Simulating Secure DNN Inference on Fully Homomorphic Encryption

Woosung Song

Seoul National University, Dept. of Computer Science and Engineering

December 10, 2021

Overview

Simulating Secure DNN Inference on FHE

- Simulating DNN inference as if it is on a secure FHE environment.
- Switching HE-incompatible ML components into approximate polynomials.
  - E.g., Conditional branches.
- Optimize and fine-tune the model to reach a higher accuracy.
- Compare the tradeoffs between accuracies and computational costs.

Homomorphic Encryption

Homomorphism Between Plaintexts and Ciphertexts

- Can perform operations on the ciphertexts without decryption
- Homomorphism between plaintext and ciphertext spaces

Let $c_1 \in Enc_{pk}(p_1), c_2 = Enc_{pk}(p_2)$.

$$Dec_{sk}(\mathrm{c}_1 \underline{+} \mathrm{c}_2) = \mathrm{p}_1 + \mathrm{p}_2$$
$$Dec_{sk}(\mathrm{c}_1 \underline{\times} \mathrm{c}_2) = \mathrm{p}_1 \times \mathrm{p}_2$$
$$Dec_{sk}(\underline{f}(\mathrm{c}_1, \mathrm{c}_2)) = f(\mathrm{p}_1, \mathrm{p}_2)$$

Fully Homomorphic Encryption

HEaaN (CKKS)

- Fully HE scheme for floating-point approximate arithmetics
- Supports $+$, $\times$, *rotate* operators for vectorized data packets (SIMD)

$$
\begin{gathered}
(a_1, a_2, \ldots, a_n) + (b_1, b_2, \ldots, b_n) = (a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n) \\
(a_1, a_2, \ldots, a_n) \times (b_1, b_2, \ldots, b_n) = (a_1 \times b_1, a_2 \times b_2, \ldots, a_n \times b_n) \\
rotate((a_1, a_2, \ldots, a_n), r) = (a_{r+1}, \ldots, a_n, a_1, \ldots, a_r)
\end{gathered}
$$

$$
\begin{gathered}
Dec_{sk}(\mathrm{c}_1 \underline{+} \mathrm{c}_2) = \mathrm{p}_1 + \mathrm{p}_2 \\
Dec_{sk}(\mathrm{c}_1 \underline{\times} \mathrm{c}_2) = \mathrm{p}_1 \times \mathrm{p}_2 \\
Dec_{sk}(\underline{rotate}(\mathrm{c}_1, r)) = rotate(\mathrm{p}_1, r)
\end{gathered}
$$

Noise Level & Bootstrapping

Noise Level

- Ciphertext's noise(error) becomes significantly larger after a multiplication
- Ciphertext lost its information when the level goes 0

Bootstrapping

- Operator in HEaaN that recovers the noise level
- Takes very very long time...
  - This is why one should care to reduce the cascaded multiplicative depths
- Enables Fully HE: No need to decrypt ciphertexts with low noise levels

Let $\mathcal{C}_l$ be a set of ciphertexts with l noise level.

$$
\begin{aligned}
Enc_{pk}(\mathrm{p}) &\in \mathcal{C}_{init} && \text{(Initial (large) noise level)} \\
\underline{+} &: \mathcal{C}_l \times \mathcal{C}_{l'} \to \mathcal{C}_{\min\{l,l'\}} \\
\underline{\times} &: \mathcal{C}_l \times \mathcal{C}_{l'} \to \mathcal{C}_{\min\{l,l'\}-1} \\
\underline{rotate} &: \mathcal{C}_l \times \mathbb{Z} \to \mathcal{C}_l \\
\underline{bootstrap} &: \mathcal{C}_{low} \to \mathcal{C}_{high}
\end{aligned}
$$

Secure DNN Inference on FHE

DNN Inference using FHE Scheme

- DNN inference from encrypted user data
- Using homormophic operators
  - $\underline{+}$, $\underline{\times}$, *rotate*
- Popular DNN components on inference stage:
  - `Matmul`, `Conv`, `Reshape`, `Sum` ...
  - `ReLU`, `MaxPool`, `Tanh`, ...

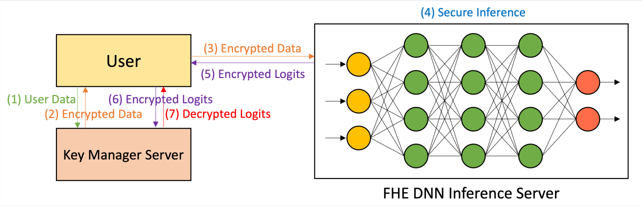

Figure: Secure DNN inference on FHE scheme

Secure DNN Inference on FHE

No Conditional Instruction in HEaaN?

- Don't know any information from the ciphertext
  - How can one implement '$ReLU(x) = \text{if } (x \geq 0) \text{ then } x \text{ else } 0$'?

Approximate Polynomials

- Approximate polynomials to implement the absolute function
  - $|x| \approx a_0 + a_1 x + a_2 x^2 + \cdots$
- Maximum degree of the approx polynomial determines noise level drop
  - Tradeoff between multiplicative depth v.s. accuracy

$$ReLU(x) = (x + |x|) \cdot 0.5$$
$$MaxPool(x_1, x_2) = \max(x_1, x_2) = 0.5 \cdot (x_1 + x_2 + |x_1 - x_2|)$$

Chebyshev's Approximation (1st Kind Expansion)

- Known to have low average error ($mean(|f - p|)$).
- Recursively generates polynomials.

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$

$$\sum_{k=1}^{n} T_i(x_k) T_j(x_k) = \begin{cases} 0 & \text{if } i \neq j \leq n \\ n & \text{if } i = j = 0 \\ n/2 & \text{if } 0 < i = j \leq n \end{cases} \qquad \text{where } x_k = -\cos(\frac{k\pi}{n})$$

- Compute approximation coefficients similar to Fourier transformation.

$$f(x) \approx \sum_{i=0}^{n} c_i T_i(x), \quad c_i = \frac{1}{d_i} \sum_{k=1}^{n} f(x_k) T_i(x_k), \quad d_i = \begin{cases} n & \text{if } i = 0 \\ n/2 & \text{if } i \neq 0 \end{cases}$$

Remez's Iterative Algorithm

Theorem (Chebyshev's Equioscillation theorem)

Among the approximate polynomial p of f s.t. $\deg(p) \leq n$, $||f - p||_\infty$ *on a domain* $[a, b]$ *is minimized iff* $\exists a \leq x_1 < x_2 < \cdots < x_{n+2} \leq b$ *s.t.* $f(x_i) - p(x_i) = (-1)^i \cdot ||f - p||_\infty$.

- Known to have minimal maximum error ($\max(|f - p|)$).
- Iteratively fetches x points and polynomial coefficients.
  - Randomly sample $n + 2$ ascending data points $(x_1, x_2, \ldots, x_{n+2})$ on $[-1, 1]$.
  - Solve a linear system of $n + 2$ equations, with $n + 2$ unknowns $c_0, \ldots, c_n, E$:

$$\left(\sum_{i=0}^{n} c_i x_k^i\right) + (-1)^k E = f(x_k)$$

  - Update $(x_1, x_2, \ldots, x_{n+2})$ to have local maximums on $||f(x) - p(x)||$, and repeat from the 2nd step.

Approximations of Absolute Function

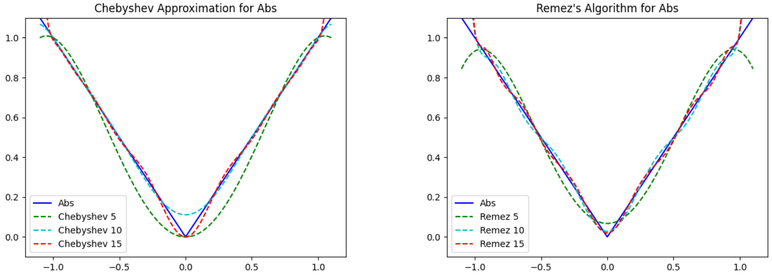

Figure: Chebyshev's (L), Remez's approximation of absolute function $y = |x|$ (R)

- Chebyshev has low average error, Remez's has low maximum error.
- Higher degree polynomials are more accurate.

Importance of Input Range Issue

Recall that $ReLU(x) = (x + |x|) \cdot 0.5$

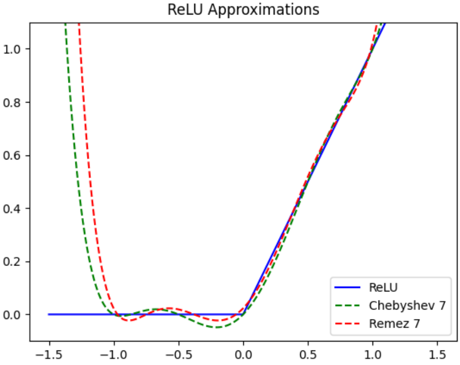

Figure: Approximations of ReLU. The function is highly inaccurate out of $[-1, 1]$ input range.

Experiment Environments

Target Datasets and Networks

- MNIST handwritten digit classification dataset.
- Simple convolutional classifier neural network.
  - `Conv2d`, `Linear`, `ReLU` and `MaxPool2d` in pytorch.
- Implemented with PyTorch for secure inference simulation.
  - Use approximation polynomials when inferencing.
  - Not necessary for training phase.

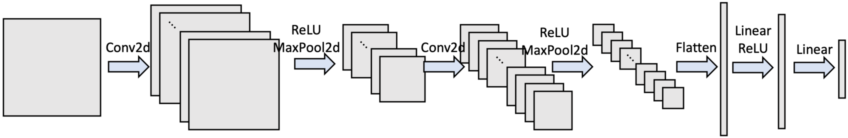

Figure: Classifier Network

Optimization & Fine-Tuning

ML Components and Losses

- Activation decay; to reduce input norms of approx. polynomials.

$$Loss(\theta, \mathrm{x}) = Loss_{orig}(\theta, \mathrm{x}) + \lambda_{act} \cdot \left(\frac{1}{N} \cdot \sum_{\mathrm{u} \in \text{inputs of approx.}} ||\mathrm{u}||_p \right)$$

- Negative slope ($u \in [0, 1)$) on ReLU (LeakyReLU)

$$LeakyReLU(x) = \begin{cases} x & \text{if } x \geq 0 \\ ux & \text{if } x < 0 \end{cases} = \left(x + \frac{1-u}{1+u} \cdot |x| \right) \cdot \frac{1+u}{2}$$

Fine-tuning

- Fine-tune with approximates
- Gradient descent training after switching HE-incompatible components into approximates

Experiment Pipelines

Common Training Pipelines

1. Train the network w/ small activation decay ($\lambda_{act,1} = 10^{-4}$) for 20 epochs
   - Learning rates: 0.1(10 epochs) $\rightarrow$ 0.01(15 epochs) $\rightarrow$ 0.001(20 epochs)
2. Train the network w/ large activation decay ($\lambda_{act,2} = 10^{-2}$) for 10 epochs
   - Learning rates: $10^{-3} \rightarrow 10^{-4} \rightarrow 10^{-5}$
3. Fine-tune the network by replacing HE-incompatible components into approximates for 10 epochs ($\lambda_{act,3} = 10^{-2}$).
   - Learning rates: $10^{-3} \rightarrow 10^{-4} \rightarrow 10^{-5}$

Control Variables

- Approximation methods (Chebyshev v.s. Remez) and degrees
- Using LeakyReLU instead of ReLU
- Using $L2$ or $L1$-norm as activation decay
- Final activation decay parameter scale ($\lambda_{act,3}$)

Github: `https://github.com/lego0901/fhe_secure_inference_simulation`

Results - LeakyReLU on L1 Activation Decay

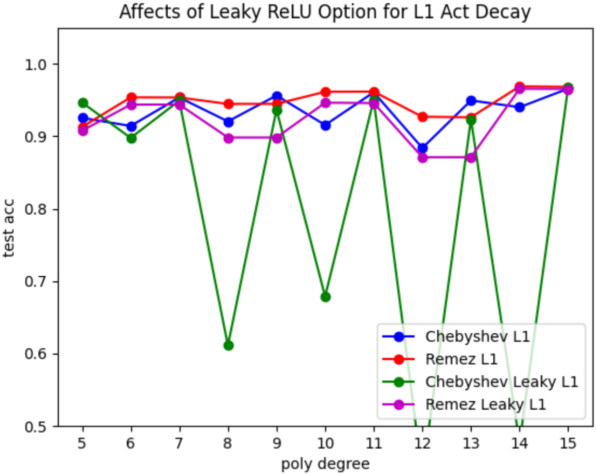

Figure: Affects of LeakyReLU. ReLU was more stable for $L1$ activation decay.

Results - LeakyReLU on L2 Activation Decay

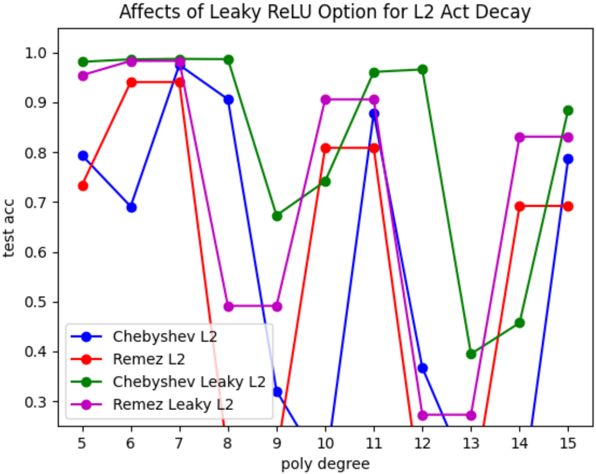

Figure: Affects of LeakyReLU. LeakyReLU was generally better for $L2$ activation decay.

Results - L1 or L2 Activation Decay Norms

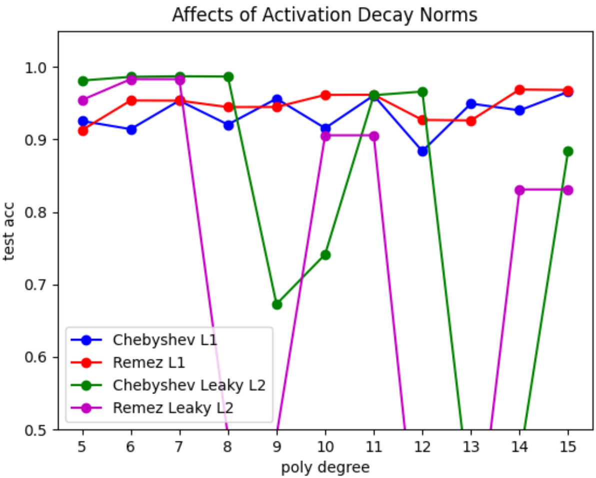

Figure: Affects of L1 and L2 activation decay norms. L2 gave impressive performances for some cases, but L1 was more stable.

Results - Activation Decay Strength ($\lambda_{act,3}$)

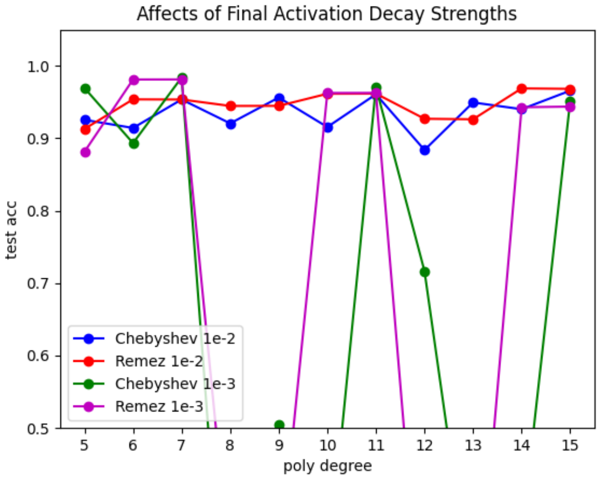

Figure: Affects of the last activation decay term ($\lambda_{act,3}$). 10^{-3} gave impressive performances for some cases, but 10^{-2} was more stable.

Results - Affects of The Fine-tune Phase

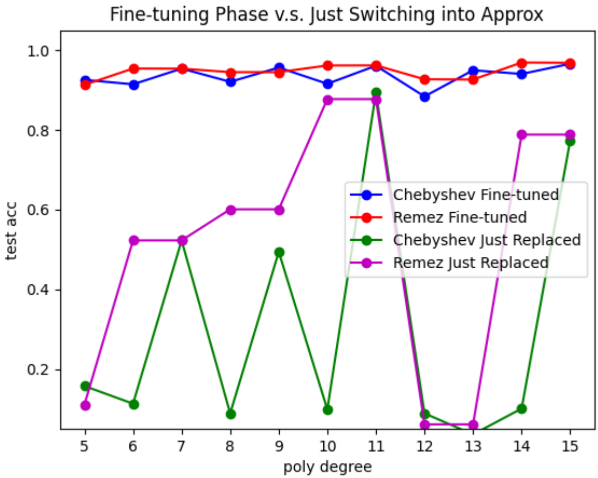

Figure: Affects of the fine-tuning phase. Just replacing ReLU and MaxPool2d into Approximates without fine-tuning gave poor performances.

Results - Affects of The Large Activation Decay Phase

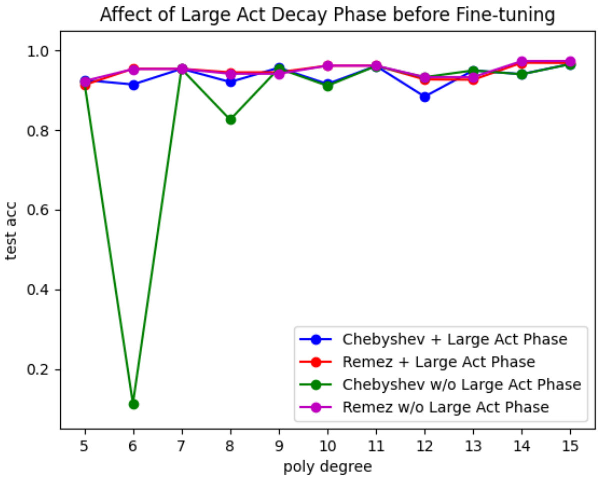

Figure: Affects of the (2nd) large activation decay phase. It provided more stable results for smaller approximation polynomial degrees.

Results - Activation Magnitude Profiles

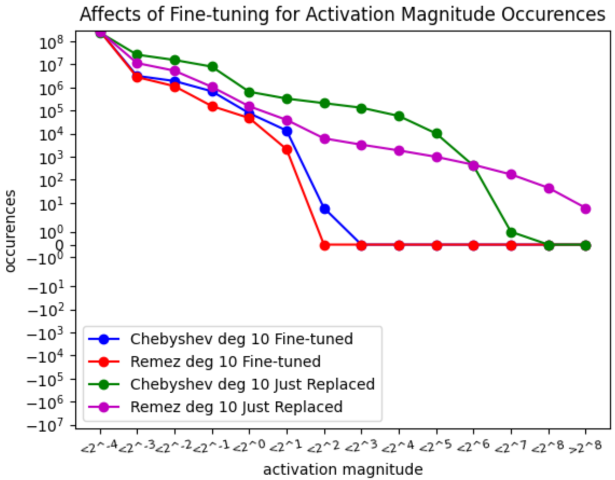

Figure: Activation magnitude profiles. The absolute value of each activation tensor element was investigated. Fine-tuning phase reduced the activation magnitudes in genernal.

Results - Top Accuracies Configs

Approximates	Act Decay	ReLU Layer	Accuracies
Chebyshev 7	L2, $\lambda_{act,3} = 10^{-2}$	LeakyReLU	98.71
Chebyshev 8	L2, $\lambda_{act,3} = 10^{-2}$	LeakyReLU	98.67
Chebyshev 6	L2, $\lambda_{act,3} = 10^{-2}$	LeakyReLU	98.63
Chebyshev 7	L1, $\lambda_{act,3} = 10^{-3}$	ReLU	98.39
Remez 7	L2, $\lambda_{act,3} = 10^{-2}$	LeakyReLU	98.31
Remez 6	L2, $\lambda_{act,3} = 10^{-2}$	LeakyReLU	98.31
Remez 10	L2, $\lambda_{act,3} = 10^{-2}$	LeakyReLU	98.21
Remez 11	L2, $\lambda_{act,3} = 10^{-2}$	LeakyReLU	98.21
Chebyshev 5	L2, $\lambda_{act,3} = 10^{-2}$	LeakyReLU	98.12
Remez 5	L1, $\lambda_{act,3} = 10^{-3}$	ReLU	98.12
Remez 6	L1, $\lambda_{act,3} = 10^{-3}$	ReLU	98.11

Table: Configurations with top accuracies

Results - Activation Magnitude Profiles

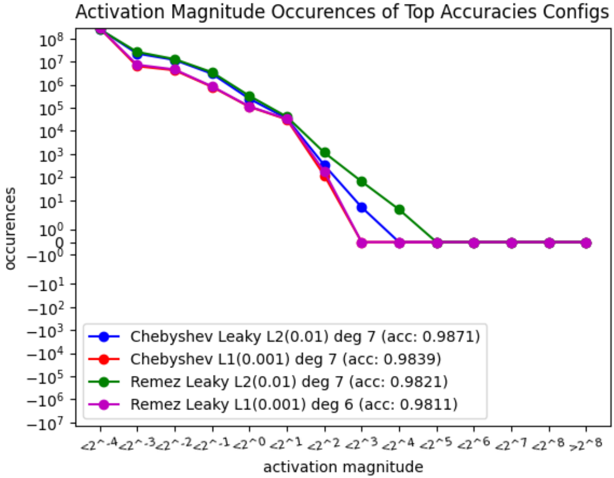

Figure: Activation magnitude profiles for remarkable configurations.

Conclusion

Summary

- Can switch HE-incompatible components into approximates.
- The accuracy drops were not critical in MNIST dataset. (98.71% in best)
- Fine-tuning phases were important to obtain better results.
  - With 'large activation decay' and 'fine-tuning' phases.
  - Fine-tuning stages reduced the magnitudes of activations, to fit into valid input ranges for the approximations; $[-1, 1]$.
- Low-degree polynomials approximations were enough. (Even better!)
  - About 6-7 degrees polynomials produced the best performances.
  - Better for all computational complexities. (resources, times, mult-depths)

Follow-up Studies

- Not 'simulating' a secure ML inference in FHE.
  - Need to implement `Matmul`, `Conv2d`, ...
- How to optimize these technique into more complex networks.
- How to make a secure training network? (not only the inference)

References I

[1] Jung Hee Cheon et al. "Bootstrapping for Approximate Homomorphic Encryption". In: *Advances in Cryptology – EUROCRYPT 2018*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Cham: Springer International Publishing, 2018, pp. 360–384. ISBN: 978-3-319-78381-9.

[2] Jung Hee Cheon et al. "Homomorphic Encryption for Arithmetic of Approximate Numbers". In: *Advances in Cryptology – ASIACRYPT 2017*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Cham: Springer International Publishing, 2017, pp. 409–437. ISBN: 978-3-319-70694-8.

[3] Jung Hee Cheon et al. "Numerical Method for Comparison on Homomorphically Encrypted Numbers". In: *Advances in Cryptology – ASIACRYPT 2019*. Ed. by Steven D. Galbraith and Shiho Moriai. Cham: Springer International Publishing, 2019, pp. 415–445. ISBN: 978-3-030-34621-8.

[4] Thore Graepel, Kristin Lauter, and Michael Naehrig. "ML Confidential: Machine Learning on Encrypted Data". In: *Information Security and Cryptology – ICISC 2012*. Ed. by Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–21. ISBN: 978-3-642-37682-5.

[5] M.H. Mudde, Alef E. Sterk, and van der Schaft Arjan. "Chebyshev approximation". In: (2017).