BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning

Chengliang Zhang†, Suyi Li†, Junzhe Xia†, Wei Wang†, Feng Yan‡, Yang Liu*

†Hong Kong University of Science and Technology
‡University of Nevada, Reno
* WeBank
Federated Learning

- Privacy concerns
  - Data breaches
- Government regulations
  - GDPR
  - CCPA

Emerging challenge: small & fragmented data

Solution: Federated Learning
Collaborative Machine Learning without Centralized Training Data [1]

**Target Scenario: Cross-Silo Horizontal FL**

- **Cross-Silo**: among organizations / institutions
  - Banks, hospitals...
  - Reliable communication and computation
  - Strong privacy requirements
  - As opposed to cross-device: edge devices
Target Scenario: Cross-Silo Horizontal FL

- **Horizontal**: datasets share same feature space [2]

  ![Horizontal FML Diagram]

  - Large overlap of features of the two data sets

- **Objective**: train a model together without revealing private data to third party (aggregator) and each other

  ![Vertical FML Diagram]

  - Large overlap of sample IDs (users) of the two data sets

Repurpose datacenter distributed training?

Gradients are not safe to share in plaintext [3]

# Federated Learning Approaches

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Efficiency</td>
<td>🚫 [6]</td>
<td></td>
<td>🚫</td>
<td>🚫</td>
</tr>
<tr>
<td>Strong Privacy</td>
<td>🚫 [4]</td>
<td></td>
<td>🚫</td>
<td>🚫</td>
</tr>
<tr>
<td>No accuracy loss</td>
<td>🚫 [5]</td>
<td></td>
<td>🚫</td>
<td>🚫</td>
</tr>
</tbody>
</table>


Additively Homomorphic Encryption for FL

• Allow computation over ciphertexts
  \[ \text{decrypt(encrypt(a) + encrypt(b))} = a + b \]
• Enables oblivious aggregation

1. Clients produce gradients
2. Encrypt gradients and upload them to Aggregator
3. Aggregator summarizes all gradient ciphertexts
4. Clients receive aggregated gradients
5. Clients decrypt and apply model update

Characterization: FL with HE

Why is HE expensive:
- Computation
- Communication
  - Plaintext: 32bit -> ciphertext: 2000+ bit

<table>
<thead>
<tr>
<th>Key Size</th>
<th>Plaintext</th>
<th>Ciphertext</th>
<th>Encryption</th>
<th>Decryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>6.87MB</td>
<td>287.64MB</td>
<td>216.87s</td>
<td>68.63s</td>
</tr>
<tr>
<td>2048</td>
<td>6.87MB</td>
<td>527.17MB</td>
<td>1152.98s</td>
<td>357.17s</td>
</tr>
<tr>
<td>3072</td>
<td>6.87MB</td>
<td>754.62MB</td>
<td>3111.14s</td>
<td>993.80</td>
</tr>
</tbody>
</table>

Paillier HE

Time breakdown of one iteration
Run on FATE, models are FMNIST, CIFAR10, and LSTM
Potential Solutions

• Accelerate HE operations
  o Limited parallelism: 3X with FPGA [9]
  o Communication stays the same

• Reduce encryption operations
  o One operation multiple data
  o “batching” gradient values
  o Compact plaintext, less inflation

plaintext: 2000 bit -> ciphertext 2000bit

Challenge:
Maintain HE’s additively property

Decrypting the sum of 2 batched ciphertexts
  =
Adding pairs separately

\[\begin{align*}
-0.3 & \quad 0 & \quad 2.6 & \quad -1.1 \\
1.2 & \quad 0.33 & \quad -4.2 & \quad -0.2 \\
\end{align*}\]

\[\begin{align*}
+ \\
\end{align*}\]

\[\begin{align*}
0.9 & \quad 0.33 & \quad -1.6 & \quad -1.3 \\
\end{align*}\]

Gradient Batching is non-trivial

All ciphertexts at aggregator: no differentiation, no permutation, no shifting

Only bit-wise additions on underlying plaintexts

Gradients are floating numbers: exponent aligning is required for addition [9]

Quantization for Batching

Floating gradient values cannot be batched -> quantization

A generic quantization method maps [-1, 1] To [0, 255]
Quantization: $255 \times \frac{-0.0079 - 1}{1 - 1} = 126$
Dequantization: $127 \times (1 - 1) / 255 + \frac{-2}{2} \times (-1) = -1$

Batching with generic quantization

<table>
<thead>
<tr>
<th>Original Value</th>
<th>Quantized Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0111 1110</td>
<td>0000 0001</td>
</tr>
<tr>
<td>-0.0079 126</td>
<td>0.0079 129</td>
</tr>
<tr>
<td>0000 0001</td>
<td>0111 1000</td>
</tr>
<tr>
<td>-0.9921 1</td>
<td>-0.0551 120</td>
</tr>
<tr>
<td>= 0111 1111</td>
<td>1111 1001</td>
</tr>
<tr>
<td>-1 127</td>
<td>-0.0475 249</td>
</tr>
</tbody>
</table>

Limitations
- Restrictive: client # is required
- Overflow easily: all positive integers
- No differentiation between positive and negative overflows
Desired quantization for aggregation

- **Flexible**
  - Aggregation results are unbatchable only with ciphertexts alone
- **Overflow-aware**
  - If overflow happens, we can tell the sign
Our Quantization & Batching Solution

Customized quantization for aggregation

- Distinguish overflow
  - *Signed* integer
- Positive and negative cancel out each other
  - *Symmetric* range
  - *Uniform* quantization

\[-1, 1\] is mapped to \([-127, 127]\)

<table>
<thead>
<tr>
<th>Bit Padding</th>
<th>Bit Value</th>
<th>Original Value</th>
<th>Quantized Value</th>
<th>Sign Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>11 111 111</td>
<td>00 000 0001</td>
<td>00 000 0001</td>
<td></td>
</tr>
<tr>
<td>-0.0079</td>
<td>-1</td>
<td>0.0079</td>
<td>+1</td>
<td></td>
</tr>
</tbody>
</table>

\[-0.9921, -1.26\] = \[-0.0475, -0.0551\]

BatchCrypt
Our Quantization & Batching Solution

Customized quantization for aggregation

- **Signed** integer
- **Symmetric** range
- **Uniform** quantization

Challenges:
1. Differentiate overflows:
   - two sign bits
2. Distinguish sign bits from value bits:
   - two’s compliment coding
3. Tolerate overflowing:
   - padding zeros in between

<table>
<thead>
<tr>
<th>original value</th>
<th>quantized value</th>
<th>sign bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 11 11 111</td>
<td>00 000 0001</td>
<td>0</td>
</tr>
<tr>
<td>-0.0079</td>
<td>-1</td>
<td>0.0079</td>
</tr>
<tr>
<td>+1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00 11 00 0010</td>
<td>00 11 11 1001</td>
<td>0</td>
</tr>
<tr>
<td>-0.9921</td>
<td>-126</td>
<td>-0.0551</td>
</tr>
<tr>
<td>-7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>01 11 00 0001</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>-127</td>
<td>-0.0475</td>
</tr>
<tr>
<td>-6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

BatchCrypt
Gradient Clipping

Gradients are *unbounded*
Quantization range is *bounded*
Clipping is required

Tradeoff:

- Higher resolution within $|\alpha|$
- More diminished range information

![Diagram showing clipping and quantization noise](image)
Gradient Clipping

Gradients are *unbounded* quantization range is *bounded*

Clipping is required

- Profiling quantization loss with a sample dataset \[10\]
  - FL has non-iid data
  - Gradients range diminishes during training: optimal shifts

- Analytical clipping with an online model
  - Model the noises with distribution fitting
  - Flexible & adaptable

\[10\] \url{http://on-demand.gputechconf.com/gtc/2017/presentation/s7310-8-bit-inference-with-tensorrt.pdf}
dACIQ: Analytical Gradient Clipping

- Gradients distribution is bell-shaped: *Gaussian* like
- Conventional gaussian fitting: *MLE, BI*
  - Requires *a lot of information*
  - Computationally intensive

- *dACIQ* proposes a Gaussian Fitting method for distributed dataset
  - Only requires *max, min, and size*
  - Computationally efficient: *online*
  - *Layer-wise* quantization

---

Introducing BatchCrypt

- Built atop FATE v1.1
- Support TensorFlow, MXNet, and extendable to other frameworks
- Implemented in Python
- Utilize Joblib, Numba for maximum parallelism
Evaluations Setup

Test Models

<table>
<thead>
<tr>
<th>Model</th>
<th>Type</th>
<th>Network</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMNIST</td>
<td>Image Classification</td>
<td>3-layer-FC</td>
<td>101.77K</td>
</tr>
<tr>
<td>CIFAR</td>
<td>Image Classification</td>
<td>AlexNet</td>
<td>1.25M</td>
</tr>
<tr>
<td>LSTM-ptb</td>
<td>Text Generation</td>
<td>LSTM</td>
<td>4.02M</td>
</tr>
</tbody>
</table>

Test Bed

- AWS
- Cluster of 10, spanning 5 locations
- C5.4xlarge instances (16 vCPUs, 32 GB memory)

<table>
<thead>
<tr>
<th>Region</th>
<th>US W.</th>
<th>Tokyo</th>
<th>US E.</th>
<th>London</th>
<th>HK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up (Mbps)</td>
<td>9841</td>
<td>116</td>
<td>165</td>
<td>97</td>
<td>81</td>
</tr>
<tr>
<td>Down (Mbps)</td>
<td>9842</td>
<td>122</td>
<td>151</td>
<td>84</td>
<td>84</td>
</tr>
</tbody>
</table>

Bandwidth from clients to aggregator
BatchCrypt’s Quantization Quality

- Negligible loss
- Quantization sometimes outperforms plain: randomness adds regularization
BatchCrypt’s Effectiveness: Computation

- Compared with stock FATE
- Batch size set to 100
- 16 bit quantization
- 23.3X for FMNIST
- 70.8X for CIFAR
- 92.8X for LSTM

Larger the model, better the results
BatchCrypt’s Effectiveness: Communication

Network traffic consumed by communication per iteration

- Compared with stock FATE
- Batch size set to 100
- 16 bit quantization
- 66X for FMNIST
- 71X for CIFAR
- 101X for LSTM
BatchCrypt’s Overhead

- Compared with plain distributed training without encryption
- Batch size set to 100
- 16 bit quantization
- Overhead significantly reduced
- Practical to deploy

Feasible to train large models now
## BatchCrypt’s Effectiveness: Convergence

Total time and communication until convergence

<table>
<thead>
<tr>
<th>Model</th>
<th>Mode</th>
<th>Epochs</th>
<th>Acc. /Loss</th>
<th>Time (h)</th>
<th>Traffic (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMNIST</td>
<td>stock</td>
<td>40</td>
<td>88.62%</td>
<td>122.5</td>
<td>2228.3</td>
</tr>
<tr>
<td></td>
<td>batch</td>
<td>68</td>
<td>88.37%</td>
<td>8.9</td>
<td>58.7</td>
</tr>
<tr>
<td></td>
<td>plain</td>
<td>40</td>
<td>88.62%</td>
<td>3.2</td>
<td>11.17</td>
</tr>
<tr>
<td>CIFAR</td>
<td>stock</td>
<td>285</td>
<td>73.79%</td>
<td>9495.6</td>
<td>16422.0</td>
</tr>
<tr>
<td></td>
<td>batch</td>
<td>279</td>
<td>74.04%</td>
<td>131.3</td>
<td>227.8</td>
</tr>
<tr>
<td></td>
<td>plain</td>
<td>285</td>
<td>73.79%</td>
<td>34.2</td>
<td>11.39</td>
</tr>
<tr>
<td>LSTM</td>
<td>stock</td>
<td>20</td>
<td>0.0357</td>
<td>8484.4</td>
<td>15347.3</td>
</tr>
<tr>
<td></td>
<td>batch</td>
<td>23</td>
<td>0.0335</td>
<td>105.2</td>
<td>175.9</td>
</tr>
<tr>
<td></td>
<td>plain</td>
<td>20</td>
<td>0.0357</td>
<td>12.3</td>
<td>10.4</td>
</tr>
</tbody>
</table>
Conclusion

• Characterized HE enabled cross-silo FL
• Designed an efficient HE batching scheme BatchCrypt
  o Codesigning quantization, coding, & batching
  o Online analytical clipping dACIQ
• Implemented, and evaluated it on AWS
  o Up to 99% cost reduction
Thank you for coming!

BatchCrypt is open sourced at https://github.com/marcoszh/BatchCrypt

Find me

https://marcoszh.github.io/

Graduating soon & seeking opportunities