A System for Scalable Privacy-Preserving Machine Learning

Abstract:

- we present new and efficient protocols for privacy preserving machine learning for linear regression, logistic regression and neural network training using the stochastic gradient descent method.
- Our protocols fall in the two-server model where data owners distribute their private data among two non-colluding servers who train various models on the joint data using secure two-party computation (2PC).
- We develop new techniques to support secure arithmetic operations on shared decimal numbers, and propose MPC-friendly alternatives to non-linear functions such as sigmoid and softmax that are superior to prior work.

Introduction

Machine learning

- Machine learning techniques are widely used in practice to produce predictive models
- Privacy-preserving machine learning via secure multiparty computation (MPC) provides a promising solution by allowing different entities to train various models on their joint data without revealing any information beyond the outcome.
- The-state-of-the-art solutions for privacy preserving linear regression [36, 20] are many orders of magnitude slower than plaintext training.
  - the bulk of computation for training takes place inside a secure 2PC for boolean circuits (e.g Yao's garbled circuit)
- In case of logistic regression and neural networks
  - the training procedure computes many instances of non-linear activation functions such as sigmoid and softmax that are expensive to compute inside a 2PC.

Introduction

Our Contributions

- We design new and efficient protocols which is several orders of magnitude more efficient than the state of the art solutions for the same problem.
- Arithmetic on shared decimal numbers.
  - represent the two shared decimal numbers as shared integers in a finite field;
  - perform a multiplication on shared integers using offline-generated multiplication triplets;
  - have each party truncate its share of the product so that a fixed number of bits represent the fractional part.
- MPC-friendly activation functions.
  - We propose a new activation function that can be seen as the sum of two RELU functions, and computed efficiently using a small garbled circuit.
- Vectorizing the protocols.
  - We show how to benet from the same vectorization techniques in the shared setting.

Introduction

Related Work

- Earlier work on privacy preserving machine learning has focused on decision trees [30], k-means clustering [27, 13], SVM classification [47, 43], linear regression [18, 19, 39] and logistic regression [41].
- These papers propose solutions based on secure multiparty computation, but appear to incur high efficiency overheads and lack implementation/evaluation.
- Privacy preserving machine learning with neural networks is more challenging.
  - [40] propose a solution, but no formal security guarantees are obtained
- A common technique used in differentially private machine learning is to introduce an additive noise to the data or the update function[8].
  - The parameters of the noise are data-independent.
  - Our system can be composed with such constructions.

Preliminaries

Machine learning

- Linear regression
  - $g(\mathbf{x}_i) = \sum_{j=1}^{d} x_{ij} w_j = \mathbf{x}_i \cdot \mathbf{w}$
- Stochastic gradient descent (SGD)
  - $$w_j := w_j - \alpha \frac{\partial C_i(\mathbf{w})}{\partial w_j}$$
- Mini-batch
  - With mini-batch, the update function can be expressed in a vectorized form:
    $$\mathbf{w} := \mathbf{w} - \frac{1}{|B|} \alpha \mathbf{X}_B^T \times (\mathbf{X}_B \times \mathbf{w} - \mathbf{Y}_B)$$
- Learning rate adjustment.
- Termination.
- Logistic Regression $\mathbf{w} := \mathbf{w} - \frac{1}{|B|} \alpha \mathbf{X}_B^T \times (f(\mathbf{X}_B \times \mathbf{w}) - \mathbf{Y}_B)$
- Neural Networks.

Preliminaries

Secure Computation

- Oblivious Transfer(OT).
  - a fundamental cryptographic primitive that is commonly used as building block in MPC
  - $(\perp; x_b) \leftarrow \mathsf{OT}(x_0, x_1; b)$
- Garbled Circuit 2PC.
  - A garbling scheme consists of a garbling algorithm that takes a random seed σ and a function f and generates a garbled circuit F and a decoding table d_{ec}; the encoding algorithm takes input x and the seed σ and generates garbled input $\hat{x}$; the evaluation algorithm takes $\hat{x}$ and F as input and returns the garbled output $\hat{z}$; and finally, a decoding algorithm that takes the decoding table d_{ec} and $\hat{z}$ and returns $f(x)$.
  - $(z_a, z_b) \leftarrow \mathsf{GarbledCircuit}(x; y, f)$
- Secret Sharing and Multiplication Triplets.
  - Additive sharing. (mostly use)
  - Boolean sharing.
  - Yao sharing.
  - refer the reader to [17] for more details.

Security Model

Architecture

- a set of clients C1, … , Cm who want to train various models on their joint data.
- secure multiparty computation
  - it requires the clients to be involved throughout the protocol.
  - unlike the two-party case, techniques for more than two are significantly more expensive.
  - not scalable to large input sizes or a large number of clients.
- server-aided setting-Server-aided MPC
  - clients can distribute (secret-share) their inputs among the two servers in a setup phase but not be involved in any future computation
  - we can benefit from a combination of efficient techniques for boolean computation such as garbled circuits and OT-extension, and arithmetic computation such as offine/online multiplication triplet shares.

Security Model

Security Definition

- m clients C1, … , Cm; two servers S0, S1; admissible adversary $\mathcal{A}$
  - The security definition should require that an adversary only learns the data of the clients it has corrupted and the final output but nothing else about the remaining honest clients' data.
- Security is defined by comparing a real and ideal interaction.
  - the real interaction $real[\mathcal{Z}, \mathcal{A}, \pi, \lambda]$: The final (single-bit) output of the environment $\mathcal{Z}$ when interacting with adversary $\mathcal{A}$ and honest parties who execute protocol π on security parameter λ.
  - the ideal interaction $ideal[\mathcal{Z}, \mathcal{S}, F_{ml}, \lambda]$: the output of the environment $\mathcal{Z}$ when interacting with adversary $\mathcal{S}$ and honest parties who run the **dummy protocol** in presence of functionality F on security parameter λ.

$$\left| \Pr\left[\text{REAL}[\mathcal{Z}, \mathcal{A}, \pi, \lambda] = 1\right] - \Pr\left[\text{IDEAL}[\mathcal{Z}, \mathcal{S}, \mathcal{F}_{ml}, \lambda] = 1\right] \right|$$

  - They must achieve the same effect in their interaction respectively.

Privacy Preserving Machine Learning

Privacy Preserving Linear Regression

- the update function: $\langle w_j \rangle := \langle w_j \rangle - \alpha \mathsf{Mul}^A\left(\sum_{k=1}^{d} \mathsf{Mul}^A(\langle x_{ik} \rangle, \langle w_k \rangle) - \langle y_i \rangle, \langle x_{ij} \rangle\right)$
- We separate our protocol into two phases: online and offline.
  - The online phase trains the model given the data
  - the offline phase consists mainly of multiplication triplet generation.
- Vectorization in the Shared Setting.
  - benefit from the mini-batch and vectorization techniques
  - $\langle \mathbf{w} \rangle := \langle \mathbf{w} \rangle - \frac{1}{|B|} \alpha \mathsf{Mul}^A(\langle \mathbf{X}_B^T \rangle, \mathsf{Mul}^A(\langle \mathbf{X}_B \rangle, \langle \mathbf{w} \rangle) - \langle \mathbf{Y}_B \rangle)$
- Arithmetic Operations on Shared Decimal Numbers.
  - two decimal numbers x and y with at most l_D bits in the fractional part.
  - $x' = 2^{l_D} x \text{ and } y' = 2^{l_D} y \qquad z = x'y'$
  - truncate the last l_D bits of z such that it has at most l_D bits representing the fractional part.
  - Positive: $z = \lfloor z \rfloor * 2^{l_D} + z_2$, where $0 \leq z_2 < 2^{l_D}$
  - Negative: $\lfloor z \rfloor = 2^l - \lfloor |z| \rfloor$

Privacy Preserving Machine Learning

Privacy Preserving Linear Regression

Protocol SGD_Linear($\langle \mathbf{X} \rangle, \langle \mathbf{Y} \rangle, \langle \mathbf{U} \rangle, \langle \mathbf{V} \rangle, \langle \mathbf{Z} \rangle, \langle \mathbf{V}' \rangle, \langle \mathbf{Z}' \rangle$):

1: $\mathcal{S}_i$ computes $\langle \mathbf{E} \rangle_i = \langle \mathbf{X} \rangle_i - \langle \mathbf{U} \rangle_i$ for $i \in \{0, 1\}$. Then parties run $\mathsf{Rec}(\langle \mathbf{E} \rangle_0, \langle \mathbf{E} \rangle_1)$ to obtain $\mathbf{E}$.
2: **for** $j = 1, \ldots, t$ **do**
3: Parties select the mini-batch $\langle \mathbf{X}_{B_j} \rangle, \langle \mathbf{Y}_{B_j} \rangle$.
4: $\mathcal{S}_i$ computes $\langle \mathbf{F}_j \rangle_i = \langle \mathbf{w} \rangle_i - \langle \mathbf{V}[j] \rangle$ for $i \in \{0, 1\}$. Then parties run $\mathsf{Rec}(\langle \mathbf{F}_j \rangle_0, \langle \mathbf{F}_j \rangle_1)$ to recover $\mathbf{F}_j$.
5: $\mathcal{S}_i$ computes $\langle \mathbf{Y}^*_{B_j} \rangle_i = -i \cdot \mathbf{E}_{B_j} \times \mathbf{F}_i + \langle \mathbf{X}_{B_j} \rangle_i \times \mathbf{F}_i + \mathbf{E}_{B_j} \times \langle \mathbf{w} \rangle_i + \langle \mathbf{Z}_j \rangle_i$ for $i \in \{0, 1\}$.
6: $\mathcal{S}_i$ compute the difference $\langle \mathbf{D}_{B_j} \rangle_i = \langle \mathbf{Y}^*_{B_j} \rangle_i - \langle \mathbf{Y}_{B_j} \rangle_i$ for $i \in \{0, 1\}$.
7: $\mathcal{S}_i$ computes $\langle \mathbf{F}'_j \rangle_i = \langle \mathbf{D}_{B_j} \rangle_i - \langle \mathbf{V}'_j \rangle_i$ for $i \in \{0, 1\}$. Parties then run $\mathsf{Rec}(\langle \mathbf{F}'_j \rangle_0, \langle \mathbf{F}'_j \rangle_1)$ to obtain $\mathbf{F}'_j$.
8: $\mathcal{S}_i$ computes $\langle \mathbf{\Delta} \rangle_i = -i \cdot \mathbf{E}^T_{B_j} \times \mathbf{F}'_j + \langle \mathbf{X}^T_{B_j} \rangle_i \times \mathbf{F}'_j + \mathbf{E}^T_{B_j} \times \langle \mathbf{D}_{B_j} \rangle_i + \langle \mathbf{Z}'_j \rangle_i$ for $i \in \{0, 1\}$.
9: $\mathcal{S}_i$ truncates its shares of Δ element-wise to get $\lfloor \langle \mathbf{\Delta} \rangle_i \rfloor$.
10: $\mathcal{S}_i$ computes $\langle \mathbf{w} \rangle_i := \langle \mathbf{w} \rangle_i - \frac{\alpha}{|B|} \lfloor \langle \mathbf{\Delta} \rangle_i \rfloor$ for $i \in \{0, 1\}$.
11: Parties run $\mathsf{Rec}^A(\langle \mathbf{w} \rangle_0, \langle \mathbf{w} \rangle_1)$ and output $\mathbf{w}$.

Figure 4: The online phase of privacy preserving linear regression.

Privacy Preserving Machine Learning

Privacy Preserving Linear Regression

- Effect of Truncation Error.
  - In Figure 5, the x-axis is the number of iterations of the SGD algorithm and the y-axis is the accuracy of the trained model on the testing dataset.
  - when we use 13 bits for the fractional part of $\boldsymbol{w}$, the privacy preserving training behaves almost exactly the same as the plaintext training.
  - 6 bits: the accuracy of our protocol oscillates during the training. This is because now the error is on the 6th bit which has a larger effect and may push the model away from the optimum.
  - 2 bits: the oscillating behavior is more extreme.
- Efficiency Discussion.
  - The dominating term in the computation cost is the matrix multiplications.
  - the total communication is $n \cdot d + (|B| + d) \cdot t = nd \cdot (1 + \frac{E}{d} + \frac{E}{|B|})$
  - In practice, E is much smaller than $|B|$ and d. Therefore, the time spent on the communication can be calculated by dividing the total communication by the bandwidth between the two parties

Privacy Preserving Machine Learning

Privacy Preserving Linear Regression

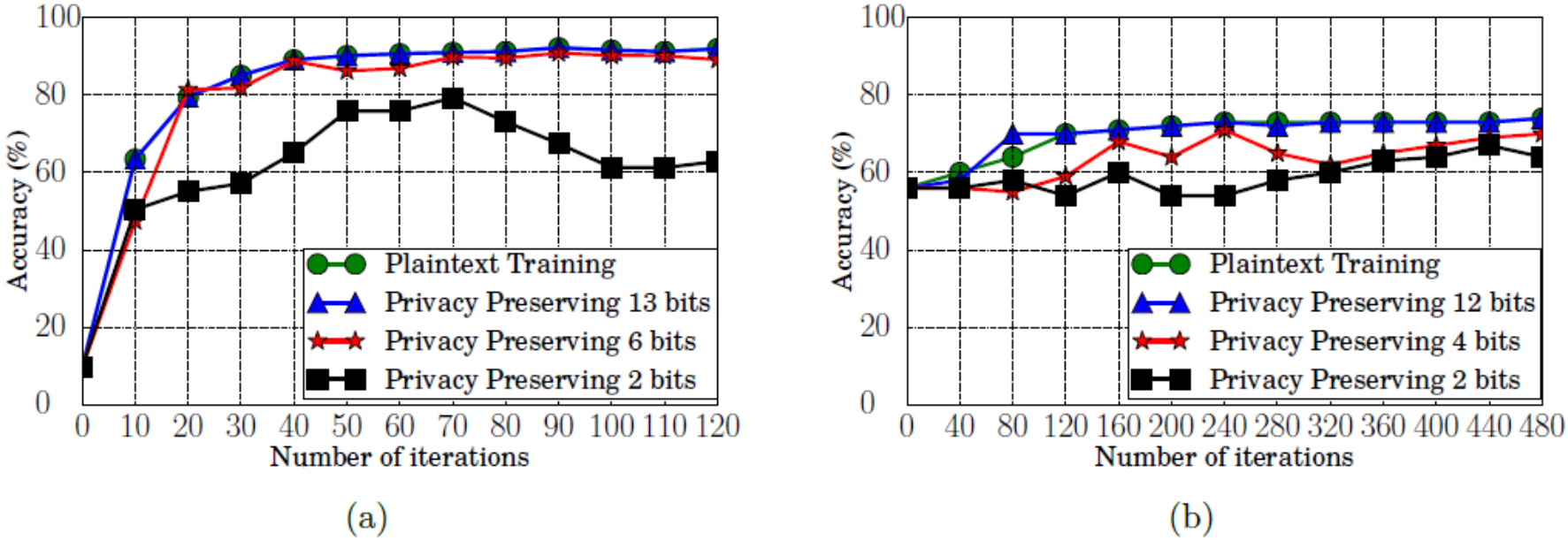


Figure 5: Comparison of accuracy of privacy preserving linear regression with truncation and plaintext training on decimal numbers. (a) MNIST dataset, $|B| = 128$, (b) Arcene dataset, $|B| = 32$.

Privacy Preserving Machine Learning

The Offline Phase

- We present two protocols for doing so based on linearly homomorphic encryption (LHE) and oblivious transfer (OT).
  - The techniques are similar to prior work (e.g., [17]) but are optimized for the vectorized scenario where we operate on matrices.
  - given shared random matrices $\langle U \rangle$ and$\langle V \rangle$, compute the shares of their product $\langle Z \rangle$
  - basic step: a $|B| \times d$ matrix $\langle A \rangle$ and a $d \times 1$matrix $\langle B \rangle$ to compute shares of a $|B| \times 1$ matrix $\langle C \rangle$ such that $C = A \times B$
  - We utilize the following relationship:

$$\mathbf{C} = \langle \mathbf{A} \rangle_0 \times \langle \mathbf{B} \rangle_0 + \langle \mathbf{A} \rangle_0 \times \langle \mathbf{B} \rangle_1 + \langle \mathbf{A} \rangle_1 \times \langle \mathbf{B} \rangle_0 + \langle \mathbf{A} \rangle_1 \times \langle \mathbf{B} \rangle$$

  - It suffices to compute $\langle \langle A \rangle_0 \times \langle B \rangle_1 \rangle$ and $\langle \langle A \rangle_1 \times \langle B \rangle_0 \rangle$ as the other two terms can be computed locally.

Privacy Preserving Machine Learning

The Offline Phase

- LHE-based generation.
  - S1 encrypts each element of $\langle B \rangle_1$ using an LHE and sends them to S0.
  - The LHE can be initiated using the cryptosystem of Paillier [37] or Damgard-Geisler-Kroigaard(DGK) [16].
  - S0 then performs the matrix multiplication on the ciphertexts, with additions replaced by multiplications and multiplications by exponentiations.
  - Finally, S0 masks the resulting ciphertexts by random values, and sends them back to S1 to decrypt.

Protocol LHE_MT($\langle \mathbf{A} \rangle_0; \langle \mathbf{B} \rangle_1$):

(Let a_{ij} be the (i,j)th element in $\langle \mathbf{A} \rangle_0$ and b_j be the jth element in $\langle \mathbf{B} \rangle_1$.)

1: $\mathcal{S}_1 \rightarrow \mathcal{S}_0$: $\mathsf{Enc}(b_j)$ for $i = 1, \ldots, d$.

2: $\mathcal{S}_0 \rightarrow \mathcal{S}_1$: $c_i = \Pi_{j=0}^{d} \mathsf{Enc}(b_j)^{a_{ij}} \cdot \mathsf{Enc}(r_i)$, for $i = 1, \ldots, |B|$.

3: $\mathcal{S}_0$ sets $\langle \langle \mathbf{A} \rangle_0 \times \langle \mathbf{B} \rangle_1 \rangle_0 = \mathbf{r}$, where $\mathbf{r} = (-r_1, \ldots, -r_{|B|})^T \mod 2^l$.

4: $\mathcal{S}_1$ sets $\langle \langle \mathbf{A} \rangle_0 \times \langle \mathbf{B} \rangle_1 \rangle_1 = (\mathsf{Dec}(c_1), \ldots, \mathsf{Dec}(c_{|B|}))^T$,

Figure 6: The offline protocol based on linearly homomorphic encryption.

Privacy Preserving Machine Learning

The Offline Phase

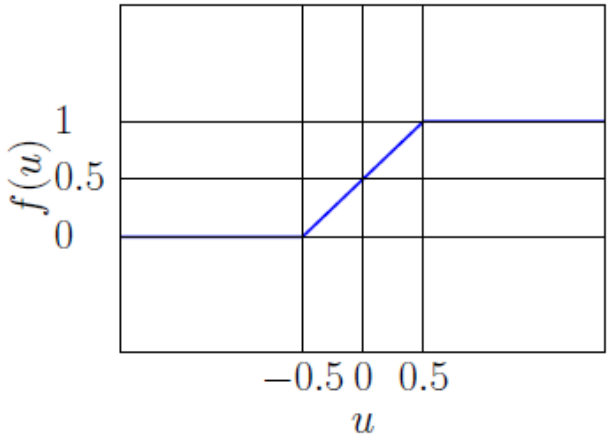
- OT-based generation.
  - We first compute the shares of the product $\langle a_{ij} \cdot b_j \rangle$ for all $i = 1, \dots, |B|$ and $j = 1, \dots, d$.
  - S1 uses each bit of b_j to select two values computed from a_{ij} using correlated OTs.
  - For $k = 1, \dots, l$, S0 sets the correlation function of COT to $f_k(x) = a_{i,j} \cdot 2^k + x \ mod \ 2^l$ and S0, S1 run $\mathrm{COT}(r_k, f_k(x), b_j[k])$
  - If $b_j[k] = 0$, S1 gets r_k; if $b_j[k] = 0$, S1 gets $a_{i,j} \cdot 2^k + r_k \ mod \ 2^l$
  - S1 sets $\langle a_{ij} \cdot b_j \rangle_1 = \sum_{k=1}^{l} (b_j[k] \cdot a_{i,j} \cdot 2^k + r_k) = a_{ij} \cdot b_j + \sum_{k=1}^{l} r_k \ mod \ 2^l$
  - S0 sets $\langle a_{ij} \cdot b_j \rangle_0 = \sum_{k=1}^{l} (-r_k) \ mod \ 2^l$
  - Finally, after computing $\langle a_{ij} \cdot b_j \rangle$, the ith element of $\langle \langle A \rangle_0 \times \langle B \rangle_1 \rangle$ can be computed by $\langle \langle A \rangle_0 \times \langle B \rangle_1 \rangle[\mathrm{i}] = \sum_{j=0}^{d} \langle a_{ij} \cdot b_j \rangle$

Privacy Preserving Machine Learning

Privacy Preserving Logistic Regression

- the main additional challenge is to compute the logistic $sigmoid$ function on shared numbers.
  - the division and the exponentiation are hard to support using a 2PC for arithmetic or boolean circuit.
  - It can be shown that approximation using a high-degree polynomial is very accurate [32].
- Secure computation friendly activation functions.

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



Privacy Preserving Machine Learning

Privacy Preserving Logistic Regression

- Secure computation friendly activation functions.
  - we use the new activation function, we have two choices when computing the backward propagation.
    - use the same update function as the logistic function
    - compute the partial derivative of the new function and substitute it into the update function.
  - The first approach yields better accuracy matching that of using the logistic function.

	Logistic	Our approaches		Polynomial Approx.		
		first	second	deg. 2	deg. 5	deg. 10
MNIST	98.64	98.62	97.96	42.17	84.64	98.54
Arcene	86	86	85	72	82	86

Table 1: Accuracy (%) comparison of different approaches for logistic regression.

Privacy Preserving Machine Learning

Privacy Preserving Logistic Regression

- The privacy preserving protocol.
  - We take advantage of techniques to switch between arithmetic sharing and Yao sharing proposed in [17].
  - following the same protocol for privacy preserving linear regression, after computing the inner product of the input data and the coefficient vector, we switch the arithmetic sharing to a Yao sharing and evaluate the activation function using a garbled circuit.
  - Then, we switch back to arithmetic sharing and continue the backward propagation.

Privacy Preserving Machine Learning

Privacy Preserving Logistic Regression

- The privacy preserving protocol.

Protocol SGD_Logistic($\langle \mathbf{X} \rangle, \langle \mathbf{Y} \rangle, \langle \mathbf{U} \rangle, \langle \mathbf{V} \rangle, \langle \mathbf{Z} \rangle, \langle \mathbf{V}' \rangle, \langle \mathbf{Z}' \rangle$):

1: Do step 1–5 as in Figure 4. Both parties obtain the shares $\langle \mathbf{U}_{B_i} \rangle = \langle \mathbf{X}_{B_i} \times \mathbf{w} \rangle$ (it was defined as $\langle \mathbf{Y}^*_{B_i} \rangle$ in Figure 4).

2: **for** every element $\langle u \rangle$ in $\langle \mathbf{U}_{B_i} \rangle$ **do**

3: $(\langle b_3 \rangle^B, \langle b_4 \rangle^B) \leftarrow$ Y2B(GarbledCircuit($\langle u \rangle_0 + \frac{1}{2}, \langle u \rangle_0 - \frac{1}{2}; \langle u \rangle_1, f$)), where f sets b_1 as the most significant bit of $(\langle u \rangle_0 + \frac{1}{2}) + \langle u \rangle_1$ and b_2 as the most significant bit of $(\langle u \rangle_0 - \frac{1}{2}) + \langle u \rangle_1$. It then outputs $b_3 = \neg b_1$ and $b_4 = b_1 \wedge (\neg b_2)$.

4: $\mathcal{S}_0$ sets $m_0 = \langle b_4 \rangle^B_0 \cdot \langle u \rangle_0 + r_1$ and $m_1 = (1 - \langle b_4 \rangle^B_0) \cdot \langle u \rangle_0 + r_1$. $\mathcal{S}_0$ and $\mathcal{S}_1$ run $(\perp; m_{\langle b_4 \rangle^B_1}) \leftarrow$ OT$(m_0, m_1; \langle b_4 \rangle^B_1)$. $m_{\langle b_4 \rangle^B_1}$ is equal to $(\langle b_4 \rangle^B_0 \oplus \langle b_4 \rangle^B_1) \cdot \langle u \rangle_0 + r_1 = b_4 \cdot \langle u \rangle_0 + r_1$.

5: P_1 sets $m_0 = \langle b_4 \rangle^B_1 \cdot \langle u \rangle_1 + r_2$ and $m_1 = (1 - \langle b_4 \rangle^B_1) \cdot \langle u \rangle_1 + r_2$. $\mathcal{S}_1$ and $\mathcal{S}_0$ run $(\perp; m_{\langle b_4 \rangle^B_0}) \leftarrow$ OT$(m_0, m_1; \langle b_4 \rangle^B_0)$. $m_{\langle b_4 \rangle^B_0}$ is equal to $b_4 \cdot \langle u \rangle_1 + r_2$.

6: $\mathcal{S}_0$ sets $m_0 = \langle b_3 \rangle^B_0 + r_3$ and $m_1 = (1 - \langle b_3 \rangle^B_0) + r_3$. $\mathcal{S}_0$ and $\mathcal{S}_1$ run $(\perp; m_{\langle b_3 \rangle^B_1}) \leftarrow$ OT$(m_0, m_1; \langle b_3 \rangle^B_1)$. $m_{\langle b_3 \rangle^B_1}$ is equivalent to $b_3 + r_3$.

7: $\mathcal{S}_0$ sets $\langle y^* \rangle_0 = m_{\langle b_4 \rangle^B_0} - r_1 - r_3$ and $\mathcal{S}_1$ sets $\langle y^* \rangle_1 = m_{\langle b_4 \rangle^B_1} + m_{\langle b_3 \rangle^B_1} - r_2$.

8: **end for**

9: Both parties set $\langle \mathbf{Y}^* \rangle_i$ as a vector of all $\langle y^* \rangle_i$s computed above and continue to step 6–12 in Figure 4.

Figure 8: Privacy preserving logistic regression protocol.

Privacy Preserving Machine Learning

Privacy Preserving Logistic Regression

- The privacy preserving protocol.
  - We take advantage of techniques to switch between arithmetic sharing and Yao sharing proposed in [17].
  - following the same protocol for privacy preserving linear regression, after computing the inner product of the input data and the coefficient vector, we switch the arithmetic sharing to a Yao sharing and evaluate the activation function using a garbled circuit.
  - Then, we switch back to arithmetic sharing and continue the backward propagation.
- Efficiency Discussion.
  - Most of the steps are exactly the same as the linear regression protocol.
  - In addition, one garbled circuit protocol and 3 extra OTs are performed in each forward propagation.
  - Therefore, the total communication overhead is $|B| \cdot t \cdot ((2l-1) \cdot 2\lambda + 3l)$.

Privacy Preserving Machine Learning

Privacy Preserving Neural Network Training

- Neural Network.
  - use the RELU function as the activation function in each neuron and the cross entropy function as the cost function..
  - All the functions, other than the RELU function, are implemented using the same techniques discussed for linear regression.
  - To evaluate the RELU function, we use the same approach as for logistic regression by switching to Yao sharing.
  - We also propose a secure computation friendly alternative to the softmax function, replace the exponentiations in the numerator with RELU functions.
- Efficiency Discussion.
  - In the online phase, The total communication is the sum of the sizes of all matrices involved in the matrix multiplication and element-wise multiplication. The total number of iterations is $5m \cdot t$.
  - In the offline phase, the total number of multiplication triplets is increased by a factor of $\sum_{i=1}^{m} d_m$ compared to regression, which is exactly the number of neurons in the neural network.

Privacy Preserving Machine Learning

Predictions and Accuracy Testing

- Privacy preserving prediction.
  - We iterate that we can hide the input data, the model, the prediction result or any combinations of them, as they can all be secret shared in our protocols.
  - In classification problems, the prediction is usually rounded to the closest class.
- Privacy preserving accuracy testing.
  - the learning rate and the number of iterations can be fixed in advance.
  - At the cost of some leakage, we propose an alternative solution that enables adjusting the rate and number of iteration in the same fashion as plaintext training.
  - To do so, we need to test the accuracy of the current model after each epoch on a testing dataset.
  - In each epoch, whether or not we adjust the learning rate or whether we terminate or not leaks one extra bit of information hence providing a trade-off between the efficiency (reduced number of epochs) and security.

Client-Aided Offline Protocol

Client-Aided Multiplication Triplets

- For each feature of each sample, the client possessing the data generates a random value u to mask the data, and generates random values v_k, v'_k, and computes $z_k = u \cdot v_k$, $z'_k = u \cdot v'_k$
- Finally, the client distributes shares of $\langle u \rangle$, $\langle v_k \rangle$, $\langle v'_k \rangle$, $\langle z_k \rangle$, $\langle z'_k \rangle$ to the two servers.
- we do not assume the clients know the partitioning of the data possession, so we can no longer utilize the vectorized equation
- it introduces overhead to the online phase
  - the computation, communication of the online phase and the storage of the two servers are increased.

The new security model

- The security model also changes with the client-aided offline phase.
- in the client-aided scenario, we change the security model to not allow collusion between a server and a client.

Experimental Results

LAN & WAN

- In the LAN, our capture the scenario where the two servers in our protocols have a high-bandwidth/low-latency network connection, but otherwise are not administered/controlled by the same party.
- The primary reason for reporting experiments in the LAN setting is more accurate benchmarking and comparison as the majority of prior work.
- contrasting our results in the LAN and WAN setting highlights the significance of network bandwidth in our various protocols.

Offline vs. online

- The offine phase includes all computation and communication that can be performed without presence of data
- the online phase consists of all data-dependent steps of the protocol.

Data sets

- The MNIST dataset [6] and the Arcene dataset [1, 24].

Experimental Results

Experiments for Linear Regression

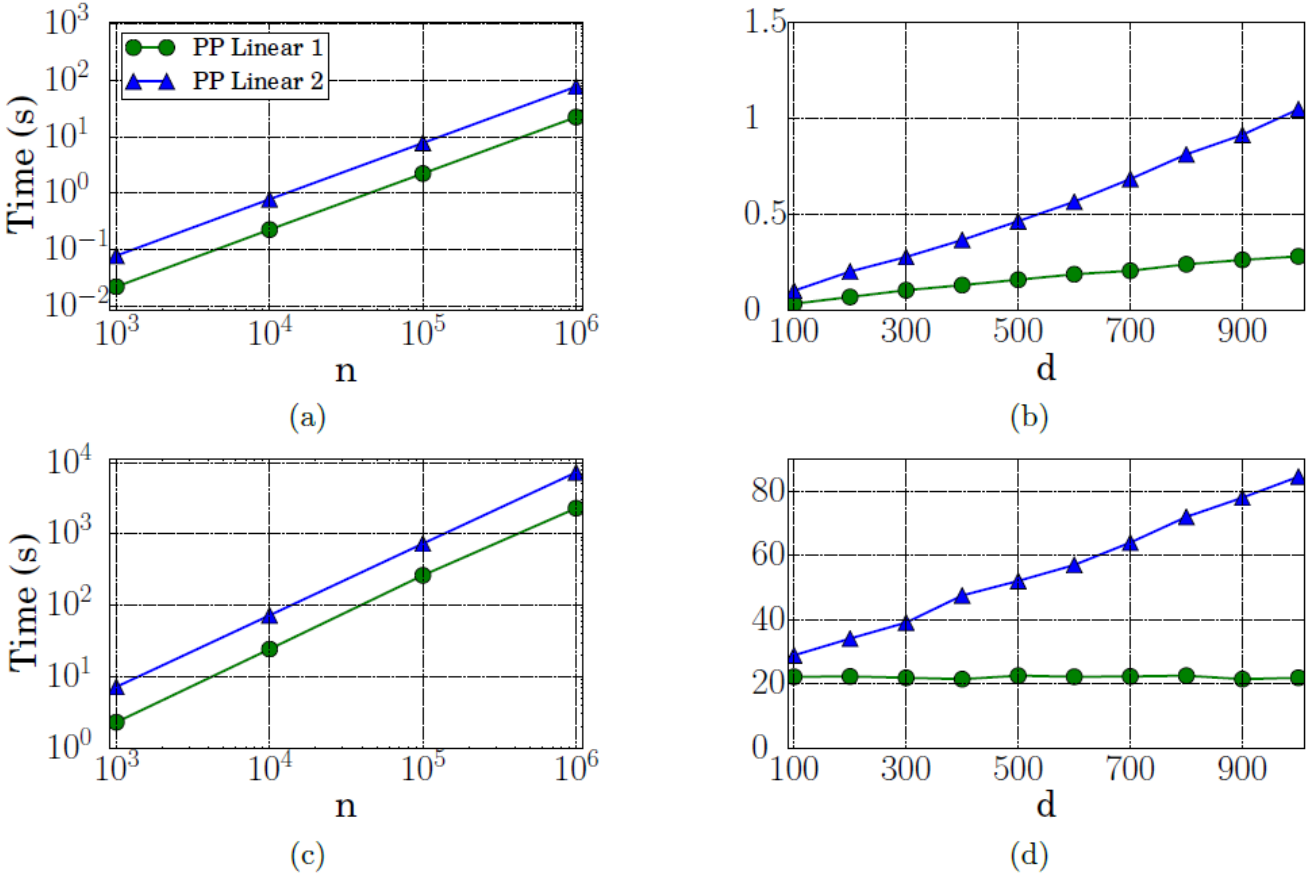


Figure 9: Online cost of privacy preserving linear regression in standard and client-aided settings. $|B|$ is set to 128. Figure (a), (b) are for LAN network and Figure (c), (d) are for WAN network. Figure (a) and (c) are in log-log scale and for $d = 784$. Figure (b) and (d) are in regular scale and for $n = 10,000$.

Experimental Results

Experiments for Linear Regression

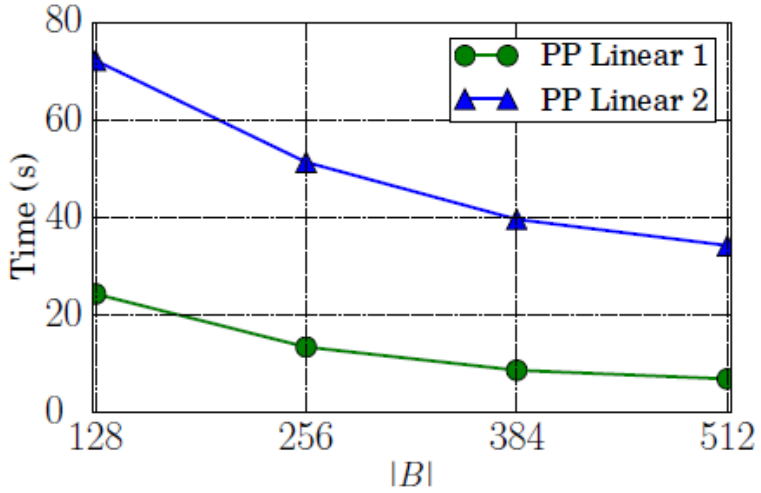


Figure 10: Performance of online phase of linear regression on WAN with different mini-batch sizes. $n = 10,000$, $d = 784$.

Experimental Results

Experiments for Linear Regression

		LHE-based			OT-based			Client aided		Dataset
n	d	LAN	WAN	Comm.	LAN	WAN	Comm.	Time	Comm.	size
1,000	100	23.9s	24.0s	2MB	0.86s	43.2s	190MB	0.028s	7MB	0.8MB
	500	83.9s	84.8s	6MB	3.8s	210.6s	1GB	0.16s	35MB	3.8MB
	1000	158.4s	163.2s	10MB	7.9s	163.2s	1.9GB	0.33s	69MB	7.6MB
10,000	100	248.4s	252.9s	20MB	7.9s	420.2s	1.9GB	0.33s	69MB	7.6MB
	500	869.1s	890.2s	60MB	39.2s	2119.1s	9.5GB	1.98s	344MB	38MB
	1000	1600.9s	1627.0s	100MB	80.0s	4097.1s	19GB	4.0s	687MB	76MB
100,000	100	2437.1s	2478.1s	200MB	88.0s	4125.1s	19GB	3.9s	687MB	76MB
	500	8721.5s	8782.4s	600MB	377.9s	20000s*	95GB	20.2s	3435MB	380MB
	1000	16000s*	16100s*	1000MB	794.0s	40000s*	190GB	49.9s	6870MB	760MB

Table 2: Performance of the offline phase. $|B| = 128$ and $E = 2$. (* means estimated via extrapolation.)

Experimental Results

Experiments for Linear Regression

- Comparison with prior work.

	MNIST	Gisette	Arcene
Cholesky	92.02%	96.7%	87%
SGD	91.95%	96.5%	86%

Table 3: Comparison of accuracy for SGD and Cholesky.

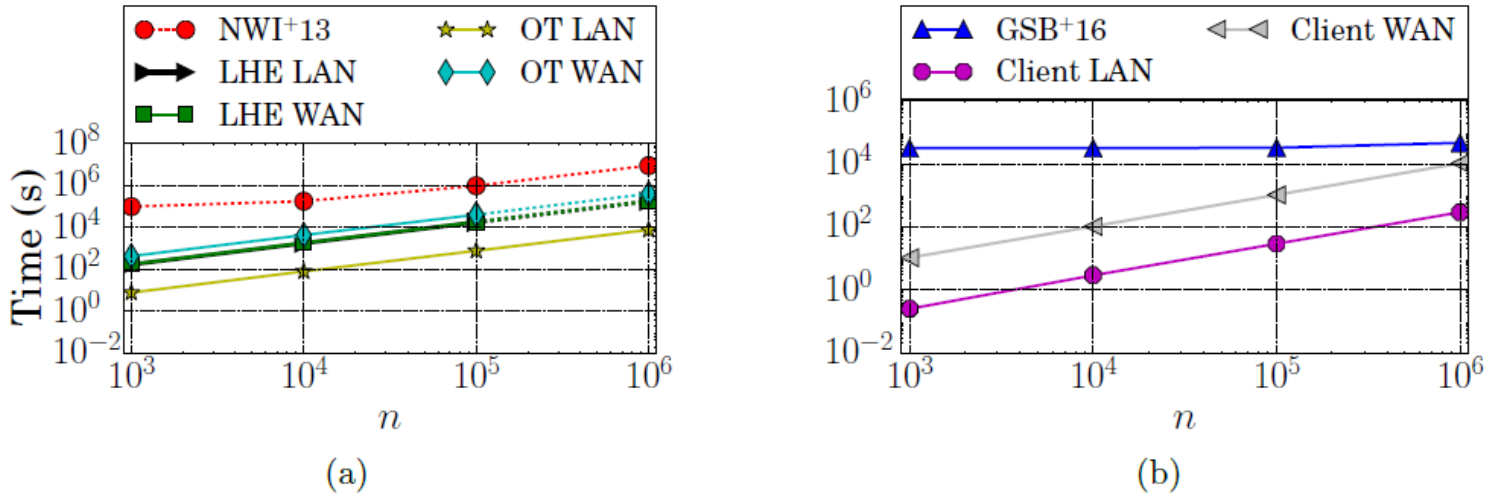


Figure 11: Efficiency comparison with prior work. Figures are in log-log scale, $d = 500$, $|B| = 128$ for our schemes.

Experimental Results

Experiments for Logistic Regression

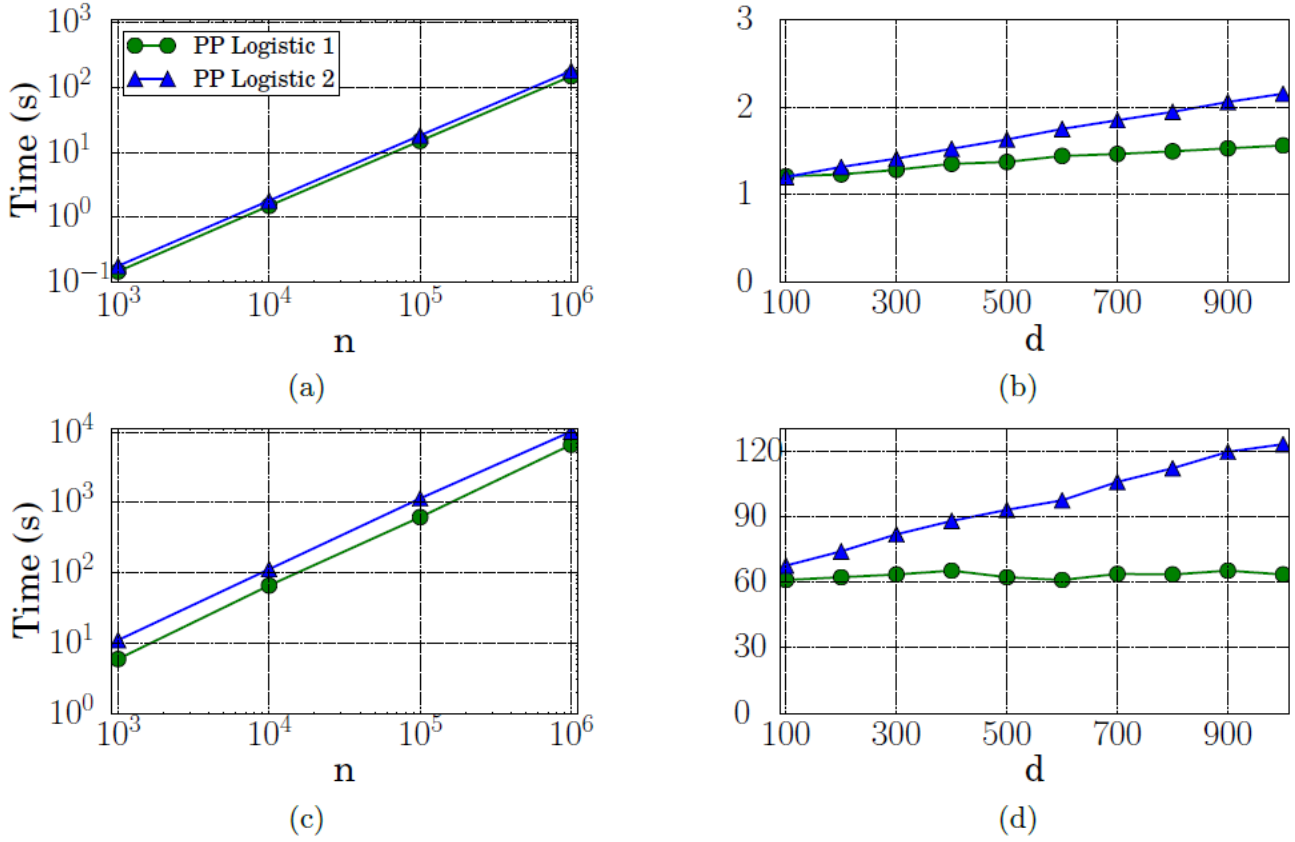


Figure 12: Online cost of privacy preserving logistic regression in the standard and client-aided setting. $|B|$ is set to 128. Figure (a), (b) are for LAN network and Figure (c), (d) are for WAN network. Figure (a) and (c) are in log-log scale, $d = 784$. Figure (b) and (d) are in regular scale, $n = 10,000$.

Experimental Results

Experiments for Neural Networks

- In terms of the accuracy
  - RELU: 93.4%
  - Square: 93.1%

	Protocol 1		Protocol 2	
	Offline	Online	Offline	Online
RELU	290,000s*	4239.7s	14951.2s	10332.3s
Square	320,000s*	653.0s	16783.9s	4260.3s

Table 4: Performance of privacy preserving neural networks training on MNIST in LAN setting. $n = 60,000$, $d = 784$.

Experimental Results

Experiments for predictions

	k	Linear (ms)		Logistic (ms)		Neural (s)	
		Online	Offline	Online	Offline	Online	Offline
LAN	1	0.20	2.5	0.59	2.5	0.18	4.7
	100	0.22	51	3.9	51	0.20	13.8
WAN	1	72	620	158	620	0.57	17.8
	100	215	2010	429	2010	1.2	472

Table 5: Online and offline performances for privacy preserving predicition. $d = 784$. The neural network is the same as the one in Section 6.3.

Experimental Results

Microbenchmarks

- Arithmetic on shared decimal numbers.

		Total (OT)	Total (LHE)	GC
LAN	$k = 1000$	0.028s	5.3s	0.13s
	$k = 10,000$	0.16s	53s	1.2s
	$k = 100,000$	1.4s	512s	11s
WAN	$k = 1000$	1.4s	6.2s	5.8s
	$k = 10,000$	12.5s	62s	68s
	$k = 100,000$	140s	641s	552s

Table 6: Comparison of our decimal multiplication and the fixed-point multiplication using garbled circuit.

Experimental Results

Microbenchmarks

- Vectorization.

	d	Online	Online Vec	OT	OT Vec	LHE	LHE Vec
LAN	100	0.37ms	0.22ms	0.21s	0.05s	67s	1.6s
	500	1.7ms	0.82ms	1.2s	0.28s	338s	5.5s
	1000	3.5ms	1.7ms	2.0s	0.46s	645s	10s
WAN	100	0.2s	0.09s	14s	3.7s	81s	2s
	500	0.44s	0.20ss	81s	19s	412s	6.2s
	1000	0.62s	0.27s	154s	34s	718s	11s

Table 7: Speedup from vectorization. $|B| = 128$.

Experimental Results

Microbenchmarks

- New logistic function.

	New Logistic	Poly Total Client-aided	Poly Total OT	Poly Total LHE
LAN	0.0045s	0.0005s	0.025s	6.8s
WAN	0.2s	0.69s	2.5s	8.5s

Table 8: Performance of our new logistic function and polynomial approximation.