Privacy Preserving Machine Learning

SecureML: An overview and discussion

Soham De

26 July 2021

WiCS Research Reading Group

Contents

Privacy Preserving Machine Learning

Privacy Preserving Machine Learning

An Overview

Traditional Machine Learning

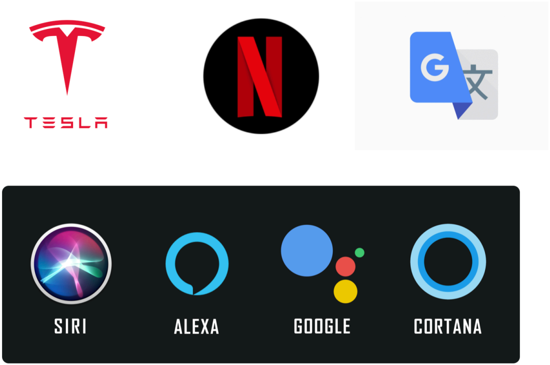


Figure 1: Present-Day applications of Machine Learning

But *why* preserve privacy in ML?

The Need for Preserving Privacy

- Primary goal of real-world mission-critical ML systems is **accuracy**

The Need for Preserving Privacy

- Primary goal of real-world mission-critical ML systems is accuracy
- Accuracy is contingent on :
  - High compute power
  - Availability of data from varied sources

The Need for Preserving Privacy

- Primary goal of real-world mission-critical ML systems is **accuracy**
- Accuracy is contingent on :
  - High compute power
  - Availability of data from varied sources
- Accumulating data from different and various sources is **not practical**

The Need for Preserving Privacy

- Primary goal of real-world mission-critical ML systems is **accuracy**
- Accuracy is contingent on :
  - High compute power
  - Availability of data from varied sources
- Accumulating data from different and various sources is **not practical**

EU's GDPR (Article 46)

"...a controller or processor may transfer personal data to a third country or an international organisation only if the controller or processor has provided **appropriate safeguards**..." (abridged)

Privacy Preserving Machine Learning

Existing Literature

Popular Approaches

Dominant approaches in literature are:

- Hardware Based
    - Trusted Enclaves (SGX)

Popular Approaches

Dominant approaches in literature are:

- Hardware Based
  - Trusted Enclaves (SGX)
- Software Based
  - Fully Homomorphic Encryption
  - Secure Multiparty Computation

This talk focuses on approaches using Secure Multiparty Computation

Preliminaries

Preliminaries

Machine Learning Primitives

Regression

Given n training data samples x_i each containing d features and the corresponding output labels y_i, **regression** is a statistical process to learn a function g such that $g(x_i) = y_i$.

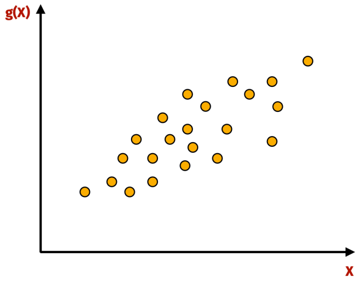


Figure 2: Simplified Illustration (N=22)

Linear Regression

- In **linear regression**, the function g is assumed to be **linear** and can be represented as the inner product of x_i with the coefficient vector w: $g(x_i) = \sum_{j=1}^{d} x_{ij} w_j = x_i \cdot w$

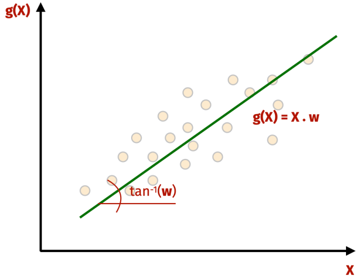


Figure 3: Simplified Linear Regression (N=22)

Linear Regression

- To learn the coefficient vector w, a **cost function** $C(w)$ is defined and w is calculated by the optimization $\text{argmin}_w C(w)$. In linear regression, a commonly used cost function is $C(w) = \frac{1}{n} C_i(w)$, where $C_i(w) = \frac{1}{2}(x_i \cdot w - y_i)^2$

Linear Regression

- To learn the coefficient vector w, a **cost function** $C(w)$ is defined and w is calculated by the optimization $\text{argmin}_w C(w)$. In linear regression, a commonly used cost function is $C(w) = \frac{1}{n} C_i(w)$, where $C_i(w) = \frac{1}{2}(x_i \cdot w - y_i)^2$
- The solution for this optimization problem can be computed by solving the linear system $(X^T \times X) \times w = X^T \times Y$

Stochastic Gradient Descent

- In each iteration, a sample (x_i, y_i) is selected and a coefficient w_j is updated as:

$$w_j := w_j - \alpha \frac{\delta C_i(w)}{\delta w_j}$$

  where α is a learning rate defining the magnitude to move towards the minimum in each iteration.

Stochastic Gradient Descent

- In each iteration, a sample (x_i, y_i) is selected and a coefficient w_j is updated as:

$$w_j := w_j - \alpha \frac{\delta C_i(w)}{\delta w_j}$$

  where α is a learning rate defining the magnitude to move towards the minimum in each iteration.
- Substituting the cost function of linear regression, the formula becomes $w_j := w_j - \alpha(x_i \cdot w - y_i)x_{ij}$

Mini-Batches

- In practice, instead of selecting one sample of data per iteration, a small batch of samples are selected randomly and w is updated by averaging the partial derivatives of all samples - this can now benefit from **vectorization**

Mini-Batches

- In practice, instead of selecting one sample of data per iteration, a small batch of samples are selected randomly and w is updated by averaging the partial derivatives of all samples - this can now benefit from **vectorization**
- With Mini-batches, the update equation becomes:

$$w_j := w_j - \frac{1}{|B|}\alpha X_B^T \times (X_B \times w - Y_B)$$

Logistic Regression

- In classification problems with two classes, the output label y is binary

Logistic Regression

- In classification problems with two classes, the output label y is binary
- Therefore, an **activation function** f is applied on top of the inner product and the relationship is expressed as: $g(x_i) = f(x_i \cdot w)$

Logistic Regression

- In classification problems with two classes, the output label y is binary
- Therefore, an **activation function** f is applied on top of the inner product and the relationship is expressed as: $g(x_i) = f(x_i \cdot w)$
- In **logistic** regression, the activation function is the **logistic** function, i.e:

$$f(u) = \frac{1}{1 + e^{-u}}$$

Logistic Regression

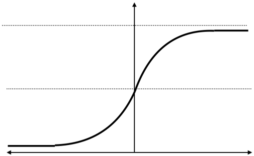


Figure 4: Sigmoid (Logistic) Activation Function

- Since the only difference between Linear and Logistic Regression is the activation function in forward propagation, the update function can be simply modified as:

$$w_j := w_j - \frac{1}{|B|}\alpha X_B^T \times (f(X_B \times w) - Y_B)$$

Neural Networks

- Neural networks are a **generalization of regression** to learn more complicated relationships between high dimensional input and output data

Neural Networks

- Neural networks are a **generalization of regression** to learn more complicated relationships between high dimensional input and output data
- Each node in the hidden layer and the output layer is an **instance of regression** and is associated with an **activation** function and a coefficient vector

Neural Networks

- Neural networks are a **generalization of regression** to learn more complicated relationships between high dimensional input and output data
- Each node in the hidden layer and the output layer is an **instance of regression** and is associated with an **activation** function and a coefficient vector
- ReLU $\left(f(u) = max(0, u)\right)$ is a widely used activation function

Neural Networks

- Neural networks are a **generalization of regression** to learn more complicated relationships between high dimensional input and output data
- Each node in the hidden layer and the output layer is an **instance of regression** and is associated with an **activation** function and a coefficient vector
- ReLU ($f(u) = max(0, u)$) is a widely used activation function
- For classification problems with multiple classes, usually a **softmax** function: $f(u_i) = \frac{e^{-u_i}}{\sum_{i=1}^{d_m} e^{-u_i}}$ is applied at the last (output layer)

Preliminaries

Cryptography Primitives

Secure Function Evaluation

Problem Definition

Given n (≥ 2) parties with private inputs X_i, where $i \in \{0, n\}$, evaluate a (publicly known) function F where, $F(X_0, X_1, \ldots X_n)$

- A naive solution involves **Secure Outsourced Computing** (SOC) and a trusted third-party, as will be described soon

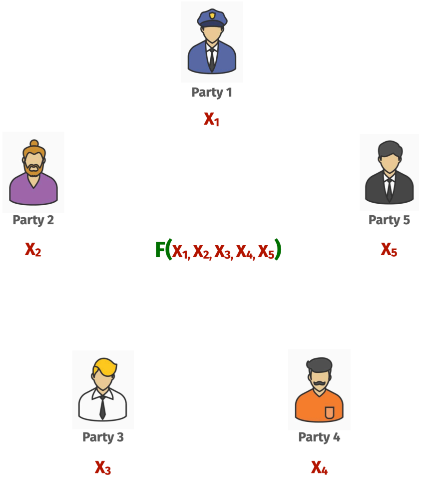
Secure Function Evaluation

Problem Definition

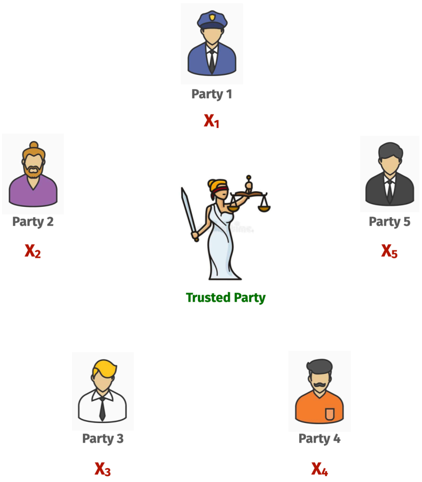
Given n (≥ 2) parties with private inputs X_i, where $i \in \{0, n\}$, evaluate a (publicly known) function F where, $F(X_0, X_1, \ldots X_n)$

- A naive solution involves **Secure Outsourced Computing** (SOC) and a trusted third-party, as will be described soon
- In the absence of such a third-party, **Secure Multiparty Computation** (S-MPC) may provide a solution

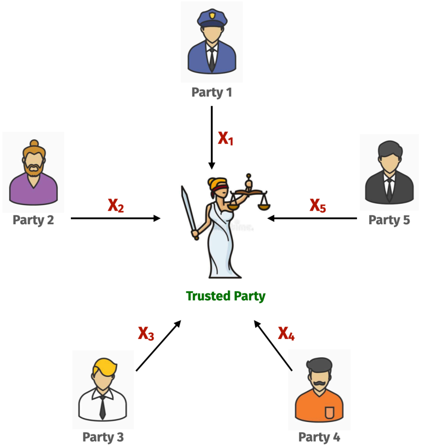
Secure Function Evaluation



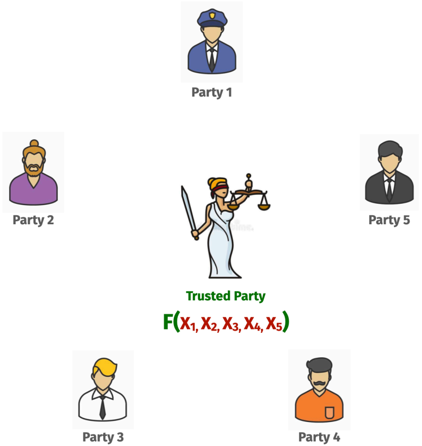
Secure Function Evaluation



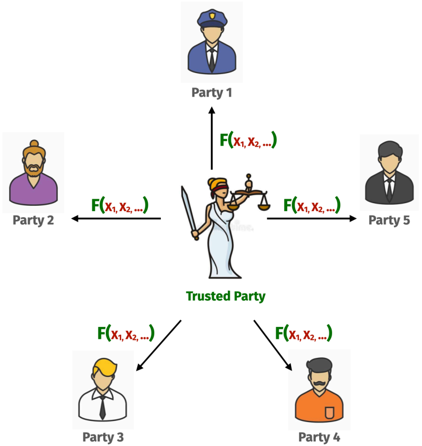
Secure Function Evaluation



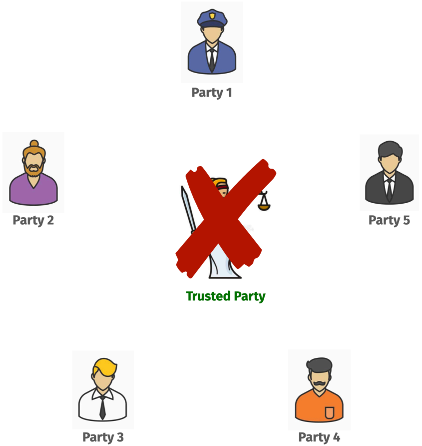
Secure Function Evaluation



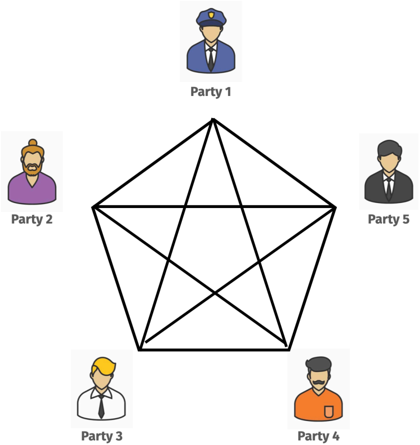
Secure Function Evaluation



Secure Function Evaluation



Secure Function Evaluation



Crypto Protocols

- Linear Layers: Multiplications, Convolutions …
    - Secure multiplication
    - Truncation

Crypto Protocols

- **Linear Layers:** Multiplications, Convolutions …
    - Secure multiplication
    - Truncation
- **Activation Functions:** ReLU, Maxpool …
    - Comparison (Garbled Circuits)

Crypto Protocols

- Linear Layers: Multiplications, Convolutions ...
  - Secure multiplication
  - Truncation
- Activation Functions: ReLU, Maxpool ...
  - Comparison (Garbled Circuits)
- Transcendental Functions: Sigmoid, tanh ...
  - More complex (refer to SIRNN, IEEE S&P'21)

SecureML

SecureML

An Overview

Novel Contributions

- New protocols for linear regression, logistic regression and neural network training

Novel Contributions

- New protocols for **linear regression**, **logistic regression** and **neural network training**
- **Truncation** for handling arithmetic on shared decimals

Novel Contributions

- New protocols for **linear regression**, **logistic regression** and **neural network training**
- **Truncation** for handling arithmetic on shared decimals
- A new MPC-friendly **activation function**

Linear Regression with Secure ML

Recall, the Linear Regression equation:

$$w_j := w_j - \frac{1}{|B|}\alpha X_B^T \times (X_B \times w - Y_B)$$

Linear Regression with Secure ML

Recall, the Linear Regression equation:

$$w_j := w_j - \frac{1}{|B|}\alpha X_B^T \times (X_B \times w - Y_B)$$

Things we need to solve are:

- An **addition** (subtraction) protocol on additive shares

Linear Regression with Secure ML

Recall, the Linear Regression equation:

$$w_j := w_j - \frac{1}{|B|}\alpha X_B^T \times (X_B \times w - Y_B)$$

Things we need to solve are:

- An **addition** (subtraction) protocol on additive shares
- A **multiplication** protocol on additive shares

Linear Regression with Secure ML

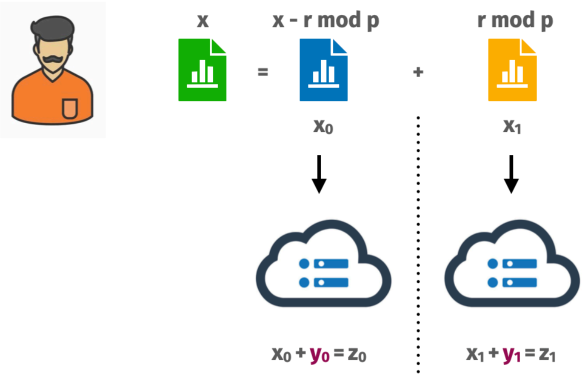
Recall, the Linear Regression equation:

$$w_j := w_j - \frac{1}{|B|}\alpha X_B^T \times (X_B \times w - Y_B)$$

Things we need to solve are:

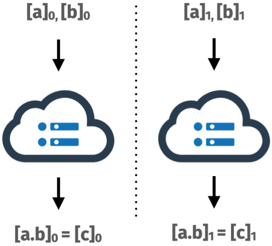
- An **addition** (subtraction) protocol on additive shares
- A **multiplication** protocol on additive shares
- Handling **Floating Point Inputs**

Addition on Secret Shares



$$z_0 + z_1 = z = x + y$$

Multiplication on Secret Shares



$[c]_0 + [c]_1 = c = a.b$

Multiplication on Secret Shares

$[a]_0, [b]_0$	$[a]_1, [b]_1$
$[u]_0, [v]_0, [z]_0$	$[u]_1, [v]_1, [z]_1$

$$[a]_0 - [u]_0, [b]_0 - [v]_0 \longrightarrow$$

$$\longleftarrow [a]_0 - [u]_0, [b]_0 - [v]_0$$

$e = a - u$	$e = a - u$
$f = b - v$	$f = b - v$
$[c]_0 = -ef + [a]_0 f + e[b]_0 + [z]_0$	$[c]_1 = [a]_1 f + e[b]_1 + [z]_1$

$$[c]_0 + [c]_1 = c = a.b$$

But what about floating point inputs?

Dealing with floating points

- **Mapping Decimals to Integers:** A decimal floating-point number x is mapped into an integer as $x' = 2^{l_D}x$ where x has at most l_D bits in the fractional part. In our implementation, we set $l_D = 13$ for double floating-point precision.

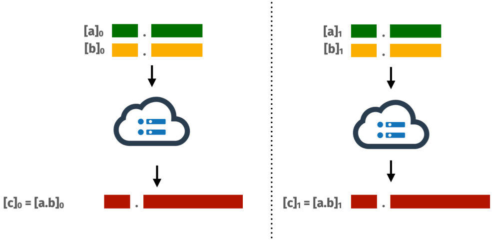
Dealing with floating points

- **Mapping Decimals to Integers:** A decimal floating-point number x is mapped into an integer as $x' = 2^{l_D}x$ where x has at most l_D bits in the fractional part. In our implementation, we set $l_D = 13$ for double floating-point precision.
- **Handling Negative Decimals:** We also note that if a decimal number z is negative, it will be represented in the field as $2^l - |z|$, where $|z|$ is its absolute value and the truncation operation (described later) changes to $z = 2^l - \lfloor |z| \rfloor$

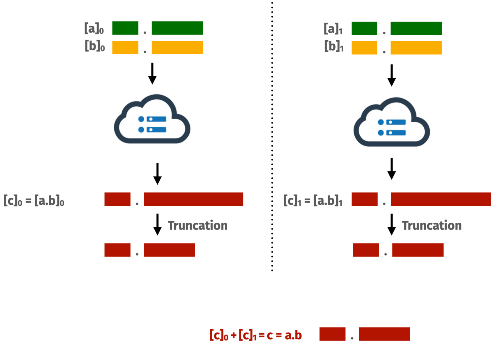
Dealing with floating points

- **Truncation Operation:** Consider the fixed-point multiplication of two decimal numbers x and y with at most l_D bits in the fractional part. We first transform the numbers to integers by letting $x' = 2^{l_D}x$ and $y' = 2^{l_D}y$ and then multiply them to obtain the product $z = x' \cdot y' = 2^{l_D}x \cdot 2^{l_D}y = 2^{2l_D}x \cdot y$. Note that z has at most $2l_D$ bits representing the fractional part of the product, so we simply truncate the last l_D bits of z such that it has at most l_D bits representing the fractional part. Mathematically speaking, if z is decomposed into two parts $z = z_1 \cdot 2l_D + z_2$, where $0 \leq z_2 < 2l_D$, then the truncation results is z_1. We denote this truncation operations by $\lfloor z \rfloor$

Truncation



Truncation



Privacy Preserving Linear Regression

Inputs: $\langle X \rangle, \langle Y \rangle, \langle U \rangle, \langle V \rangle, \langle Z \rangle, \langle V' \rangle, \langle Z' \rangle$

$$S_i \leftarrow \langle E \rangle_i = \langle X \rangle_i - \langle U \rangle_i$$

$$\text{Obtain } E = \mathrm{Rec}^A(\langle E \rangle_0, \langle E \rangle_1)$$

$$S_i \leftarrow \langle F_j \rangle_i = \langle w \rangle_i - \langle V_j \rangle_i$$

$$\text{Obtain } F = \mathrm{Rec}^A(\langle F \rangle_0, \langle F \rangle_1)$$

$$S_i \leftarrow \langle Y^*_{B_j} \rangle_i = i \cdot E_{B_j} \cdot F_j + \langle X_{B_j} \rangle_i \cdot F_j + E_{B_j} \cdot \langle w \rangle_i + \langle Z_j \rangle_i$$

$$S_i \leftarrow \langle D_{B_j} \rangle_i = \langle Y^*_{B_j} \rangle_i - \langle Y_{B_j} \rangle_i$$

$$S_i \leftarrow \langle F'_j \rangle_i = \langle D_{B_j} \rangle_i - \langle V'_j \rangle_i$$

$$\text{Obtain } F' = \mathrm{Rec}^A(\langle F' \rangle_0, \langle F' \rangle_1)$$

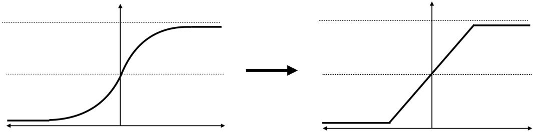
$$S_i \leftarrow \langle \Delta \rangle_i = i \cdot E^T_{B_j} \cdot F'_j + \langle X^T_{B_j} \rangle_i \cdot F'_j + E^T_{B_j} \cdot \langle D_{B_j} \rangle_i + \langle Z'_j \rangle_i$$

$$S_i \leftarrow \langle w \rangle_i := \; \leftarrow \langle w \rangle_i = \frac{\alpha}{|B|} \lfloor \langle \Delta \rangle_i \rfloor$$

Output: $w = \mathrm{Rec}^A(\langle w \rangle_0, \langle w \rangle_1)$

New Activation Function

$$f(x) = \begin{cases} 0 & \text{if } x < -\frac{1}{2} \\ x + \frac{1}{2} & \text{if} - \frac{1}{2} \leq x \leq \frac{1}{2} \\ 1 & \text{if } x > \frac{1}{2} \end{cases}$$



fin.