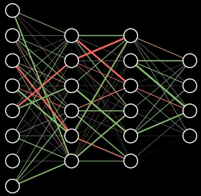


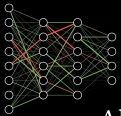
garbled neural networks are practical

. . .according to

marshall ball	columbia university
brent carmer	galois inc
tal malkin	columbia university
mike rosulek	oregon state university
nichole schimanski	galois inc

PPML @ ACM CCS 2019; Nov 15

private classifier



x

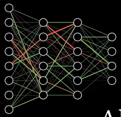
Alice ⟶ ⟵ ⟶ Bob

prediction(x)

ML classification as service

usually only **weights** are private
black-box reconstruction of weights!

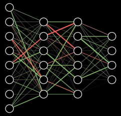
private classifier vs. public classifier



x

Alice $\longrightarrow$ $\longleftarrow$ $\longrightarrow$ Bob

prediction(x)



x_A x_B

Alice $\longrightarrow$ $\longleftarrow$ $\longrightarrow$ Bob

prediction($x_A + x_B$)

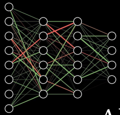
ML classification as service

usually only **weights** are private
black-box reconstruction of weights!

e.g., spam filtering on secret shared data

less to hide $\implies$ (potentially) cheaper

private classifier vs. public classifier



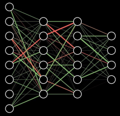
x

Alice $\longrightarrow$ $\longleftarrow$ $\longrightarrow$ Bob

prediction(x)

ML classification as service

usually only **weights** are private
black-box reconstruction of weights!



x_A x_B

Alice $\longrightarrow$ $\longleftarrow$ $\longrightarrow$ Bob

prediction($x_A + x_B$)

e.g., spam filtering on secret shared data

less to hide $\Longrightarrow$ (potentially) cheaper

In this work: both

FHE

+ low communication
+ low round complexity
– high computation
– low-depth only

secret-sharing

+ low communication
– high round complexity

garbled circuits

+ low round complexity
+ good for boolean
– bad for arithmetic

FHE

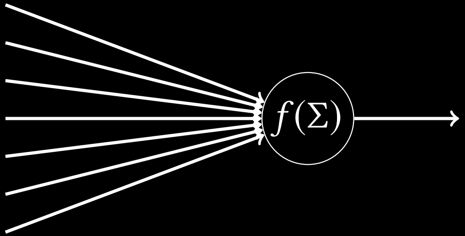
+ low communication
+ low round complexity
– high computation
– low-depth only

secret-sharing

+ low communication
– high round complexity

garbled circuits

+ low round complexity
+ good for boolean
– bad for arithmetic



conventional wisdom:

NN are mostly arithmetic operations $\implies$ GC is a bad fit

outline of this work

Garbling scheme of Ball-Malkin-Rosulek-2016

- very efficient for arithmetic operations
- inefficient for comparisons, etc

outline of this work

Garbling scheme of Ball-Malkin-Rosulek-2016

- very efficient for arithmetic operations
- inefficient for comparisons, etc

$\Downarrow$

Improvements to BMR16 garbling techniques

- targeting common NN activation functions
- approximate activations $\Rightarrow$ accuracy-efficiency tradeoff

Implementation @ `github.com/GaloisInc/fancy-garbling`

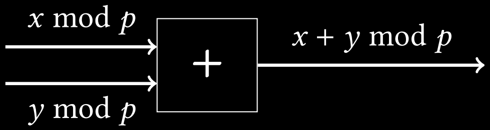
- TensorFlow model support

XONN Sadegh Riazi, Samragh, Chen, Laine, Lauter, Koushanfar	ours
garbled circuit NN evaluation	garbled circuit NN evaluation
binarized NN : $\{0, 1\}$ signals	discretized NN : $\{0, \ldots, N\}$ signals
requires retraining	can discretize existing NN
comparable performance	comparable performance

future work $\Longrightarrow$ combine techniques, retrain NN with smaller discretization

circuit model for Ball-Malkin-Rosulek-16 garbling

free addition:

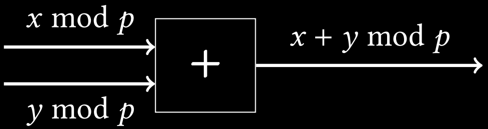


free to garble

($\Rightarrow$ free multiplication by constant)

circuit model for Ball-Malkin-Rosulek-16 garbling

free addition:



free to garble

($\Rightarrow$ free multiplication by constant)

unary/projection gates:

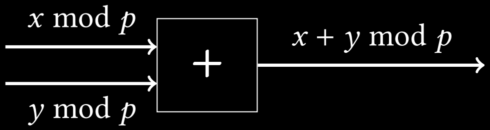
arbitrary $\varphi : \mathbb{Z}_p \to \mathbb{Z}_q$

cost to garble: $p - 1$ ciphertexts

(moduli should remain small)

circuit model for Ball-Malkin-Rosulek-16 garbling

free addition:



free to garble

($\Rightarrow$ free multiplication by constant)

unary/projection gates:

arbitrary $\varphi : \mathbb{Z}_p \to \mathbb{Z}_q$

cost to garble: $p - 1$ ciphertexts

(moduli should remain small)

[BMR16] construct multiplication, high-fan-in boolean, etc from these building blocks

residue representation:

$$[\![x]\!]_{\mathsf{crt}} = \left(x \bmod p_1, x \bmod p_2, \ldots x \bmod p_k\right)$$

addition mod $\prod_i p_i$:	free ✓
multiplication mod $\prod_i p_i$:	cheap: $O(\sum_i p_i)$ ✓
other things (comparisons, etc):	expensive

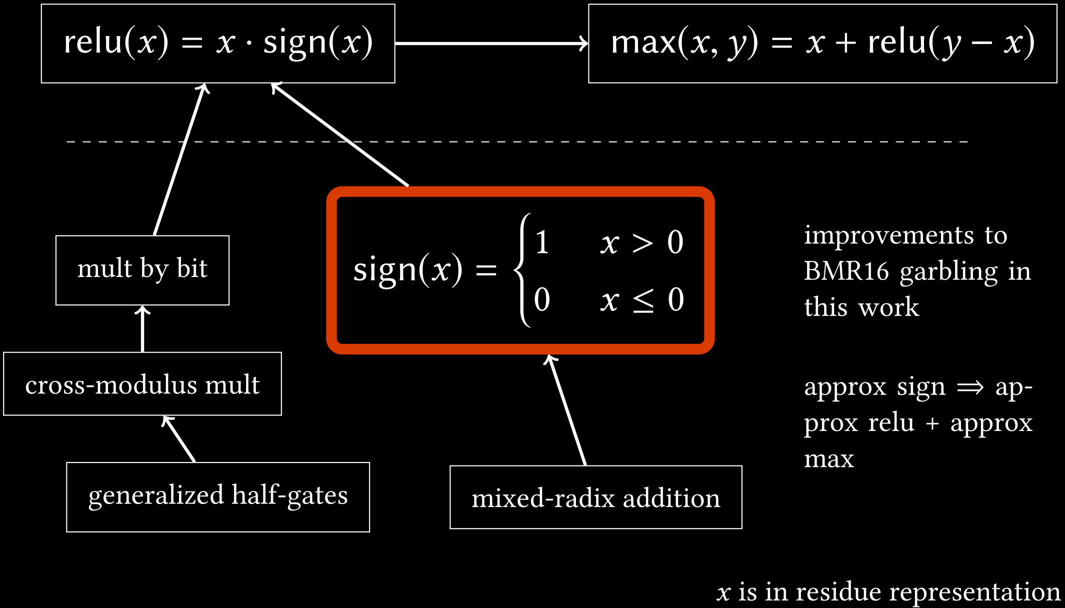
residue representation:

$$[\![x]\!]_{\mathsf{crt}} = \left(x \bmod p_1, x \bmod p_2, \ldots x \bmod p_k\right)$$

linear combination, public weights:	free ✓
linear combination, private weights:	cheap: $O(\sum_i p_i)$ ✓
activation function, max-pooling, etc:	expensive

$\boxed{\mathrm{relu}(x) = x \cdot \mathrm{sign}(x)} \longrightarrow \boxed{\max(x, y) = x + \mathrm{relu}(y - x)}$

x is in residue representation


$\mathrm{relu}(x) = x \cdot \mathrm{sign}(x)$
$\max(x, y) = x + \mathrm{relu}(y - x)$
mult by bit
cross-modulus mult
generalized half-gates
$\mathrm{sign}(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$
mixed-radix addition
improvements to BMR16 garbling in this work
approx sign $\Rightarrow$ approx relu + approx max
x is in residue representation

$$[\![x]\!]_{\mathsf{crt}} = (\underbrace{x \bmod p_1}_{x_1}, \ldots, \underbrace{x \bmod p_k}_{x_k})$$

$$\boxed{\mathrm{sign}(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}}$$

$$[\![x]\!]_{\mathsf{crt}} = (\underbrace{x \bmod p_1}_{x_1}, \ldots, \underbrace{x \bmod p_k}_{x_k});\ P = \prod_i p_i$$

$$\boxed{\mathrm{sign}(x) = \begin{cases} 1 & x > P/2 \\ 0 & x \leq P/2 \end{cases}}$$

$$[\![x]\!]_{\text{crt}} = (\underbrace{x \bmod p_1}_{x_1}, \ldots, \underbrace{x \bmod p_k}_{x_k}); \; P = \prod_i p_i$$

$$\boxed{\operatorname{sign}(x) = \begin{cases} 1 & x > P/2 \\ 0 & x \leq P/2 \end{cases}}$$

$$x \equiv \sum_i \alpha_i x_i \pmod{P}$$

$$[\![x]\!]_{\mathrm{crt}} = (\underbrace{x \bmod p_1}_{x_1}, \ldots, \underbrace{x \bmod p_k}_{x_k});\; P = \textstyle\prod_i p_i$$

$$\boxed{\mathrm{sign}(x) = \begin{cases} 1 & x > P/2 \\ 0 & x \le P/2 \end{cases}}$$

$$x \equiv \textstyle\sum_i \alpha_i x_i \pmod P$$

$$\Longrightarrow x = \textstyle\sum_i \alpha_i x_i + qP \text{ (over } \mathbb{Z}\text{)}$$

$$[\![x]\!]_{\mathrm{crt}} = (\underbrace{x \bmod p_1}_{x_1}, \ldots, \underbrace{x \bmod p_k}_{x_k}); \; P = \prod_i p_i$$

$$\boxed{\mathrm{sign}(x) = \begin{cases} 1 & x > P/2 \\ 0 & x \leq P/2 \end{cases}}$$

$$x \equiv \sum_i \alpha_i x_i \pmod{P}$$

$$\Longrightarrow x = \sum_i \alpha_i x_i + qP \text{ (over } \mathbb{Z})$$

$$\Longrightarrow \frac{x}{P} = \sum_i \frac{\alpha_i x_i}{P} + q$$

$$[\![x]\!]_{\mathrm{crt}} = (\underbrace{x \bmod p_1}_{x_1}, \ldots, \underbrace{x \bmod p_k}_{x_k}); \; P = \prod_i p_i$$

$$\boxed{\operatorname{sign}(x) = \begin{cases} 1 & x > P/2 \\ 0 & x \leq P/2 \end{cases}}$$

$$x \equiv \sum_i \alpha_i x_i \pmod{P}$$

$$\Rightarrow x = \sum_i \alpha_i x_i + qP \text{ (over } \mathbb{Z})$$

$$\Rightarrow \frac{x}{P} = \sum_i \frac{\alpha_i x_i}{P} + q$$

$$\Rightarrow \left[\frac{x}{P}\right]_1 = \left[\sum_i \frac{\alpha_i x_i}{P}\right]_1$$

$$[\![x]\!]_{\mathrm{crt}} = (\underbrace{x \bmod p_1}_{x_1}, \ldots, \underbrace{x \bmod p_k}_{x_k}); \; P = \prod_i p_i$$

$$\boxed{\mathrm{sign}(x) = \begin{cases} 1 & x > P/2 \\ 0 & x \leq P/2 \end{cases}}$$

$$x \equiv \sum_i \alpha_i x_i \pmod{P}$$

$$\Longrightarrow x = \sum_i \alpha_i x_i + qP \text{ (over } \mathbb{Z}\text{)}$$

$$\Longrightarrow \frac{x}{P} = \sum_i \frac{\alpha_i x_i}{P} + q$$

$$\Longrightarrow \left[\frac{x}{P}\right]_1 = \left[\sum_i \frac{\alpha_i x_i}{P}\right]_1$$

our approach:

approximate fractions inside $\sum$

add fractions (ignore integer part)

check whether result $> 1/2$

$$\mathrm{sign} = 1 \Leftrightarrow \left[\sum_i \frac{\alpha_i x_i}{P}\right]_1 > \frac{1}{2}$$

$$[\![x]\!]_{\mathrm{crt}} = (\underbrace{x \bmod p_1}_{x_1}, \ldots, \underbrace{x \bmod p_k}_{x_k});\ P = \prod_i p_i$$

$$\boxed{\mathrm{sign}(x) = \begin{cases} 1 & x > P/2 \\ 0 & x \le P/2 \end{cases}}$$

$$x \equiv \sum_i \alpha_i x_i \pmod{P}$$

$$\Longrightarrow x = \sum_i \alpha_i x_i + qP \text{ (over } \mathbb{Z}\text{)}$$

$$\Longrightarrow \frac{x}{P} = \sum_i \frac{\alpha_i x_i}{P} + q$$

$$\Longrightarrow \left[\frac{x}{P}\right]_1 = \left[\sum_i \frac{\alpha_i x_i}{P}\right]_1$$

$$\mathrm{sign} = 1 \Leftrightarrow \left[\sum_i \frac{\alpha_i x_i}{P}\right]_1 > \frac{1}{2}$$

our approach:

approximate fractions inside $\sum$

- ⋆ e.g., 10-bit binary

add fractions (ignore integer part)

- ⋆ (non-free) binary adder circuit

check whether result $> 1/2$

- ⋆ most significant bit of $\sum$

$$[\![x]\!]_{\mathrm{crt}} = (\underbrace{x \bmod p_1}_{x_1}, \ldots, \underbrace{x \bmod p_k}_{x_k});\ P = \textstyle\prod_i p_i$$

$$\boxed{\mathrm{sign}(x) = \begin{cases} 1 & x > P/2 \\ 0 & x \le P/2 \end{cases}}$$

$$x \equiv \textstyle\sum_i \alpha_i x_i \pmod P$$

$$\Longrightarrow x = \textstyle\sum_i \alpha_i x_i + qP \text{ (over } \mathbb{Z})$$

$$\Longrightarrow \frac{x}{P} = \textstyle\sum_i \frac{\alpha_i x_i}{P} + q$$

$$\Longrightarrow \left[\frac{x}{P}\right]_1 = \left[\textstyle\sum_i \frac{\alpha_i x_i}{P}\right]_1$$

$$\mathrm{sign} = 1 \Leftrightarrow \left[\textstyle\sum_i \frac{\alpha_i x_i}{P}\right]_1 > \frac{1}{2}$$

our approach:

approximate fractions inside $\sum$

- ★ e.g., 10-bit binary

add fractions (ignore integer part)

- ★ (non-free) binary adder circuit

check whether result $> 1/2$

- ★ most significant bit of $\sum$

choice of “fixed-pt resolution” affects **cost** & **accuracy**

# primes	$\prod_i p_i$	fixed-point resolution	correct	cost
7	$2^{19.0}$	$108360 = 86 \cdot 7 \cdot 6^2 \cdot 5$	$= 100\%$	637
		$10560 = 88 \cdot 6 \cdot 5 \cdot 4$	$\geq 99.99\%$	470
		$1200 = 60 \cdot 5 \cdot 4$	$\geq 99.9\%$	315
8	$2^{23.2}$	$1975680 = 98 \cdot 9 \cdot 8^2 \cdot 7 \cdot 5$	$= 100\%$	1078
		$107100 = 102 \cdot 7 \cdot 6 \cdot 5^2$	$\geq 99.999\%$	770
		$10920 = 78 \cdot 7 \cdot 5 \cdot 4$	$\geq 99.99\%$	574
		$1170 = 78 \cdot 5 \cdot 3$	$\geq 99.9\%$	385
9	$2^{27.7}$	$31933300 = 76 \cdot 7^5 \cdot 5^2$	$= 100\%$	1534
		$119700 = 114 \cdot 7 \cdot 6 \cdot 5^2$	$\geq 99.999\%$	933
		$12600 = 84 \cdot 6 \cdot 5^2$	$\geq 99.99\%$	696
		$1260 = 140 \cdot 9$	$\geq 99.9\%$	465
10	$2^{32.6}$	$791920800 = 202 \cdot 11^2 \cdot 6^4 \cdot 5^2$	$= 100\%$	2294
		$128520 = 102 \cdot 7 \cdot 6^2 \cdot 5$	$\geq 99.999\%$	1122
		$13440 = 112 \cdot 6 \cdot 5 \cdot 4$	$\geq 99.99\%$	843
		$1330 = 190 \cdot 7$	$\geq 99.9\%$	547

# primes	$\prod_i p_i$	fixed-point resolution	correct	cost
7	$2^{19.0}$	$108360 = 86 \cdot 7 \cdot 6^2 \cdot 5$	$= 100\%$	637
		$10560 = 88 \cdot 6 \cdot 5 \cdot 4$	$\geq 99.99\%$	470
		$1200 = 60 \cdot 5 \cdot 4$	[illegible]	[illegible]
8	$2^{23.2}$	$1975680 = 98 \cdot 9 \cdot 8^2 \cdot 7 \cdot$ [illegible]	[illegible]	[illegible]
		$107100 = 102 \cdot$ [illegible]	[illegible]	[illegible]
		[illegible]	[illegible]	[illegible]
		[illegible]	[illegible]	385
[illegible]	[illegible]	[illegible]	$= 100\%$	1534
		[illegible] $6 \cdot 5^2$	$\geq 99.999\%$	933
		[illegible] $84 \cdot 6 \cdot 5^2$	$\geq 99.99\%$	696
		$1260 = 140 \cdot 9$	$\geq 99.9\%$	465
10	$2^{32.6}$	$791920800 = 202 \cdot 11^2 \cdot 6^4 \cdot 5^2$	$= 100\%$	2294
		$128520 = 102 \cdot 7 \cdot 6^2 \cdot 5$	$\geq 99.999\%$	1122
		$13440 = 112 \cdot 6 \cdot 5 \cdot 4$	$\geq 99.99\%$	843
		$1330 = 190 \cdot 7$	$\geq 99.9\%$	547

99.99% accuracy @ 33–50% cost

	weights	time (s)	comm (MB)	accuracy (%)
boolean garbling	private	>300	3407	96.8
ours	private	1.98	128	96.8

MNIST-A (128+128+10 neurons) @ 22-bit discretization

	weights	time (s)	comm (MB)	accuracy (%)
boolean garbling	private	>300	3407	96.8
boolean garbling	public	53	618	96.8
ours	private	1.98	128	96.8
ours	public	0.12	4.43	96.8

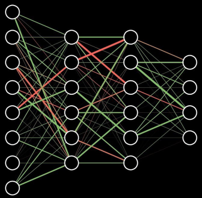
MNIST-A (128+128+10 neurons) @ 22-bit discretization

	weights	time (s)	comm (MB)	accuracy (%)
boolean garbling	private	>300	3407	96.8
boolean garbling	public	53	618	96.8
ours	private	1.98	128	96.8
ours	public	0.12	4.43	96.8
ours (99.99% activation)	private	1.98	127	95.7
ours (99.99% activation)	public	0.10	2.77	95.7

MNIST-A (128+128+10 neurons) @ 22-bit discretization

	weights	time (s)	comm (MB)	accuracy (%)
boolean garbling	private	>300	3407	96.8
boolean garbling	public	53	618	96.8
ours	private	1.98	128	96.8
ours	public	0.12	4.43	96.8
ours (99.99% activation)	private	1.98	127	95.7
ours (99.99% activation)	public	0.10	2.77	95.7
SecureML	private	4.88	-	93.1
MiniONN	private	1.04	47.6	97.6
Gazelle	private	0.03	0.5	-
XONN	private	0.13	4.29	97.6

MNIST-A (128+128+10 neurons) @ 22-bit discretization



garbled circuits are better than you thought

for arithmetic computations, NNs

. . . especially for NNs with public weights

approximate activation functions are great!

`ia.cr/2019/338`

`github.com/GaloisInc/fancy-garbling`