + r_j$$

$$\rightarrow x_j$$

Protocol for Select

Select function: input a selector bit $s \in \{0, 1\}$ and an n-bit payload $x \in U_N$, returns x if s = 1 and 0 otherwise ⟹ $select_n(s, x) = s \cdot x$

offset function for $select_n$:

$$
\begin{aligned}
\mathsf{select}_n^{[(\mathsf{r}_1^{\mathsf{in}}, \mathsf{r}_2^{\mathsf{in}}), \mathsf{r}^{\mathsf{out}}]}(\hat{s}, \hat{x}) &= (\hat{s} - \mathsf{r}_1^{\mathsf{in}} + 2 \cdot 1\{\hat{s} < \mathsf{r}_1^{\mathsf{in}}\}) \cdot (\hat{x} - \mathsf{r}_2^{\mathsf{in}}) \\
&\quad + \mathsf{r}^{\mathsf{out}} \mod 2^n \\
&= \hat{s} \cdot \hat{x} - \mathsf{r}_1^{\mathsf{in}} \cdot \hat{x} - \hat{s} \cdot \mathsf{r}_2^{\mathsf{in}} + \mathsf{r}_1^{\mathsf{in}} \cdot \mathsf{r}_2^{\mathsf{in}} + \mathsf{r}^{\mathsf{out}} \\
&\quad + 2 \cdot 1\{\hat{s} = 0 \text{ and } \mathsf{r}_1^{\mathsf{in}} = 1\} \cdot (\hat{x} - \mathsf{r}_2^{\mathsf{in}}) \mod 2^n
\end{aligned}
$$

Here, use the fact that $1\{\hat{s} < r_1^{in}\} = 1\{\hat{s} = 0 \ and\ r_1^{in} = 1\}$ as $\hat{s}$ and r_1^{in} are single bit values

Protocol for Select

$$\mathsf{select}_n^{[(\mathsf{r}_1^{\mathsf{in}},\mathsf{r}_2^{\mathsf{in}}),\mathsf{r}^{\mathsf{out}}]}(\hat{s},\hat{x}) = (\hat{s} - \mathsf{r}_1^{\mathsf{in}} + 2\cdot 1\{\hat{s} < \mathsf{r}_1^{\mathsf{in}}\})\cdot(\hat{x} - \mathsf{r}_2^{\mathsf{in}}) + \mathsf{r}^{\mathsf{out}} \mod 2^n$$

$$= \hat{s}\cdot\hat{x} - \mathsf{r}_1^{\mathsf{in}}\cdot\hat{x} - \hat{s}\cdot\mathsf{r}_2^{\mathsf{in}} + \mathsf{r}_1^{\mathsf{in}}\cdot\mathsf{r}_2^{\mathsf{in}} + \mathsf{r}^{\mathsf{out}} + 2\cdot 1\{\hat{s} = 0 \text{ and } \mathsf{r}_1^{\mathsf{in}} = 1\}\cdot(\hat{x} - \mathsf{r}_2^{\mathsf{in}}) \mod 2^n$$

if $\hat{s} = 0$:

$$select_n = -r_1^{in}\cdot\hat{x} + (r_1^{in}\cdot r_2^{in} + r^{out}) + 2\cdot\hat{x}\cdot r_1^{in} - 2\cdot r_2^{in}\cdot r_1^{in} \mod 2^n$$

$$= r_1^{in}\cdot\hat{x} + (r_1^{in}\cdot r_2^{in} + r^{out}) - 2\cdot r_2^{in}\cdot r_1^{in} \mod 2^n$$

if $\hat{s} = 1$:

$$select_n = \hat{x} - r_1^{in}\cdot\hat{x} - r_2^{in} + (r_1^{in}\cdot r_2^{in} + r^{out}) \mod 2^n$$

Protocol for Select

Select Π_n^{select}

$\mathsf{Gen}_n^{\mathsf{select}}((\mathsf{r}_1^{\mathsf{in}}, \mathsf{r}_2^{\mathsf{in}}), \mathsf{r}^{\mathsf{out}})$:

$u = \mathsf{extend}(\mathsf{r}_1^{\mathsf{in}}, n)$
$w = u \cdot \mathsf{r}_2^{\mathsf{in}} + \mathsf{r}^{\mathsf{out}}$
$z = 2 \cdot u \cdot \mathsf{r}_2^{\mathsf{in}}$
share $(u, \mathsf{r}_2^{\mathsf{in}}, w, z)$
For $b \in \{0,1\}$, $k_b = u_b || \mathsf{r}_{2,b}^{\mathsf{in}} || w_b || z_b$

$\mathsf{Eval}_n^{\mathsf{select}}(b, k_b, (\hat{s}, \hat{x}))$:

Parse k_b as $u_b || \mathsf{r}_{2,b}^{\mathsf{in}} || w_b || z_b$
if $\hat{s} = 0$ **then**
  return $\hat{y}_b = u_b \cdot \hat{x} + w_b - z_b$
else
  return $\hat{y}_b = b \cdot \hat{x} - u_b \cdot \hat{x} - \mathsf{r}_{2,b}^{\mathsf{in}} + w_b$
end if

if $\hat{s} = 0$

$$\hat{y}_b = u_b \cdot \hat{x} + w_b - z_b$$
$$= r_{1,b}^{in} \cdot \hat{x} + (r_{1,b}^{in} \cdot r_{2,b}^{in} + r_b^{out}) - 2 \cdot r_{1,b}^{in} \cdot r_{2,b}^{in}$$

if $\hat{s} = 1$

$$\hat{y}_b = \mathrm{b} \cdot \hat{x} - u_b \cdot \hat{x} - r_{2,b}^{in} + w_b$$
$$= \mathrm{b} \cdot \hat{x} - r_{1,b}^{in} \cdot \hat{x} - r_{2,b}^{in} + (r_{1,b}^{in} \cdot r_{2,b}^{in} + r_b^{out})$$

if b=1

$$\hat{y}_b = \hat{x} - r_{2,b}^{in} - r_{1,b}^{in} \cdot \hat{x} + (r_{1,b}^{in} \cdot r_{2,b}^{in} + r_b^{out})$$

if b=0

$$\hat{y}_b = - r_{2,b}^{in} - r_{1,b}^{in} \cdot \hat{x} + (r_{1,b}^{in} \cdot r_{2,b}^{in} + r_b^{out})$$

Evaluation

	Accuracy		Time (in min)		Comm. (in GB)	
Dataset	PIRANHA	ORCA	PIRANHA	ORCA	PIRANHA	ORCA
MNIST	96.8 (−0.3%)	97.1	56 (4.0×)	14	2,168.4 (43.2×)	50.2
CIFAR-10	55 (−4.6%)	59.6	1170 (22.5×)	52	65231.3 (98.5×)	662.4
	55 (−14%)	69	1170 (9.1×)	128	65231.3 (39.4×)	1656.1

PIRANHA is the current state-of-the-art in accelerating secure training using GPUs

ORCA compared to PIRANHA:

- higher accuracy
- generated models in 4 − 22× less time
- with 43−98× less communication

Evaluation

Model	LLAMA	CrypTen	ORCA	
			$n = 64$	time $[n, f]$
VGG-16	54.93 (103.6×)	13.19 (24.9×)	1.21 (2.3×)	0.53 [32, 12]
ResNet-50	45.83 (67.4×)	5.76 (8.5×)	0.93 (1.4×)	0.68 [37, 12]
ResNet-18	12.03 (85.9×)	2.97 (21.2×)	0.28 (2.0×)	0.14 [32, 10]

Runtime (in seconds) for secure ImageNet inference

ORCA enables **sub-second ImageNet-scale** inference of VGG-16 and ResNet-50 and outperforms state-of-the-art by an order of magnitude

参考资料

[1] ML:【[双语字幕]吴恩达深度学习deeplearning.ai-哔哩哔哩】 https://b23.tv/2VmdFfv
[2] ML: 【(强推)李宏毅2021/2022春机器学习课程-哔哩哔哩】 https://b23.tv/Lpeh2BS
[3] 论文阅读笔记：SecureNN: 3-Party Secure Computation for Neural Network Training - 知乎 (zhihu.com)
[4] 函数秘密分享（二） - 知乎 (zhihu.com)
[5] Wagh S, Gupta D, Chandran N. SecureNN: 3-party secure computation for neural network training[J]. Proceedings on Privacy Enhancing Technologies, 2019.
[6] Ng L K L, Chow S S M. SoK: cryptographic neural-network computation[C]//2023 IEEE Symposium on Security and Privacy (SP). IEEE, 2023: 497-514.
[7] Jawalkar N, Gupta K, Basu A, et al. Orca: Fss-based secure training and inference with gpus[C]//2024 IEEE Symposium on Security and Privacy (SP). IEEE, 2024: 597-616.
[8] E. Boyle, N. Chandran, N. Gilboa, D. Gupta, Y. Ishai, N. Kumar, and M. Rathee, “Function secret sharing for mixed-mode and fixed-point secure computation,” in EUROCRYPT, 2020.
[9] E. Boyle, N. Gilboa, and Y. Ishai, “Secure computation with preprocessing via function secret sharing,” in TCC, 2019.
[10] K. Gupta, D. Kumaraswamy, N. Chandran, and D. Gupta, “Llama: A low latency math library for secure inference,” in PETS, 2022.