Industrial Scale Privacy Preserving Deep Neural Network

Longfei Zheng, Chaochao Chen*, Yingting Liu, Bingzhe Wu, Xibin Wu, Li Wang, Lei Wang, Jun Zhou
Ant Financial Services Group
{zlf206411,chaochao.ccc,yingting,fengyuan.wbz,xibin.wxb,raymond.wangl,shensi.wl,jun.zhoujun}@antfin.com

Abstract

Deep Neural Network (DNN) has been showing great potential in kinds of real-world applications such as fraud detection and distress prediction. Meanwhile, data isolation has become a serious problem currently, i.e., different parties cannot share data with each other. To solve this issue, most research leverages cryptographic techniques to train secure DNN models for multi-parties without compromising their private data. Although such methods have strong security guarantee, they are difficult to scale to deep networks and large datasets due to its high communication and computation complexities. To solve the scalability of the existing secure Deep Neural Network (DNN) in data isolation scenarios, in this paper, we propose an industrial scale privacy preserving neural network learning paradigm, which is secure against semi-honest adversaries. Our main idea is to split the computation graph of DNN into two parts, i.e., the computations related to private data are performed by each party using cryptographic techniques, and the rest computations are done by a neutral server with high computation ability. We also present a defender mechanism for further privacy protection. We conduct experiments on real-world fraud detection dataset and financial distress prediction dataset, the encouraging results demonstrate its practicalness.

Introduction

Deep Neural Network (DNN) has been showing great potential in kinds of machine learning tasks and successfully applying in various applications such as computer vision (Howard et al. 2017), sales forecasting (Chen et al. 2019), recommender system (Zhu et al. 2019), and financial distress prediction (Wei-Sen Chen 2009), due to its powerful representation ability. Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction (LeCun, Bengio, and Hinton 2015). The general structure of a neural network is shown in Figure 1. Meanwhile, data isolation has become a serious problem currently, especially with kinds of national data protection regulations coming into force. That is, different organizations (parties) are reluctant or cannot share sensitive data with each other due to competition or regulation reasons. Such data isolation problem has limited the power of DNN, since DNN usually achieves better performance with more data.

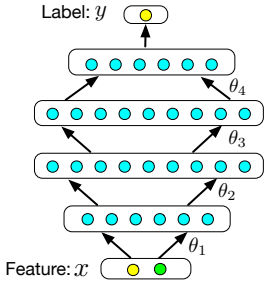


Figure 1: Structure of a traditional neural network.

To solve this problem, existing researches adopted cryptographic techniques, e.g., homomorphic encryption (Gilad-Bachrach et al. 2016) or secure multi-party computation (Mohassel and Zhang 2017), for multi-parties to train privacy preserving neural networks. Although such cryptographic based neural networks have strong security guarantee, they are difficult to scale to deep network structures and large datasets due to its high communication and computation complexities. However, in industry, the real-world applications happened to have two characteristics: (1) the datasets are large due to millions of data are hold by big companies, and (2) the neural network structures are deep so as to learn the patterns in big data. Therefore, efficiency becomes a main challenge when applying existing privacy preserving neural networks in practice.

To ingeniously balance the privacy and scalability, in this paper, we propose an industrial-scale privacy preserving neural network learning paradigm. Motivated by split learning (Vepakomma et al. 2018), we split the computation graph of DNN into two kinds, i.e., the computations related to private data are performed by each party using cryptographic techniques, and the rest computations are done by a neutral server with high computation ability. To this end, both private data and model are hold by data holders, and the heavy non-private data related computations are done by a neutral server. To further protect data privacy, we propose a defender mechanism when training the model so that the neural server cannot infer the private input of data holders from the hidden layers. Therefore, our proposal not only preserves data private, but also has good scalability. We conduct

*Corresponding author

experiments on real-world fraud detection and distress prediction datasets, the results demonstrate that our proposed privacy preserving neural network has almost the same performance with the traditional neural model.

Our main contributions are summarized as follows:

- We propose an industrial-scale privacy preserving neural network learning paradigm with defender mechanism, which not only preserves data privacy, but also has good scalability.
- We implement our model on decentralized network settings, where computation nodes have their own private data and they can train privacy preserving neural network models.
- Our proposal is verified on real-world datasets and the results show its superiority.

Related Work

We first simply review deep learning models and then describe two popular types of privacy preserving neural network models.

Deep learning. Deep Neural Network (DNN) has been showing great power in kinds of machine learning tasks, since it can learn complex functions by composing multiple non-linear modules to transform representations from low-level raw inputs to high-level abstractions (Gu et al. 2019). Mathematically, the forward procedure of a DNN can be defined as a representation function f that maps an input $\mathbf{X}$ to an output y, i.e., $y = f(\mathbf{X}, \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is model parameter. Assume a DNN has L layers, then f is composed of L sub-functions $f_{l|l\in[1,L]}$, which are connected in a chain. That is, $f(\mathbf{X}) = f_L f_{L-1} ... f_1(\mathbf{X}, \boldsymbol{\theta}_1)$, as is shown in Figure 1.

Cryptographic based methods. These methods use cryptographic techniques, e.g., secret sharing and homomorphic encryption, to build approximated neural networks models (Mohassel and Zhang 2017; Wagh, Gupta, and Chandran 2019), since the nonlinear active functions are not cryptographically computable. These models are difficult to scale to deep networks and large datasets due to the high communication and computation complexities of the cryptographic techniques. In this paper, we use cryptographic techniques for data holders to calculate the hidden layers securely.

Split neural graph based methods. These methods split the computation graph of neural networks into two parts, i.e., let data holders calculate the private data related computations individually and get a hidden layer, and then let a server makes the rest computations (Gupta and Raskar 2018; Vepakomma et al. 2018; Osia et al. 2019; Gu et al. 2019). For example, Gu et al. (Gu et al. 2019) proposed to enclose sensitive computation in a trusted execution environment, i.e., Intel Software Guard Extensions (McKeen et al. 2013), to mitigate input information disclosures, and then delegate non-sensitive workloads with hardware-assisted deep learning acceleration. Our model differs from them in mainly two aspects. First, we use cryptographic techniques for data holders to calculate the hidden layers collaboratively rather than compute them based on their plaintext data individually. Seconds, we propose a defender mechanism when training the model so that the neutral server cannot infer the private input of data holders from the hidden layers. Therefore, our model has better privacy guarantee.

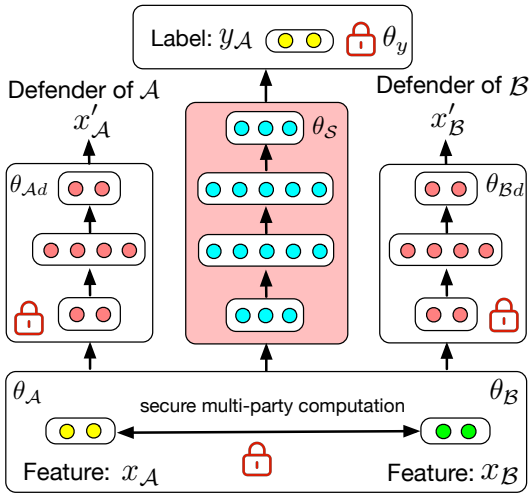


Figure 2: The proposed privacy preserving neural network. The middle pink part is performed on a neutral server and the rest are done by data holders.

The Proposed Method

Problem Description

We start from a concrete example. Suppose there are two financial companies, i.e., $\mathcal{A}$ and $\mathcal{B}$, who both need to detect fraud users. $\mathcal{A}$ has some user features ($\mathbf{X}_A$) and labels ($\mathbf{y}_A$), and $\mathcal{B}$ has features ($\mathbf{X}_B$) for the same batch of users. Although $\mathcal{A}$ can build a Deep Neural Network (DNN) for fraud detection using its own data, the model performance can be improved by incorporating features of $\mathcal{B}$. However, these two companies can not share data with each other due to the fact that leaking users' private data is against regulations. This is a classic data isolation problem. It is challenging for both parties to build a privacy preserving neural network collaboratively without compromising their private data. In this paper, we only consider the situation where two data holders have the same sample set, one of them ($\mathcal{A}$) has partial features and labels, and the other ($\mathcal{B}$) has the rest partial features. Our proposal can be naturally extended to more than two parties.

Proposal Overview

We propose a novel privacy preserving neural network learning framework for the above challenge. As described in related work, DNN can be defined as a layer-wise representation function. Motivated by the existing work (Gupta and Raskar 2018; Vepakomma et al. 2018; Osia et al. 2019; Gu et al. 2019), we propose to decouple the computation graph of DNN into two kinds, i.e., the computations related to private data are performed by each party using cryptographic techniques, and the rest computations are done by a neutral server with high computation ability. Here, the private data are the input and output of the neural network, which corresponding to the private features and labels from data holders.

Specifically, we divide the model parameters (θ) into three parts, *the computations that are related to private features on both data holders* ($\theta_{\mathcal{A}}$ and $\theta_{\mathcal{B}}$), *the computations related to private labels on a data holder* (θ_y), and *the rest heavy hidden layer related computations on a neutral server* ($\theta_{\mathcal{S}}$). As shown in Figure 2, the first two parts are private data related computations and therefore are performed by data holders themselves using secure multi-party computation techniques, and the rest computations can be done by a neutral server which has rich computation resources. Moreover, as has been pointed out by literature, attackers may recover the raw input data given the hidden layers of a DNN. To prevent the neutral server inferring the private input of data holders from the hidden layers, we propose a *defender mechanism* when training the model. Our solution is against *semi-honest adversary*, i.e., the corrupted participants will still strictly follow the protocol but may want to learn more information. We will describe each module in details in the following subsections.

Private Feature Related Computations

Private feature related computations refer to data holders collaboratively calculate the hidden layer of a DNN using their own private data. Here, data holders want to (1) calculate a common function, i.e., $f_1(\mathbf{X}_A, \mathbf{X}_B)$, collaboratively and (2) keep their features, i.e., $\mathbf{X}_A$ and $\mathbf{X}_B$, private. Secure multi-party computation (Yao 1982) was born to solve this problem. Mathematically, $\mathcal{A}$ and $\mathcal{B}$ have partial features ($\mathbf{X}_A$ and $\mathbf{X}_B$) and partial model parameters ($\boldsymbol{\theta}_A$ and $\boldsymbol{\theta}_B$), respectively, and they want to compute the output of the first hidden layer collaboratively. That is, $\mathcal{A}$ and $\mathcal{B}$ want to compute $\mathbf{h}_1 = f_1(\mathbf{X}_A \oplus \mathbf{X}_B, \boldsymbol{\theta}_A \oplus \boldsymbol{\theta}_B)$, where $\oplus$ denotes concatenation operation and f_1 is the active function.

We propose to solve the above problem using secret sharing (Shamir 1979). The main technique used is secret sharing based matrix addition and multiplication on fixed-point numbers. Please refer to (Mohassel and Zhang 2017) for more details. Assuming f_1 is a linear active function, we propose a secure protocol in Algorithm 1. Note that the nonlinear active functions can be approximated by using polynomials or Taylor expansion (Hardy et al. 2017). To this end, $\mathcal{A}$ and $\mathcal{B}$ each obtains a partial share of the hidden layer, i.e., $\mathbf{h}_1 = \langle \mathbf{h}_1 \rangle_A + \langle \mathbf{h}_1 \rangle_B$.

Hidden Layer Related Computations

After $\mathcal{A}$ and $\mathcal{B}$ obtain the shares of the first hidden layer, they send them to a neutral server for hidden layer related computations, i.e., $\mathbf{h}_L = f(\mathbf{h}_l, \boldsymbol{\theta}_S)$. This is the same as the traditional neural networks. Given l-th hidden layer $\mathbf{h}_l$, where $1 \leq l \leq L-1$ and L be the number of hidden layers, the $(l+1)$-th hidden layer can be calculated by

$$\mathbf{h}_{l+1} = f_l(\mathbf{h}_l, \boldsymbol{\theta}_l), \tag{1}$$

where $\boldsymbol{\theta}_l$ is the parameters in l-th layer, and f_l is the active function of the l-th layer. These are the most time-consuming computations, because there are many nonlinear operations, e.g., max pooling, are not cryptographically friendly. We leave these heavy computations on a neutral server who has strong computation power. To this end, our model can scale to large dataset.

Private Label Related Computations

After the neutral server finishes the hidden layer related computations, it sends the final hidden layer $\mathbf{h}_L$ to the data holder who has the label, i.e., $\mathcal{A}$ in this case, for computing predictions. That is

$$\hat{\mathbf{y}} = \delta(\mathbf{h}_L, \boldsymbol{\theta}_L), \tag{2}$$

where δ is designed based on different prediction tasks, e.g., δ be the logistic function for a binary classification task.

Strengthening Privacy with Defender Mechanism

To further protect data privacy, we propose a defender mechanism when training the model so that the neural server cannot infer the private input of data holders from the hidden layers. As can be seen in Figure 2, the defender tries to learn a representation that maps hidden layer to the private input (features), just as an attacker would do. Given the hidden layer ($\mathbf{h}_1$), the recovered input of $\mathcal{A}$ and $\mathcal{B}$ are $f(\mathbf{h}_1, \boldsymbol{\theta}_{\mathcal{A}d})$ and $f(\mathbf{h}_1, \boldsymbol{\theta}_{\mathcal{B}d})$, respectively. Therefore, to protect the private features being recovered, the defender losses of $\mathcal{A}$ and $\mathcal{B}$ are

$$\max_{\boldsymbol{\theta}_{\mathcal{A}d}} d(\mathbf{X}_A, f(\mathbf{h}_1, \boldsymbol{\theta}_{\mathcal{A}d})), \tag{3}$$

$$\max_{\boldsymbol{\theta}_{\mathcal{B}d}} d(\mathbf{X}_B, f(\mathbf{h}_1, \boldsymbol{\theta}_{\mathcal{B}d})), \tag{4}$$

where $\boldsymbol{\theta}_{\mathcal{A}d}$ and $\boldsymbol{\theta}_{\mathcal{B}d}$ are the defender model of $\mathcal{A}$ and $\mathcal{B}$ and $d(\cdot, \cdot)$ measures the distance between original input and recovered input. With the present of the defender, it becomes difficult for the server to infer the input given the hidden layer and the corresponding input. We will empirically study the effect of the defender on privacy protection in experiments.

Putting All together

Our model consists of the private feature related computations, hidden layer related computations, private label related computations, and the defender. The first three parts compute the output loss of the DNN, and the last part is the defender loss. Thus, the total loss becomes

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) - \lambda \cdot (d(\mathbf{X}_A, f(\mathbf{h}_1, \boldsymbol{\theta}_{\mathcal{A}d})) + d(\mathbf{X}_B, f(\mathbf{h}_1, \boldsymbol{\theta}_{\mathcal{B}d}))), \tag{5}$$

where λ is the defender weight, and $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ is designed based on different prediction tasks, e.g., $\mathcal{L}$ be the logistic loss for a binary classification task.

Learning Model Parameters

The loss function in Eq. (5) is difficult to be solved due to the complex architectures. We learn the loss function using iterative optimization method via gradient descent using back propagation (LeCun, Bengio, and Hinton 2015), as summarized in Algorithm 2. Both forward computation and backward computation need communication between $\mathcal{A}$, $\mathcal{B}$, and the server, in a decentralized manner. During training, all the private data ($\mathbf{X}_A$, $\mathbf{X}_B$, and $\mathbf{y}$) and private data related model parameters ($\boldsymbol{\theta}_A$, $\boldsymbol{\theta}_B$, and $\boldsymbol{\theta}_y$) are kept by data holders. Therefore, data privacy is kept to a large extent.

Algorithm 1: Data holders $\mathcal{A}$ and $\mathcal{B}$ securely calculate the first hidden layer using secret sharing

Input: features of $\mathcal{A}$ and $\mathcal{B}$ ($\mathbf{X}_A$ and $\mathbf{X}_B$) and current models of $\mathcal{A}$ and $\mathcal{B}$ ($\boldsymbol{\theta}_A$ and $\boldsymbol{\theta}_B$)
Output: The share of first hidden layer for $\mathcal{A}$ and $\mathcal{B}$

$\mathcal{A}$ and $\mathcal{B}$ locally generate $\langle \mathbf{X}_A \rangle_1$ and $\langle \mathbf{X}_A \rangle_2$, and $\langle \mathbf{X}_B \rangle_1$ and $\langle \mathbf{X}_B \rangle_2$, respectively
$\mathcal{A}$ and $\mathcal{B}$ locally generate $\langle \boldsymbol{\theta}_A \rangle_1$ and $\langle \boldsymbol{\theta}_A \rangle_2$, and $\langle \boldsymbol{\theta}_B \rangle_1$ and $\langle \boldsymbol{\theta}_B \rangle_2$, respectively
$\mathcal{A}$ distributes $\langle \mathbf{X}_A \rangle_2$ and $\langle \boldsymbol{\theta}_A \rangle_2$ to $\mathcal{B}$
$\mathcal{B}$ distributes $\langle \mathbf{X}_B \rangle_1$ and $\langle \boldsymbol{\theta}_B \rangle_1$ to $\mathcal{A}$
$\mathcal{A}$ locally calculates $\langle \mathbf{X} \rangle_1 = \langle \mathbf{X}_A \rangle_1 \oplus \langle \mathbf{X}_B \rangle_1$, $\langle \boldsymbol{\theta} \rangle_1 = \langle \boldsymbol{\theta}_A \rangle_1 \oplus \langle \boldsymbol{\theta}_B \rangle_1$, and $\langle \mathbf{X} \rangle_1 \times \langle \boldsymbol{\theta} \rangle_1$
$\mathcal{B}$ locally calculates $\langle \mathbf{X} \rangle_2 = \langle \mathbf{X}_A \rangle_2 \oplus \langle \mathbf{X}_B \rangle_2$, $\langle \boldsymbol{\theta} \rangle_2 = \langle \boldsymbol{\theta}_A \rangle_2 \oplus \langle \boldsymbol{\theta}_B \rangle_2$, and $\langle \mathbf{X} \rangle_2 \times \langle \boldsymbol{\theta} \rangle_2$
$\mathcal{A}$ and $\mathcal{B}$ calculate $\langle \mathbf{X} \rangle_1 \times \langle \boldsymbol{\theta} \rangle_2$ and $\langle \mathbf{X} \rangle_2 \times \langle \boldsymbol{\theta} \rangle_1$ using secret sharing matrix multiplication, $\mathcal{A}$ get $\langle \mathbf{X}_1 \times \boldsymbol{\theta}_2 \rangle_A$ and $\langle \mathbf{X}_2 \times \boldsymbol{\theta}_1 \rangle_A$, $\mathcal{B}$ gets $\langle \mathbf{X}_1 \times \boldsymbol{\theta}_2 \rangle_B$ and $\langle \mathbf{X}_2 \times \boldsymbol{\theta}_1 \rangle_B$
$\mathcal{A}$ locally calculates $\langle \mathbf{X} \times \boldsymbol{\theta} \rangle_A = \langle \mathbf{X} \rangle_1 \times \langle \boldsymbol{\theta} \rangle_1 + \langle \mathbf{X}_1 \times \boldsymbol{\theta}_2 \rangle_A + \langle \mathbf{X}_2 \times \boldsymbol{\theta}_1 \rangle_A$
$\mathcal{B}$ locally calculates $\langle \mathbf{X} \times \boldsymbol{\theta} \rangle_B = \langle \mathbf{X} \rangle_2 \times \langle \boldsymbol{\theta} \rangle_2 + \langle \mathbf{X}_1 \times \boldsymbol{\theta}_2 \rangle_B + \langle \mathbf{X}_2 \times \boldsymbol{\theta}_1 \rangle_B$
return $\langle \mathbf{X} \times \boldsymbol{\theta} \rangle_A$ for $\mathcal{A}$ and $\langle \mathbf{X} \times \boldsymbol{\theta} \rangle_B$ for $\mathcal{B}$

Algorithm 2: Privacy preserving neural network

Input: Features of $\mathcal{A}$ ($\mathbf{X}_A$), features of $\mathcal{B}$ ($\mathbf{X}_B$), a neutral server ($\mathcal{S}$), and the number of iteration (T)
Output: Trained model ($\boldsymbol{\theta}$) and defender ($\boldsymbol{\theta}_{\mathcal{A}d}$ and $\boldsymbol{\theta}_{\mathcal{B}d}$)

$\mathcal{A}$, $\mathcal{B}$, and the neutral server initialize model parameters
for $t = 1$ *to* T **do**
  for *each mini-batch in training datasets* **do**
    # Forward computation
    $\mathcal{A}$ and $\mathcal{B}$ collaboratively learn the first hidden layer based on Algorithm 1 and send the result to $\mathcal{S}$
    $\mathcal{A}$ and $\mathcal{B}$ calculate the recovered input by $f(\mathbf{h}_1, \boldsymbol{\theta}_{\mathcal{A}d})$ and $f(\mathbf{h}_1, \boldsymbol{\theta}_{\mathcal{B}d})$, respectively
    $\mathcal{S}$ calculates the rest hidden layers by $\mathbf{h}_L = f(\mathbf{h}_l, \boldsymbol{\theta}_S)$
    $\mathcal{S}$ sends $\mathbf{h}_L$ back to $\mathcal{A}$
    $\mathcal{A}$ makes predictions by Eq. (2)
    # Backward computation
    Update model parameters $\boldsymbol{\theta}$, including $\boldsymbol{\theta}_A$, $\boldsymbol{\theta}_B$, $\boldsymbol{\theta}_S$, and $\boldsymbol{\theta}_y$, using the gradient $\Delta_{\boldsymbol{\theta}}$Eq. (5)
    Update defender parameters $\boldsymbol{\theta}_d$ using the gradient $\Delta_{\boldsymbol{\theta}_{\mathcal{A}d}}$Eq. (3) and $\Delta_{\boldsymbol{\theta}_{\mathcal{B}d}}$Eq. (4)
  end
end
return Trained model ($\boldsymbol{\theta}$) and defender ($\boldsymbol{\theta}_{\mathcal{A}d}$ and $\boldsymbol{\theta}_{\mathcal{B}d}$)

It is worth noticed that our proposal can be generalized to the situations that the data holders collaboratively calculate i ($1 \leq i \leq L$) hidden layers instead of the first hidden layer only. Therefore, the existing method (Mohassel and Zhang 2017) is one of our special cases, i.e., $\mathcal{A}$ and $\mathcal{B}$ collaboratively calculate all the neural networks using secure multi-party computation techniques, without the neutral server.

Implementation

Communication and computation are two key parts of the decentralized implementation. We will describe our solution in details.

Communication. The communication includes two parts, the communication between data holder $\mathcal{A}$ and data holder $\mathcal{B}$, and the communication between both data holders and server. We adopt Google's gRPC protocol[1] to make connection and exchange data between server and data holders. Before training, we configure detailed parameters for server and data holders, such as IP addresses, gateways, and dataset locations. At the beginning of the training, server and data holders shake hands to build connection. After that, they exchange data to finish model training following Algorithm 2.

Computation. The computation are mainly in two parts, i.e., the computations by data holders and the computations by server. First, we implement the computations by data holders using Python by ourselves. Second, for the heavy computations by server, we choose TensorFlow[2] as backend to perform forward and backward computations. Note that our proposal can be easily implemented by using other deep learning platforms such as PyTorch.

Empirical Study

Experimental Settings

Datasets. To test the effectiveness of our proposed model, we choose two public benchmark datasets from Kaggle, both of which are binary classification tasks. The first one is a fraud detection dataset (Dal Pozzolo et al. 2014), where

[1] https://grpc.io/

[2] https://www.tensorflow.org/

Table 1: Comparison results on two datasets in terms of AUC

AUC	NN	P^2N^2
Fraud Detection	0.9270	0.9231
Financial Distress	0.9379	0.9314

there are 28 features and 284,807 transactions. The other one is financial distress dataset, where there are 85 features and 3,672 transactions. After we encode the categorical features, there are 556 features in total. We assume these features are hold by two parties, and each of them has equal partial features. Moreover, we randomly split the fraud detection dataset into two parts: 80% as training dataset and the rest as test dataset. We also randomly split the financial distress dataset into 70% and 30%, since the test dataset needs more samples. We repeat experiments five times and report their average results.

Metrics. We adopt Area Under the receiver operating characteristic curve (AUC) as the evaluation metric, since both datasets are binary classification tasks. In practice, AUC is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance, and therefore, the higher the better.

Hyper-parameters. For the Fraud detection dataset, we use a multi-layer perception with 2 hidden layers whose dimensions are [8,8]. We choose Sigmoid as the activation function (Han and Moraga 1995) and use gradient descent as the optimizer. We set the learning rate to 0.001. For the Financial distress dataset, we use a multi-layer perception with 3 hidden layers with dimensions [400, 16, 8], we choose Relu as the activation function (H.R.Hahnloser et al. 2000) in the last layer and Sigmoid function in the other layers, use gradient descent as the optimizer, and set the learning rate to 0.006.

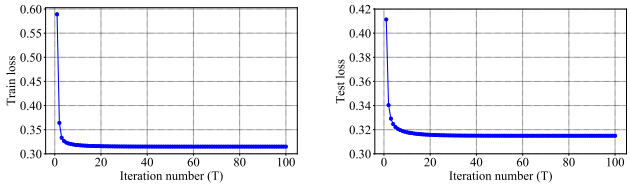
Comparison Results

To study the effectiveness of our proposed Privacy Preserving Neural Network (P^2N^2), we compare it with the traditional neural network (NN) and report the AUC performances on both datasets in Table 1, where we set the defender weight $\lambda = 0$. From it, we can see that P^2N^2 achieves almost the same prediction performance as NN, which is consistent with the exiting research (Mohassel and Zhang 2017). Besides, we show the average training loss and average test loss w.r.t the iteration on two datasets in Figure 3 and Figure 4, respectively, where we can see that P^2N^2 converges steadily without over-fitting. The results demonstrate the practicalness of our proposed model.

Efficiency Results

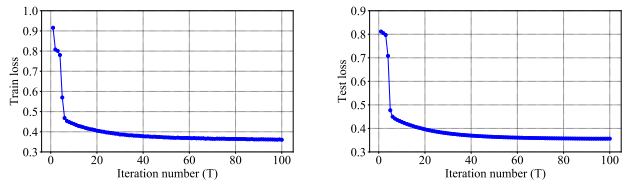
We now study the efficiency of our proposed P^2N^2, including the comparison of P^2N^2 and NN, the running time of P^2N^2 with different training data size and bandwidth.

Comparison of training time. First, to study the efficiency of (P^2N^2), we compare the training time of P^2N^2 and NN on both datasets. Note that P^2N^2 is implemented on three PCs



(a) *Training loss* (b) *Test loss*

Figure 3: Average loss of P^2N^2 on fraud detection dataset.



(a) *Training loss* (b) *Test loss*

Figure 4: Average loss of P^2N^2 on financial distress dataset.

which are used as the server and the data-holders in Local Area Network (LAN), and we currently ignore the communication delay between data holders and server. The results are summarized in Table 2, where we set batch size to 5000. From it, we find that P^2N^2 is 70× slower than NN on the fraud detection dataset and 240× slower on the financial distress dataset. This is because the first hidden layer is calculated by using secret sharing technique instead of plaintext computations, which takes extra communication time, and the first layer dimension on the fraud detection dataset is smaller than that on the financial distress dataset. The results indicate that if all the hidden layers are computed using secure multi-party computation techniques, similar as the existing privacy preserving neural networks, the running time will be much longer than our proposal (depend on the depth of the hidden layers).

Running time with different bandwidth. Second, we study the training time of P^2N^2 on the fraud detection dataset by varying network bandwidth. The result is shown in Figure 5. From it, we find that with the increase of network bandwidth, the training time of P^2N^2 first rapidly decreases and then tends to be stable. The result indicates that the efficiency of our proposed P^2N^2 heavily relies on the network status.

Running time with different data size. Furthermore, we study the running time of P^2N^2 with different data size, where fix the network bandwidth to 100 M/bps. We do this by varying the proportion of training data size using the fraud detection dataset, and report the running time of P^2N^2 in Figure 6. From it, we find that the running time of P^2N^2 scales linearly with the training data size. The results indicate that our proposed P^2N^2 can be scale to large dataset.

Effect of Defender

We finally study the effects of the defender on both model accuracy and its privacy preserving ability.

Influence on accuracy. We first vary the defender weight λ in $\{10^{-5}, 10^{-4}, ..., 10^{-1}\}$ and study its influence on

Table 2: Comparison of training time (in seconds) on both datasets

Training time	NN	P^2N^2
Fraud detection	21.52	1478.32
Financial distress	5.07	1196.67

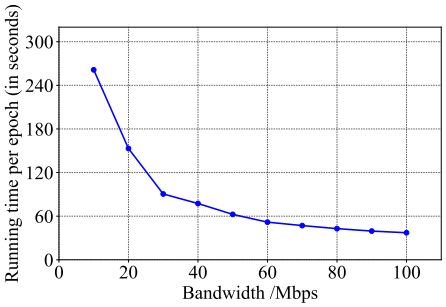


Figure 5: Running time of P^2N^2 with different bandwidth.

our model performance, where we use the fraud detection dataset. We report the results in Figure 8. Note that $\lambda = 0$ indicates the absence of such a defender. We observe that, with the increase of λ, the accuracy of P^2N^2 first slightly increases and then quickly decreases. This is because, the objective function in Eq. (5) has two parts, i.e., the cross-entropy loss that determines model accuracy and the defender loss which determines the privacy preserving ability, and they are balanced by λ. When λ is a small value (but bigger than 0), it works like a penalty term which prevents the model from overfitting to a certain extent. However, when λ is too big, P^2N^2 pays more attention to the defender loss rather than the cross-entropy loss. Therefore, the accuracy starts to decrease quickly.

Influence on privacy preserving ability. Second, to visually demonstrate the effectiveness of our proposed defender strategy, we conduct the following experiments on the MNIST dataset—a handwritten digits classification dataset (LeCun 1998), where each private input record is a handwritten digit from 0 to 9. During private input recovery experiments, we assume a serious private information leakage situation, i.e., the neutral server obtains some of the private input and the corresponding hidden layer of the training dataset. Based on these leaked information, the server can learn an attacker that maps the hidden layer to private input. After it, the attacker can easily recover the input of other records given their hidden layers.

We compare the recovery result with and without the defender. We use a three-layer fully-connected neural network, i.e., a multi-layer perception, as the defender. The network structure of the defender is (728, 512, 128, 10), where 728 is the dimension of each handwritten digit, 512 and 128 are hidden layer dimensions, and 10 is the output dimension (classification number). During experiments, we assume two data holders have evenly partial features of a digit. We choose Relu (LeCun, Bengio, and Hinton 2015) as the active function, mean squared error as the distant function $d(\cdot, \cdot)$ in Eq. (5), Adam (Kingma and Ba 2014) as the optimizer, and set learning rate to 0.01.

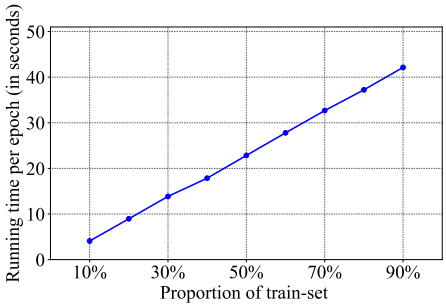


Figure 6: Running time of P^2N^2 with different datasize.

We report the input recovery result in Figure 7, where (a) is the randomly selected original handwritten digits, (b) and (c) are the corresponding recovered results with and without the defender, respectively. We set the defender weight $\lambda = 100$ on MNIST dataset, since the attack loss is small comparing with that on fraud detection dataset. From it, we can clearly see that, with the presence of the defender, it becomes more difficult to recognize the recovered digits. The results indicate that the defender mechanism can effectively preventing the server from recovering the private input of data holders.

Note that one can still make a good guess on the recovered digits from Figure 7 (C), even with the defender mechanism. This is because, in our experiments, we assumed that the neutral server has obtained some private input from data holders. This is quite serious data leakage situation and is very difficult to appear in practice, since these private input are hold by different data holders. Nevertheless, our experiments demonstrated the effectiveness of the defender mechanism.

Conclusion

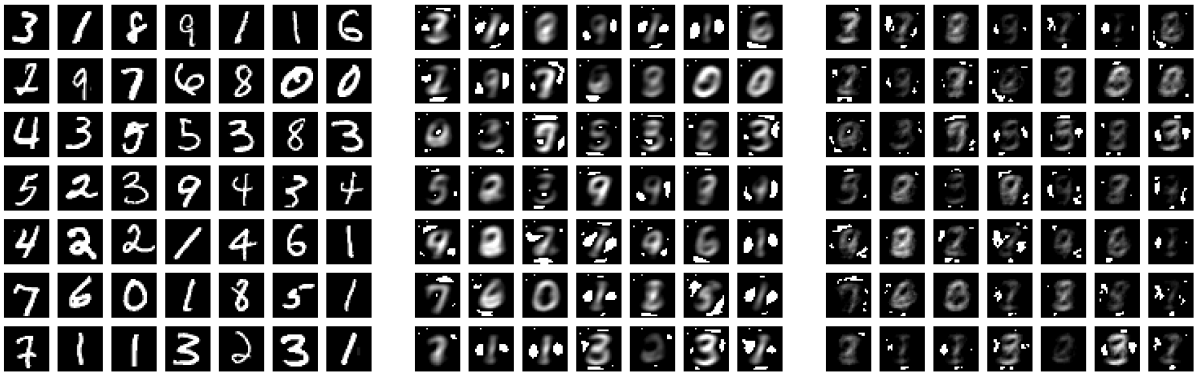
In this paper, we proposed a privacy preserving neural network learning paradigm that can scale to large datasets. Our motivation is to split the computation graph of DNN into two parts, i.e., the computations related to private data are performed by data holders using cryptographical techniques, and the rest of the computations are done by a neutral server with high computation ability. Our model achieved promising results on real-world fraud detection dataset and financial distress dataset. In the future, we would like to deploy our proposal in real-world applications in Ant Financial.

References

Chen, C.; Liu, Z.; Zhou, J.; Li, X.; Qi, Y.; Jiao, Y.; and Zhong, X. 2019. How much can a retailer sell? sales forecasting on tmall. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 204–216. Springer.

Dal Pozzolo, A.; Caelen, O.; Le Borgne, Y.-A.; Waterschoot, S.; and Bontempi, G. 2014. Learned lessons in credit card fraud detection from a practitioner perspective. *Expert systems with applications* 41(10):4915–4928.

Gilad-Bachrach, R.; Dowlin, N.; Laine, K.; Lauter, K.; Naehrig, M.; and Wernsing, J. 2016. Cryptonets: Apply-



(a) Original input (b) Without defender (c) With defender

Figure 7: Private input recovery results on MNIST dataset. (a) is the randomly selected original handwritten digits, (b) is the corresponding recovered result by the server without the presence of the defender, and (c) is the recovered result by the server but with the presence of the defender on data holders.

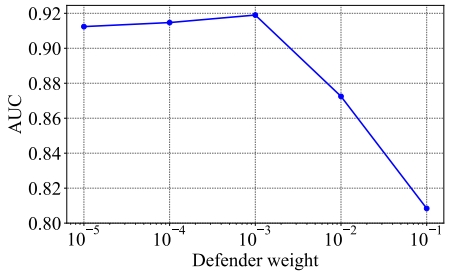


Figure 8: Effect of defender weight λ on P^2N^2.

ing neural networks to encrypted data with high throughput and accuracy. In *ICML*, 201–210.

Gu, Z.; Huang, H.; Zhang, J.; Su, D.; Lamba, A.; Pendarakis, D.; and Molloy, I. 2019. Securing input data of deep learning inference systems via partitioned enclave execution. *CoRR* abs/1807.00969.

Gupta, O., and Raskar, R. 2018. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications* 116:1–8.

Han, J., and Moraga, C. 1995. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *IWANN*, 195–201. CORE.

Hardy, S.; Henecka, W.; Ivey-Law, H.; Nock, R.; Patrini, G.; Smith, G.; and Thorne, B. 2017. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677*.

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

H.R.Hahnloser, R.; Sarpeshkar, R.; A. Mahowald, M.; J. Douglas, R.; and Seung, H. S. 2000. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature* 405:947–951.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *nature* 521(7553):436.

LeCun, Y. 1998. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

McKeen, F.; Alexandrovich, I.; Berenzon, A.; Rozas, C. V.; Shafi, H.; Shanbhogue, V.; and Savagaonkar, U. R. 2013. Innovative instructions and software model for isolated execution. *Hasp@ isca* 10(1).

Mohassel, P., and Zhang, Y. 2017. Secureml: A system for scalable privacy-preserving machine learning. In *S&P*, 19–38. IEEE.

Osia, S. A.; Shamsabadi, A. S.; Taheri, A.; Katevas, K.; Sajadmanesh, S.; Rabiee, H. R.; Lane, N. D.; and Haddadi, H. 2019. A hybrid deep learning architecture for privacy-preserving mobile analytics. *arXiv preprint arXiv:1703.02952*.

Shamir, A. 1979. How to share a secret. *Communications of the ACM* 22(11):612–613.

Vepakomma, P.; Gupta, O.; Swedish, T.; and Raskar, R. 2018. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564*.

Wagh, S.; Gupta, D.; and Chandran, N. 2019. Securenn: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies* 1:24.

Wei-Sen Chen, Y.-K. D. 2009. Using neural networks and data mining techniques for the financial distress prediction model. *Expert Systems with Applications* 36:4075–4086.

Yao, A. C. 1982. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science*, 160–164. IEEE.

Zhu, F.; Chen, C.; Wang, Y.; Liu, G.; and Zheng, X. 2019. Dtcdr: A framework for dual-target cross-domain recommendation. In *CIKM*, 1533–1542. ACM.