Using Intel SGX to Improve Private Neural Network Training and Inference

Ryan Karl
University of Notre Dame
Notre Dame, Indiana, United States
rkarl@nd.edu

Jonathan Takeshita
University of Notre Dame
Notre Dame, Indiana, United States
jtakeshi@nd.edu

Taeho Jung
University of Notre Dame
Notre Dame, Indiana, United States
tjung@nd.edu

ABSTRACT

In our modern data-driven society, the importance of leveraging machine learning (ML) algorithms to make critical business and government decisions continues to grow. To dramatically improve performance, such algorithms are often outsourced to the cloud, but within privacy and security sensitive domains, this presents several challenges to data owners for ensuring that their data is protected from malicious parties. One practical solution to these problems comes from Trusted Execution Environments (TEEs), which utilize several hardware technologies to isolate sensitive computations from untrusted software. This paper investigates a new technique utilizing a TEE to allow for the high performance training and execution of Deep Neural Networks (DNNs), an ML algorithm that has recently been used with great success in a variety of challenging tasks, including face and speech recognition.

* *All authors confirm that this manuscript is unpublished and revisions can be incorporated into the manuscript prior to their next submission for publication.*

KEYWORDS

Privacy-preserving Deep Learning, Intel SGX, Training and Inference

ACM Reference Format:
Ryan Karl, Jonathan Takeshita, and Taeho Jung. 2020. Using Intel SGX to Improve Private Neural Network Training and Inference. In *Symposium and Bootcamp on the Science of Security, April 7–8, 2020, Lawrence, KS.* ACM, New York, NY, USA, 2 pages. https://doi.org/XX.XXXX/XXXXXXX.XXXXXXX

1 BACKGROUND AND APPLICATIONS

Machine learning (ML) is increasingly used in a variety of data-driven decision making settings where security is of paramount importance. Given the growth in the popularity of cloud-based ML frameworks, which hide the complexity of ML algorithms from users, the number of attack points continues to grow. Trusted Execution Environments (TEEs), e.g, Intel SGX, ARM TrustZone, etc., offer a practical solution to this problem. TEEs use a variety of hardware and software technologies to isolate potentially sensitive code from untrusted applications, while still providing users with the functionality to attest their code was correctly executed without any tampering from an adversary (Figure 1) [1].

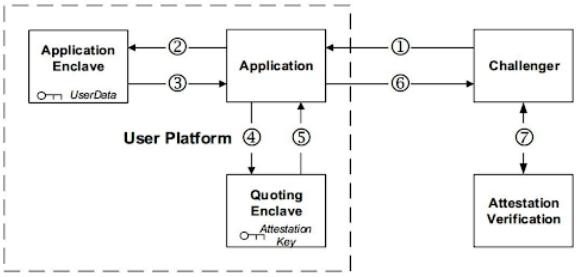

Fig. 1 Diagram of SGX Enclave Creation

Our approach, inspired by the Slalom framework and verifiable ASICs [6], is notably different from ML outsourcing based on purely cryptographic methods [2, 4, 5]. With this framework, computations are delegated between two co-located processors, which support an outsourcing protocol with efficiency that is orders-of-magnitude faster than existing work, while not requiring that the DNN be executed or trained fully in a TEE. By utilizing Freivald's algorithm [3], an efficient verifiable scheme that allows for outsourced matrix multiplication, we can support private ML training.

2 RELATED WORK

Previous work introduced Chiron, a system for privacy-preserving Machine Learning as a service (MLaaS) [4]. Note that the MLaaS framework assumes that an individual data provider will train their own ML model using hardware and algorithms owned by an untrusted party. Another similar work features a setup where several data providers train a shared ML model [2]. This paper does not focus on leveraging differential privacy to confront problems relating to model performance or data size, and instead is interested in hiding memory access patterns for SGX based training to avoid side-channel attacks. Our work is most similar to the Slalom framework [6], which allows for efficient privacy-preserving NN inference (but not training) using via a TEE. They leverage a cryptographic blinding method along with Freivald's algorithm [3], a method for delegating matrix multiplication to an untrusted GPU that can be verified for correctness after the computation is complete.

Contribution: Our work composes DL training in such a way that the one-time pad scheme can be used in tandem with the activation function so that training can be supported and delegated in an iterative fashion to a co-located GPU. Our approach will expedite private training and inference using a TEE by allowing more computations to occur in an untrusted GPU without needing

HotSos '20, April 7–8, 2020, Lawrence, KS

ACM ISBN XXX-X-XXXX-XXXX-X/XX/XX…$XX.00
https://doi.org/XX.XXXX/XXXXXXX.XXXXXXX

to frequently retrieve intermediate results for processing in the TEE.

3 PROPOSED METHODOLOGY

For our inference method, the linear layers are outsourced and then verified via Freivald's algorithm [3]. Then, the inputs of the linear layers are encrypted with a pre-computed pseudorandom stream (base on the one-time pad scheme) to guarantee their privacy. Notice that only two values are needed to continue the backpropagation algorithm for a given layer of the neural network: (1) the cost C_o where $C_o = a^{(L)} - y)^2$ where $a^{(L)}$ is the activation at layer L (L is the total number of layers) and y is the output of the neural network, and (2) the activation at layer L, denoted $a^{(L)} = \sigma(w^{(L)}a^{(L-1)} + b^{(L)}$ where $w^{(L)}$ is the weight at layer L, $a^{(L-1)}$ is the activation at layer $L-1$, and $b^{(L)}$ is the bias at layer L. For ease of notation we represent the input of the activation function in future sections as $z^{(L)} = w^{(L)}a^{(L-1)} + b^{(L)}$. We model the dependencies in Figure 2.

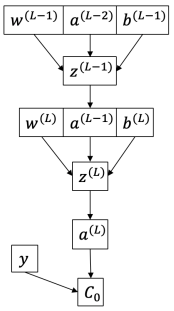

Fig. 2 Diagram of Backpropagation Dependencies

To compute backpropagation, we take the partial derivative of C_o with respect to $w^{(L)}$, with respect to $b^{(L)}$, and with respect to $a^{(L-1)}$. More formally, we would compute theses three equations:

$$\frac{\partial C_o}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}}\frac{\partial a^{(L)}}{\partial z^{(L)}}\frac{\partial C_o}{\partial a^{(L)}} = a^{(L-1)}\sigma'(z^{(L)})2(a^{(L)-y}) \quad (1)$$

$$\frac{\partial C_o}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}}\frac{\partial a^{(L)}}{\partial z^{(L)}}\frac{\partial C_o}{\partial a^{(L)}} = \sigma'(z^{(L)})2(a^{(L)-y}) \quad (2)$$

$$\frac{\partial C_o}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}}\frac{\partial a^{(L)}}{\partial z^{(L)}}\frac{\partial C_o}{\partial a^{(L)}} = w^{(L)}\sigma'(z^{(L)})2(a^{(L)-y}) \quad (3)$$

For our protocol we do the following:

(1) **Initialize:** The *TEE* is initialized with the model F and the input x_1, and the untrusted server S is initialized with F.
(2) **Preprocess:** For each element $i \in [1, n]$, the *TEE* generates random masking value r_i by randomly sampling $r_i \leftarrow F^{m_i}$, and then calculates the associated demasking value as $u_i = r_i W_i$. They then mask the input x_i as $\widetilde{x_i} = x_i + r_i$
(3) **Online:** For each element $i \in [1, n]$, the *TEE* sends $\widetilde{x_i}$ to the untrusted server S, who computes $\widetilde{y_i} = \widetilde{x_i}\widetilde{W_i}$, and sends $\widetilde{y_i}$ back to the *TEE*.
(4) **Verify:** For each element $i \in [1, n]$, the *TEE* checks that the untrusted S computed $\widetilde{y_i}$ correctly by computing $y_i = \widetilde{y_i} - u_i$ and calling Freivalds(y_i, x_i, W_i). If Freivalds successfully verifies that the $\widetilde{y_i}$ is correct, we continue (otherwise we know that S behaved maliciously and may abort). The *TEE* then computes the activation function as $x_{i+1} = \sigma(y_i)$.
(5) **Return:** The algorithm then returns y_n.

Notice that for the sigmoid function, which we write as $\sigma(k) = \frac{1}{1+e^{-k}}$ and the derivative as $\sigma'(k) = \frac{1}{1+e^{-k}}(1 - \frac{1}{1+e^{-k}}) = \sigma(k)(1 - \sigma(k))$, if we were to mask the input x with an additive noise r, for the first layer of the network we would have $w^{(0)}(x + r) + b^{(0)} = w^{(0)}x + w^{(0)}r + b^{(0)} = a^{(0)} + w^{(0)}r$. If we input this into σ, we have $\sigma(a^{(0)} + w^{(0)}r) = \frac{1}{1+e^{-(a^{(0)})+w^{(0)}r}} = \frac{1}{1+e^{-(a^{(0)})}e^{w^{(0)}r}}$. If we compute the derivative σ' we have $\sigma'(a^{(0)} + w^{(0)}r) = \frac{1}{1+e^{-a^{(0)}}e^{(w^{(0)}r)}}(1 - \frac{1}{1+e^{-a^{(0)}}e^{(w^{(0)}r)}})$. This means that we would need to compute the three equations described previously and somehow remove the random masking value (a nontrivial task):

$$\frac{\partial C_o}{\partial w^{(L)}} = a^{(L-1)}\frac{1}{1+e^{-a^{(L)}}e^{(w^{(L)}r)}}(1 - \frac{1}{1+e^{-a^{(L)}}e^{(w^{(L)}r)}})2(a^{(L)-y}) \quad (4)$$

$$\frac{\partial C_o}{\partial b^{(L)}} = \frac{1}{1+e^{-a^{(L)}}e^{(w^{(L)}r)}}(1 - \frac{1}{1+e^{-a^{(L)}}e^{(w^{(L)}r)}})2(a^{(L)-y}) \quad (5)$$

$$\frac{\partial C_o}{\partial a^{(L-1)}} = w^{(L)}\frac{1}{1+e^{-a^{(L)}}e^{(w^{(L)}r)}}(1 - \frac{1}{1+e^{-a^{(L)}}e^{(w^{(L)}r)}})2(a^{(L)-y}) \quad (6)$$

Note the Arctan function is $\sigma(k) = tan^{-1}(k)$ and its derivative is $\sigma'(k) = \frac{1}{k^2+1}$), and we would need to remove the random mask from the following equations:

$$\frac{\partial C_o}{\partial w^{(L)}} = a^{(L-1)}\frac{1}{(w^{(L)}r + a^{(L)})^2 + 1}2(a^{(L)-y}) \quad (7)$$

$$\frac{\partial C_o}{\partial b^{(L)}} = \frac{1}{(w^{(L)}r + a^{(L)})^2 + 1}2(a^{(L)-y}) \quad (8)$$

$$\frac{\partial C_o}{\partial a^{(L-1)}} = w^{(L)}\frac{1}{(w^{(L)}r + a^{(L)})^2 + 1}2(a^{(L)-y}) \quad (9)$$

Note the SoftMax function is $\sigma(k) = log(1+e^k)$ and its derivative is $\sigma'(k) = (\frac{1}{1+e^{-x}})$, so we would need to remove the random mask from the following equations:

$$\frac{\partial C_o}{\partial w^{(L)}} = a^{(L-1)}\frac{1}{1+e^{-a^{(L)}}e^{(w^{(L)}r)}}2(a^{(L)-y}) \quad (10)$$

$$\frac{\partial C_o}{\partial b^{(L)}} = \frac{1}{1+e^{-a^{(L)}}e^{(w^{(L)}r)}}2(a^{(L)-y}) \quad (11)$$

$$\frac{\partial C_o}{\partial a^{(L-1)}} = w^{(L)}\frac{1}{1+e^{-a^{(L)}}e^{(w^{(L)}r)}}2(a^{(L)-y}) \quad (12)$$

REFERENCES

[1] McKeen et al. 2013. Innovative instructions and software model for isolated execution.
[2] Ohrimenko et al. 2016. Oblivious multi-party machine learning on trusted processors. , 619–636 pages.
[3] Rusins Freivalds. 1977. Probabilistic Machines Can Use Less Running Time. , 842 pages.
[4] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. 2018. Chiron: Privacy-preserving machine learning as a service.
[5] Nick Hynes, Raymond Cheng, and Dawn Song. 2018. Efficient deep learning on multi-source private data.
[6] Florian Tramer and Dan Boneh. 2018. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware.