Theme Article: Federated Machine Learning

A Secure Federated Transfer Learning Framework

Yang Liu
WeBank

Yan Kang
WeBank

Chaoping Xing
Shanghai Jiao Tong University

Tianjian Chen
WeBank

Qiang Yang
Hong Kong University of Science and Technology

***Abstract*—Machine learning relies on the availability of vast amounts of data for training. However, in reality, data are mostly scattered across different organizations and cannot be easily integrated due to many legal and practical constraints. To address this important challenge in the field of machine learning, we introduce a new technique and framework, known as federated transfer learning (FTL), to improve statistical modeling under a data federation. FTL allows knowledge to be shared without compromising user privacy and enables complementary knowledge to be transferred across domains in a data federation, thereby enabling a target-domain party to build flexible and effective models by leveraging rich labels from a source domain. This framework requires minimal modifications to the existing model structure and provides the same level of accuracy as the nonprivacy-preserving transfer learning. It is flexible and can be effectively adapted to various secure multiparty machine learning tasks.**

Recent artificial intelligence (AI) achievements have been depending on the availability of massive amounts of labeled data. For example, AlphaGo has been trained using a dataset containing 30 million moves from 160 000 actual games. The ImageNet dataset has over 14 million images. However, across various industries, most applications only have access to small or poor quality datasets. Labeling data is very expensive, especially in fields which require human expertise and

Digital Object Identifier 10.1109/MIS.2020.2988525
Date of publication 22 April 2020; date of current version 3 September 2020.

Published by the IEEE Computer Society

domain knowledge. In addition, data needed for a specific task may not all be stored in one place. Many organizations may only have unlabeled data, and some other organizations may have very limited amounts of labels. It has been increasingly difficult from a legislative perspective for organizations to combine their data, too. For example, General Data Protection Regulation,[1] a new bill introduced by the European Union, contains many terms that protect user privacy and prohibit organizations from exchanging data without explicit user approval. How to enable the large number of businesses and applications that have only small data (few samples and features) or weak supervision (few labels) to build effective and accurate AI models while complying with data privacy and security laws is a difficult challenge.

To address this challenge, Google introduced a federated learning system[2] in which a global machine learning model is updated by a federation of distributed participants while keeping their data stored locally. Their framework requires all contributors share the same feature space. On the other hand, secure machine learning with data partitioned in the feature space has also been studied.[3] These approaches are only applicable in the context of data with either common features or common samples under a federation. In reality, however, the set of such common entities may be small, making a federation less attractive and leaving the majority of the nonoverlapping data underutilized.

In this article, we propose federated transfer learning (FTL) to address the limitations of existing federated learning approaches. It leverages transfer learning[4] to provide solutions for the entire sample and feature space under a federation. Our contributions are as follows.

1) We formalize the research problem of FTL in a privacy-preserving setting to provide solutions for federation problems beyond the scope of existing federated learning approaches.
2) We provide an end-to-end solution to the proposed FTL problem and show that the performance of the proposed approach in terms of convergence and accuracy is comparable to nonprivacy-preserving transfer learning.
3) We provide some novel approaches to incorporate additively HE and secret sharing using beaver triples into two-party computation (2PC) with neural networks under the FTL framework such that only minimal modifications to the neural network is required and the accuracy is almost lossless.

RELATED WORK

Recent years have witnessed a surge of studies on encrypted machine learning. For example, Google introduced a secure aggregation scheme to protect the privacy of aggregated user updates under their federated learning framework.[5] CryptoNets[6] adapted neural network computations to work with data encrypted via HE. SecureML[7] is a multiparty computing scheme which uses secret-sharing and Yao's Garbled Circuit for encryption and supports collaborative training for linear regression, logistic regression (LR), and neural networks.

Transfer learning aims to build an effective model for an application with a small dataset or limited labels in a target domain by leveraging knowledge from a different but related source domain. In recent years, there have been tremendous progress in applying transfer learning to various fields such as image classification and sentiment analysis. The performance of transfer learning relies on how related the domains are. Intuitively, parties in the same data federation are usually organizations from the same industry. Therefore, they can benefit more from knowledge propagation. To the best of our knowledge, FTL is the first framework to enable federated learning to benefit from transfer learning.

PRELIMINARIES AND SECURITY DEFINITION

Consider a source domain dataset $\mathcal{D}_A := \{(x_i^A, y_i^A)\}_{i=1}^{N_A}$ where $x_i^A \in R^a$ and $y_i^A \in \{+1, -1\}$ is the ith label, and a target domain $\mathcal{D}_B := \{x_j^B\}_{j=1}^{N_B}$ where $x_j^B \in R^a$. $\mathcal{D}_A$ and $\mathcal{D}_B$ are separately held by two private parties and cannot be exposed to each other legally. We assume that there exists a limited set of cooccurrence samples $\mathcal{D}_{AB} := \{(x_i^A, x_i^B)\}_{i=1}^{N_{AB}}$ and a small set of labels for data from domain B in party A's dataset: $\mathcal{D}_c := \{(x_i^B, y_i^A)\}_{i=1}^{N_c}$, where N_c is the number of available target labels. Without loss

of generality, we assume all labels are in party A, but all the deduction here can be adapted to the case where labels exist in party B. One can find the set of commonly shared sample IDs in a privacy-preserving setting by masking data IDs with encryption techniques (e.g., the RSA scheme). We utilize the RSA Intersection module of the FATE* framework to align cooccurrence samples of the two parties. Given the above setting, the objective is for the two parties to build a transfer learning model to predict labels for the target-domain party accurately without exposing data to each other.

In this article, we adopt a security definition in which all parties are *honest-but-curious*. We assume a threat model with a semihonest adversary $\mathcal{D}$ who can corrupt at most one of the two data clients. For a protocol P performing $(O_A, O_B) = P(I_A, I_B)$, where O_A and O_B are the party A's and party B's outputs and I_A and I_B are their inputs, respectively, P is secure against A if there exists an infinite number of (I'_B, O'_B) pairs such that $(O_A, O'_B) = P(I_A, I'_B)$. It allows flexible control of information disclosure compared to complete zero knowledge security.

PROPOSED APPROACH

In this section, we introduce our proposed transfer learning model.

Deep neural networks have been widely adopted in transfer learning to find implicit transfer mechanisms. Here, we explore a general scenario in which hidden representations of A and B are produced by two neural networks $u_i^A = \mathrm{Net}^A(x_i^A)$ and $u_i^B = \mathrm{Net}^B(x_i^B)$, where $u^A \in \mathbb{R}^{N_A \times d}$ and $u^B \in \mathbb{R}^{N_B \times d}$, and d is the dimension of the hidden representation layer. The neural networks Net^A and Net^B serve as feature transformation functions that project the source features of party A and party B into a common feature subspace in which knowledge can be transferred between the two parties. Any other symmetric feature transformation techniques can be applied here to form the common feature subspace. However, neural networks can help us build an end-to-end solution to the proposed FTL problem.

To label the target domain, a general approach is to introduce a prediction function $\varphi(u_j^B) = \varphi(u_1^A, y_1^A ... u_{N_A}^A, y_{N_A}^A, u_j^B)$. Without losing much generality, we assume $\varphi(u_j^B)$ is linearly separable. That is, $\varphi(u_j^B) = \Phi^A \mathcal{G}(u_j^B)$. In our experiment, we use a translator function, $\varphi(u_j^B) = \frac{1}{N_A} \sum_i^{N_A} y_i^A u_i^A (u_j^B)'$, where $\Phi^A = \frac{1}{N_A} \sum_i^{N_A} y_i^A u_i^A$ and $\mathcal{G}(u_j^B) = (u_j^B)'$. We can then write the training objective using the available labeled set as

$$\underset{\Theta^A, \Theta^B}{\operatorname{argmin}} \mathcal{L}_1 = \sum_{i=1}^{N_c} \ell_1(y_i^A, \varphi(u_i^B)) \tag{1}$$

where Θ^A, Θ^B are the training parameters of Net^A and Net^B, respectively. Let L_A and L_B be the number of layers for Net^A and Net^B, respectively. Then, $\Theta^A = \{\theta_l^A\}_{l=1}^{L_A}$, $\Theta^B = \{\theta_l^B\}_{l=1}^{L_B}$ where θ_l^A and θ_l^B are the training parameters for the lth layer. ℓ_1 denotes the loss function. For logistic loss, $\ell_1(y, \varphi) = \log(1 + \exp(-y\varphi))$.

In addition, we also aim to minimize the alignment loss between A and B in order to achieve feature transfer learning in a federated learning setting

$$\underset{\Theta^A, \Theta^B}{\operatorname{argmin}} \mathcal{L}_2 = \sum_{i=1}^{N_{AB}} \ell_2(u_i^A, u_i^B) \tag{2}$$

where ℓ_2 denotes the alignment loss. Typical alignment losses can be $-u_i^A (u_i^B)'$ or $||u_i^A - u_i^B||_F^2$. For simplicity, we assume that it can be expressed in the form of $\ell_2(u_i^A, u_i^B) = \ell_2^A(u_i^A) + \ell_2^B(u_i^B) + \kappa u_i^A (u_i^B)'$, where κ is a constant

The final objective function is

$$\underset{\Theta^A, \Theta^B}{\operatorname{argmin}} \mathcal{L} = \mathcal{L}_1 + \gamma \mathcal{L}_2 + \frac{\lambda}{2}(\mathcal{L}_3^A + \mathcal{L}_3^B) \tag{3}$$

where γ and λ are the weight parameters, and $\mathcal{L}_3^A = \sum_l^{L_A} ||\theta_l^A||_F^2$ and $\mathcal{L}_3^B = \sum_l^{L_B} ||\theta_l^B||_F^2$ are the regularization terms. Now we focus on obtaining the gradients for updating Θ^A, Θ^B in back propagation. For $i \in \{A, B\}$, we have

$$\frac{\partial \mathcal{L}}{\partial \theta_l^i} = \frac{\partial \mathcal{L}_1}{\partial \theta_l^i} + \gamma \frac{\partial \mathcal{L}_2}{\partial \theta_l^i} + \lambda \theta_l^i. \tag{4}$$

Under the assumption that A and B are not allowed to expose their raw data, a privacy-preserving approach needs to be developed to compute (3) and (4). Here, we adopt a second-order Taylor approximation for logistic loss

*https://github.com/FederatedAI/FATE

$$\ell_1(y,\varphi) \approx \ell_1(y,0) + \frac{1}{2}C(y)\varphi + \frac{1}{8}D(y)\varphi^2 \tag{5}$$

and the gradient is

$$\frac{\partial \ell_1}{\partial \varphi} = \frac{1}{2}C(y) + \frac{1}{4}D(y)\varphi \tag{6}$$

where $C(y) = -y$, $D(y) = y^2$.

In the following two sections, we will discuss two alternative constructions of the secure FTL protocol: the first one is leveraging additively homomorphic encryption (HE), and the second one is utilizing the secret sharing based on beaver triples. We carefully design the FTL protocol such that only minimal information needs to be encrypted or secretly shared between parties. Besides, the FTL protocol is designed to be compatible with other HE and secret sharing schemes with minimal modifications.

FTL USING HE

Additively HE and polynomial approximations have been widely used for privacy-preserving machine learning. Applying (5) and (6), and additively HE (denoted as $[[\cdot]]$), we obtain the privacy preserved loss function and the corresponding gradients for the two domains as

$$\begin{aligned}[[\mathcal{L}]] = &\sum_i^{N_c}([[\ell_1(y_i^A,0)]] + \frac{1}{2}C(y_i^A)\Phi^A[[\mathcal{G}(u_i^B)]] \\ &+ \frac{1}{8}D(y_i^A)\Phi^A[[(\mathcal{G}(u_i^B))'\mathcal{G}(u_i^B)]](\Phi^A)') \\ &+ \gamma\sum_i^{N_{AB}}([[\ell_2^B(u_i^B)]] + [[\ell_2^A(u_i^A)]] + \kappa u_i^A[[(u_i^B)']]) \\ &+ \left[\left[\frac{\lambda}{2}\mathcal{L}_3^A\right]\right] + \left[\left[\frac{\lambda}{2}\mathcal{L}_3^B\right]\right]\end{aligned} \tag{7}$$

$$\begin{aligned}\left[\left[\frac{\partial \mathcal{L}}{\partial \theta_l^B}\right]\right] = &\sum_i^{N_c}\frac{\partial(\mathcal{G}(u_i^B))'\mathcal{G}(u_i^B)}{\partial u_i^B}\left[\left[\left(\frac{1}{8}D(y_i^A)(\Phi^A)'\Phi^A\right)\right]\right]\frac{\partial u_i^B}{\partial \theta_l^B} \\ &+ \sum_i^{N_c}\left[\left[\frac{1}{2}C(y_i^A)\Phi^A\right]\right]\frac{\partial \mathcal{G}(u_i^B)}{\partial u_i^B}\frac{\partial u_i^B}{\partial \theta_l^B} \\ &+ \sum_i^{N_{AB}}\left([[\gamma\kappa u_i^A]]\frac{\partial u_i^B}{\partial \theta_l^B} + \left[\left[\gamma\frac{\partial \ell_2^B(u_i^B)}{\partial \theta_l^B}\right]\right]\right) + [[\lambda\theta_l^B]]\end{aligned} \tag{8}$$

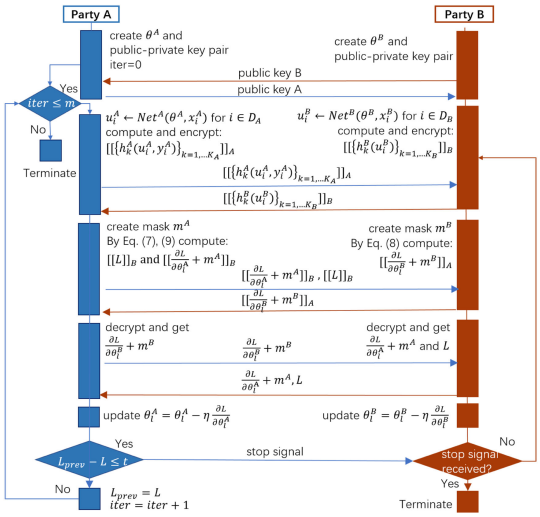

Figure 1. HE-based FTL algorithm workflow.

$$\begin{aligned}\left[\left[\frac{\partial \mathcal{L}}{\partial \theta_l^A}\right]\right] = &\sum_j^{N_A}\sum_i^{N_c}(\frac{1}{4}D(y_i^A)\Phi^A[[\mathcal{G}(u_i^B)'\mathcal{G}(u_i^B)]] \\ &+ \frac{1}{2}C(y_i^A)[[\mathcal{G}(u_i^B)]])\frac{\partial \Phi^A}{\partial u_j^A}\frac{\partial u_j^A}{\partial \theta_l^A} \\ &+ \gamma\sum_i^{N_{AB}}\left([[\kappa u_i^B]]\frac{\partial u_i^A}{\partial \theta_l^A} + \left[\left[\frac{\partial \ell_2^A(u_i^A)}{\partial \theta_l^A}\right]\right]\right) + [[\lambda\theta_l^A]].\end{aligned} \tag{9}$$

FTL Algorithm—HE Based

With (7)–(9), we now design a federated algorithm for solving the transfer learning problem. See Figure 1. Let $[[\cdot]]_A$ and $[[\cdot]]_B$ be HE operators with public keys from A and B, respectively. A and B initialize and execute their respective neural networks Net^A and Net^B locally to obtain hidden representations u_i^A and u_i^B. A then computes and encrypts components $\{h_k(u_i^A, y_i^A)\}_{k=1,2,...K_A}$ and sends to B to assist calculations of gradients of Net^B. In the current scenario, $K_A = 3$, $h_1^A(u_i^A, y_i^A) = \{[[\frac{1}{8}D(y_i^A)(\Phi^A)'(\Phi^A)]]_A\}_{i=1}^{N_c}$, $h_2^A(u_i^A, y_i^A) = \{[[\frac{1}{2}C(y_i^A)\Phi^A]]_A\}_{i=1}^{N_c}$, and $h_3^A(u_i^A, y_i^A) = \{[[\gamma\kappa u_i^A]]_A\}_{i=1}^{N_{AB}}$. Similarly, B then computes and encrypts components $\{h_k^B(u_i^B)\}_{k=1,2,...K_B}$ and sends to A to assist calculations of gradients of Net^A and loss $\mathcal{L}$. In the current scenario, $K_B = 4$, $h_1^B(u_i^B) = \{[[(\mathcal{G}(u_i^B))'\mathcal{G}(u_i^B)]]_B\}_{i=1}^{N_c}$, $h_2^B(u_i^B) = \{[[\mathcal{G}(u_i^B)]]_B\}_{i=1}^{N_c}$, $h_3^B(u_i^B) = \{[[\kappa u_i^B]]_B\}_{i=1}^{N_{AB}}$, and $h_4^B(u_i^B) = [[\frac{\lambda}{2}\mathcal{L}_3^B]]_B$.

To prevent A's and B's gradients from being exposed, A and B further mask each gradient with an encrypted random value. They then

send the masked gradients and loss to each other and decrypt the values locally. *A* can send termination signals to *B* once the loss convergence condition is met. Otherwise, they unmask the gradients, update the model parameters with their respective gradients, and move on to next iteration. Once the model is trained, FTL can provide predictions for unlabeled data from party *B*. Algorithm 1 summaries the prediction process.

Algorithm 1. HE-Based FTL: Prediction

Require: Model parameters Θ^A, Θ^B, $\{x_j^B\}_{j\in N_B}$
1: *B* do:
2: $u_j^B \leftarrow \mathrm{Net}^B(\Theta^B, x_j^B)$;
3: Encrypt $\{[[\mathcal{G}(u_j^B)]]_B\}_{j\in\{1,2,\ldots,N_B\}}$ and send it to *A*;
4: *A* do:
5: Create a random mask m^A;
6: Compute $[[\varphi(u_j^B)]]_B = \Phi^A[[\mathcal{G}(u_j^B)]]_B$ and send $[[\varphi(u_j^B) + m^A]]_B$ to *B*;
7: *B* do:
8: Decrypt $\varphi(u_j^B) + m^A$ and send results to *A*;
9: *A* do:
10: Compute $\varphi(u_j^B)$ and y_j^B and send y_j^B to *B*;

Security Analysis

Theorem 1. *The protocol in the FTL training Algorithm (see Figure 1) and Algorithm 1 is secure under our security definition, provided that the underlying additively HE scheme is secure.*

Proof. The training protocol in Figure 1 and Algorithm 1 do not reveal any information, because all *A* and *B* learned are the masked gradients. In each iteration, *A* and *B* create new random masks. The strong randomness and secrecy of the masks secure the information against the other party.[8] During training, *A* learns its own gradients at each step, but this is not enough for *A* to learn any information from *B* based on the impossibility of solving n equations with more than n unknowns.[8] In other words, there exist an infinite number of inputs from *B* that result in the same gradients *A* receives. Similarly, *B* cannot learn any information from *A*. Therefore, as long as the encryption scheme is secure, the proposed protocol is secure. During evaluation, *A* learns the predicted result for each sample from *B*, which is a scalar product, from which *A* cannot infer *B*'s private information. *B* learns only the label, from which *B* cannot infer *A*'s private information. □

At the end of the training process, each party (*A* or *B*) only obtains the model parameters associated with its own features, and remains oblivious to the data structure of the other party. At inference time, the two parties need to collaboratively compute the prediction results. Note that the protocol does not deal with the situation in which one (or both) of the two parties is (are) malicious. If *A* fakes its inputs and submits only one nonzero input, it might be able to tell the value of u_i^B at the position of that input. It still cannot tell x_i^B or Θ_B, and neither party can obtain the correct prediction results.

FTL USING SECRET SHARING

Throughout this section, assume that any private value v is shared between the two parties where A keeps $\langle v\rangle_A$ and B keeps $\langle v\rangle_B$ such that $v = \langle v\rangle_A + \langle v\rangle_B$. To make it possible for the performance to be comparable with the previous construction, assume that $\ell_1(y, \varphi)$ and $\frac{\partial \ell_1}{\partial \varphi}$ can be approximated by the second-order Taylor expansion following (5) and (6). So $\mathcal{L}, \frac{\partial \mathcal{L}}{\partial \theta_\ell^A}$, and $\frac{\partial \mathcal{L}}{\partial \theta_\ell^B}$ can be expressed as the following:

$$\begin{aligned}\mathcal{L} = &\sum_i^{N_C} \ell_1(y_i^A, 0) + \frac{1}{2}C(y_i^A)\Phi^A\mathcal{G}(u_i^B)\\ &+ \left(\frac{1}{8}D(y_i^A)\Phi^A\mathcal{G}(u_i^B)\right)\left(\Phi^A\mathcal{G}(u_i^B)\right)\\ &+ \gamma\sum_i^{N_{AB}}\left(\ell_2^B(u_i^B) + \ell_2^A(u_i^A) + \kappa u_i^A(u_i^B)'\right)\\ &+ \frac{\lambda}{2}\left(\mathcal{L}_3^A + \mathcal{L}_3^B\right)\end{aligned} \tag{10}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \theta_\ell^B} = &\sum_i^{N_C}\frac{1}{2}C(y_i^A)\Phi^A\frac{\partial \mathcal{G}(u_i^B)}{\partial \theta_\ell^B}\\ &+ 2\left[\left(\frac{1}{8}D(y_1^A)\Phi^A\mathcal{G}(u_i^B)\right)\left(\Phi^A\frac{\partial\left(\mathcal{G}(u_i^B)\right)}{\partial \theta_\ell^B}\right)\right]\\ &+ \sum_i^{N_{AB}}\left(\gamma\kappa u_i^A\frac{\partial u_i^B}{\partial \theta_\ell^B} + \gamma\frac{\partial \ell_2^B(u_i^B)}{\partial \theta_\ell^B}\right) + \lambda\theta_\ell^B\end{aligned} \tag{11}$$

$$\frac{\partial \mathcal{L}}{\partial \theta_l^A} = \sum_i^{N_C} \frac{1}{2} C(y_i^A) \frac{\partial \Phi^A}{\partial \theta_\ell^A} \mathcal{G}(u_i^B) + 2\left[\left(\frac{1}{8} D(y_i^A)\Phi^A \mathcal{G}(u_i^B)\right)\left(\frac{\partial \Phi^A}{\partial \theta_\ell^A}\mathcal{G}(u_i^B)\right)\right] + \gamma \sum_i^{N_{AB}} \left(\kappa u_i^B \frac{\partial u_i^A}{\partial \theta_\ell^A} + \frac{\partial \ell_2^A(u_i^A)}{\partial \theta_\ell^A}\right) + \lambda \theta_\ell^A. \tag{12}$$

In this case, the whole process can be performed securely if secure matrix addition and multiplication can be constructed. Since operations with public matrices or adding two private matrices can simply be done using the shares without any communication, the remaining operation that requires discussion is secure matrix multiplication. Beaver's triples are used to help in the matrix multiplication.

Algorithm 2. Secure Matrix Multiplication

Require: M, N two matrices to be multiplied with dimensions $m \times n$ and $n \times k$ respectively and secretly shared between A and B. Triple $(D, E, F = DE)$ of matrices with dimension $m \times n, n \times k$, and $m \times k$ respectively secretly shared between A and B.

1: *A* do:
2: $\langle \delta \rangle_A \leftarrow \langle M \rangle_A - \langle D \rangle_A, \langle \gamma \rangle_A \leftarrow \langle N \rangle_A - \langle E \rangle_A$ and send them to B;
3: *B* do:
4: $\langle \delta \rangle_B \leftarrow \langle M \rangle_B - \langle D \rangle_B, \langle \gamma \rangle_B \leftarrow \langle N \rangle_B - \langle E \rangle_B$ and send them to A;
5: *A* and *B* recovers $\delta = \langle \delta \rangle_A + \langle \delta \rangle_B$ and $\gamma = \langle \gamma \rangle_A + \langle \gamma \rangle_B$.
6: *A* do:
7: $\langle P \rangle_A \leftarrow \langle M \rangle_A \gamma + \delta \langle N \rangle_A + \langle F \rangle_A$;
8: *B* do:
9: $\langle P \rangle_B \leftarrow \langle M \rangle_B \gamma + \delta \langle N \rangle_B + \langle F \rangle_B - \delta\gamma$;

Secure Matrix Multiplication Using Beaver Triples

First, we briefly recall how to perform the matrix multiplication given that the two parties have already shared a Beaver's triple. Suppose that the computation required is to obtain $P = MN$ where the dimensions of M, N, and P are $m \times n, n \times k$, and $m \times k$, respectively. As assumed, the matrices M and N have been secretly shared by the two parties where A keeps $\langle M \rangle_A$ and $\langle N \rangle_A$ and B keeps $\langle M \rangle_B$ and $\langle N \rangle_B$. To assist with the calculation, in the preprocessing phase, A and B have generated three matrices D, E, F of dimension $m \times n, n \times k$, and $m \times k$, respectively where A keeps $\langle D \rangle_A, \langle E \rangle_A$, and $\langle F \rangle_A$ while B keeps $\langle D \rangle_B, \langle E \rangle_B$, and $\langle F \rangle_B$ such that $DE = F$.

It is easy to see that $\langle P \rangle_A + \langle P \rangle_B = MN$, which is what is required for this protocol. As can be seen, this method guarantees efficient online computation in the cost of having offline phase where players generated the Beavers triples. So next we discuss the scheme to generate the triples.

Algorithm 3. Offline Secure Matrix Multiplication

Require: U and V, two matrices to be multiplied with dimensions $m \times n$ and $n \times k$ respectively; U is owned by A and V is owned by B.

1: Invite a third party C;
2: *A* do:
3: Randomly choose $\langle U \rangle_0$ and set $\langle U \rangle_1 = U - \langle U \rangle_0$;
4: Send $\langle U \rangle_1$ to B and $\langle U \rangle_0$ to C;
5: *B* do:
6: Randomly choose $\langle V \rangle_0$ and set $\langle V \rangle_1 = V - \langle V \rangle_0$
7: Send $\langle V \rangle_0$ to A and $\langle V \rangle_1$ to C;
8: *C* do:
9: Compute $\tilde{W} = \langle U \rangle_0 \langle V \rangle_1$;
10: Randomly choose $\langle \tilde{W} \rangle_A$ and set $\langle \tilde{W} \rangle_B = \tilde{W} - \langle \tilde{W} \rangle_A$;
11: Send $\langle \tilde{W} \rangle_A$ to A and $\langle \tilde{W} \rangle_B$ to B;
12: *A* do:
13: Set $\langle W \rangle_A = \langle U \rangle_0 \langle V \rangle_0 + \langle U \rangle_1 \langle V \rangle_0 + \langle \tilde{W} \rangle_A$;
14: *B* do:
15: Set $\langle W \rangle_B = \langle U \rangle_1 \langle V \rangle_1 + \langle \tilde{W} \rangle_B$;

Beaver Triples Generation

In the preprocessing phase, the Beaver's triples generation protocol uses a subprotocol to perform the secure matrix multiplication with the help of a third party, which we will call Carlos. Recall that having two matrices U and V owned respectively by A and B, they want to calculate UV securely with the help of Carlos. In order to do this, A and B individually generate shares for U and V, respectively. That is, we have $U = \langle U \rangle_0 + \langle U \rangle_1$ and $V = \langle V \rangle_0 + \langle V \rangle_1$. Then, we have $UV = (\langle U \rangle_0 + \langle U \rangle_1) \cdot (\langle V \rangle_0 + \langle V \rangle_1) = \langle U \rangle_0 \langle V \rangle_0 + \langle U \rangle_0 \langle V \rangle_1 + \langle U \rangle_1 \langle V \rangle_0 + \langle U \rangle_1 \langle V \rangle_1$. So if A sends $\langle U \rangle_1$ to B and B sends $\langle V \rangle_0$ to A:

1) $\langle U\rangle_0\langle V\rangle_0$ can be privately calculated by A
2) $\langle U\rangle_1\langle V\rangle_1$ can be privately calculated by B
3) $\langle U\rangle_1\langle V\rangle_0$ can be privately calculated by both A and B
4) However, no one can calculate $\langle U\rangle_0\langle V\rangle_1$ yet. This is what Carlos will calculate.

By the use of Algorithm 3, Algorithm 4 generates triple (D, E, F) such that:

1) $DE = F$
2) A holds $\langle D\rangle_A, \langle E\rangle_A$, and $\langle F\rangle_A$ without learning anything about $(D, E, F), \langle D\rangle_B, \langle E\rangle_B$, and $\langle F\rangle_B$.
3) B holds $\langle D\rangle_B, \langle E\rangle_B$, and $\langle F\rangle_B$ without learning anything about $(D, E, F), \langle D\rangle_A, \langle E\rangle_A$, and $\langle F\rangle_A$.

Algorithm 4. Beaver Triples Generation

Require: The dimensions of the required matrices, $m \times n, n \times k$, and $m \times k$;
1: *A* do:
2: Randomly choose $\langle D\rangle_A$ and $\langle E\rangle_A$;
3: *B* do:
4: Randomly choose $\langle D\rangle_B$ and $\langle E\rangle_B$;
5: *A* and *B* do:
6: Perform Algorithm 3 with $U = \langle D\rangle_A$ and $V = \langle E\rangle_B$ to get $W = \langle D\rangle_A\langle E\rangle_B$ such that A holds $\langle W\rangle_A$ and B holds $\langle W\rangle_B$;
7: Perform Algorithm 3 with the role of A and B reversed, $U = \langle D\rangle_B$ and $V = \langle E\rangle_A$ to get $Z = \langle D\rangle_1\langle E\rangle_0$ such that A holds $\langle Z\rangle_A$ and B holds $\langle Z\rangle_B$;
8: *A* do:
9: Set $\langle F\rangle_A = \langle D\rangle_A\langle E\rangle_A + \langle W\rangle_A + \langle Z\rangle_A$;
10: *B* do:
11: Set $\langle F\rangle_B = \langle D\rangle_B\langle E\rangle_B + \langle W\rangle_B + \langle Z\rangle_B$;

Finally, during offline phase, A and B also requested Carlos to generate sufficient number of shares for zero matrices with various dimensions.

FTL Algorithm—Secret Sharing Based

Before discussing our FTL protocol that is constructed based on Beaver triples, we first give some notation to simplify (10)–(12) based on the parties needed to complete the calculation.

1) Let $\mathcal{L}_A = \sum_i^{N_C} \ell_1(y_i^A, 0) + \gamma \sum_i^{N_{AB}} \ell_2^A(u_i^A) + \frac{\lambda}{2}\mathcal{L}_3^A$.
2) Let $\mathcal{L}_B = \gamma \sum_i^{N_{AB}} \ell_2^B(u_i^B) + \frac{\lambda}{2}(\mathcal{L}_3^B)$.
3) Let

$$\mathcal{L}_{AB} = \frac{1}{2}\sum_i^{N_C} C(y_i^A)\Phi^A\mathcal{G}(u_i^B) + \frac{1}{8}\left(D(y_i^A)\Phi^A\mathcal{G}(u_i^B)\right)\left(\Phi^A\mathcal{G}(u_i^B)\right) + \gamma\kappa\sum_i^{N_{AB}} u_i^A(u_i^B)'$$

with

- For A :
  - $\mathcal{L}_{AB}^{(A,1)} = \left(\frac{1}{2}C(y_i^A)\Phi^A\right)_{i=1,\cdots,N_C}$,
  - $\mathcal{L}_{AB}^{(A,2)} = \left(\frac{1}{8}D(y_i^A)\Phi^A\right)_{i=1,\cdots,N_C}$
  - $\mathcal{L}_{AB}^{(A,3)} = \left(\Phi^A\right)_{i=1,\cdots,N_C}$ and
  - $\mathcal{L}_{AB}^{(A,4)} = (\gamma\kappa u_i^A)_{i=1,\cdots,N_{AB}}$.
- For B :
  - $\mathcal{L}_{AB}^{(B,1)} = \left(\mathcal{G}(u_i^B)\right)_{i=1,\cdots,N_C}$, and
  - $\mathcal{L}_{AB}^{(B,2)} = (u_i^B)_{i=1,\cdots,N_C}$.

Then

$$\mathcal{L} = \mathcal{L}_A + \mathcal{L}_B + \sum_i^{N_C}\mathcal{L}_{AB}^{(A,1)}(i)\mathcal{L}_{AB}^{(B,1)}(i) + \left(\mathcal{L}_{AB}^{(A,2)}(i)\mathcal{L}_{AB}^{(B,1)}(i)\right)\left(\mathcal{L}_{AB}^{(A,3)}(i)\mathcal{L}_{AB}^{(B,1)}(i)\right) + \sum_i^{N_{AB}}\mathcal{L}_{AB}^{(A,4)}(i)\left(\mathcal{L}_{AB}^{(B,2)}(i)\right)'.$$

4) Let $D_B^{(B,\ell)} = \sum_i^{N_{AB}} \gamma\frac{\partial\ell_2^B(u_i^B)}{\partial\theta_\ell^B} + \lambda\theta_\ell^B$.
5) Let

$$D_{AB}^{(B,\ell)} = \sum_i^{N_C}\frac{1}{2}C(y_i^A)\Phi^A\frac{\partial\mathcal{G}(u_i^B)}{\partial\theta_\ell^B} + 2\left(\frac{1}{8}D(y_i^A)\Phi^A\right)\mathcal{G}(u_i^B)(\Phi^A)\left(\frac{\partial\mathcal{G}(u_i^B)}{\partial\theta_\ell^B}\right) + \sum_i^{N_{AB}}\gamma\kappa u_i^A\frac{\partial u_i^B}{\partial\theta_\ell^B}$$

with

- For B:
  - $D_{AB,B,1}^{(B,\ell)} = \left(\frac{\partial\mathcal{G}(u_i^B)}{\partial\theta_\ell^B}\right)_{i=1,\cdots,N_C,}$
  - $D_{AB,B,2}^{(B,\ell)} = \left(\frac{\partial u_i^B}{\partial\theta_\ell^B}\right)_{i=1,\cdots,N_{AB.}}$

Then

$$\frac{\partial \mathcal{L}}{\partial \theta_\ell^B} = D_B^{B,\ell} + \sum_i^{N_C} \mathcal{L}_{AB}^{(A,1)}(i) D_{AB,B,1}^{(B,\ell)}(i) + 2\Big(\mathcal{L}_{AB}^{(A,2)}(i)\mathcal{L}_{AB}^{(B,1)}(i)\Big)\Big(\mathcal{L}_{AB}^{(A,3)}(i) D_{AB,B,1}^{(B,\ell)}(i)\Big) + \sum_i^{N_{AB}} \mathcal{L}_{AB}^{(A,4)}(i) D_{AB,B,2}^{(B,\ell)}(i).$$

6) Let $D_A^{(A,\ell)} = \sum_i^{N_{AB}} \gamma \frac{\partial \ell_2^A(u_i^A)}{\partial \theta_\ell^A} + \lambda \theta_\ell^A$.
7) Let

$$D_{AB}^{(A,\ell)} = \sum_i^{N_C} \frac{1}{2} C(y_i^A) \frac{\partial \Phi^A}{\partial \theta_\ell^A} \mathcal{G}(u_i^B) + 2\left(\frac{1}{8} D(y_i^A)\Phi^A\right)\mathcal{G}(u_i^B)\left(\frac{\partial \Phi^A}{\partial \theta_\ell^A}\right)\mathcal{G}(u_i^B) + \gamma \sum_i^{N_{AB}} \left(\kappa u_i^B \frac{\partial u_i^A}{\partial \theta_\ell^A}\right)$$

with

- For A :
  - $D_{AB,A,1}^{(A,\ell)} = \left(\frac{1}{2} C(y_i^A) \frac{\partial \Phi^A}{\partial \theta_\ell^A}\right)_{i=1,\cdots,N_C}$,
  - $D_{AB,A,2}^{(A,\ell)} = \left(\frac{\partial \Phi^A}{\partial \theta_\ell^A}\right)_{i=1,\cdots,N_C}$ and
  - $D_{AB,A,3}^{(A,\ell)} = \left(\gamma\kappa \frac{\partial u_i^A}{\partial \theta_\ell^A}\right)_{i=1,\cdots,N_{AB}}$.

Then

$$\frac{\partial \mathcal{L}}{\partial \theta_\ell^A} = D_A^{(A,\ell)} + \sum_i^{N_C} D_{AB,A,1}^{(A,\ell)}(i)\mathcal{L}_{AB}^{(B,1)}(i) + 2\Big(\mathcal{L}_{A,B}^{(A,2)}\mathcal{L}_{AB}^{(B,1)}(i)\Big)\Big(D_{AB,A,2}^{(A,\ell)}(i)\mathcal{L}_{AB}^{(B,1)}(i)\Big) + \gamma \sum_i^{N_{AB}} \Big(D_{AB,A,3}^{(A,\ell}(i)(u_i^B)'\Big).$$

To perform the training scheme, both A and B first initialize and execute their respective neural networks Net^A and Net^B locally to obtain u_i^A and u_i^B. A computes $\{h_k^A(u_i^A, y_i^A)\}_{k=1,\cdots,K_A}$. Then, for each k, she randomly chooses a mask $\langle h_k^A(u_i^A, y_i^A)\rangle_A$ and sets $\langle h_k^A(u_i^A, y_i^A)\rangle_B = h_k^A(u_i^A, y_i^A) - \langle h_k^A(u_i^A, y_i^A)\rangle_A$. Then, A sends $\langle h_k^A(u_i^A, y_i^A)\rangle_B$ to B for $k = 1, \cdots, K_A$. Similarly, B computes $\{h_k^B(u_i^B)\}_{k=1,\cdots,K_B}$ and for each k, he randomly chooses $\langle h_k^B(u_i^B)\rangle_B$ and sets $\langle h_k^B(u_i^B)\rangle_A = h_k^B(u_i^B) - \langle h_k^B(u_i^B)\rangle_B$, which is then sent to A.

In our scenario, $K_A = 7$, with

1) $h_1^A(u_i^A, y_i^A) = \mathcal{L}_{AB}^{(A,1)}$,
2) $h_2^A(u_i^A, y_i^A) = \mathcal{L}_{AB}^{(A,2)}$,
3) $h_3^A(u_i^A, y_i^A) = \mathcal{L}_{AB}^{(A,3)}$,
4) $h_4^A(u_i^A, y_i^A) = \mathcal{L}_{AB}^{(A,4)}$,
5) $h_5^A(u_i^A, y_i^A) = \mathcal{D}_{AB,A,1}^{(A,\ell)}$,
6) $h_6^A(u_i^A, y_i^A) = \mathcal{D}_{AB,A,2}^{(A,\ell)}$,
7) $h_7^A(u_i^A, y_i^A) = \mathcal{D}_{AB,A,3}^{(A,\ell)}$,

and $K_B = 4$, with

1) $h_1^B(u_i^B) = \mathcal{L}_{AB}^{(B,1)}$,
2) $h_2^B(u_i^B) = \mathcal{L}_{AB}^{(B,2)}$,
3) $h_3^B(u_i^B) = \mathcal{D}_{AB,B,1}^{(B,\ell)}$,
4) $h_4^B(u_i^B) = \mathcal{D}_{AB,B,2}^{(B,\ell)}$.

Algorithm 5. FTL Training: Beaver Triples Based

Require: A holds $h_k^A, \mathcal{L}_A$, and $D_A^{(A,\ell)}$ while B holds $h_k^B, \mathcal{L}_B$, and $D_B^{(B,\ell)}$. In the offline phase, they have also generated sufficient triples with the appropriate dimensions. We also require a threshold ϵ for termination condition;

1: Calculate $\mathcal{L}_{AB}$ with $3N_C + N_{AB}$ inner products of length d and two real number multiplications. A receives $\langle \mathcal{L}_{AB}\rangle_A$ and B receives $\langle \mathcal{L}_{AB}\rangle_B$. A sets $\langle \mathcal{L}\rangle_A = \mathcal{L}_A + \langle \mathcal{L}_{AB}\rangle_A$ and B sets $\langle \mathcal{L}\rangle_B = \mathcal{L}_B + \langle \mathcal{L}_{AB}\rangle_B$;
2: Both A and B publish their shares so they can individually recover $\mathcal{L}$;
3: For each $\theta_\ell^B \in \Theta^B$, calculate $D_{AB}^{B,\ell}$ with $3N_C + N_{AB}$ inner product of vectors of length d and two real number multiplications. A receives $\langle D_{AB}^{(B,\ell)}\rangle_A$ and sets $\langle \frac{\partial \mathcal{L}}{\partial \theta_\ell^B}\rangle_A = \langle D_{AB}^{(B,\ell)}\rangle_A$. In the same time, B receives $\langle D_{AB}^{(B,\ell)}\rangle_B$ and sets $\langle \frac{\partial \mathcal{L}}{\partial \theta_\ell^B}\rangle_B = D_B^{(B,\ell)} + \langle D_{AB}^{(B,\ell)}\rangle_B$;
4: A sends $\langle \frac{\partial \mathcal{L}}{\partial \theta_\ell^B}\rangle_A$ to B;
5: B recovers $\frac{\partial \mathcal{L}}{\partial \theta_\ell^B} = \langle \frac{\partial \mathcal{L}}{\partial \theta_\ell^B}\rangle_A + \langle \frac{\partial \mathcal{L}}{\partial \theta_\ell^B}\rangle_B$;
6: B updates θ_ℓ^B;
7: For each $\theta_\ell^A \in \Theta^A$, calculate $D_{AB}^{A,\ell}$ with $3N_C + N_{AB}$ inner product of vectors of length d and two real number multiplications. A receives $\langle D_{AB}^{(A,\ell)}\rangle_A$ and sets $\langle \frac{\partial \mathcal{L}}{\partial \theta_\ell^A}\rangle_A = D_A^{(A,\ell)} + \langle D_{AB}^{(A,\ell)}\rangle_A$. In the same time, B receives $\langle D_{AB}^{(A,\ell)}\rangle_B$ and sets $\langle \frac{\partial \mathcal{L}}{\partial \theta_\ell^A}\rangle_B = \langle D_{AB}^{(A,\ell)}\rangle_B$;
8: B sends $\langle \frac{\partial \mathcal{L}}{\partial \theta_\ell^A}\rangle_B$ to A;
9: A recovers $\frac{\partial \mathcal{L}}{\partial \theta_\ell^A} = \langle \frac{\partial \mathcal{L}}{\partial \theta_\ell^A}\rangle_A + \langle \frac{\partial \mathcal{L}}{\partial \theta_\ell^A}\rangle_B$;
10: A updates θ_ℓ^A;
11: B updates θ_ℓ^B;
12: Repeat as long as $\mathcal{L}_{\text{prev}} - \mathcal{L} \geq \epsilon$;

In addition, A privately computes $\mathcal{L}_A$ and $D_A^{(A,\ell)}$ while B privately computes $\mathcal{L}_B$ and $D_B^{(B,\ell)}$. Algorithm 6 provides the training protocol for one iteration based on Beaver triples generated by Algorithm 4.

After the training is completed, we proceed to the prediction phase. Recall that after the training phase, A has the optimal value for Θ^A while B has the optimal value for Θ^B. Suppose that now B wants to learn the label for $\{x_j^B\}_{j\in N_B}$. The protocol can be found in Algorithm 6.

Algorithm 6. FTL Prediction: Beaver Triples Based

Require: A holds the optimal parameter Θ^A and B holds the optimal parameter Θ^B and unlabeled data $\{x_j^B\}_{j\in N_B}$;

1: B do:
2: Calculate $u_j^B = \mathrm{Net}^B(\Theta^B, x_j^B)$;
3: Calculate $\mathcal{G}(u_j^B)$;
4: Randomly choose $\langle \mathcal{G}(u_j^B)\rangle_B$;
5: Set $\langle \mathcal{G}(u_j^B)\rangle_A = \mathcal{G}(u_j^B) - \langle \mathcal{G}(u_j^B)\rangle_B$ and send it to A;
6: A do:
7: Calculate Φ^A
8: Randomly choose $\langle \Phi^A\rangle_A$;
9: Set $\langle \Phi^A\rangle_B = \Phi^A - \langle \Phi^A\rangle_A$ and send it to B;
10: Perform secure matrix multiplication from Algorithm 2 so A receives $\langle \Phi^A \mathcal{G}(u_j^B)\rangle_A$ and B receives $\langle \Phi^A \mathcal{G}(u_j^B)\rangle_B$.
11: B sends $\langle \Phi^A \mathcal{G}(u_j^B)\rangle_B$ to A;
12: A recovers $\varphi(u_j^B) = \Phi^A \mathcal{G}(u_j^B)$, calculates y_j^B and sends it to B;

Theorem 2. *The protocols in Algorithms 3–6 are information theoretically secure against at most one passive adversary.*

Proof. Note that in all of these algorithms, the only information that any party receives regarding any private values is only the share for an n-out-of-n secret sharing scheme. So by the property of n-out-of-n secret sharing scheme, no one can learn any information about the private values they are not supposed to learn. After the calculation, the same thing can be said since each party only learns about a share of a secret sharing scheme and they cannot learn any information regarding values they are not supposed to learn from there. ■ □

Remark 1. Using the argument by Mohassel and Zhang[7] we can improve the efficiency in the following manner; for each matrix A, it is always masked by the same random matrix. This optimization does not affect the security of the protocol while significantly improves the efficiency.

EXPERIMENTAL EVALUATION

In this section, we report experiments conducted on public datasets including: 1) NUS-WIDE dataset[9] and 2) Kaggle's *Default-of-Credit-Card-Clients*[10] ("Default-Credit") to validate our proposed approach. We study the effectiveness and scalability of the approach with respect to various key factors, including the number of overlapping samples, the dimension of hidden common representations, and the number of features.

The NUS-WIDE dataset consists of 634 low-level features from Flickr images as well as their associated tags and ground truth labels. There are in total 81 ground truth labels. We use the top 1000 tags as text features and combine all the low-level features including color histograms and color correlograms as image features. We consider solving a one-versus-all classification problem with a data federation formed between party A and party B, where A has 1 000 text tag features and labels, whereas party B has 634 low-level image features.

The "Default-Credit" dataset consists of credit card records including user demographics, history of payments, and bill statements, etc., with users' default payments as labels. After applying one-hot encoding to categorical features, we obtain a dataset with 33 features and 30 000 samples. We then split the dataset both in the feature space and the sample space to simulate a two-party federation problem. Specifically, we assign each sample to party A, party B or both so that there exists a small number of samples overlapping between A and B. All labels are on the side of party A. We will examine the scalability (see the "Scalability" section) of the FTL algorithm by dynamically splitting the feature space.

Impact of Taylor Approximation

We studied the effect of Taylor approximation by monitoring and comparing the training loss decay and the performance of prediction. Here, we test the convergence and precision of

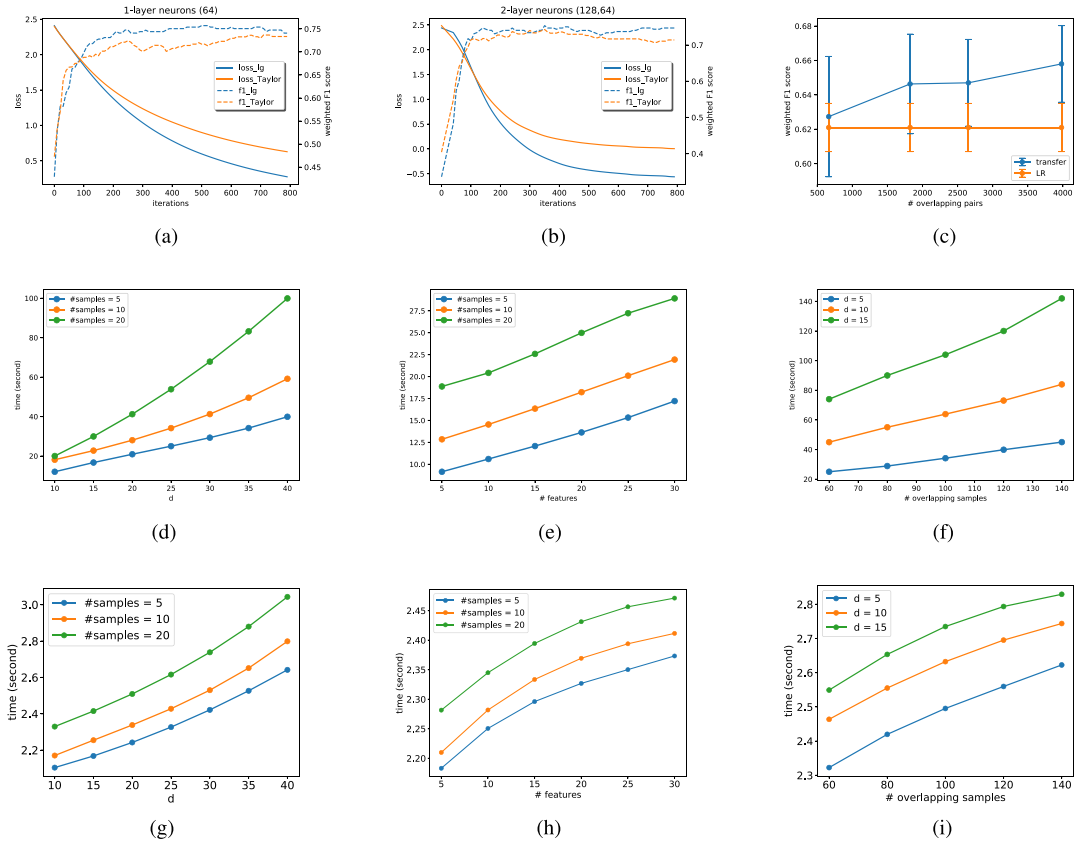

Figure 2. Experiment results. (a) Learning loss (1-layer). (b) Learning loss (2-layer). (c) F1 versus # overlapping pairs. (d) Time versus d (HE). (e) Time versus # features (HE). (f) Time versus # samples (HE). (g) Time versus d (SS). (h) Time versus # features (SS). (I) Time versus # samples (SS).

the algorithm using the NUS-WIDE data and neural networks with different levels of depth. In the first case, Net^A and Net^B both have one autoencoder layer with 64 neurons, respectively. In the second case, Net^A and Net^B both have two autoencoder layers with 128 and 64 neurons, respectively. In both cases, we used 500 training samples, 1396 overlapping pairs, and set $\gamma = 0.05, \lambda = 0.005$. We summarize the results in Figure 2(a) and (b). We found that the loss decays at a similar rate when using Taylor approximation as compared to using the full logistic loss, and the weighted F1 score of the Taylor approximation approach is also comparable to the full logistic approach. The loss converges to a different minima in each of these cases. As we increased the depth of the neural networks, the convergence and the performance of the model did not decay.

Most existing secure deep learning frameworks suffer from accuracy loss when adopting privacy-preserving techniques.[7] Using only low-degree Taylor approximation, the drop in accuracy in our approach is much less than the state-of-art secure neural networks with similarly approximations.

Performance

We tested SS-based FTL (SST), HE-based FTL with Taylor loss (TLT), and FTL with logistic loss (TLL). For the self-learning approach, we picked three machine learning models: 1) LR, 2) SVM, and 3) stacked autoencoders (SAEs). The SAEs are of the same structure as the ones we used for transfer learning, and are connected to a logistic layer for classification. We picked three of the most frequently occurring labels in the NUS-WIDE dataset, i.e., water, person, and sky, to conduct one versus others binary classification tasks. For each experiment, the number of overlapping samples we used is half of the total number of samples in that category. We varied

Table 1. Comparison of weighted F1 scores.

Tasks	N_c	SST	TLT	TLL	LR	SVMs	SAEs
water vs. others	100	**0.698** $\pm$ 0.011	0.692 $\pm$ 0.062	0.691 $\pm$ 0.060	0.685 $\pm$ 0.020	0.640 $\pm$ 0.016	0.677 $\pm$ 0.048
water vs. others	200	**0.707** $\pm$ 0.013	0.702 $\pm$ 0.010	0.701 $\pm$ 0.007	0.672 $\pm$ 0.023	0.643 $\pm$ 0.038	0.662 $\pm$ 0.010
person vs. others	100	**0.703** $\pm$ 0.015	0.697 $\pm$ 0.010	0.697 $\pm$ 0.020	0.694 $\pm$ 0.026	0.619 $\pm$ 0.050	0.657 $\pm$ 0.030
person vs. others	200	**0.735** $\pm$ 0.004	0.733 $\pm$ 0.009	**0.735** $\pm$ 0.010	0.720 $\pm$ 0.004	0.706 $\pm$ 0.023	0.707 $\pm$ 0.008
sky vs. others	100	0.708 $\pm$ 0.015	0.700 $\pm$ 0.022	**0.713** $\pm$ 0.006	0.694 $\pm$ 0.016	0.679 $\pm$ 0.018	0.667 $\pm$ 0.009
sky vs. others	200	**0.724** $\pm$ 0.014	0.718 $\pm$ 0.033	0.718 $\pm$ 0.024	0.696 $\pm$ 0.026	0.680 $\pm$ 0.042	0.684 $\pm$ 0.056

the size of the training sample set and conducted three tests for each experiment with different random partitions of the samples. The parameters λ and γ are optimized via cross validation.

Figure 2(c) shows the effect of varying the number of overlapping samples on the performance of transfer learning. The overlapping sample pairs are used to bridge the hidden representations between the two parties. The performance of FTL improves as the overlap between datasets increases.

The comparison of F-score (mean $\pm$ std) among SST, TLT, TLL and the several other machine learning models is shown in Table 1. We observe that SST, TLT, and TLL yield comparable performance across all tests. This demonstrates that SST can achieve plain-text level accuracy while TLT can achieve almost lossless accuracy although Taylor approximation is applied. The three FTL models outperform baseline self-learning models significantly using only a small set of training samples under all experimental conditions. In addition, performance improves as we increased the number of training samples. The results demonstrated the robustness of FTL.

Scalability

We study the scalability using Default-Credit dataset because it allows us to conveniently choose features when we do experiments. Specifically, we study how the training time scales with the number of overlapping samples, the number of target-domain features, and the dimension of hidden representations, denote as d. Based on the algorithmic detail of proposed transfer learning approach, the communication cost for B sending a message to A can be calculated by formula $\text{Cost}_{B\rightarrow A} = n * (d^2 + d) * ct$, where ct is the size of the message and n is the number of samples sent. The same cost applies when sending message from A to B.

To speed up the secure FTL algorithm, we preform compute-intensive operations in parallel. The logic flow of parallel secure FTL algorithm includes three stages: parallel encryption, parallel gradient calculation, and parallel decryption. Detailed logic flow is shown in Figure 3.

On parallel encryption stage, we parallelly encrypt components that will be sent to the other party. On parallel gradient calculation stage, we parallelly perform operations, including matrix multiplication and addition, on encrypted components to calculate encrypted gradients. On parallel decryption stage, we parallelly decrypt masked loss and gradients. Finally, the two parties exchange decrypted masked gradients that will be used to update neural networks. With 20 partitions, the parallel scheme can boost the secure FTL 100x than sequential scheme.

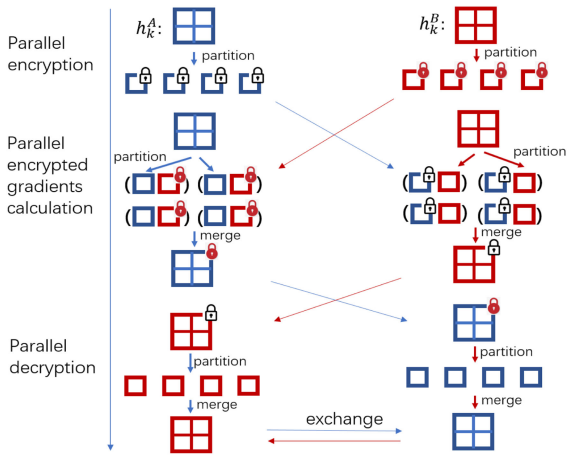

Figure 3. Logic flow of parallel secure FTL.

Table 2. Comparison of training time between SS and HE with the increasing dimension of hidden representation denoted by d, the increasing number of target-domain features, and the increasing number of overlapping samples, respectively.

protocol \ d	15	20	25	30	35	40
HE training time (sec)	29.12	41.03	53.88	66.74	81.91	101.02
SS training time (sec)	2.41	2.52	2.61	2.73	2.89	3.04
protocol \ # features	5	10	15	20	25	30
HE training time (sec)	17.82	20.45	21.67	24.03	27.12	28.58
SS training time (sec)	2.28	2.35	2.39	2.43	2.46	2.48
protocol \ # samples	60	80	100	120	140	
HE training time (sec)	74.21	89.91	111.12	123.79	146.48	
SS training time (sec)	2.55	2.65	2.74	2.79	2.83	

Figure 2(d)–(f) illustrates that with parallelism applied, the running time of HE-based FTL grows approximately linearly with respect to the size of the hidden representation dimension, the number of target-domain features, as well as the number of overlapping samples, respectively.

Figure 2(g)–(i) illustrates how the training time varies with the three key factors in the SS setting. The communication cost can be simplified as $O(d^2)$ if keeping other factors constant. As illustrated in Figure 2(g), however, the increasing rate of the training time is approaching linear rather than $O(d^2)$. We conjecture that this is due to the computational efficiency of SS-based FTL. Besides, as illustrated in Figure 2(h) and (i), respectively, as the feature sizes or overlapping samples increase, the increasing rate of training time drops.

Furthermore, we compare the scalability of SS-based with that of HE-based FTL along the axis of the hidden representation dimension, the number of features, and the number of overlapping samples, respectively. The results are presented in Table 2. We notice that SS-based FTL is running much faster than HE-based FTL. Overall, SS-based FTL speeds up by 1-2 orders of magnitude compared with HE-based FTL. In addition, as shown in the three tables, the increasing rate of the training time of SS-based FTL is much slower than that of HE-based FTL.

CONCLUSIONS AND FUTURE WORK

In this article, we proposed a secure FTL framework to expand the scope of existing secure federated learning to broader real-world applications. Two secure approaches, namely, HE and secret sharing are proposed in this article for preserving privacy. The HE approach is simple, but computationally expensive. The biggest advantages of the secret sharing approach include 1) there is no accuracy loss and 2) computation is much faster than HE approach. The major drawback of the secret sharing approach is that one has to offline generate and store many triplets before online computation.

We demonstrated that, in contrast to existing secure deep learning approaches which suffer from accuracy loss, FTL is as accurate as nonprivacy-preserving approaches, and is superior to nonfederated self-learning approaches. The proposed framework is a general privacy-preserving FTL solution that is not restricted to specific models.

In future research, we will continue improving the efficiency of the FTL framework by using distributed computing techniques with less expensive computation and communication schemes.

REFERENCES

1. EU, "Regulation (EU) 2016/679 of the European Parliament and of the Council on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)," 2016. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT
2. H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," 2016. [Online]. Available: http://arxiv.org/abs/1602.05629
3. A. Gascón *et al.*, "Secure linear regression on vertically partitioned datasets," *IACR Cryptology ePrint Archive*, vol. 2016, p. 892, 2016.
4. S. J. Pan, X. Ni, J.-T. Sun, Q. Yang, and Z. Chen, "Cross-domain sentiment classification via spectral feature alignment," in *Proc. 19th Int. Conf. World Wide Web*, 2010, pp. 751–760.
5. K. Bonawitz *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. Conf. Comput. Commun. Secur.*, 2017, pp. 1175–1191.
6. N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy," Tech. Rep. MSR-TR-2016-3, 2016.

7. P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," *IEEE Symp. Secur. Privacy*, pp. 19–38, 2017, doi: 10.1109/SP.2017.12.
8. W. Du, Y. S. Han, and S. Chen, "Privacy-preserving multivariate statistical analysis: Linear regression and classification," in *Proc. 4th SIAM Int. Conf. Data Mining*, 2004, pp. 222–233.
9. T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng, "NUS-WIDE: A real-world web image database from National University of Singapore," in *Proc. ACM Int. Conf. Image Video Retrieval*, 2009, Art. no. 48.
10. Kaggle, "Default of credit card clients dataset:https://www.kaggle.com/uciml/default-of-credit-card-clients-dataset," 2019. [Online]. Available: https://www.kaggle.com/uciml/default-of-credit-card-clients-dataset

Yang Liu is currently a Senior Researcher with the AI Department of WeBank, Shenzhen, China. Her research interests include federated learning, transfer learning, multiagent systems, statistical mechanics, and applications of these technologies in the financial industry. She received the Ph.D. degree from Princeton University, Princeton, NJ, USA, in 2012 and the Bachelor's degree from Tsinghua University, Beijing, China, in 2007. She holds multiple patents. Her research has been published in leading scientific journals such as ACM TIST and Nature. Contact her at yangliu@webank.com

Yan Kang is currently an Engineer and Researcher with the AI Department of WeBank, Shenzhen, China. His work is focusing on the research and implementation of privacy-preserving machine learning and federated transfer learning techniques. He received the Ph.D. degree from the University of Maryland, Baltimore, MD, USA. He participated in multiple projects collaborated with the National Institute of Standards and Technology and had been working on system design and implementation for more than four years. Contact him at yangkang@webank.com

Chaoping Xing is currently a Chair Professor with the Shanghai Jiaotong University, Shanghai, China, since September 2019. He received the Ph.D. degree in 1990 from the University of Science and Technology of China, Hefei, China. From March of 1998 to November of 2007, he was working with the National University of Singapore as an Assistant/Associate/Full Professor. From December of 2007 to August of 2019, he was working with Nanyang Technological University as a Full Professor. He has been working on the areas of cryptography and coding theory. Contact him at xingcp@sjtu.edu.cn.

Tianjian Chen is currently the Deputy General Manager with the AI Department of WeBank, Shenzhen, China. He is now responsible for building the Banking Intelligence Ecosystem based on Federated Learning Technology. Before joining WeBank, he was the Chief Architect of Baidu Finance, Principal Architect of Baidu. He has more than 12 years of experience in large-scale distributed system design and enabling technology innovations in various application fields. Contact him at tobychen@webank.com.

Qiang Yang is currently the Head of AI at WeBank, Shenzhen, China, (Chief AI Officer) and Chair Professor with the Computer Science and Engineering Department of the Hong Kong University of Science and Technology. His research interests include transfer learning, automated planning, federated learning, and case-based reasoning. He is a Fellow of several international societies, including ACM, AAAI, IEEE, IAPR, and AAAS. He received the B.Sc. degree in astrophysics from Peking University, Beijing, China, in 1982, and the Ph.D. degree from the Computer Science Department in 1989 and the M. Sc. degree in astrophysics in 1985, both from the University of Maryland, College Park, MD, USA. He is Fellow of the IEEE. Contact him at qyang@cse.ust.hk.