Annual Computer Security Applications Conference 2023

Secure Softmax/Sigmoid for Machine-learning Computation

Yu Zheng*, Qizhi Zhang*, Sherman S. M. Chow
Yuxiang Peng, Sijun Tan, Lichun Li, Shan Yin

香港中文大學 The Chinese University of Hong Kong
蚂蚁集团 ANT GROUP
Berkeley UNIVERSITY OF CALIFORNIA
NORTHEASTERN UNIVERSITY 1923
ByteDance

Rundown

- <u>Secure</u> Machine Learning Background
  - Secret share: 2 computing parties  + 1 commodity server 
  - Against semi-honest adversary

Rundown

- Secure Machine Learning Background
  - Secret share: 2 computing parties  + 1 commodity server 
  - Against semi-honest adversary
- Non-Linearity Challenges and Sigmoid/Softmax in Crypto
- **New Protocols** for Nonlinear Functions
  - Local-sigmoid via Fourier series
  - Quasi-softmax via ordinary differential equation

Rundown

- Secure Machine Learning Background
  - Secret share: 2 computing parties  + 1 commodity server 
  - Against semi-honest adversary
- Non-Linearity Challenges and Sigmoid/Softmax in Crypto
- **New Protocols** for Nonlinear Functions
  - Local-sigmoid via Fourier series
  - Quasi-softmax via ordinary differential equation
- Experiments and System Performance
- Conclusion

Secure Machine Learning

- Machine learning attains great performance
- Privacy concerns over sensitive data, e.g., health, finance.

Secure Machine Learning

- Machine learning attains great performance
- Privacy concerns over sensitive data, e.g., health, finance.
- Most SML frameworks support simpler inference tasks

LLAMA [PoPets'22], GForce [Usenix Sec'21], SiRNN [S&P'21], CryptFlow2 [CCS'20], etc.

Secure Machine Learning

- Machine learning attains great performance
- Privacy concerns over sensitive data, e.g., health, finance.
- Most SML frameworks support simpler inference tasks

LLAMA [PoPets'22], GForce [Usenix Sec'21], SiRNN [S&P'21], CryptFlow2 [CCS'20], etc.

- Training is more complicated to do with cryptography
  - It produces fluctuating computation results.
  - It requires non-linear computation such as those in activation layers.

Crypto. Challenges in Secure Training

- Crypto. excels primarily with *finite fields* and *linear functions*.
  - **Accuracy**: *expand* finite field to cater to fluctuating ranges.
- But, increase computational & **communication** overheads.
  - Secure protocols for exact computation of non-linearity are known to be heavyweight.

Crypto. Challenges in Secure Training

- Crypto. excels primarily with *finite fields* and *linear functions*.
  - **Accuracy**: *expand* finite field to cater to fluctuating ranges.
- But, increase computational & **communication** overheads.
  - Secure protocols for exact computation of non-linearity are known to be heavyweight.
- Not until recently, start to have secure training frameworks.
  - CrypTen [NeurIPS'21], CryptGPU [S&P'21], Piranha [Usenix Sec'22], etc.
  - Support more complex activation, including softmax and sigmoid.
  - Achieve high computational performance over AlexNet (60M param) and VGG-16 (138M param) .

Communication Bottleneck

- However, large communication overhead persists as a major concern.
  - Prominently, Piranha, a GPU platform for secure computation, reports **94%+** of the training time consumed by communication in a wide-area network (WAN) setting.

Communication Bottleneck

- However, large communication overhead persists as a major concern.
  - Prominently, Piranha, a GPU platform for secure computation, reports **94%+** of the training time consumed by communication in a wide-area network (WAN) setting.
- Informally, sigmoid/softmax combine e^x , $1/x$, or Σe^x.
  - e^x and $1/x$ are unbounded and continuous.

Communication Bottleneck

- However, large communication overhead persists as a major concern.
  - Prominently, Piranha, a GPU platform for secure computation, reports **94%+** of the training time consumed by communication in a wide-area network (WAN) setting.
- Informally, sigmoid/softmax combine e^x , $1/x$, or Σe^x.
  - e^x and $1/x$ are unbounded and continuous.
- Securely computing them with efficiency is challenging.
  - Existing works either separately approximate or replace them.
  - Below, we detail secure sigmoid and softmax one by one.

State-of-The-Art Secure Sigmoid

- **Sigmoid(x) =** $e^x/(e^x + 1)$: (-inf, +inf) $\mapsto$ **[0, 1].**
  - Binary classification.
  - It squashes any input to a value in (0,1).

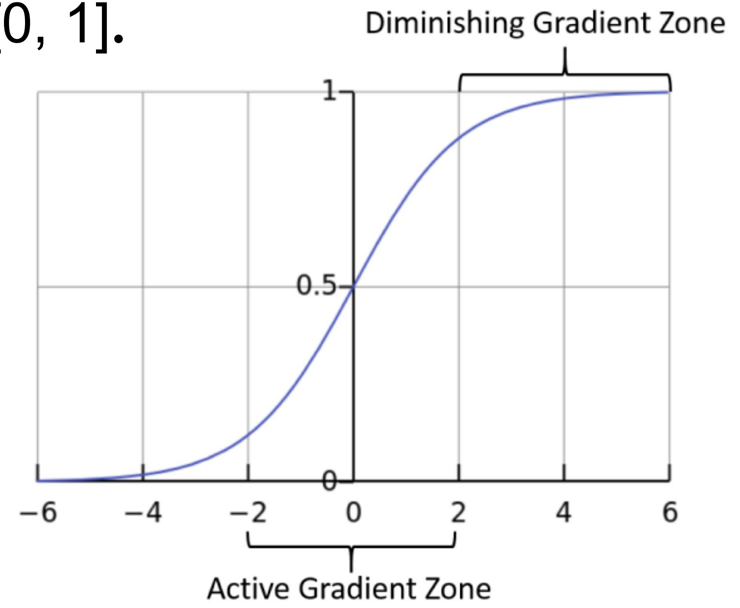
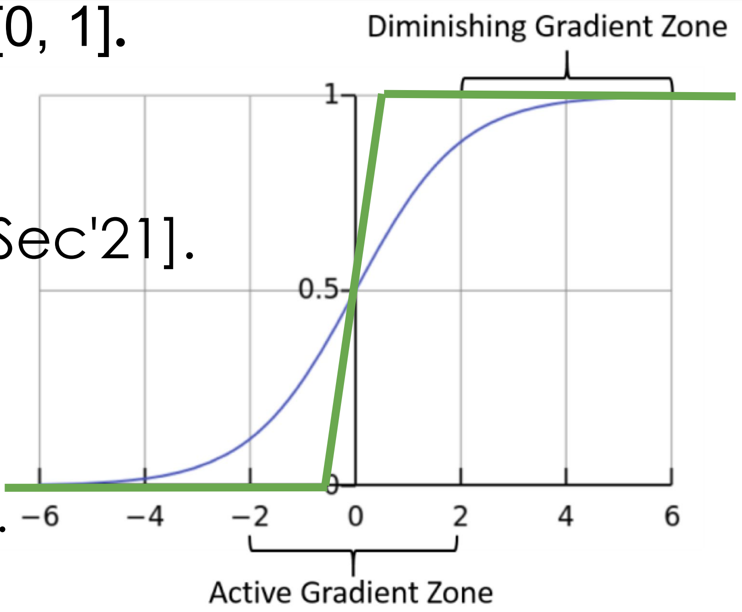


*Figure is from Google image.

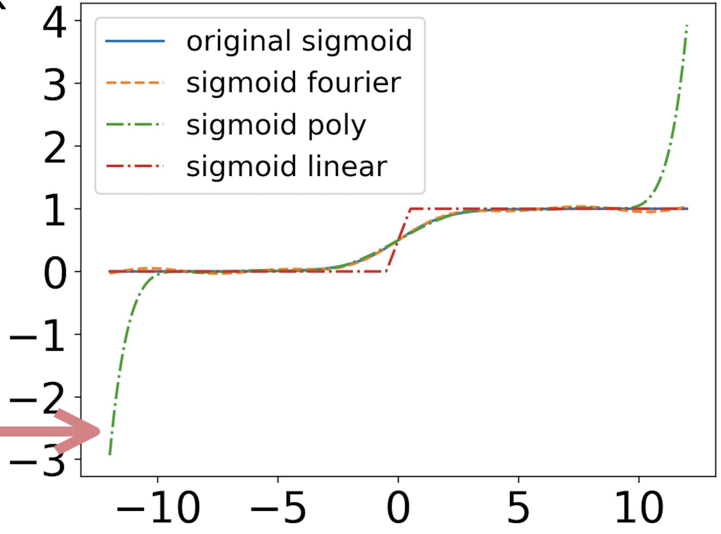
State-of-The-Art Secure Sigmoid

- **Sigmoid(**x**) =** $e^x/(e^x + 1)$: (-inf, +inf) ↦ [0, 1].
  - Binary classification.
  - It squashes any input to a value in (0,1).
- ABY-series protocols [CCS'18, Usenix Sec'21].
  - Piecewise linear approx.
  - x + 0.5 for |x| < 0.5; x = 0 or x = 1.
  - Requires comparison to identify pieces.
  - Relatively inaccurate approx. near to 0.



State-of-The-Art Secure Sigmoid

- **Sigmoid**(x) **=** $e^x/(e^x + 1)$: (-inf, +inf) ↦ [0, 1].
  - It squashes any input to a value in (0,1).
- ABY-series protocols [CCS'18, Usenix Sec'21]
  - Piecewise linear approx.
- Chebyshev polynomial
  - [CCS'21 Workshop]
  - Linear to #terms
  - Possibly result in gradient explosion



State-of-The-Art Secure Sigmoid

- **Sigmoid(**x**) =** $e^x/(e^x + 1) : (-\inf, +\inf) \mapsto [0, 1]$.
  - It squashes any input to a value in (0,1).
- ABY-series protocols [CCS'18, Usenix Sec'21].
  - Piecewise linear approx.
- Chebyshev polynomial [CCS'21 workshop]
- How about their communication costs?

Protocol	Offline (bits)	Online (bits)	Overall (bits)	Round
ABY-series	-	~800	-	5
Polyn. (5,8)	320 ~ 512	1280 ~ 2048	1600 ~ 2560	1

State-of-The-Art Secure Sigmoid

- **Sigmoid(x) =** $e^x/(e^x + 1)$: (-inf, +inf) ↦ **[0, 1].**
  - It squashes any input to a value in (0,1).
- ABY-series protocols [CCS'18, Usenix Sec'21].
  - Piec
- Cheby
- How a

Can we expect **<40 bits** online in **1** round?

Protocol	Offline (bits)	Online (bits)	Overall (bits)	Round
ABY-series	-	~800	-	5
Polyn. (5,8)	320 ~ 512	1280 ~ 2048	1600 ~ 2560	1

New Sigmoid Approx. and Protocol

- Local-Sigmoid definition.
  - Sigmoid in [-*a*, *a*].
  - High accuracy in range.
  - Bounded error out of range

Sigmoid Function and Its Derivative

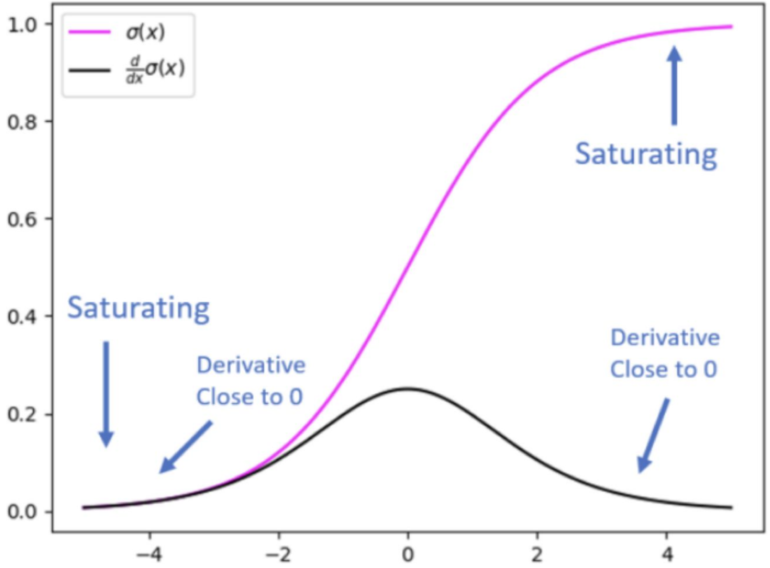
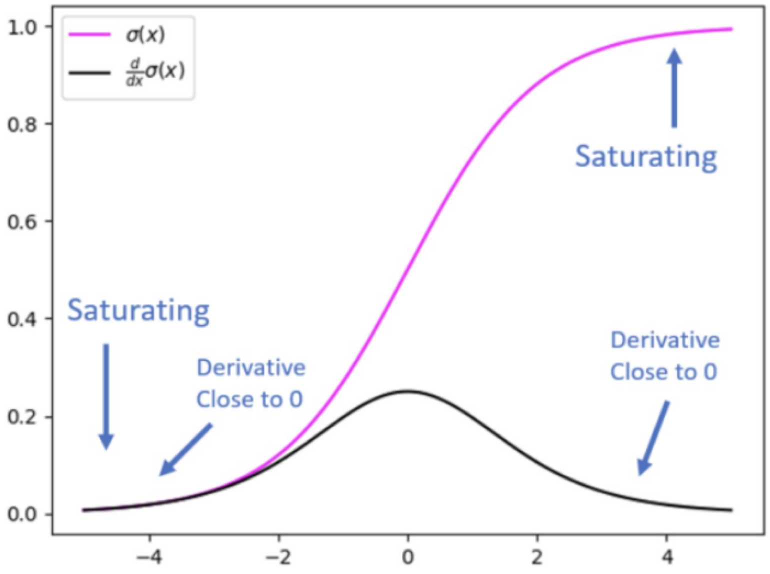


*Figure is from Google image.

New Sigmoid Approx. and Protocol

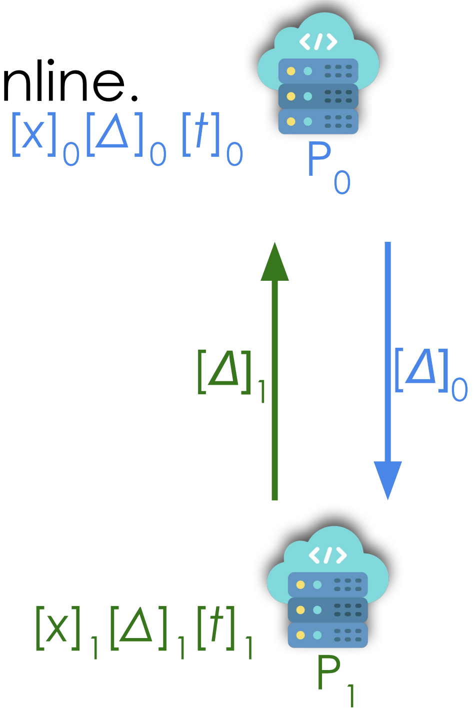
- Local-Sigmoid definition.
  - Sigmoid in [-*a*, *a*].
  - High accuracy in range.
  - Bounded error out of range
- Fourier approximation.
  - LSig(x) = $a + \boldsymbol{b}\sin(x\boldsymbol{k})$.
  - Mask t, shared value $\Delta = x - t$.
  - No secure comparison is required.
  - $\sin(\Delta + t) = \sin(\Delta)\cos(t) + \cos(\Delta)\sin(t)$

Sigmoid Function and Its Derivative



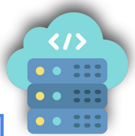
New Sigmoid Approx. and Protocol

- Jointly compute LSig(x) = $a + \boldsymbol{b}\sin((\Delta + t)\boldsymbol{k})$ online.
  - Public parameters a, $\boldsymbol{b}$, $\boldsymbol{k}$.
  - P_0 holds $[\Delta]_0 = [x]_0 - [t]_0$; P_1 holds $[\Delta]_1 = [x]_1 - [t]_1$.
  - P_0 sends $[\Delta]_0$; P_1 sends $[\Delta]_1$. [1 round]



New Sigmoid Approx. and Protocol

- Jointly compute LSig(x) = $a + \boldsymbol{b}\sin((\Delta + t)\boldsymbol{k})$ online.
  - Public parameters a, $\boldsymbol{b}$, $\boldsymbol{k}$.
  - P_0 holds $[\Delta]_0 = [x]_0 - [t]_0$; P_1 holds $[\Delta]_1 = [x]_1 - [t]_1$.
  - P_0 sends $[\Delta]_0$; P_1 sends $[\Delta]_1$. [1 round]
  - P_0 and P_1 compute $\Delta = [\Delta]_0 + [\Delta]_1$.



$[x]_0$ Δ $[t]_0$ P_0

$[x]_1$ Δ $[t]_1$ P_1

New Sigmoid Approx. and Protocol

- Jointly compute LSig(x) = $a + \boldsymbol{b}\sin((\Delta + t)\boldsymbol{k})$ online.
  - Public parameters a, $\boldsymbol{b}$, $\boldsymbol{k}$.
  - P_0 holds $[\Delta]_0 = [x]_0 - [t]_0$; P_1 holds $[\Delta]_1 = [x]_1 - [t]_1$
  - P_0 sends $[\Delta]_0$; P_1 sends $[\Delta]_1$. [1 round]
  - P_0 and P_1 compute $\Delta = [\Delta]_0 + [\Delta]_1$.
  - P_0 and P_1 locally compute $\sin(\Delta\boldsymbol{k})$, $\cos(\Delta\boldsymbol{k})$

$[x]_0$ Δ $[t]_0$ P_0

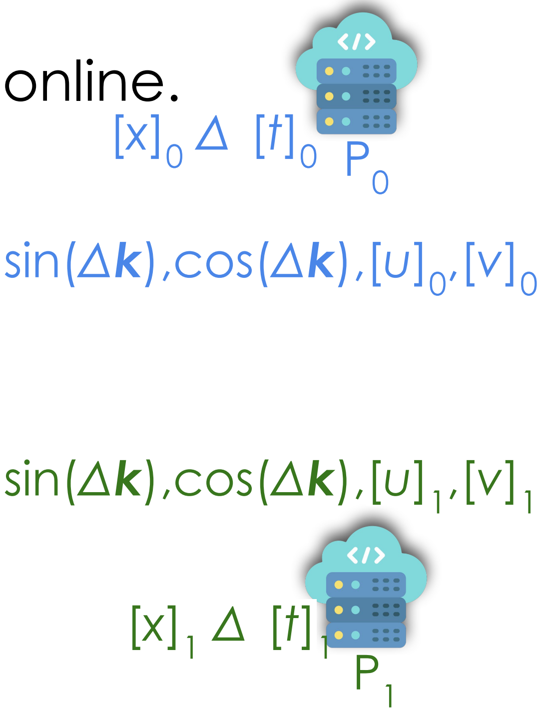
$\sin(\Delta\boldsymbol{k}), \cos(\Delta\boldsymbol{k})$

$\sin(\Delta\boldsymbol{k}), \cos(\Delta\boldsymbol{k})$

$[x]_1$ Δ $[t]_1$ P_1

New Sigmoid Approx. and Protocol

- Jointly compute LSig(x) = $a + \boldsymbol{b}\sin((\Delta + t)\boldsymbol{k})$ online.
  - Public parameters a, $\boldsymbol{b}$, $\boldsymbol{k}$.
  - P_0 holds $[\Delta]_0 = [x]_0 - [t]_0$; P_1 holds $[\Delta]_1 = [x]_1 - [t]_1$
  - P_0 sends $[\Delta]_0$; P_1 sends $[\Delta]_1$. [1 round]
  - P_0 and P_1 compute $\Delta = [\Delta]_0 + [\Delta]_1$.
  - P_0 and P_1 locally compute $\sin(\Delta\boldsymbol{k})$, $\cos(\Delta\boldsymbol{k})$.
- Secret-shared outputs
  - P_0 gets $a + \boldsymbol{b}(\sin(\Delta\boldsymbol{k})[v]_0 + \cos(\Delta\boldsymbol{k})[u]_0)$.
  - P_1 gets $a + \boldsymbol{b}(\sin(\Delta\boldsymbol{k})[v]_1 + \cos(\Delta\boldsymbol{k})[u]_1)$.



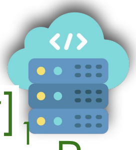
New Sigmoid Approx. and Protocol

- Jointly compute LSig(x) = $a + \boldsymbol{b}\sin((\Delta + t)\boldsymbol{k})$ online.
  - Public parameters a, $\boldsymbol{b}$, $\boldsymbol{k}$.
  - P_0 holds $[\Delta]_0 = [x]_0 - [t]_0$; P_1 holds $[\Delta]_1 = [x]_1 - [t]_1$
  - P_0 sends $[\Delta]_0$; P_1 sends $[\Delta]_1$. [1 round]
  - P_0 and P_1 compute $\Delta = [\Delta]_0 + [\Delta]_1$.
  - P_0 and P_1 locally compute $\sin(\Delta\boldsymbol{k})$, $\cos(\Delta\boldsymbol{k})$.
- Secret-shared outputs
  - P_0 gets $a + \boldsymbol{b}(\sin(\Delta\boldsymbol{k})[v]_0 + \cos(\Delta\boldsymbol{k})[u]_0)$.
  - P_1 gets $a + \boldsymbol{b}(\sin(\Delta\boldsymbol{k})[v]_1 + \cos(\Delta\boldsymbol{k})[u]_1)$.
- What are $[u]_0, [v]_0, [t]_0$ and $[u]_1, [v]_1, [t]_1$?

$[x]_0$ Δ $[t]_0$ P_0

$\sin(\Delta\boldsymbol{k}), \cos(\Delta\boldsymbol{k}), [u]_0, [v]_0$

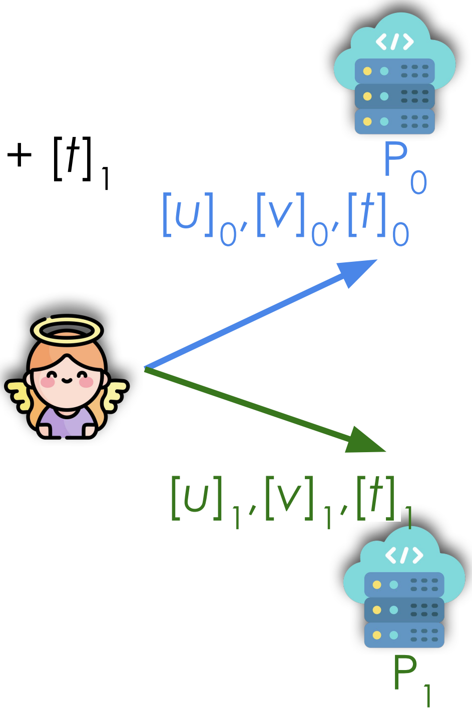
$\sin(\Delta\boldsymbol{k}), \cos(\Delta\boldsymbol{k}), [u]_1, [v]_1$



$[x]_1$ Δ $[t]_1$ P_1

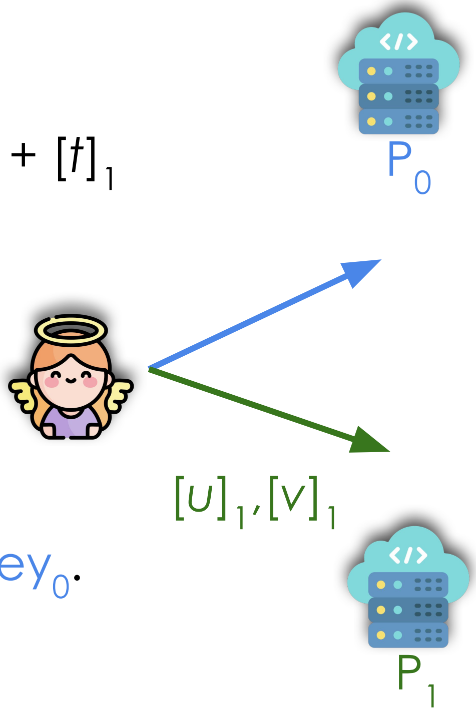
New Sigmoid Approx. and Protocol

- What are $[u]_0, [v]_0, [t]_0$ and $[u]_1, [v]_1, [t]_1$?
  - $\sin(t\mathbf{k}) = [u]_0 + [u]_1$; $\cos(t\mathbf{k}) = [v]_0 + [v]_1$; $t = [t]_0 + [t]_1$
  - Randomness independent to private x.
  - Generated by a crypto commodity server.
  - In a pre-computation phase offline.



New Sigmoid Approx. and Protocol

- What are $[u]_0,[v]_0,[t]_0$ and $[u]_1,[v]_1,[t]_1$?
  - $\sin(t\mathbf{k}) = [u]_0 + [u]_1$; $\cos(t\mathbf{k}) = [v]_0 + [v]_1$; $t = [t]_0 + [t]_1$
  - Randomness independent to private x.
  - Generated by a crypto commodity server.
  - In a pre-computation phase offline.
- Optimize offline communication?
  - Use PRF with a synchronized counter.
  - &P_0 generate $[u]_0,[v]_0,[t]_0$ using the same key_0.
  - &P_1 generate $[t]_1$ using the same key_1.
  - computes and sends $[u]_1,[v]_1$ to P_1.



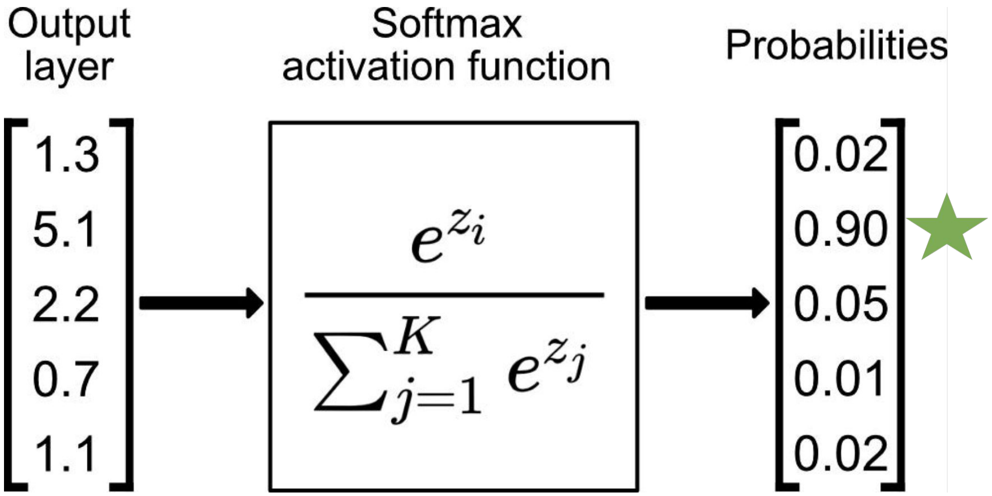
Communication Costs

- Achieved **< 40bits** online in **1** round!

Protocol	Offline (bits)	Online (bits)	Overall (bits)	Round
ABY-series	-	∼800	-	5
Polyn. (K=5,8)	320 ∼ 512	1280 ∼ 2048	1600 ∼ 2560	1
Ours (m=4, K=5)	640	**36**	676	**1**
Ours (m=4, K=8)	1024	**36**	1060	**1**
Ours (m=5, K=5)	640	**38**	678	**1**
Ours (m=5, K=8)	1024	**38**	1062	**1**

State-of-The-Art Secure Softmax

- **Softmax(**x**)** = $e^{z\_i}/(\Sigma e^{z\_j})$: $R^m \mapsto [0, 1]^m$.
  - It squashes any input vector to a probability vector.
  - Multi-classification.



State-of-The-Art Secure Softmax

- **Softmax(**x**)** = $e^{z\_i}/(\Sigma e^{z\_j})$: $R^m \mapsto [0, 1]^m$.
  - It squashes any input vector to a probability vector.
- Crypten [NeurIPS'22] follows exact computation.
  - It requires secure maximum, exponentiation, and division.
  - Secure maximum takes O(log m) rounds for removing the largest input and mitigating overflow of e^{z_i} and Σe^{z_j}.

State-of-The-Art Secure Softmax

- **Softmax(**x**)** = $e^{z\_i}/(\Sigma e^{z\_j})$: $R^m \mapsto [0, 1]^m$.
  - It squashes any input vector to a probability vector.
- Crypten [NeurIPS'22] follows exact computation.
  - It requires secure maximum, exponentiation, and division.
  - Secure maximum takes O(log m) rounds for removing the largest input and mitigating overflow of e^{z_i} and Σe^{z_j}.
- ASM protocol replaces exponential function with ReLU.
  - It is adopted in SecureNN [PoPETS'19], Falcon [PoPETS'21].
  - It relies on manual efforts in tuning the model [Keller and Sun, ICML'22].

State-of-The-Art Secure Softmax

- How about their communication costs?

Protocol	#Class	Online (bits)	Overall (bits)	Round
ASM protocol	10	-	3M	704
ASM protocol	100	-	30M	704
ASM protocol	1000	-	302M	704
Crypten	10	783250	982K	171
Crypten	100	8536390	11M	300
Crypten	1000	86067790	108M	430

State-of-The-Art Secure Softmax

- How a

Can we expect **<10% communication costs** in **<35** rounds?

Protoco				nd
ASM protocol	10	-		704
ASM protocol	100	-	30M	704
ASM protocol	1000	-	302M	704
Crypten	10	783250	982K	171
Crypten	100	8536390	11M	300
Crypten	1000	86067790	108M	430

New Softmax Approx. and Protocol

- Formulate Quasi-Softmax (QSMax) capturing probability distribution of softmax's outputs.

New Softmax Approx. and Protocol

- Formulate Quasi-Softmax (QSMax) capturing probability distribution of softmax's outputs.
- For an input vector $\mathbf{x}$ and iteration step r, we instantiate a vector function QSMax $\mathbf{g}()$, which is an ordinary differential equation solved by Euler formula.
  - $\mathbf{g}(0) = [1/m, \ldots, 1/m]$

    for $i = 1, \ldots, r$ **do**

    $$\mathbf{g}(i/r) = \mathbf{g}((i-1)/r) + (\mathbf{x} - \langle \mathbf{x},\mathbf{g}((i-1)/r)\rangle \mathbf{1}) * \mathbf{g}((i-1)/r)/r$$
- $\mathbf{g}(r/r) = \mathbf{g}(1)$, *iteratively limits to real softmax.*

New Softmax Approx. and Protocol

- Formulate Quasi-Softmax (QSMax) capturing probability distribution of softmax's outputs.
- For an input vector **x** and iteration step *r*, we instantiate a <u>vector</u> function QSMax ***g**()*, which is an ordinary differential equation solved by Euler formula.
  - ***g**(0) = [1/m, ... , 1/m]*

    for *i = 1, ..., r* **do**

    $$\mathbf{g}(i/r) = \mathbf{g}((i-1)/r) + (\mathbf{x} - \langle \mathbf{x}, \mathbf{g}((i-1)/r) \rangle \mathbf{1}) * \mathbf{g}((i-1)/r)/r$$
- ***g**(r/r) = **g**(1), iteratively limits to real softmax.*
- Notably, loop function contains only <u>two multiplications</u> and additions.

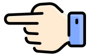
New Softmax Approx. and Protocol

- Formulate Quasi-Softmax (QSMax) capturing probability distribution of softmax's outputs.
- For an input vector **x** and iteration step *r*, we instantiate a vector function QSMax $\mathbf{g}()$, which is an ordinary differential equation solved by Euler formula.
  - $\mathbf{g}(0) = [1/m, \ldots, 1/m]$

    for $i = 1, \ldots, r$ **do**

    $$\mathbf{g}(i/r) = \mathbf{g}((i-1)/r) + (\mathbf{x} - \langle \mathbf{x}, \mathbf{g}((i-1)/r) \rangle \mathbf{1}) * \mathbf{g}((i-1)/r)/r$$
- $\mathbf{g}(r/r) = \mathbf{g}(1)$, *iteratively limits to real softmax.*
- Notably, loop function contains only two multiplications and additions.

New Softmax Approx. and Protocol

- Jointly compute QSMax(x) online.
  - $\mathbf{g}(0) = [1/m, \ldots, 1/m]$ 

    for $i = 1, \ldots, r$ **do**

    $$\mathbf{g}(i/r) = \mathbf{g}((i-1)/r) + (\mathbf{x} - \langle \mathbf{x}, \mathbf{g}((i-1)/r) \rangle \mathbf{1}) * \mathbf{g}((i-1)/r)/r$$

$[1/m, \ldots, 1/m]$ 

P_0

$[0, \ldots, 0]$ 

P_1

New Softmax Approx. and Protocol

- Jointly compute QSMax(x) online.
  - $\boldsymbol{g}(0) = [1/m, \ldots, 1/m]$

    for $i = 1, \ldots, r$ **do**

    $$\boldsymbol{g}(i/r) = \boldsymbol{g}((i-1)/r) + (\boldsymbol{x} - \langle \boldsymbol{x}, \boldsymbol{g}((i-1)/r) \rangle \mathbf{1}) * \boldsymbol{g}((i-1)/r)/r$$

$[1/m, \ldots, 1/m]$

$[\mathbf{x}]_0 = [\mathbf{x}]_0/r$

P_0

$[\mathbf{x}]_1 = [\mathbf{x}]_1/r$

$[0, \ldots, 0]$

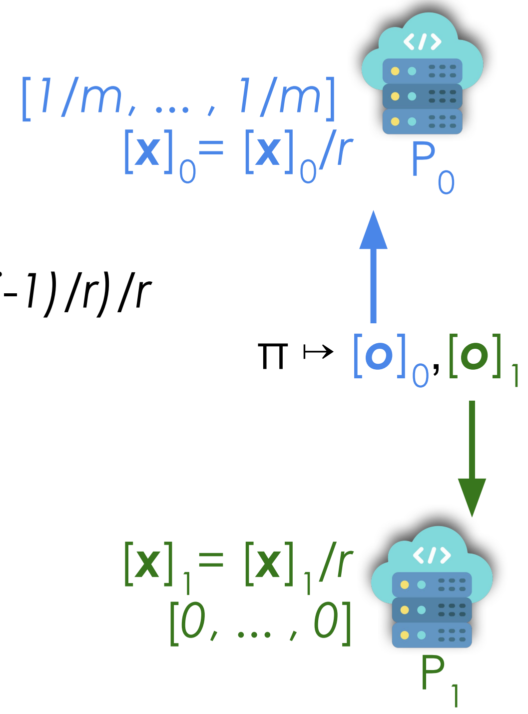
P_1

New Softmax Approx. and Protocol

- Jointly compute QSMax(x) online.
  - $\mathbf{g}(0) = [1/m, \ldots, 1/m]$

    for $i = 1, \ldots, r$ **do**

    $$\mathbf{g}(i/r) = \mathbf{g}((i-1)/r) + (\mathbf{x} - \langle \mathbf{x}, \mathbf{g}((i-1)/r) \rangle \mathbf{1}) * \mathbf{g}((i-1)/r)/r$$
- Jointly invoke secure multiplication π.
  - Inner product with *m* dimensions parallelly.

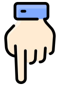


New Softmax Approx. and Protocol

- Jointly compute QSMax(x) online.
  - $\mathbf{g}(0) = [1/m, \ldots, 1/m]$

    for $i = 1, \ldots, r$ **do**

    $$\mathbf{g}(i/r) = \mathbf{g}((i-1)/r) + (\mathbf{x} - \langle \mathbf{x}, \mathbf{g}((i-1)/r) \rangle \mathbf{1}) * \mathbf{g}((i-1)/r)/r$$
- Jointly invoke secure multiplication π.
  - Inner product with m dimensions parallelly.



$[1/m, \ldots, 1/m]$

$[\mathbf{x}]_0 = [\mathbf{x}]_0/r$

$[\mathbf{q}]_0 = (\Sigma[\mathbf{p}_i]_0)\mathbf{1}$

P_0

$[\mathbf{q}]_1 = (\Sigma[\mathbf{p}_i]_1)\mathbf{1}$

$[\mathbf{x}]_1 = [\mathbf{x}]_1/r$

$[0, \ldots, 0]$

P_1

New Softmax Approx. and Protocol

- Jointly compute QSMax(x) online
  - $\mathbf{g}(0) = [1/m, \dots, 1/m]$

    for $i = 1, \dots, r$ **do**

    $$\mathbf{g}(i/r) = \mathbf{g}((i-1)/r) + (\mathbf{x} - \langle \mathbf{x}, \mathbf{g}((i-1)/r) \rangle \mathbf{1}) * \mathbf{g}((i-1)/r)/r$$
- Jointly invoke secure multiplication π.
  - Inner product with *m* dimensions parallelly.

$[1/m, \dots, 1/m]$

$[\mathbf{x}]_0 = [\mathbf{x}]_0/r$

$[\mathbf{w}]_0 = [\mathbf{x}]_0 - [\mathbf{q}]_0$

P_0

$[\mathbf{w}]_1 = [\mathbf{x}]_1 - [\mathbf{q}]_1$

$[\mathbf{x}]_1 = [\mathbf{x}]_1/r$

$[0, \dots, 0]$

P_1

New Softmax Approx. and Protocol

- Jointly compute QSMax(x) online
  - $\mathbf{g}(0) = [1/m, \ldots, 1/m]$

    for $i = 1, \ldots, r$ **do**

    $$\mathbf{g}(i/r) = \mathbf{g}((i-1)/r) + (\mathbf{x} - \langle \mathbf{x}, \mathbf{g}((i-1)/r)\rangle \mathbf{1}) * \mathbf{g}((i-1)/r)/r$$
- Jointly invoke secure multiplication π.
  - Inner product with *m* dimensions parallelly.
  - Element-wise product with *m* dimensions.

$[1/m, \ldots, 1/m]$

$[\mathbf{x}]_0 = [\mathbf{x}]_0/r$

$[\mathbf{w}]_0 = [\mathbf{x}]_0 - [\mathbf{q}]_0$

P_0

$\pi \mapsto [\mathbf{t}]_0, [\mathbf{t}]_1$

$[\mathbf{w}]_1 = [\mathbf{x}]_1 - [\mathbf{q}]_1$

$[\mathbf{x}]_1 = [\mathbf{x}]_1/r$

$[0, \ldots, 0]$

P_1

New Softmax Approx. and Protocol

- Jointly compute QSMax(x) online
  - $\mathbf{g}(0) = [1/m, \dots, 1/m]$

    for $i = 1, \dots, r$ **do**

    $\mathbf{g}(i/r) = \mathbf{g}((i-1)/r) + (\mathbf{x} - <\mathbf{x},\mathbf{g}((i-1)/r)>\mathbf{1})*\mathbf{g}((i-1)/r)/r$
- Jointly invoke secure multiplication π.
  - Inner product with *m* dimensions parallelly.
  - Element-wise product with *m* dimensions.

$[1/m, \dots, 1/m]$

$[\mathbf{x}]_0 = [\mathbf{x}]_0/r$

P_0

$[\mathbf{p}]_0 = [\mathbf{x}]_0 - [\mathbf{q}]_0$

$[\mathbf{g}((i-1)/r)]_0 + [\mathbf{t}]_0$

$[\mathbf{g}((i-1)/r)]_1 + [\mathbf{t}]_1$

$[\mathbf{p}]_1 = [\mathbf{x}]_1 - [\mathbf{q}]_1$

$[\mathbf{x}]_1 = [\mathbf{x}]_1/r$

$[0, \dots, 0]$

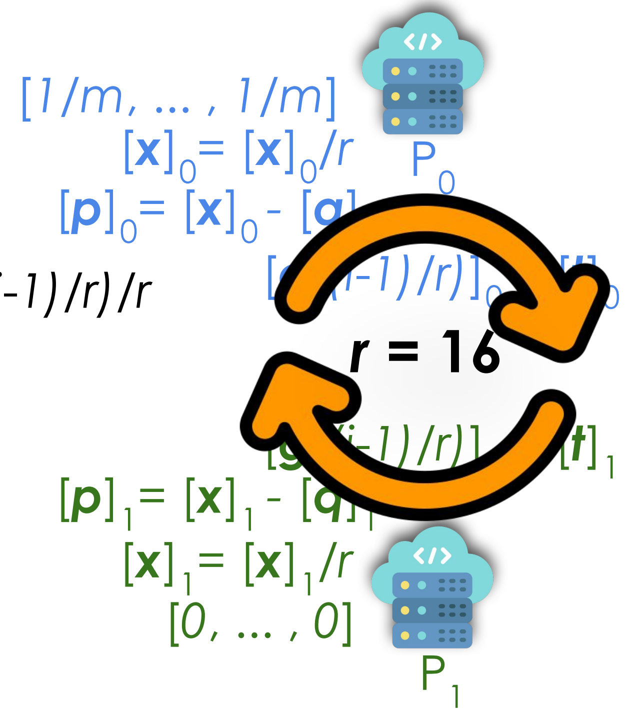
P_1

New Softmax Approx. and Protocol

- Jointly compute QSMax(x) online
  - $\mathbf{g}(0) = [1/m, \ldots, 1/m]$

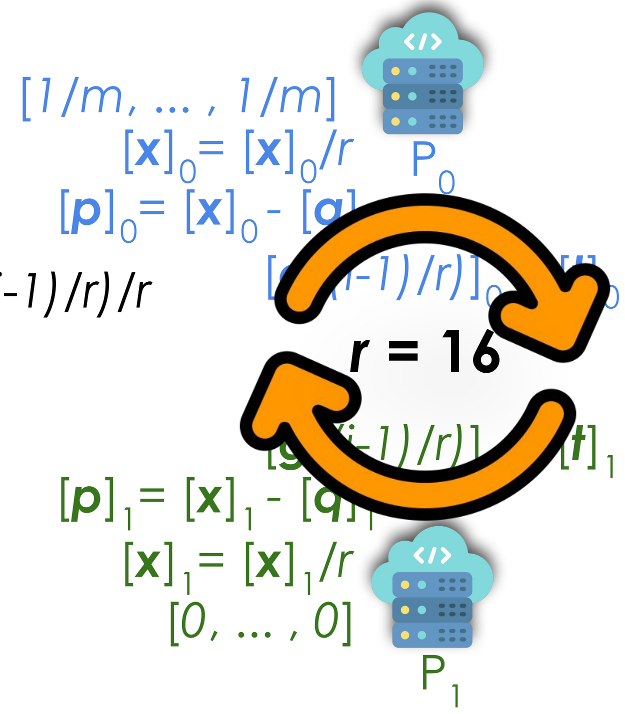
    for $i = 1, \ldots, r$ **do**

    $\mathbf{g}(i/r) = \mathbf{g}((i-1)/r) + (\mathbf{x} - \langle \mathbf{x}, \mathbf{g}((i-1)/r) \rangle \mathbf{1}) * \mathbf{g}((i-1)/r)/r$
- Jointly invoke secure multiplication π.
  - Inner product with m dimensions parallelly.
  - Element-wise product with m dimensions.



New Softmax Approx. and Protocol

- Jointly compute QSMax(x) online
  - $\mathbf{g}(0) = [1/m, \ldots, 1/m]$
    for $i = 1, \ldots, r$ **do**
    $$\mathbf{g}(i/r) = \mathbf{g}((i-1)/r) + (\mathbf{x} - \langle \mathbf{x}, \mathbf{g}((i-1)/r) \rangle \mathbf{1}) * \mathbf{g}((i-1)/r)/r$$
- Jointly invoke secure multiplication π.
  - Inner product with *m* dimensions parallelly.
  - Element-wise product with *m* dimensions.
- Secret-shared outputs.
  - P_0 gets $[\mathbf{g}(1)]_0$.
  - P_1 gets $[\mathbf{g}(1)]_0$.



Communication Costs

- Achieved **< 10%** communication costs in **32** rounds!

Protocol	#Class	Online (bits)	Overall (bits)	Round
ASM protocol	10	-	3M	704
Crypten		783K	982K	171
Ours		63K	**84K**	**32**
ASM protocol	100	-	30M	704
Crypten		8.5M	11M	300
Ours		616K	**821K**	**32**
ASM protocol	1000	-	302M	704
Crypten		86M	108M	430
Ours		6M	**8M**	**32**

Experiments & System Performance

- Datasets: MNIST, CIFAR-10
- Models: AlexNet, LeNet, VGG-16, ResNet, Networks A-B-C-D.
- Communication reduces by **57%-77%.**
- Accuracy
  - reaches a higher accuracy for AlexNet, VGG-16 compared with Piranha [Usenix Sec'22].
  - reaches a similar accuracy for Networks A-B-C-D compared with SPDZ-QT [Keller and Sun, ICML'22].
- Training time
  - **10%-60%** speed-up in LAN & **56%-78%** speed-up in WAN.

Conclusion

- Propose two cryptography-friendly approximations for secure computation of softmax and sigmoid, leading to expedited private training with much lower communication.
- Provide both C++ & Python implementation for different programming preference.
- Shed light on protocol design for bounded nonlinear functions, avoiding unbounded intermediate functions (e^x, $1/x$).
- Extend the realm of secure computation to encompass solutions for differential equations with rational polynomial or trigonometric functions coefficients.