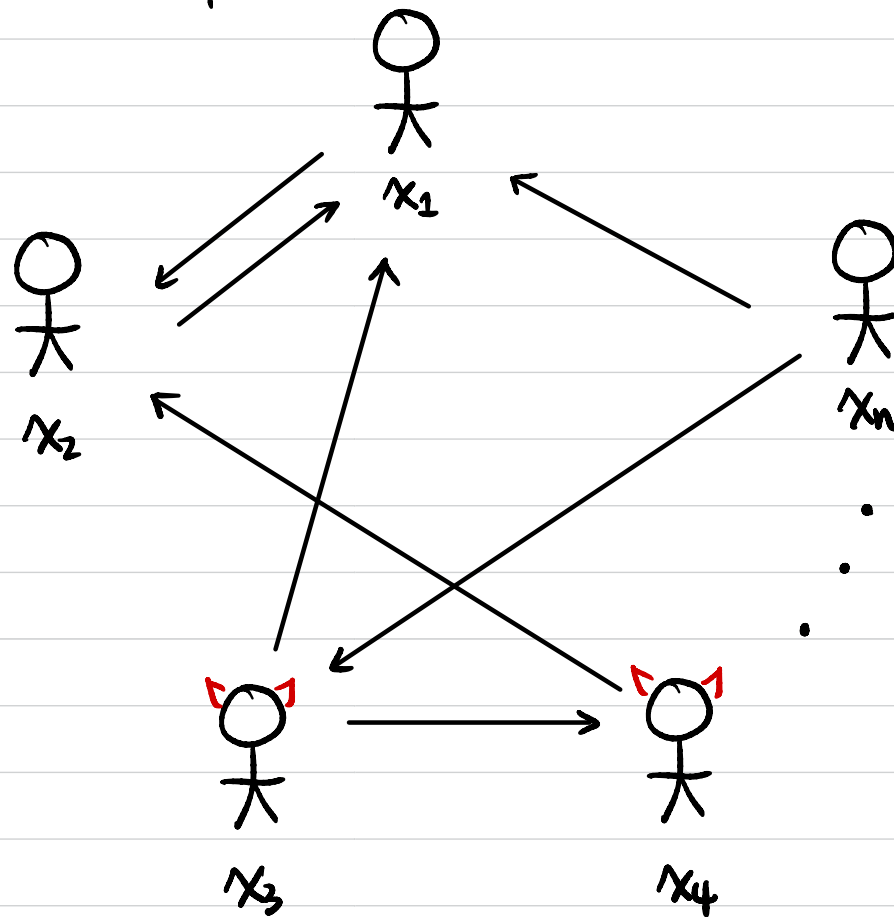


CSCI 1515 Applied Cryptography

This Lecture:

- GMW: Semi-Honest MPC for Any Function (Continued)
- GMW Compiler: Malicious MPC for Any Function
- Privacy-Preserving Machine Learning

Secure Multi-Party Computation (MPC)



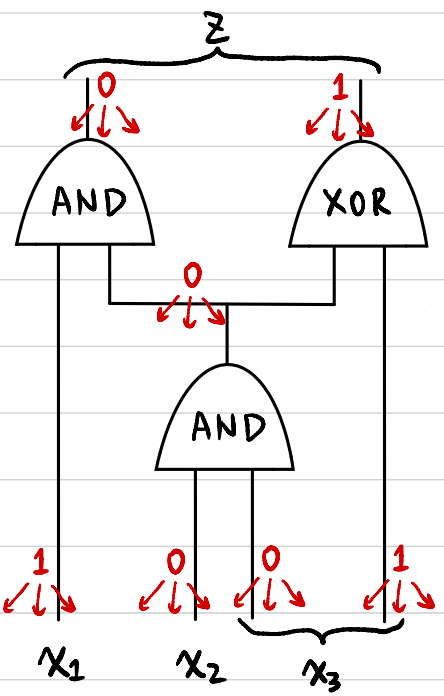
$$z = f(x_1, \dots, x_n)$$

Allowed adversarial behavior:

- **Semi-honest:** Follow the protocol description honestly.
- **Malicious:** Can deviate arbitrarily from the protocol description.

MPC for any function with $t \leq n-1$ (GMW)

Each party P_i holds a random share $v_i^w \in \{0,1\}$ s.t. $\bigoplus_{i=1}^n v_i^w = v^w$



Inputs:

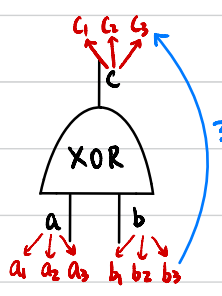
For each input wire w :

If it's from party P_k with input value $v^w \in \{0,1\}$.

P_k randomly samples $v_i^w \in \{0,1\}$ s.t. $\bigoplus_{i=1}^n v_i^w = v^w$

→ Sends v_i^w to party P_i .

XOR gates:

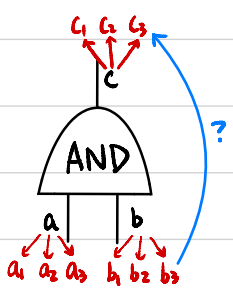


GIVEN: $\bigoplus_{i=1}^n a_i = a$ $\bigoplus_{i=1}^n b_i = b$

WANT: $\{c_i\}$ s.t. $\bigoplus_{i=1}^n c_i = c = a \oplus b$

$$c_i = a_i \oplus b_i$$

AND gates:



GIVEN: $\bigoplus_{i=1}^n a_i = a$ $\bigoplus_{i=1}^n b_i = b$

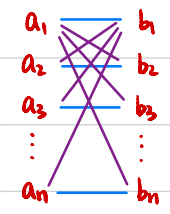
WANT: $\{c_i\}$ s.t. $\bigoplus_{i=1}^n c_i = c = a \cdot b$

$$c_i = a_i \cdot b_i \oplus \sum_{j \neq i} r_{i,j}^{(1)} \oplus \sum_{j \neq i} r_{j,i}^{(2)}$$

$$a \cdot b = \left(\sum_{i=1}^n a_i \right) \cdot \left(\sum_{i=1}^n b_i \right) \pmod{2}$$

$$= \left(\sum_{i=1}^n a_i \cdot b_i \right) + \left(\sum_{i \neq j} a_i \cdot b_j \right) \pmod{2}$$

\uparrow P_i locally \leftarrow Reshare $r_{i,j}^{(1)}$ $r_{j,i}^{(2)}$



Outputs:

For each output wire w :

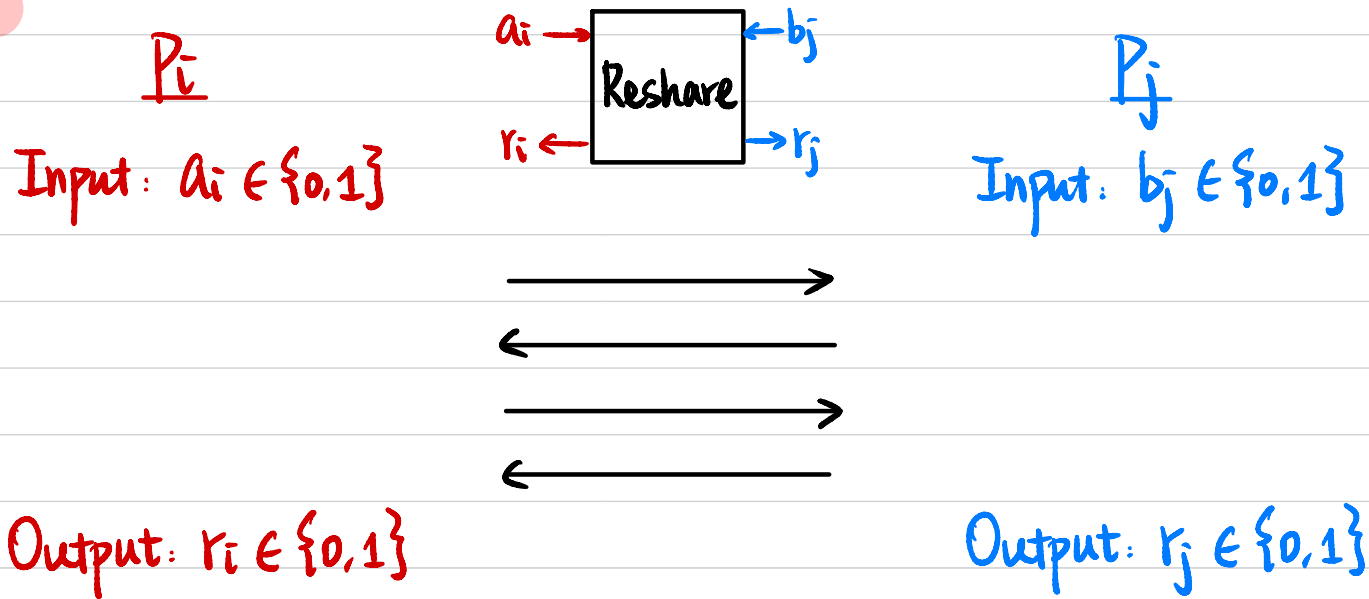
Each party P_i holds a random share $v_i^w \in \{0,1\}$

→ Sends v_i^w to all parties

Each party computes the value $v^w = \bigoplus_{i=1}^n v_i^w$

MPC for any function with $t \leq n-1$ (GMW)

Reshare:



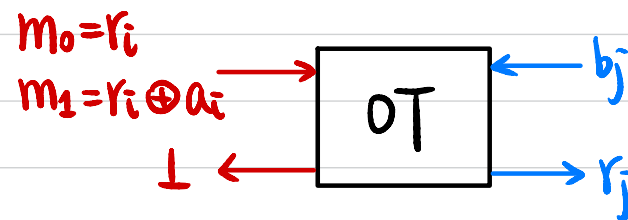
WANT: Random $r_i, r_j \in \{0,1\}$ st. $r_i \oplus r_j = a_i \cdot b_j$

1) P_i randomly samples $r_i \leftarrow \{0,1\}$

2) How to let P_j learn r_j st. $r_i \oplus r_j = a_i \cdot b_j$?

If $b_j = 0 \Rightarrow r_j = r_i$

If $b_j = 1 \Rightarrow r_j = r_i \oplus a_i$



MPC for any function with $t \leq n-1$ (GMW)

Computational Complexity?

$O(\#AND \cdot n)$ per party

Communication Complexity?

$O(\#AND \cdot n^2)$ total

Round Complexity? $O(\text{depth of AND gates})$



What could go wrong against malicious adversaries?

GMW Compiler

Given a semi-honest protocol:

Once inputs & randomness are fixed, protocol is deterministic.

Step 1: Each party P_i commits to its input x_i & randomness r_i to be used in the semi-honest protocol.

Step 2: Run semi-honest protocol.

Along with every message, prove in ZK that the message is computed correctly (based on its input, randomness, transcript so far)

Semi-honest OT
(OT protocol that's secure
against semi-honest adv.)

Yao's Garbled
Circuit

GMW

Semi-honest ZPC
for any function

Semi-honest MPC
for any function

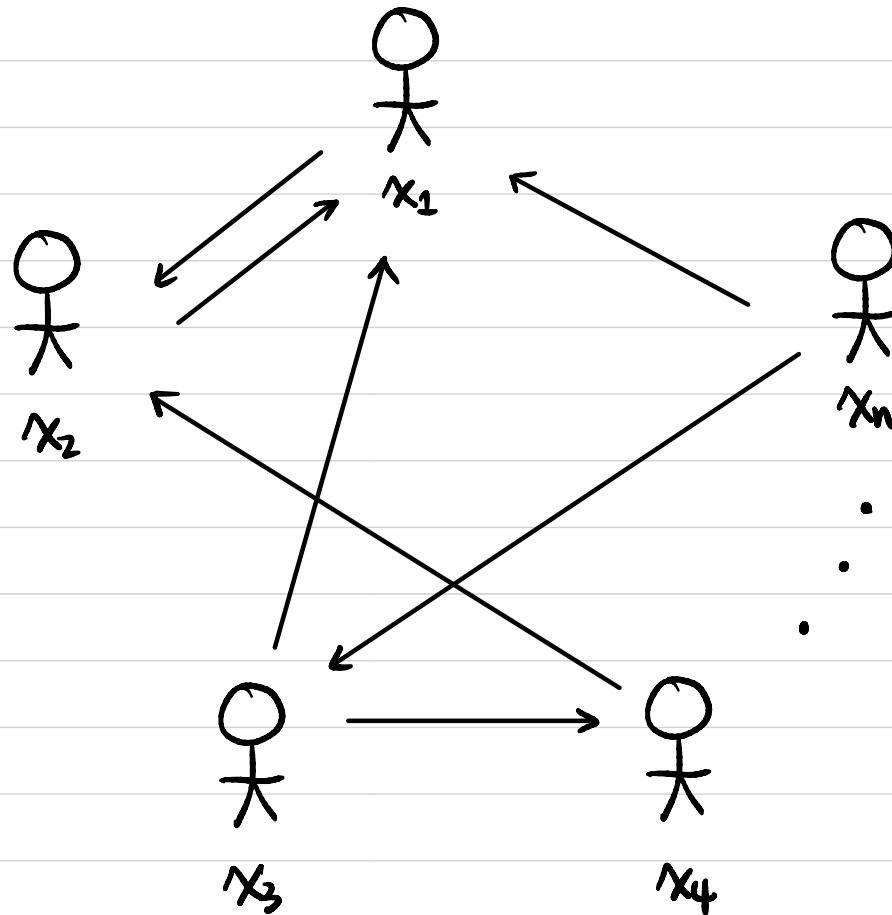
cut-and-choose
with commitments

GMW Compiler
with ZKP

Malicious ZPC
for any function

Malicious MPC
for any function

Privacy-Preserving Machine Learning (PPML)

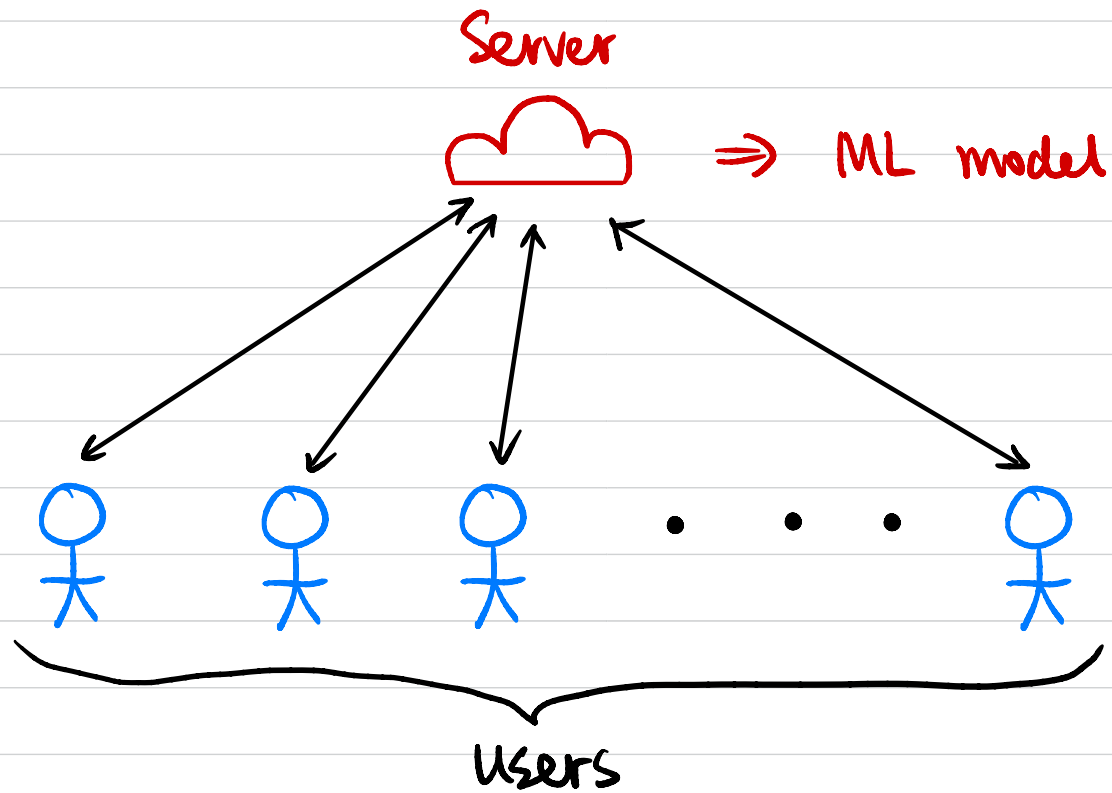


ML model



ML inference

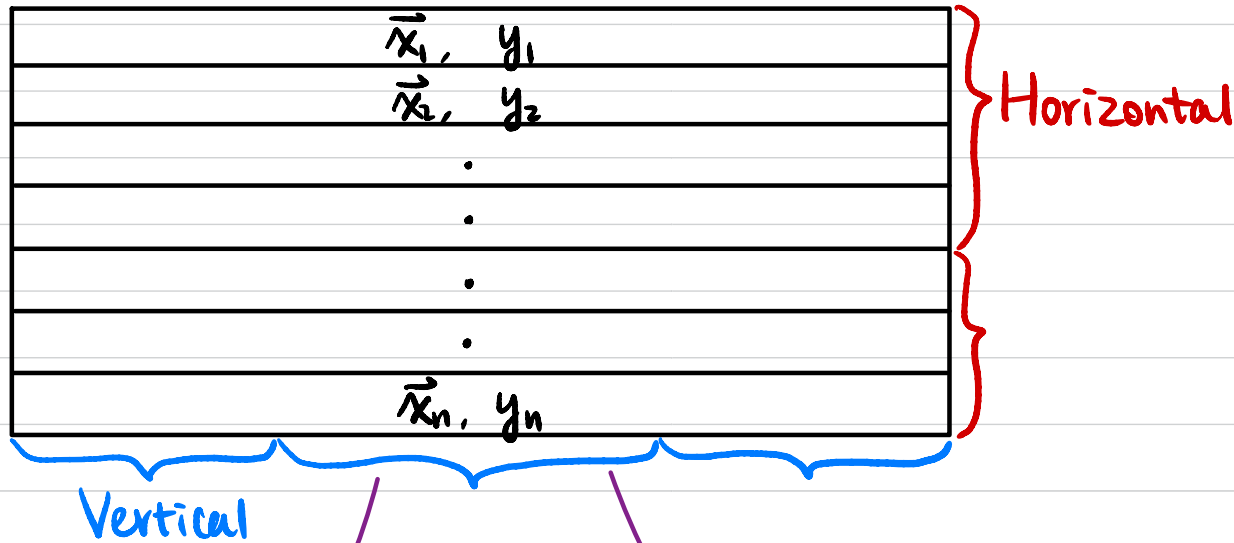
Federated Learning (FL)



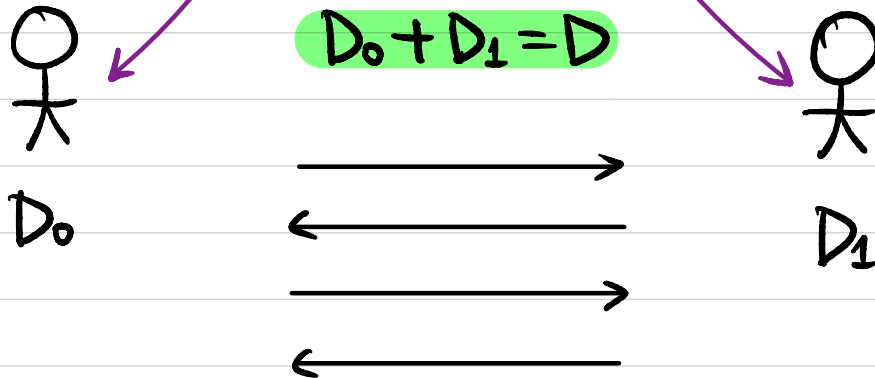
Application: Google mobile keyboard prediction

Privacy-Preserving Machine Learning (PPML)

Horizontal / Vertical Data Partitioning



PPML Framework



Invariant: Two parties hold secret shares of all inputs & intermediate values.

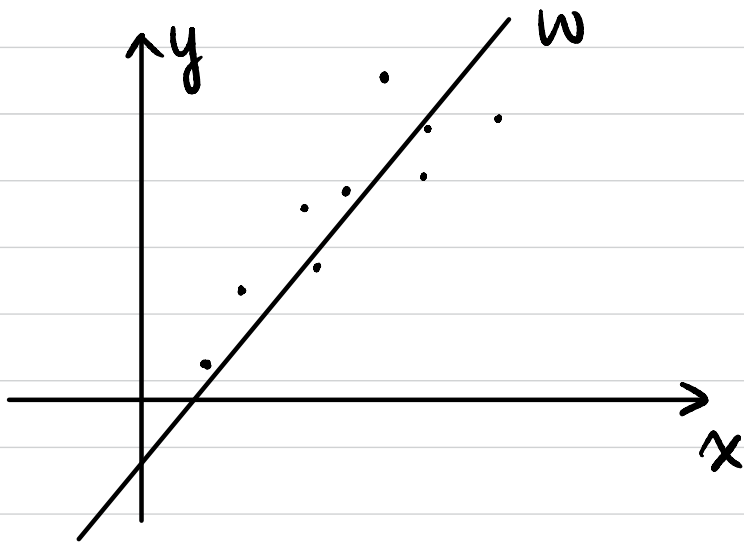
Machine Learning Background

Linear Regression

Data Points (\vec{x}, y)

ML Model: coefficient vector \vec{w}

$$g(\vec{x}) = \langle \vec{x}, \vec{w} \rangle$$



For each data point (\vec{x}_i, y_i) ,

Define loss function

$$L_i(\vec{w}) := \frac{1}{2} (\langle \vec{x}_i, \vec{w} \rangle - y_i)^2$$

$$\text{Total loss: } L(\vec{w}) := \frac{1}{N} \sum_{i \in [N]} L_i(\vec{w})$$

Goal: Find \vec{w} that minimizes $L(\vec{w})$.

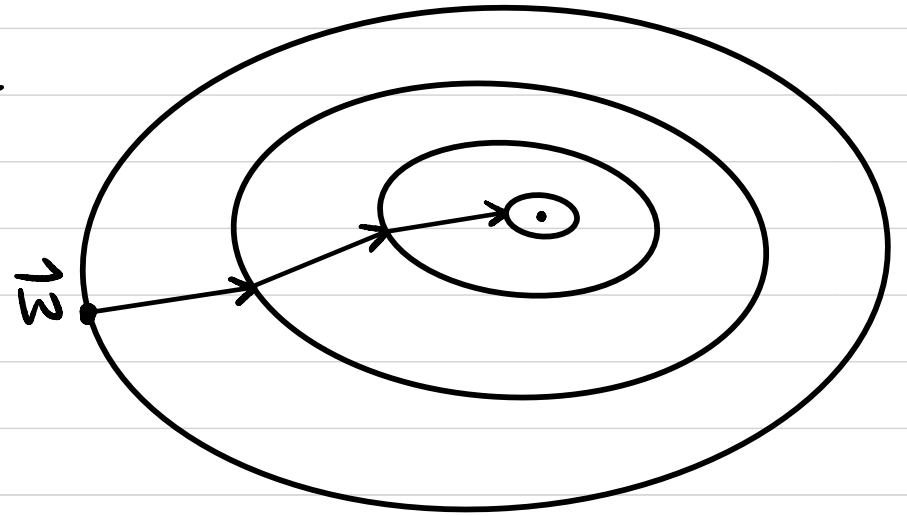
Machine Learning Background

Stochastic Gradient Descent (SGD)

- \vec{w} initialized with arbitrary value
- Given a data point (\vec{x}_i, y_i) :

$$\vec{w} \leftarrow \vec{w} - \eta \cdot \nabla L_i(\vec{w})$$

learning rate gradient

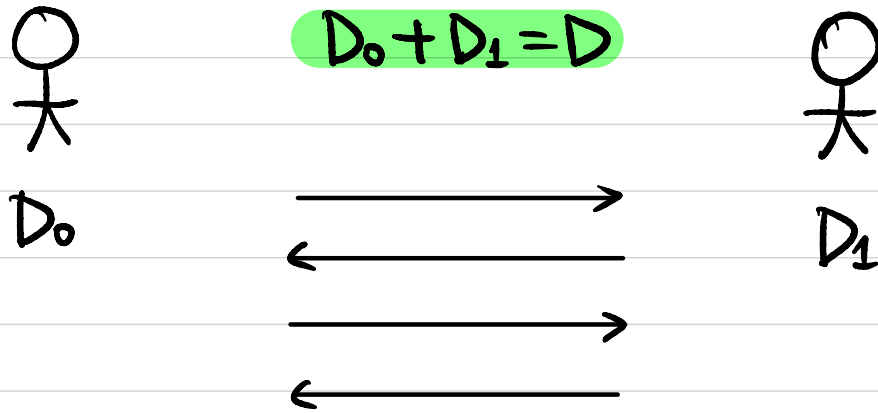


$$\vec{w} \leftarrow \vec{w} - \eta \cdot (\underbrace{\langle \vec{x}_i, \vec{w} \rangle}_{y_i^*} - y_i) \cdot \vec{x}_i$$

$y_i^* = \langle \vec{x}_i, \vec{w} \rangle$ (forward propagation)

backward propagation

PPML for Linear Regression



Invariant: Two parties hold secret shares of all inputs & intermediate values.

Initialization: Randomly sample \vec{w}_0, \vec{w}_1 respectively.

SGD: $\vec{w} \leftarrow \vec{w} - \eta \cdot (\langle \vec{x}_i, \vec{w} \rangle - y_i) \cdot \vec{x}_i$

The equation shows the update rule for the weight vector \vec{w} . The terms \vec{w} , \vec{x}_i , and y_i are circled in green. A purple arrow labeled "public" points to the dot product term $\langle \vec{x}_i, \vec{w} \rangle$. An upward arrow points to the updated \vec{w} on the left side of the equation.

How to get a secret share of the updated \vec{w} ?

Machine Learning Background

Logistic Regression

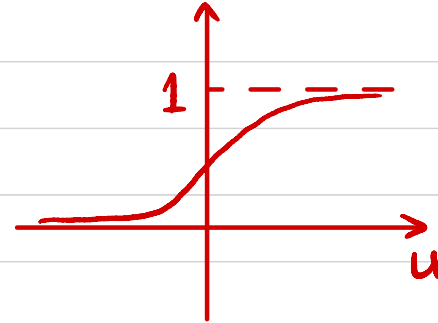
Data Points (\vec{x}, y)

ML Model: coefficient vector \vec{w}

$$g(\vec{x}) = f(\langle \vec{x}, \vec{w} \rangle)$$

↑
activation function

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



For each data point (\vec{x}_i, y_i) ,

$$\text{Define loss function } L_i(\vec{w}) := -y_i \cdot \log y_i^* - (1 - y_i) \cdot \log(1 - y_i^*)$$

$$\text{Total loss: } L(\vec{w}) := \frac{1}{N} \sum_{i \in [N]} L_i(\vec{w})$$

$$y_i^* := f(\langle \vec{x}_i, \vec{w} \rangle)$$

Goal: Find \vec{w} that minimizes $L(\vec{w})$.

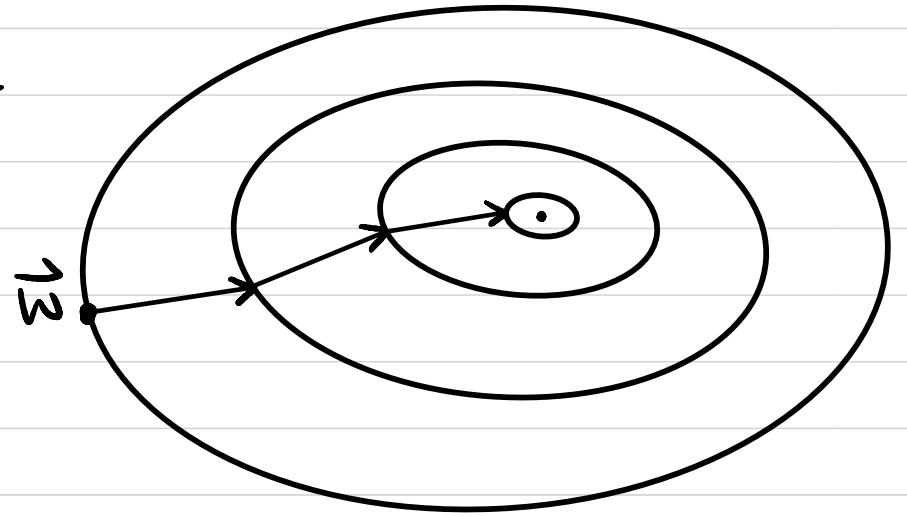
Machine Learning Background

Stochastic Gradient Descent (SGD)

- \vec{w} initialized with arbitrary value
- Given a data point (\vec{x}_i, y_i) :

$$\vec{w} \leftarrow \vec{w} - \eta \cdot \nabla L_i(\vec{w})$$

learning rate gradient

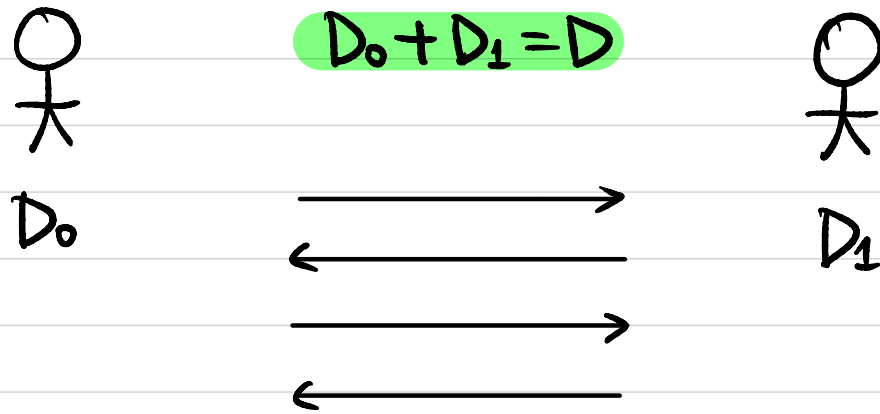


$$\vec{w} \leftarrow \vec{w} - \eta \cdot (f(\langle \vec{x}_i, \vec{w} \rangle) - y_i) \cdot \vec{x}_i$$

$$y_i^* = \langle \vec{x}_i, \vec{w} \rangle \text{ (forward propagation)}$$

backward propagation

PPML for Logistic Regression



Invariant: Two parties hold secret shares of all inputs & intermediate values.

Initialization: Randomly sample \vec{w}_0, \vec{w}_1 respectively.

SGD:
$$\vec{w} \leftarrow \vec{w} - \eta \cdot (f(\langle \vec{x}_i, \vec{w} \rangle) - y_i) \cdot \vec{x}_i$$

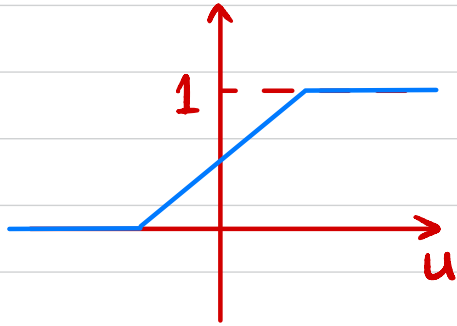
The diagram shows the SGD update equation. The left side is \vec{w} with an upward arrow. The right side is $\vec{w} - \eta \cdot (f(\langle \vec{x}_i, \vec{w} \rangle) - y_i) \cdot \vec{x}_i$. The terms \vec{w} , \vec{x}_i , and y_i are circled in green. A purple arrow labeled "public" points to the $f(\langle \vec{x}_i, \vec{w} \rangle)$ term.

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

How to get a secret share of the updated \vec{w} ?

PPML for Logistic Regression

Approximating $f(u)$



$$\begin{array}{c} [u] \\ \swarrow \quad \searrow \\ u_0 + u_1 \end{array}$$

$$\text{Piecewise polynomial } f(u) = \begin{cases} 0 & \text{if } u < -\frac{1}{2} \\ u + \frac{1}{2} & \text{if } u \in [-\frac{1}{2}, \frac{1}{2}] \\ 1 & \text{if } u > \frac{1}{2} \end{cases}$$

$$b_1 := \begin{cases} 1 & \text{if } u < -\frac{1}{2} \\ 0 & \text{otherwise} \end{cases} \quad [b_1]$$

$$b_2 := \begin{cases} 1 & \text{if } u > \frac{1}{2} \\ 0 & \text{otherwise} \end{cases} \quad [b_2]$$

$$f(u) = b_1 \cdot 0 + b_2 \cdot 1 + (1 - b_1) \cdot (1 - b_2) \cdot (u + \frac{1}{2})$$