Secure Softmax/Sigmoid for Machine-Learning Computation

今天介绍的论文是港中文郑宇博士等人发表在ACSAC'2023的关于机器学习中Softmax和Sigmoid安全近似优化的工作。论文和开源代码链接如下：

https://github.com/alipay/Antchain-MPC/tree/sec_softmoid

0. 背景和动机

在隐私保护机器学习中，Softmax和Sigmoid是两种常见的激活函数，但是其计算却非常复杂。为了使得两种函数的计算在MPC下更加高效、MPC-Friendly，本文分别提出了基于常微分方程（ordinary differential equations，ODE）和傅立叶级数（Fourier series）的近似算法，从而实现了常数轮的Softmax近似算法和一轮通信的Sigmoid近似算法。如下表所示，本文方案的理论通信量和通信轮数如下，实验也显示和CryptGPU，Piranha和MP-SPDZ相比，本文取得了较好的提升，对于Softmax和Sigmoid的安全计算，本文的方法可以减少83%~95%的通信量。

Table 1: Communication Comparison of Secure Computation of Softmax and Sigmoid

Frameworks	Softmax		Sigmoid		Party
	Communication	Round	Communication	Round	
SecureNN [33]	$8mn(n+1)\log p + 24mn(n+1)$	$11n$	n.a.	n.a.	3
ABY2.0 [25]	n.a.	n.a.	$(2x_1+15)\lambda + 2x_1 + 4x_2 + 18n + 2$	$\log_4 n + 2$	2
CrypTen [15]	$275(m-1)n\log n + 39mn + 2m + 106n - 2$	$39\log m + 41$	$96n$	16	2+
Ours	$(8mn+2n)r$	$2r$	$< 12n$	1	2+

Security parameter is $\lambda = 128$. r is a preset value representing the number of iterations. m is the number of classes. n is the bit-length of fixed-point numbers, usually $n = 64$. x_1, x_2 are parameters linear to the number of AND gates. p is the minimal prime greater than n.

1. 安全模型和基础协议

本文采用2PC+1dealer架构，其中P_0和P_1承担主要在线计算任务而T负责生成Beaver三元组等关联随机数。本文在诚实大多数场景下抵抗半诚实敌手，即敌手最多腐化一个计算参与方。

2. Softmax的安全计算

给定输入向量$\vec{x} = (x_1, \ldots, x_m)$，输出向量

$$\mathrm{Softmax}(\vec{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

2.1 Quasi-Softmax近似

和之前针对Softmax近似计算的工作不同，本文观察到输入的分布差距而不是误差绝对值对模型训练准确率的影响更大。基于此，作者从概率角度提出了quasi-Softmax (QSMax)来近似计算Softmax。QSMax的定义如下：

Qsasi-Softmax 定义： 函数$f:\mathbb{R}^m \to [0,1]^m$是QSMax如果满足以下性质：

- $f(\vec{x})$是定义在$[0,1]^m$上的概率分布，即$\sum_{i=1}^m [f(\vec{x})]_i = 1$且对于任意的$i=1,\dots,m$有$[f(\vec{x})]_i \geq 0$;
- $[f(\vec{x})]_i \leq [f(\vec{x})]_j$ 当且仅当 $x_i \leq x_j$，其中$i,j=1,\dots,m$。

为了将多元的Softmax计算转化为单元函数，定义$\vec{f}(t)=\mathrm{softmax}(t\vec{x})$。当$t=1$时，得到$\vec{f}(1)=\mathrm{QSMax}(\vec{x})$。同时$\vec{f}(0)=\vec{1}/m$，且$\vec{f}'(t)=(\vec{x}-\langle\vec{x},\vec{f}(t)\rangle\vec{1})*\vec{f}(t)$。
本文将计算多元函数$f(\vec{x})$转化为计算单元单数$\vec{g}(\frac{i}{r})$，$\vec{g}$的输出为m维向量，使得$\vec{g}$为一个迭代解。明文下迭代近似计算算法如下：

$\vec{\mathrm{g}}(0) = [1/m, \dots, 1/m]$
for $i = 1, 2, 3, \dots, r$ **do**
  $\vec{\mathrm{g}}(\frac{i}{r}) = \vec{\mathrm{g}}(\frac{i-1}{r}) + (\vec{x} - \langle \vec{x}, \vec{\mathrm{g}}(\frac{i-1}{r}) \rangle \vec{1}) * \vec{\mathrm{g}}(\frac{i-1}{r}) \cdot \frac{1}{r}$
end for

其中r是超参数。进行完r次迭代计算之后，输出的$\vec{g}(1)$即为Softmax的近似结果。

2.2 Quasi-Softmax安全计算协议

Protocol 1 Π_{QSMax}: Quasi-Softmax via ODE

P_0 Input	$\langle\vec{x}\rangle_0, r$	P_0 Output	$\langle \mathrm{QSMax}(\vec{x})\rangle_0$
P_1 Input	$\langle\vec{x}\rangle_1, r$	P_1 Output	$\langle \mathrm{QSMax}(\vec{x})\rangle_1$

1: {Initial-Value Processing}
2: P_0: Set $\langle\vec{x}\rangle_0 = \mathrm{FR}(\frac{1}{r}) \cdot \langle\vec{x}\rangle_0$ and $\langle\vec{g}(0)\rangle_0 = \vec{1}/m$
3: P_1: Set $\langle\vec{x}\rangle_1 = \mathrm{FR}(\frac{1}{r}) \cdot \langle\vec{x}\rangle_1$ and $\langle\vec{g}(0)\rangle_1 = \vec{0}$
4: {Iteration for $\vec{f}(\frac{i}{r})$ via Euler Formula}
5: **for** $i = 1, 2, 3, \ldots, r$ **do**
6: $\quad P_0, P_1$: $\langle\vec{o}\rangle_0, \langle\vec{o}\rangle_1 = \Pi_{\times}(\langle\vec{x}\rangle_0, \langle\vec{g}(\frac{i-1}{r})\rangle_0; \langle\vec{x}\rangle_1, \langle\vec{g}(\frac{i-1}{r})\rangle_1)$
7: $\quad P_0$: Compute $\langle\vec{q}\rangle_0 = (\sum_{i=1}^{m} \langle[\vec{o}]_i\rangle_0) \cdot \vec{1}$
8: $\quad P_1$: Compute $\langle\vec{q}\rangle_1 = (\sum_{i=1}^{m} \langle[\vec{o}]_i\rangle_1) \cdot \vec{1}$
9: $\quad P_0, P_1$: $\langle\vec{t}\rangle_{\mathrm{L}}, \langle\vec{t}\rangle_{\mathrm{R}} = \Pi_{\times}(\langle\vec{q}\rangle_0, \langle\vec{g}(\frac{i-1}{r})\rangle_0; \langle\vec{q}\rangle_1, \langle\vec{g}(\frac{i-1}{r})\rangle_1)$
10: $\quad P_0$: Compute $\langle\vec{\delta}_{\mathrm{ot}}\rangle_0 = \langle\vec{o}\rangle_0 - \langle\vec{t}\rangle_0$
11: $\quad P_1$: Compute $\langle\vec{\delta}_{\mathrm{ot}}\rangle_1 = \langle\vec{o}\rangle_1 - \langle\vec{t}\rangle_1$
12: $\quad P_0$: Compute $\langle\vec{g}(\frac{i}{r})\rangle_0 = \langle\vec{g}(\frac{i-1}{r})\rangle_0 + \langle\vec{\delta}_{\mathrm{ot}}\rangle_0$
13: $\quad P_1$: Compute $\langle\vec{g}(\frac{i}{r})\rangle_1 = \langle\vec{g}(\frac{i-1}{r})\rangle_1 + \langle\vec{\delta}_{\mathrm{ot}}\rangle_1$
14: **end for**
15: P_0: $\langle \mathrm{QSMax}(\vec{x})\rangle_0 = \langle\vec{g}(1)\rangle_0$
16: P_1: $\langle \mathrm{QSMax}(\vec{x})\rangle_1 = \langle\vec{g}(1)\rangle_1$

在Quasi-Softmax近似计算算法的基础上，本文提出了高效的安全计算协议Π_{QSMax}。如上图所示，该协议只需要加法和乘法，且该协议之需要在第6步和第9步计算两次乘法，其余部分都是本地加法计算。假设$\vec{x}$是维度为m的向量，每个元素是n比特的，r是迭代计算次数，那么协议Π_{QSMax}只需要$6mnr + 2nr$比特的在线通信量和$2mnr$的预计算通信量，通信轮数为$2r$。

更多关于近似计算方面的分析请参考原文。

3. Sigmoid的安全计算

之前的Sigmoid近似计算算法在效率和精确率等方面存在薄弱，本文提出了一种利用傅立叶级数精确近似的方法。同时，级数计算中的sine函数可以用MPC—Friendly的方式高效计算，因此本文方法非常高效。

3.1 Local Sigmoid计算

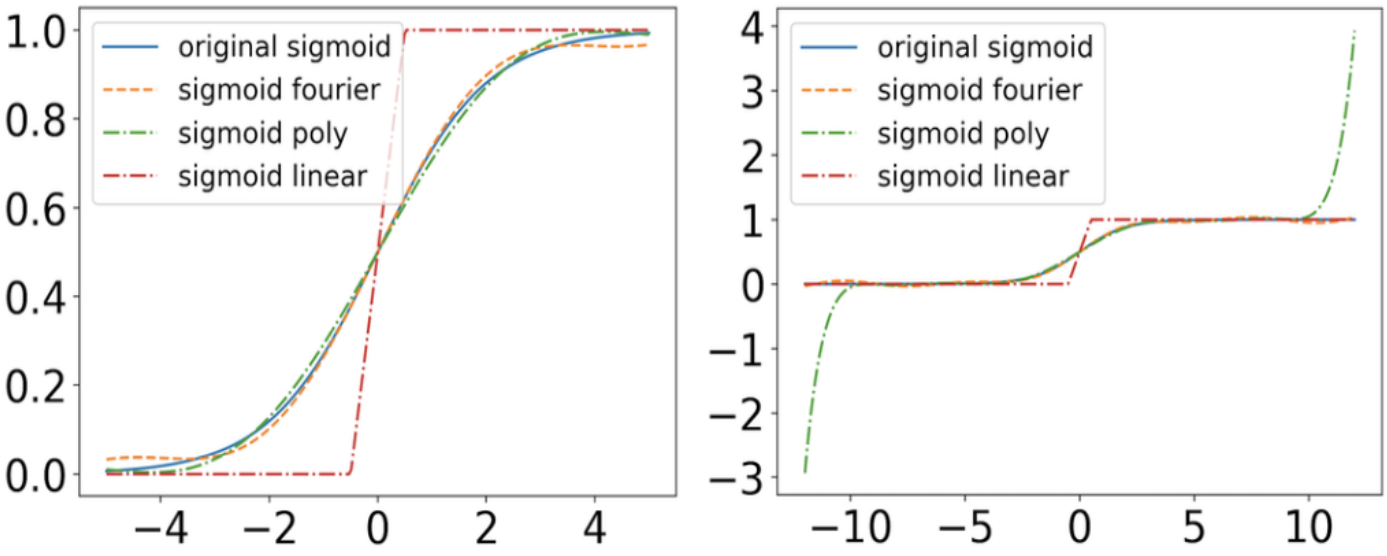
首先，本文提出了基于6项傅立叶级数的近似计算的公式：

$$\begin{aligned}\mathrm{LSig}(x) = \alpha &+ \beta_1 sin(2\pi x/2^m) + \beta_2 sin(4\pi x/2^m) \\ &+ \beta_3 sin(6\pi x/2^m) + \beta_4 + \beta_4 sin(8\pi x/2^m) \\ &+ \beta_5 sin(10\pi x/2^m)\end{aligned}$$

其中，系数配置如下：

$$\alpha = 0.5, \vec{\beta} = [0.61727893, -0.03416704, 0.16933091, -0.04596946, 0.08159136], m = 5$$

如下图所示，和之前的近似方法相比，本文的方案更加精确，且对于较大范围的输入区间也能精确近似。



(a) Comparison in small range (b) Comparison in large range

Figure 1: Comparison for different approximations

3.2 Local Sigmoid的安全计算协议

直观上来看，上述近似方法需要计算sin函数，在MPC下计算sin貌似没有高效计算的方法。但是，记秘密分享$x = \delta_x + t$，本文利用了如下公式：

$$sin(x\vec{k}) = sin(\delta_x\vec{k} + t\vec{k}) = sin(\delta_x\vec{k}) * cos(t\vec{k}) + cos(\delta_x\vec{k})sin(t\vec{k})$$

因为P_0和P_1分别持有δ_x和t，上述计算中的各个项可以在两方各自本地计算完成。

基于上述公式，安全计算协议构造如下：

Protocol 2 Π_{LSig}: Local-Sigmoid via Fourier Series

P_0 Input	$\langle x\rangle_0, \mathsf{key}_0$	P_0 Output	$\langle \mathsf{LSig}(x)\rangle_0$
P_1 Input	$\langle x\rangle_1, \mathsf{key}_1$	P_1 Output	$\langle \mathsf{LSig}(x)\rangle_1$

1: *Offline Phase*
2: P_0, T: $\langle t\rangle_0, \langle \vec{u}\rangle_0, \langle \vec{v}\rangle_0 \leftarrow \mathsf{PRF}_0(\mathsf{key}_0)$
3: P_1, T: $\langle t\rangle_1 \leftarrow \mathsf{PRF}_1(\mathsf{key}_1)$
4: T: Compute $t = \langle t\rangle_0 + \langle t\rangle_1$, $\langle \vec{u}\rangle_1 = \mathsf{FR}(\sin(t\vec{k})) - \langle \vec{u}\rangle_0$, $\langle \vec{v}\rangle_1 = \mathsf{FR}(\cos(t\vec{k})) - \langle \vec{v}\rangle_0$
5: T: Send $\langle \vec{u}\rangle_1, \langle \vec{v}\rangle_1$ to P_1
6: *Online Phase*
7: P_0: Compute $\delta_{\langle x\rangle_0} = \langle x\rangle_0 - \langle t\rangle_0 \mod 32$
8: P_1: Compute $\delta_{\langle x\rangle_1} = \langle x\rangle_1 - \langle t\rangle_1 \mod 32$
9: P_0: Send $\delta_{\langle x\rangle_0}$ to P_1 ▷ 5 bits for integral part
10: P_1: Send $\delta_{\langle x\rangle_1}$ to P_0
11: $\mathsf{P}_0, \mathsf{P}_1$: Compute $\delta_x = \delta_{\langle x\rangle_0} + \delta_{\langle x\rangle_1}$
12: $\mathsf{P}_0, \mathsf{P}_1$: Compute $\vec{p} = \mathsf{FR}(\sin(\delta_x\vec{k}))$ and $\vec{q} = \mathsf{FR}(\cos(\delta_x\vec{k}))$
13: P_0: $\langle \mathsf{LSig}(x)\rangle_0 = \alpha + \vec{\beta}(\vec{p} * \langle \vec{v}\rangle_0 + \vec{q} * \langle \vec{u}\rangle_0)$
14: P_1: $\langle \mathsf{LSig}(x)\rangle_1 = \alpha + \vec{\beta}(\vec{p} * \langle \vec{v}\rangle_1 + \vec{q} * \langle \vec{u}\rangle_1)$

可以看到该协议只需要在第9、10步进行一轮双向通信。令x是n比特整数，其中p比特表示小数部分，$g(x) = a_0 + \sum_{k=1}^{K} sin\frac{2k\pi x}{2^m}$，那么调用协议$\Pi_{\mathrm{LSig}}$计算$g(x)$只需要一轮在线通信，在线通信量为

$2(m+p)$比特，预计算通信量为$2Kn$比特。

本文的方法和Squirrel的方法有些类似，不过Squirrel只是用傅立叶级数近似了Sigmoid计算中的一段，因此比本文的方法需要更多的通信量和通信轮数。更多关于本文的正确性和近似精度分析请参考原文。

4. 实验结果

本文的代码已经在Github开源。本文进行了大量的实验，包括对Softmax、Sigmoid计算的通信量测量，安全训练中的通信和时间开销等。这里只简单列几个实验结果展示方法的效果，更多的实验数据和分析请参考原文。

Table 2: Communication Comparison for Softmax

Protocol	Offline (bit)	Online (bit)	Overall (bit)	Round
$(m = 10)$				
ASM	-	-	3017195	704
ComDiExp	198912	783250	982162	171
$\Pi_{\text{QSMax}}(r = 8)$	10240	31744	41984	16
$\Pi_{\text{QSMax}}(r = 16)$	20480	63488	83968	32
$\Pi_{\text{QSMax}}(r = 32)$	40960	126976	167936	64
$\Pi_{\text{QSMax}}(r = 64)$	81920	253952	335872	128
$(m = 100)$				
ASM	-	-	30171944	704
ComDiExp	2174592	8536390	10710982	300
$\Pi_{\text{QSMax}}(r = 8)$	102400	308224	410624	16
$\Pi_{\text{QSMax}}(r = 16)$	204800	616448	821248	32
$\Pi_{\text{QSMax}}(r = 32)$	409600	1232896	1642496	64
$\Pi_{\text{QSMax}}(r = 64)$	819200	2465792	3284992	128
$(m = 1000)$				
ASM	-	-	301719448	704
ComDiExp	21931392	86067790	107999182	430
$\Pi_{\text{QSMax}}(r = 8)$	1024000	3073024	4097024	16
$\Pi_{\text{QSMax}}(r = 16)$	2048000	6146048	8194048	32
$\Pi_{\text{QSMax}}(r = 32)$	4096000	12292096	16388096	64
$\Pi_{\text{QSMax}}(r = 64)$	8192000	24584192	32776192	128

Table 3: Communication Comparison for Sigmoid

Protocol	Offline (bit)	Online (bit)	Overall (bit)	Rd
Piece. Linear Approx.	-	≈ 800	-	5
Poly. Approx. ($K = 5$)	320	1280	1600	1
Poly. Approx. ($K = 8$)	512	2048	2560	1
Squirrel ($K = 5$)	-	2560 + 1792	-	3
Squirrel ($K = 8$)	-	4096 + 1792	-	3
LLAMA	-	128	-	1
Π_{LSig} ($m = 4, K = 5$)	640	36	676	1
Π_{LSig} ($m = 4, K = 8$)	1024	36	1060	1
Π_{LSig} ($m = 5, K = 5$)	640	38	678	1
Π_{LSig} ($m = 5, K = 8$)	1024	38	1062	1

Table 7: Communication for Large Models (GB/batch)

Framework	AlexNet (CIFAR10)		VGG16 (CIFAR10)	
CryptGPU[$]	1.37	[↓ 57.7%]	7.55	[↓ 61.3%]
Falcon[$]	0.62×3	[↓ 68.8%]	1.78×3	[↓ 45.3%]
Piranha[$]	0.58×3	[↓ 66.7%]	4.26×3	[↓ 77.2%]
Ours	0.39 ($b = 5$), 0.58 ($b = 7$)		2.925 (batch of 2^5)	

[$] From Table 4 in [35], which reports per-party communication.
We use the same pooling and free truncation as in Piranha for a fair comparison.