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
private classifier

ML classification as service

usually only weights are private black-box reconstruction of weights!
private classifier vs. public classifier

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

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 $\Rightarrow$ (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

conventional wisdom: NN are mostly arithmetic ⟷ GC is a bad
**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 \(\Rightarrow\) 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

Improvements to BMR16 garbling techniques

- targeting common NN activation functions
- approximate activations ⇒ 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

binarized NN: \{0, 1\} signals
requires retraining
comparable performance

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

garbled circuit NN evaluation

discretized NN: \{0, \ldots, N\} signals
can discretize existing NN
comparable performance
circuit model for Ball-Malkin-Rosulek-16 garbling

free addition:

\[ \begin{align*}
& x \mod p \\
& y \mod p \\
\downarrow & \quad + & \quad \downarrow \\
\quad & x + y \mod p
\end{align*} \]

free to garble

(⇒ free multiplication by constant)

modulo should remain small
circuit model for Ball-Malkin-Rosulek-16 garbling

free addition:

\[ x \mod p + y \mod p \mod p \]

free to garble

(\Rightarrow \text{free multiplication by constant})

unary/projection gates:

\[ \phi_x \mod p \rightarrow \phi(x) \mod q \]

arbitrary \( \phi : \mathbb{Z}_p \rightarrow \mathbb{Z}_q \)

cost to garble: \( p - 1 \) ciphertexts

(moduli should remain small)
circuit model for Ball-Malkin-Rosulek-16 garbling

free addition:

\[ x \mod p \quad + \quad y \mod p \rightarrow x + y \mod p \]

free to garble

(⇒ free multiplication by constant)

unary/projection gates:

\[ x \mod p \rightarrow \phi(x) \mod q \]

arbitrary \( \phi : \mathbb{Z}_p \rightarrow \mathbb{Z}_q \)

cost to garble: \( p - 1 \) ciphertexts

(moduli should remain small)

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

$$\llbracket x \rrbracket_{\text{crt}} = \left( x \mod p_1, x \mod p_2, \ldots x \mod p_k \right)$$

- 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:

\[
\left\lfloor x \right\rfloor_{\text{crt}} = \left( x \mod p_1, x \mod p_2, \ldots x \mod p_k \right)
\]

- linear combination, public weights: free ✓
- linear combination, private weights: cheap: \( O(\Sigma_i p_i) \) ✓
- activation function, max-pooling, etc: expensive
relu($x$) = $x \cdot \text{sign}(x)$

max($x$, $y$) = $x + \text{relu}(y - x)$

$x$ is in residue representation
ReLU function:
\[ \text{relu}(x) = x \cdot \text{sign}(x) \]

Max function:
\[ \text{max}(x, y) = x + \text{relu}(y - x) \]

Sign function:
\[ \text{sign}(x) = \begin{cases} 
1 & x > 0 \\
0 & x \leq 0 
\end{cases} \]

- mult by bit
- cross-modulus mult
- generalized half-gates
- mixed-radix addition

Improvements to BMR16 garbling in this work:
approx \text{sign} \Rightarrow \text{approx relu} + \text{approx max}

$x$ is in residue representation.
\[
[x]_{\text{crt}} = (x \mod p_1, \ldots, x \mod p_k)
\]

\[
\text{sign}(x) = \begin{cases} 
1 & x > 0 \\
0 & x \leq 0 
\end{cases}
\]
\[
[[x]]_{\text{crt}} = (x \mod p_1, \ldots, x \mod p_k); \quad P = \prod_i p_i
\]

\[
\text{sign}(x) = \begin{cases} 
1 & x > P/2 \\
0 & x \leq P/2 
\end{cases}
\]
\[
[x]_{\text{crt}} = (x \mod p_1, \ldots, x \mod p_k); \quad P = \prod_i p_i
\]

\[
x \equiv \sum_i \alpha_i x_i \pmod{P}
\]

\[
\text{sign}(x) = \begin{cases} 
1 & x > P/2 \\
0 & x \leq P/2
\end{cases}
\]
\[
[x]_{\text{crt}} = \left( x \mod p_1, \ldots, x \mod p_k \right); \quad P = \prod_i p_i
\]

\[
x \equiv \sum_i \alpha_i x_i \pmod{P}
\]

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

\[
\text{sign}(x) = \begin{cases} 
1 & x > P/2 \\
0 & x \leq P/2
\end{cases}
\]
\[
\begin{align*}
[x]_{\text{crt}} &= (x \mod p_1, \ldots, x \mod p_k); 
P &= \prod_i p_i \\
x &= \sum_i \alpha_i x_i \pmod{P} \\
\Rightarrow x &= \sum_i \alpha_i x_i + qP \quad \text{(over } \mathbb{Z}) \\
\Rightarrow \frac{x}{P} &= \sum_i \frac{\alpha_i x_i}{P} + q
\end{align*}
\]

\[
\text{sign}(x) = \begin{cases} 
1 & x > P/2 \\
0 & x \leq P/2
\end{cases}
\]
\[
[x]_{\text{crt}} = (x \mod p_1, \ldots, x \mod p_k); P = \prod_i p_i
\]

\[
x = \sum_i \alpha_i x_i \pmod{P}
\]

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

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

\[
\Rightarrow \left[\frac{x}{P}\right]_1 = \left[\sum_i \frac{\alpha_i x_i}{P}\right]_1
\]
\[\llbracket x \rrbracket_{\text{crt}} = (x \mod p_1, \ldots, x \mod p_k); \quad P = \prod_i p_i\]

\[\text{sign}(x) = \begin{cases} 
1 & x > P/2 \\
0 & x \leq P/2
\end{cases}\]

our approach:

approximate fractions inside \(\sum\)

add fractions (ignore integer part)

check whether result > 1/2

\[\begin{align*}
\text{sign} = 1 & \iff \left\lfloor \sum_i \frac{\alpha_i x_i}{P} \right\rfloor_1 > \frac{1}{2}
\end{align*}\]
\([x]_{\text{crt}} = (x \mod p_1, \ldots, x \mod p_k); P = \prod_i p_i\]

\[
x \equiv \sum_i \alpha_i x_i \pmod{P}
\]

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

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

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

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

\[
\text{sign}(x) = \begin{cases} 1 & x > P/2 \\ 0 & x \leq P/2 \end{cases}
\]

**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 \(\sum\)**
\[ \llbracket x \rrbracket_{\text{crt}} = (x \mod p_1, \ldots, x \mod p_k); \quad P = \prod_i p_i \]

\[ x = \sum_i \alpha_i x_i \pmod{P} \]

\[ x = \sum_i \alpha_i x_i + qP \quad \text{(over } \mathbb{Z}) \]

\[ \frac{x}{P} = \sum_i \frac{\alpha_i x_i}{P} + q \]

\[ \llbracket \frac{x}{P} \rrbracket_1 = \llbracket \sum_i \frac{\alpha_i x_i}{P} \rrbracket_1 \]

\[ \text{sign}(x) = \begin{cases} 
1 & x > P/2 \\
0 & x \leq P/2 
\end{cases} \]

**our approach:**

approximate fractions inside \( \Sigma \)
- e.g., 10-bit binary

add fractions (ignore integer part)
- (non-free) binary adder circuit

check whether result > 1/2
- most significant bit of \( \Sigma \)

choice of “fixed-pt resolution” affects **cost & accuracy**
<table>
<thead>
<tr>
<th># primes</th>
<th>$\Pi_i p_i$</th>
<th>fixed-point resolution</th>
<th>correct</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>$2^{19.0}$</td>
<td>$108360 = 86 \cdot 7 \cdot 6^2 \cdot 5$</td>
<td>= 100%</td>
<td>637</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$10560 = 88 \cdot 6 \cdot 5 \cdot 4$</td>
<td>$\geq 99.99%$</td>
<td>470</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1200 = 60 \cdot 5 \cdot 4$</td>
<td>$\geq 99.9%$</td>
<td>315</td>
</tr>
<tr>
<td>8</td>
<td>$2^{23.2}$</td>
<td>$1975680 = 98 \cdot 9 \cdot 8^2 \cdot 7 \cdot 5$</td>
<td>= 100%</td>
<td>1078</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$107100 = 102 \cdot 7 \cdot 6 \cdot 5^2$</td>
<td>$\geq 99.999%$</td>
<td>770</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$10920 = 78 \cdot 7 \cdot 5 \cdot 4$</td>
<td>$\geq 99.99%$</td>
<td>574</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1170 = 78 \cdot 5 \cdot 3$</td>
<td>$\geq 99.9%$</td>
<td>385</td>
</tr>
<tr>
<td>9</td>
<td>$2^{27.7}$</td>
<td>$31933300 = 76 \cdot 7^5 \cdot 5^2$</td>
<td>= 100%</td>
<td>1534</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$119700 = 114 \cdot 7 \cdot 6 \cdot 5^2$</td>
<td>$\geq 99.999%$</td>
<td>933</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$12600 = 84 \cdot 6 \cdot 5^2$</td>
<td>$\geq 99.99%$</td>
<td>696</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1260 = 140 \cdot 9$</td>
<td>$\geq 99.9%$</td>
<td>465</td>
</tr>
<tr>
<td>10</td>
<td>$2^{32.6}$</td>
<td>$791920800 = 202 \cdot 11^2 \cdot 6^4 \cdot 5^2$</td>
<td>= 100%</td>
<td>2294</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$128520 = 102 \cdot 7 \cdot 6^2 \cdot 5$</td>
<td>$\geq 99.999%$</td>
<td>1122</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$13440 = 112 \cdot 6 \cdot 5 \cdot 4$</td>
<td>$\geq 99.99%$</td>
<td>843</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1330 = 190 \cdot 7$</td>
<td>$\geq 99.9%$</td>
<td>547</td>
</tr>
<tr>
<td># primes</td>
<td>$\Pi_i p_i$</td>
<td>fixed-point resolution</td>
<td>correct</td>
<td>cost</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
<td>------------------------</td>
<td>----------</td>
<td>-------</td>
</tr>
<tr>
<td>7</td>
<td>$2^{19.0}$</td>
<td>$108360 = 86 \cdot 7 \cdot 6^2 \cdot 5$</td>
<td>$= 100%$</td>
<td>637</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$10560 = 88 \cdot 6 \cdot 5 \cdot 4$</td>
<td>$\geq 99.99%$</td>
<td>470</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1200 = 60 \cdot 5 \cdot 4$</td>
<td>$\geq 99.9%$</td>
<td>115</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1975680 = 98 \cdot 9 \cdot 8^2 \cdot 7$</td>
<td>$= 100%$</td>
<td>678</td>
</tr>
<tr>
<td>8</td>
<td>$2^{23.2}$</td>
<td>$107100 = 102 \cdot 5^2$</td>
<td>$\geq 99.999%$</td>
<td>933</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$10920 = 64 \cdot 6 \cdot 5^2$</td>
<td>$\geq 99.99%$</td>
<td>696</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1260 = 140 \cdot 9$</td>
<td>$\geq 99.9%$</td>
<td>465</td>
</tr>
<tr>
<td>10</td>
<td>$2^{32.6}$</td>
<td>$791920800 = 202 \cdot 11^2 \cdot 6^4 \cdot 5^2$</td>
<td>$= 100%$</td>
<td>2294</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$128520 = 102 \cdot 7 \cdot 6^2 \cdot 5$</td>
<td>$\geq 99.999%$</td>
<td>1122</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$13440 = 112 \cdot 6 \cdot 5 \cdot 4$</td>
<td>$\geq 99.99%$</td>
<td>843</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1330 = 190 \cdot 7$</td>
<td>$\geq 99.9%$</td>
<td>547</td>
</tr>
</tbody>
</table>

99.99% accuracy @ 33–50% cost
<table>
<thead>
<tr>
<th>weights</th>
<th>time (s)</th>
<th>comm (MB)</th>
<th>accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean garbling private</td>
<td>&gt;300</td>
<td>3407</td>
<td>96.8</td>
</tr>
<tr>
<td>ours private</td>
<td>1.98</td>
<td>128</td>
<td>96.8</td>
</tr>
</tbody>
</table>

MNIST-A (128+128+10 neurons) @ 22-bit discretization
<table>
<thead>
<tr>
<th>weights</th>
<th>time (s)</th>
<th>comm (MB)</th>
<th>accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean garbling</td>
<td>&gt;300</td>
<td>3407</td>
<td>96.8</td>
</tr>
<tr>
<td>boolean garbling</td>
<td>53</td>
<td>618</td>
<td>96.8</td>
</tr>
<tr>
<td>ours</td>
<td>1.98</td>
<td>128</td>
<td>96.8</td>
</tr>
<tr>
<td>ours</td>
<td>0.12</td>
<td>4.43</td>
<td>96.8</td>
</tr>
</tbody>
</table>

**MNIST-A (128+128+10 neurons) @ 22-bit discretization**
<table>
<thead>
<tr>
<th>weights</th>
<th>time (s)</th>
<th>comm (MB)</th>
<th>accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean garbling private</td>
<td>&gt;300</td>
<td>3407</td>
<td>96.8</td>
</tr>
<tr>
<td>boolean garbling public</td>
<td>53</td>
<td>618</td>
<td>96.8</td>
</tr>
<tr>
<td>ours private</td>
<td>1.98</td>
<td>128</td>
<td>96.8</td>
</tr>
<tr>
<td>ours public</td>
<td>0.12</td>
<td>4.43</td>
<td>96.8</td>
</tr>
<tr>
<td>ours (99.99% activation) private</td>
<td>1.98</td>
<td>127</td>
<td>95.7</td>
</tr>
<tr>
<td>ours (99.99% activation) public</td>
<td>0.10</td>
<td>2.77</td>
<td>95.7</td>
</tr>
</tbody>
</table>

MNIST-A (128+128+10 neurons) @ 22-bit discretization
<table>
<thead>
<tr>
<th>Model</th>
<th>Privacy</th>
<th>weights</th>
<th>time (s)</th>
<th>comm (MB)</th>
<th>accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean garbling</td>
<td>private</td>
<td>&gt;300</td>
<td>3407</td>
<td>96.8</td>
<td></td>
</tr>
<tr>
<td>boolean garbling</td>
<td>public</td>
<td>53</td>
<td>618</td>
<td>96.8</td>
<td></td>
</tr>
<tr>
<td>ours</td>
<td>private</td>
<td>1.98</td>
<td>128</td>
<td>96.8</td>
<td></td>
</tr>
<tr>
<td>ours</td>
<td>public</td>
<td>0.12</td>
<td>4.43</td>
<td>96.8</td>
<td></td>
</tr>
<tr>
<td>ours (99.99% activation)</td>
<td>private</td>
<td>1.98</td>
<td>127</td>
<td>95.7</td>
<td></td>
</tr>
<tr>
<td>ours (99.99% activation)</td>
<td>public</td>
<td>0.10</td>
<td>2.77</td>
<td>95.7</td>
<td></td>
</tr>
<tr>
<td>SecureML</td>
<td>private</td>
<td>4.88</td>
<td>-</td>
<td>93.1</td>
<td></td>
</tr>
<tr>
<td>MiniONN</td>
<td>private</td>
<td>1.04</td>
<td>47.6</td>
<td>97.6</td>
<td></td>
</tr>
<tr>
<td>Gazelle</td>
<td>private</td>
<td>0.03</td>
<td>0.5</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>XONNN</td>
<td>private</td>
<td>0.13</td>
<td>4.29</td>
<td>97.6</td>
<td></td>
</tr>
</tbody>
</table>

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