

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

The importance of leveraging machine learning (ML) algorithms to make critical business and government decisions continues to grow. To improve performance, such algorithms are often outsourced to the cloud, but within privacy sensitive domains this presents several challenges for ensuring data is protected from malicious parties. One practical solution to these problems comes from Trusted Execution Environments (TEEs), which utilize 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 speech and face recognition.

KEYWORDS

Privacy-preserving Deep Learning, Intel SGX, Neural Networks

ACM Reference Format:

Ryan Karl, Jonathan Takeshita, and Taeho Jung. 2020. Using Intel SGX to Improve Private Neural Network Training and Inference. In *Hot Topics in the Science of Security Symposium (HotSoS '20)*, April 7–8, 2020, Lawrence, KS, USA. ACM, New York, NY, USA, 2 pages. <https://doi.org/10.1145/3384217.3386399>

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]. 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.

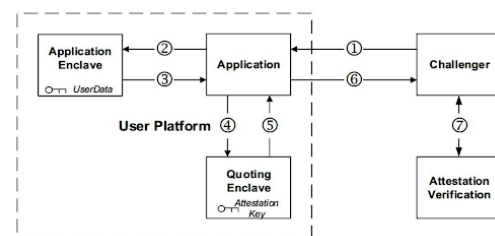


Fig. 1 Diagram of SGX Enclave Creation

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 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 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.

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 to 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 activation at layer l : $a^{(l)} = \sigma(w^{(l)} a^{(l-1)} + b^{(l)})$, 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 . (2) The cost at the final layer l : $C_o = (a^{(l)} - y)^2$, where y is the label of the input.

The input of the activation function at layer l is denoted as $z^{(l)} = w^{(l)}a^{(l-1)} + b^{(l)}$. We model the dependencies in Figure 2.

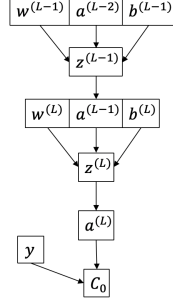


Fig. 2 Diagram of Backpropagation Dependencies

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 these three equations:

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

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

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

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) **Encode:** 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 $\tilde{x}_i = x_i + r_i$.
- (3) **Outsource to GPU:** For each element $i \in [1, n]$, the TEE sends \tilde{x}_i to the untrusted server S , who computes $\tilde{y}_i = \tilde{x}_i \tilde{W}_i$, and sends \tilde{y}_i back to the TEE.
- (4) **Verify:** For each element $i \in [1, n]$, the TEE checks that the untrusted S computed \tilde{y}_i correctly by computing $y_i = \tilde{y}_i - u_i$ and calling Freivalds(y_i, x_i, W_i). If Freivalds successfully verifies that the \tilde{y}_i is correct, we continue. Otherwise, we abort with an error. The TEE then computes the activation function as $x_{i+1} = \sigma(y_i)$.
- (5) **Decode:** The algorithm then returns y_n . We can continue iterating through steps 2-5 for each layer until we reach the final layer of the NN.
- (6) **Update:** We then update the weights via backpropagation using the outsourcing described in steps 2-5 (we traverse the NN backwards) to mask each gradient update computation outsourced to the GPU until reaching the desired accuracy.

To build an additive masking scheme as described above, we might use the sigmoid function $\sigma(k) = \frac{1}{1+e^{-k}}$ as the activation function. The derivative of it is $\sigma'(k) = \frac{1}{1+e^{-k}} (1 - \frac{1}{1+e^{-k}}) = \sigma(k)(1 - \sigma(k))$, so 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:

$$\sigma'(a^{(0)} + w^{(0)}r) = \frac{1}{1 + e^{-a^{(0)}} e^{w^{(0)}r}} \left(1 - \frac{1}{1 + e^{-a^{(0)}} e^{w^{(0)}r}}\right)$$

This means that we would need to compute the three equations described previously and somehow remove the random masking value from (a nontrivial task):

$$\frac{\partial C_o}{\partial w^{(l)}} = a^{(l-1)} \frac{1}{1 + e^{-a^{(l)}} e^{w^{(l)}r}} \left(1 - \frac{1}{1 + e^{-a^{(l)}} e^{w^{(l)}r}}\right) 2(a^{(l)} - y)$$

$$\frac{\partial C_o}{\partial b^{(l)}} = \frac{1}{1 + e^{-a^{(l)}} e^{w^{(l)}r}} \left(1 - \frac{1}{1 + e^{-a^{(l)}} e^{w^{(l)}r}}\right) 2(a^{(l)} - y)$$

$$\frac{\partial C_o}{\partial a^{(l-1)}} = w^{(l)} \frac{1}{1 + e^{-a^{(l)}} e^{w^{(l)}r}} \left(1 - \frac{1}{1 + e^{-a^{(l)}} e^{w^{(l)}r}}\right) 2(a^{(l)} - y)$$

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 (a more intuitive task):

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

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

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

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

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

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

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

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.
- [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.