Privacy Preserving Back-Propagation Neural Network Learning Made Practical with Cloud Computing

Jiawei Yuan, *Student Member, IEEE,* Shucheng Yu, *Member, IEEE,*

Abstract—To improve the accuracy of learning result, in practice multiple parties may collaborate through conducting joint Back-propagation neural network learning on the union of their respective data sets. During this process no party wants to disclose her/his private data to others. Existing schemes supporting this kind of collaborative learning are either limited in the way of data partition or just consider two parties. There lacks a solution that allows two or more parties, each with an arbitrarily partitioned data set, to collaboratively conduct the learning. This paper solves this open problem by utilizing the power of cloud computing. In our proposed scheme, each party encrypts his/her private data locally and uploads the ciphertexts into the cloud. The cloud then executes most of the operations pertaining to the learning algorithms over ciphertexts without knowing the original private data. By securely offloading the expensive operations to the cloud, we keep the computation and communication costs on each party minimal and independent to the number of participants. To support flexible operations over ciphertexts, we adopt and tailor the BGN'doubly homomorphic' encryption algorithm for the multi-party setting. Numerical analysis and experiments on commodity cloud show that our scheme is secure, efficient and accurate.

Index Terms—Privacy reserving, Learning, Neural Network, Back-Propagation, Cloud computing, Computation Outsource.

1 Introduction

Back-propagation[18] is an effective method for learning neural networks and has been widely used in various applications. The accuracy of the learning result, despite other facts, is highly affected by the volume of high quality data used for learning. As compared to learning with only local data set, collaborative learning improves the learning accuracy by incorporating more data sets into the learning process[21], [11], [20]: the participating parties carry out learning not only on their own data sets, but also on others' data sets. With the recent remarkable growth of new computing infrastructures such as Cloud Computing, it has been more convenient than ever for users across the Internet, who may not even know each other, to conduct joint/collaborative learning through the shared infrastructure[13], [14].

Despite the potential benefits, one crucial issue pertaining to the Internet-wide collaborative neural network learning is the protection of data privacy for each participant. In particular, the participants from different trust domains may not want to disclose their private data sets, which may contain privacy or proprietary information, to anybody else. In applications such as healthcare, disclosure of sensitive data, e.g., protected health information (PHI)[2], is not only a privacy issue but of legal concerns according to the privacy rules such as Health Insurance Probability and Accountability Act(HIPAA)[1]. In order to embrace the Internet-wide collaborative learning, it is imperative to provide a solution that allows the participants, who lack mutual trust, to conduct neural network learning jointly without disclosing their respective private data sets. Preferably, the solution shall be efficient and scalable enough to support an arbitrary number of participants, each possessing arbitrarily partitioned data sets.

Challenges. Theoretically, secure multi-party computation (SMC)[22] can be used to solve problems of this kind. But the extremely high computation and communication complexity of SMC, due to the circuit size, usually makes it far from practical even in the two-party case. In order to provide practical solutions for privacy preserving back-propagation neural (BPN) network learning, three main challenges need to be met simultaneously: 1) To protect each participant's private dataset and intermediate results generated during the BPN network learning process, it requires secure computation of various operations, e.g. addition, scalar product and the nonlinear sigmoid function, which are needed by the BPN network algorithm; 2) To ensure the practicality of the proposed solution, the computation/communication cost introduced to each participant shall be affordable. In order to accommodate a large range of collaborative learning, the proposed solution shall consider system scalability. In particular, it shall be able to support an arbitrary number of participants without introducing tremendous computation/communication costs to each participant. 3) For collaborative training, the training data sets may be owned by different parties and par-

• *The preliminary version of this paper appeared in Securecomm 2012 [23]. J.Yuan and S.Yu are with Deptartment of Computer Science, University of Arkansas at Little Rock, Little Rock, AR, 72204. E-mail: {jxyuan,sxyu1}@ualr.edu*

titioned in arbitrary ways rather than a single way of partition.

Related Work. Several privacy preserving BPN network learning schemes have been proposed recently. Schlitter[19] introduces a privacy preserving BPN network learning scheme that enables two or more parties to jointly perform BPN network learning without disclosing their respective private data sets. But the solution is proposed only for horizontal partitioned data. Moreover, this scheme cannot protect the intermediate results, which may also contain sensitive data, during the learning process. Chen et. al.[6] proposes a privacy preserving BPN network learning algorithm for two-party scenarios. This scheme provides strong protection for data sets including intermediate results. However, it just supports vertically partitioned data. To overcome this limitation, Bansal et. al.[4] enhanced this scheme and proposed a solution for arbitrarily partitioned data. Nevertheless, this enhanced scheme, just like [6], was proposed for the two-party scenario. Directly extending them to the multi-party setting will introduce a computation/communication complexity quadratic in the number of participants. In practical implementation, such a complexity represents a tremendous cost on each party considering the already expensive operations on the underlying groups such as Elliptic Curves. However, [4] just considers the two-party scenario though it supports arbitrarily partitioned dataset. To our best knowledge, none of existing schemes have solved all these challenges at the same time. There still lacks an efficient and scalable solution that supports collaborative BPN network learning with privacy preservation in the multi-party setting and allows arbitrarily partitioned datasets.

Our Contribution. In this work, we address this open problem by incorporating the computing power of the cloud. The main idea of our scheme can be summarized as follows: each participant first encrypts her/his private data with the system public key and then uploads the ciphertexts to the cloud; cloud servers then execute most of the operations pertaining to the learning process over the ciphertexts and return the encrypted results to the participants; the participants jointly decrypt the results with which they update their respective weights for the BPN network. During this process, cloud servers learn no privacy data of a participant even if they collude with all the rest participants. Through off-loading the computation tasks to the resource-abundant cloud, our scheme makes the computation and communication complexity on each participant *independent* to the number of participants and is thus highly scalable. For privacy preservation, we decompose most of the sub-algorithms of BPN network into simple operations such as addition, multiplication, and scalar product. To support these operations over ciphertexts, we adopt the BGN(Boneh, Goh and Nissim) 'doubly homomorphic' encryption algorithm[5] and tailor it to split the decryption capability among multiple participants for collusion-resistance decryption. As decryption of [5] is limited to small numbers, we introduce a novel design in our scheme such that arbitrarily large numbers can be efficiently decrypted. To protect the intermediate data during the learning process, we introduce a novel random sharing algorithm to randomly split the data without decrypting the actual value. Thorough security analysis shows that our proposed scheme is secure. Experiments conducted on Amazon Elastic Compute Cloud(Amazon EC2)[15], over real datasets from UCI Machine Learning Repository[12], show that our scheme significantly outperform existing ones in computation/communication cost and accuracy loss.

Our contribution can be summarized as follows:

- To our best knowledge, this paper is the first that provides privacy preservation for multi-party (more than two parties) collaborative BPN network learning over arbitrarily partitioned data;
- Thorough analysis investigating privacy and efficiency guarantees of proposed scheme is presented; real experiments on Amazon Cloud further show our scheme's several magnitudes lower computation/communicational costs than the existing ones.
- We tailor [5] to support multi-party secure scalar product and introduce designs that allows decryption of arbitrary large messages. These improvements can be used as independent general solutions for other related applications.

The rest of this paper is organized as follows. Section 2 presents the models and assumptions. In section 3 we introduce technique preliminaries which is followed by detailed description of our proposed scheme in section 4. Section 5 evaluates our proposed scheme. We conclude our work in section 6.

2 MODELS AND ASSUMPTIONS

2.1 System Model

We consider a system composed of three major parties: a *trusted authority* (TA), the *participating parties* (data owner) and the *cloud servers* (or *cloud*). TA is the party only responsible for generating and issuing encryption/decryption keys for all the other parties. It will not participate in any computation other than key generation and issuing. Each participating party s, denoted as P_s, owns a private data set and wants to perform collaborative BPN network learning with all other participating parties. That is, they will collaboratively conduct learning over the arbitrarily partitioned data set, which is private and cannot be disclosed during the whole learning process. We assume that each participating party stays online with broadband access to the cloud and is equipped with one or several contemporary computers, which can work in parallel if there are more than one.

2.2 Security Model

Our scheme assumes the existence of a trusted authority who is trusted by all the parties, TA has the knowledge of system secret key and will not participate in any computation besides the key generation and issuing. TA is allowed to learn about each party's private data whenever necessary. We claim that the existence of TA is useful when investigation is needed in case some malicious party intentionally interrupts the system, say using bogus data sets. In real life, parties such as the government agents or organization alliances can be the TA. Although the existence of TA is helpful, we leave the completely distributed solution as a future work.

The participating parties do not fully trust each other. Therefore, they do not want to disclose their respective private data(except for the final weights learned by the network) to any other parties than TA. The cloud is not fully trusted by the participating parties either, i.e., the cloud is not allowed to learn about the sensitive information, such as original data sets and intermediate data. In this paper, we follows the curious-but-honest model[8]. That is, all the parties (i.e., all the participating parties and the cloud) will honestly follow our protocol but try to discover others' private data as much as possible. A number malicious of participating parties may collude among themselves and/or with the cloud.

3 TECHNIQUE PRELIMINARIES

3.1 Arbitrarily Partitioned Data

In this paper, we consider arbitrarily partitioned data [4] among multi-parties, say Z parties($Z > 2$). For arbitrarily partitioned data, each party $P_s, 1 \le s \le Z$, holds parts of the data set without any specific order. Specifically, consider a data set D with N rows $\{DB^1, DB^2, \cdots, DB^N\}$, and each row $DB^v, 1 \le v \le N$ has m attributes$\{x_1^v, x_2^v \cdots, x_m^v\}$. DB_s^v is the subset of data set owned by P_s, we have $DB^v = DB_1^v \bigcup DB_2^v \cdots \bigcup DB_Z^v$ and $DB_1^v \bigcap DB_2^v \cdots \bigcap DB_Z^v = \emptyset$. For each DB^v, P_s has t_s^v attributes(i.e. $|DB_s^v| = t_s^v$), where $\sum_{s=1}^{Z} t_s^v = m$, and each $t_s^v, 1 \le v \le N$ does not have to be equal. When $t_s^1 = t_s^2 = \cdots = t_s^N$ and the attributes owned by a party in each row are at the same position, the arbitrary partition becomes vertical partition. Similarly, it is horizontal partition if each P_s completely holds some DB^v.

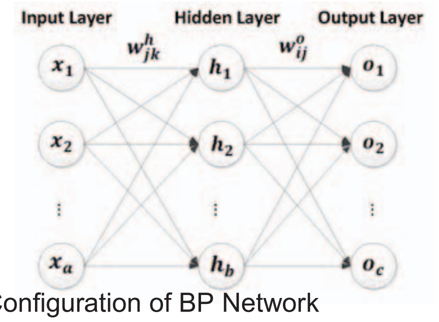

Fig. 1. Configuration of BP Network

3.2 Back-Propagation Neural Network Learning

Back-Propagation neural network learning algorithm is mainly composed of two stages: $feed\ forward$ and $error\ back-propagation$. Fig.1 shows is a configuration for a three layer(a-b-c) BP network. We use vector$\{x_1, x_2, \cdots, x_a\}$ to denote the values of input nodes, vector$\{h_1, h_2, \cdots, h_b\}$ and vector$\{o_1, o_2, \cdots, o_c\}$ to denote the values of hidden layer nodes and output layer nodes respectively. w_{jk}^h is used to represent the weight connecting the input layer node k and the hidden layer node j. w_{ij}^o denotes the weight connecting j and the output layer node i, where $1 \le k \le a, 1 \le j \le b, 1 \le i \le c$. During the BPN network learning process,

Algorithm 1: Back-Propagation Neural Network Learning Algorithm

```
Input: $N$ input sample vectors $\vec{V_i}, 1 \le i \le N$, with $a$ dimensions,
       $iteration_{max}$, learning rate $\eta$, target value $t_i$, sigmoid function
       $f(x) = \frac{1}{1+e^{-x}}$
Output: Network with final weights:
       $w_{jk}^h, w_{ij}^o, 1 \le k \le a, 1 \le j \le b, 1 \le i \le c$
begin
    Randomly Initialize all $w_{jk}^h, w_{ij}^o$.
    for $iteration = 1, 2 \cdots, iteration_{max}$ do
        for $sample = 1, 2 \cdots, N$ do
            //Feed Forward Stage:
            for $j = 1, 2 \cdots, b$ do
                $h_j = f(\sum_{k=1}^{a} x_k * w_{jk}^h)$
            for $i = 1, 2 \cdots, c$ do
                $o_i = f(\sum_{j=1}^{a} h_j * w_{ij}^o)$
            if $Error = \frac{1}{2}\sum_{i=1}^{c}(t_i - o_i)^2 > threshold$ then
                //Back-Propagation Stage:
                $\Delta w_{ij}^o = -(t_i - o_i) * h_j$
                $\Delta w_{jk}^h = -h_j(1-h_j)x_k \sum_{i=1}^{c}[(t_i - o_i) * w_{ij}^o)]$
                $w_{ij} = w_{ij} - \eta \Delta w_{ij}$
                $w_{jk}^h = w_{jk}^h - \eta \Delta w_{jk}^h$
            else
                //Learning Finish
                break
```

the goal is to model a given function by modifying internal weights of input to generate an expected output. As described in Algorithm 1, all the weights are initialized as small random numbers[10], [7], [16]. In the $Feed\ Forward\ Stage$, values at each layer are calculated using the weights, the sigmoid function, and the values at the previous layer. In the $Back-Propagation\ stage$, the algorithm checks whether the error between output values and target values is within the threshold. If not, all the weights will be modified according to Eq.1,2 and the learning procedure is repeated. The learning will not be terminated until the error is within the threshold or the max number of iterations is exceeded. After the learning, the final weights on each link are used to generate the learned network. Ref.[18] describes details of the BPN network learning algorithm.

$$\Delta w_{ij}^o = -(t_i - o_i)h_j \tag{1}$$

$$\Delta w_{jk}^h = -h_j(1-h_j)x_k \sum_{i=1}^{c}[(t_i - o_i) * w_{ij}^o)] \tag{2}$$

3.3 BGN Homomorphic Encryption

Homomorphic encryption enables operations on plaintexts to be performed on their respective ciphertexts without disclosing the plaintexts. Most existing homomorphic encryption schemes only support single operation - either addition or multiplication. In [5], Boneh et.

al. introduced a public-key 'doubly homomorphic' encryption scheme(called 'BGN' for short), which simultaneously supports one multiplication and unlimited number of addition operations. Therefore, given ciphertexts $C(m_1), C(m_2), \cdots, C(m_i)$ and $C(\hat{m}_1), C(\hat{m}_2), \cdots, C(\hat{m}_i)$, one can compute $C(m_1\hat{m}_1 + m_2\hat{m}_2 + \cdots + m_i\hat{m}_i)$ without knowing the plaintext, where $C()$ is the ciphertext of message m_i or $\hat{m}_i$, encrypted by the system's public key. We refer to Ref. [5] for details of the BGN scheme.

We shall point out that the BGN scheme is designed for two parties. Moreover, due to message decryption involves solving discrete logarithm of the ciphertext using Pollard's lambda method, BGN scheme just works with small numbers. While it is easy to make BGN decrypt long messages (in which the message is treated as a bit string) using a mode of operation, it is non-trivial to let BGN efficiently decrypt large numbers (wherein the final message is interpreted by its value and unknown to the encryptor after homomorphic operations).

4 OUR PROPOSED SCHEME

4.1 Problem Statement and Scheme Overview

Problem Statement: In this paper, we aim at enabling multiple parties to jointly conduct BPN network learning without revealing their private data. The input data sets owned by the parties can be arbitrarily partitioned. The computational and communicational costs on each party shall be practically efficient and the system shall be scalable.

Specifically, we consider a 3-layer(a-b-c configuration) neural network for simplicity but it can be easily extended to multi-layer neural networks. The learning data set for the neural network, which has N samples(denoted as vector$\{x_1^m, x_2^m, \ldots, x_a^m\}$, $1 \le m \le N$), is arbitrary partitioned into $Z(Z \ge 2)$ subsets. Each party P_s holds $x_{1s}^m, x_{2s}^m, \cdots, x_{as}^m$ and have:

$$x_{11}^m + x_{12}^m + \cdots + x_{1Z}^m = x_1^m \tag{3}$$

$$\cdots\cdots$$

$$x_{a1}^m + x_{a2}^m + \cdots + x_{aZ}^m = x_a^m \tag{4}$$

Each attribute in sample $\{x_1^m, x_2^m, \ldots, x_a^m\}, 1 \le m \le N$, is possessed by only one party - if P_s possesses $x_k^m, 1 \le k \le a$, then $x_{ks}^m = x_k^m$; otherwise $x_{ks}^m = 0$. In this paper, w_{jk}^h denotes the weight used to connect the input layer node k and the hidden layer node j; w_{ij}^o denotes the weight used to connect the hidden layer node j and the output layer node i, where $1 \le k \le a, 1 \le j \le b, 1 \le i \le c$ and a, b, c are the number of nodes of each layer in Figure.1. For collaborative learning, the main task for all the parties is to jointly execute the operations defined in the Feed Forward stage and the Back-Propagation stage as shown in Algorithm 1. During each learning stage, except for the final learned network, neither the input data of each party nor the intermediate results(i.e., weights, value of hidden layer node, value of output layer node) generated can be revealed to anybody other than TA.

Scheme Overview: To achieve the above goals, the main idea of our proposed scheme is to implement a privacy preserving equivalence for each step of the original BPN netowrk learning algorithm described in Algorithm 1. Different from the original BPN network learning algorithm, our proposed scheme lets each party encrypt her/his input data set and upload the encrypted data to the cloud, allowing the cloud servers to perform most of the operations, i.e., additions and scalar products. To support these operations over ciphertexts, we adopt and tailor the BGN 'doubly homomorphic' encryption[5] for data encryption. Nevertheless, as the BGN algorithm just supports one step multiplication over ciphertext, the intermediate results, e.g., the intermediate products or scalar products, shall be first securely decrypted and then encrypted to support consecutive multiplication operations as described in Algorithm 1. For privacy preservation, however, the decrypted results known to each party cannot be the actual intermediate values, e.g., the values of the hidden layer. For this purpose, we design a secret sharing algorithm that allows the parties to decrypt only the random shares of the intermediate values. The random shares allow the parties to collaboratively execute the following steps without knowing the actual intermediate values. Data privacy is thus well protected. The overall algorithm is described in Algorithm 2 which is the privacy preserving equivalence of Algorithm 1. To support the operations defined in Algorithm 2, we propose three other cloud based algorithms: Algorithm 3 for secure scalar product and addition, Algorithm 4 for secure random sharing, and Algorithm 5 for sigmoid function approximation. After the entire process of the privacy preserving learning, all the parties jointly establish a neural network representing the whole data set without disclosing any private data to each other.

4.2 Privacy Preserving Multi-Party Neural Network Learning

In this section, we introduce our cloud based privacy preserving multi-party BPN network learning algorithm over arbitrarily partitioned data. As we described in Algorithm 2, all the parties generate and assign random weights w_{jks}^h and w_{ijs}^o to each P_s and make agreement on the max number of learning iteration $iteration_{max}$, the learning rate η, error threshold and target value t_i of each output layer node at the beginning of learning. In the Feed Forward Stage, all the parties agree on the terms of approximation for the sigmoid function according to their accuracy requirement(details given in section 4.5) and obtain random shares h_{js} for value of hidden layer node and o_{is} for value of output layer node. After the Feed Forward Stage, all the parties work together to check whether the network has reached the error threshold. If not, they proceed to the Back-Propagation Stage, which aims at modifying the weights so as to achieve correct weights in the neural network. For the weights of each output layer node w_{ij}^o, each P_s obtains random shares of the changes in weights, denoted as

Algorithm 2: Privacy Preserving Multi-Party BPN network Learning Algorithm

```
begin
    Input: each $P_s$'s data set for $N$ data
        samples,$x_{1s}^v, x_{2s}^v, \cdots, x_{as}^v, 1 \le v \le N$, $w_{jks}^{hv}$ and $w_{ijs}^{ov}$ for $N$
        samples, $iteration_{max}$, $\eta$, target value $t_i$
    Output: Network with final weights:
        $w_{jk}^h, w_{ij}^o, 1 \le k \le a, 1 \le j \le b, 1 \le i \le c$
    for $iteration = 1, 2, \cdots, iteration_{max}$ do
        for $v = 1, 2, \cdots, N$ do
            //Feed Forward Stage: for $j = 1, 2, \cdots, b$ do
                Using Algorithm 3 and Algorithm 4, each $P_s$ obtain
                random shares $\varphi_{vs}$ for $\sum_{k=1}^{a}(x_{k1}^v + x_{k2}^v + \cdots +$
                $x_{kZ}^v) * (w_{jk1}^{hv} + w_{jk2}^{hv} + \cdots + w_{jkZ}^{hv})$
                Using Algorithm 5, all the parties compute the sigmoid
                function and obtain the random shares $h_{vjs}$,
                $\sum_{s=1}^{Z} h_{vjs} = h_{vj}$ and $h_{vj} = f(\sum_{s=1}^{Z} \varphi_{vs})$, where
                $f()$ is the approximation for the sigmoid function as
                described in section4.6.
            for $i = 1, 2, \cdots, c$ do
                Using Algorithm 3,Algorithm 4 and Algorithm 5, each $P_s$
                obtain random shares $o_{vis}$ for $f(\sum_{j=1}^{b}(h_{vj1} +$
                $h_{vj2} + \cdots + h_{vjZ}) * (w_{ij1}^{ov} + w_{ij2}^{ov} + \cdots + w_{ijZ}^{ov}))$
            Using Algorithm 3, all the parties and cloud calculate
            $Error = \frac{1}{2}\sum_{i=1}^{c}(t_i - o_i)^2$
            if $Error > threshold$ then
                //Back-Propagation Stage:
                for $i = 1, 2, \cdots, c$ do
                    //(step 1)
                    Using Algorithm 4 and Algorithm 3, each $P_s$
                    obtains random share $\Delta w_{ijs}^{ov}$ for
                    $\Delta w_{ij}^{ov} = (-(t_{vi} - \sum_{s=1}^{Z} o_{vis}) * (\sum_{s=1}^{Z} h_{vjs})$
                for $j = 1, 2, \cdots, b$ do
                    //(step 2)
                    Using Algorithm 4 and Algorithm 3, each $P_s$
                    obtains random share $\mu_s^v$ for
                    $\sum_{i=1}^{c}[(\sum_{s=1}^{Z} o_{vis} - t_{vi}) * (\sum_{s=1}^{Z} w_{ijs}^{ov})]$
                    //(step 3)
                    Using Algorithm 4 and Algorithm 3, each $P_s$
                    obtains random share $\kappa_s^v$ for
                    $\sum_{s=1}^{Z} x_{ks}^v * \sum_{s=1}^{Z} \mu_s^v$
                    //(step 4)
                    Using Algorithm 4 and Algorithm 3, each $P_s$
                    obtains random share $\vartheta_s^v$ for
                    $\sum_{s=1}^{Z} h_{vjs} * (1 - \sum_{s=1}^{Z} h_{vjs})$
                    //(step 5)
                    Using Algorithm 4 and Algorithm 3, each $P_s$
                    obtains random share $\Delta w_{jks}^{hv}$ for
                    $\Delta w_{jk}^{hv} = \sum_{s=1}^{Z} \kappa_s^v * \sum_{s=1}^{Z} \vartheta_s^v$
                Each $P_s$ updates $w_{ijs}^{ov} = w_{ijs}^{ov} - \eta * \Delta w_{ijs}^{ov}$ and
                $w_{jks}^{hv} = w_{jks}^{hv} - \eta * \Delta w_{jks}^{hv}$
            else
                Learning Finished;
```

Δw_{ijs}^o, for Δw_{ij}^o (cf. Eq.1) by using Algorithm 3 and Algorithm 4. In order to compute the changes in the weight w_{jk}^h of each hidden layer node (cf. Eq.2), our proposed scheme computes the following items one by one - $\sum_{i=1}^{c}[(t_i - o_i) * w_{ij}^o)]$, $x_k \sum_{i=1}^{c}[(t_i - o_i) * w_{ij}^o)]$, $-h_j(1-h_j)$ and Δw_{jk}^h, as shown in step 2 - 5 respectively in Algorithm 2. Then, each P_s obtains the random shares (μ_s,κ_s,ϑ_s,Δw_{jks}^h) of each item using Algorithm 3 and Algorithm 4. Finally, P_s updates its own weights with its shares and the learning rate η.

4.3 Secure Scalar Product and Addition with Cloud

In this subsection, we tailor Ref.[5] and propose an algorithm that allows multiple parties to perform secure scalar product and homomorphic addition operations on ciphertexts using cloud computing. Specifically, each party encrypts her/his data with the system public key and uploads the ciphertexts to the cloud. The cloud servers compute the sum of original messages based on their ciphertexts. If the original messages are vectors, the cloud computes the scalar product of the vectors. During this process, the cloud does not need to decrypt nor learn about the original messages. The final result of the sum or scalar product is returned to all the parties in ciphertext. Decrypting the results needs the participation of all the parties using Pollard's lambda method. Due to efficiency limitation of the Pollard's lambda method, the algorithm in [5] can only work well with relative small numbers. We overcome such a limitation and make it suitable for large numbers. Our algorithm is presented in Algorithm 3.

Algorithm 3: Secure Scalar Product and Addition

- **Key Generation:**
  TA generates two cyclic groups G and G_1 of order $n = q_1 q_2$, where q_1 and q_2 are large primes, and a bilinear map $e : G \times G \rightarrow G_1$. Then it picks two random generators $g, u \in G$ and computes $h = u^{q_2}$. TA splits q_1 as $q_1 = (q_{11} + q_{12} + \cdots + q_{1Z})\ mod\ n$, where q_{1s} is randomly chosen from Z_n for $1 \le s \le Z$. For $1 \le s \le Z$, TA sends q_{1s} to party P_s as her/his secret key. The public key is published as $PK = (n, G, G_1, e, g, h)$, and the system master key is $SK = q_1$ which is only known to TA.
- **Encryption:** Given a message M, encrypt it as: $C = g^M h^r \in G$, $r \overset{R}{\leftarrow} Z_n$.
- **Secure Scalar Product:** Given the ciphertexts of vector $(M_{11}, M_{12}, \cdots, M_{1v})$ and $(M_{21}, M_{22}, \cdots, M_{2v})$, the cloud computes their scalar product as:
  $$C(prod) = h_1^r * \prod_{i=1}^{v} e(C_{1i}, C_{2i}),$$
  where $h_1 = e(g, h)$,C_{1i} and C_{2i} are the ciphertexts of message M_{1i} and M_{2i} respectively.
- **Secure Addition:** Given the ciphertexts of message $M_1, M_2, \cdots, M_v$, the cloud computes their sum as:
  $$C(sum) = \prod_{i=1}^{v} C_i$$
- **Decryption:** Without loss of generality, we just demonstrate the decryption of $C(sum)$ as follows. The cloud broadcasts $C(sum)$ to each party. On receiving the ciphertext, each party P_s computes $C(sum)^{q_{1s}}$ and returns the result to the cloud.
  With the results from all the parties, the cloud computes:
  $$\prod_{j=1}^{Z} C(sum)^{q_{1s}} = C(sum)^{q_1}.$$
  Since $C(sum) = \prod_{i=1}^{v} C_i = \prod_{i=1}^{v} g^{M_i} h^{r_i}$, we have:
  $$C(sum)^{q_1} = (g^{\sum_{i=1}^{v} M_i} \prod_{i=1}^{v} h^{r_i})^{q_1} = (g^{q_1})^{\sum_{i=1}^{t} M_i}$$
  Note that $h^{q_1} = 1$. $\sum_{i=1}^{v} M_i$ can be efficiently solved using Pollard's lambda method[17] given g^{q_1}. The encrypted scalar product can be decrypted jointly in the similar way.

Decryption of Large Numbers: Message decryption in the BGN algorithm involves solving the discrete log using Pollard's lambda method[17]. On a single contemporary computer, for example, the Pollard's lambda method is able to decrypt numbers of up to 30-40 bits within a reasonable time slot (e.g., in minutes or hours). Decryption of larger numbers is usually believed less practical in terms of the time complexity. In practice, however, it is hard to guarantee that the final results (numbers) are always small enough for the Pollard's lambda method to efficiently decrypt. This is either because the numbers contained in the vectors are too large, or the vectors are too long (of high dimension). To overcome this limitation, we propose to let the data holders divide the numbers, if they are large, into several numbers, and the cloud then decrypt the smaller "chunks" with which the final result can be recovered. The decryption process can be parallelized for efficiency. Assuming that the cloud is able to efficiently decrypt the result if each input number is less than d bits, our solution for supporting large numbers can be described as follows. Without loss of generality, we just consider the scalar product operation over input numbers of $3d$ bits.

Let $V_A = (A_1, A_2, \cdots, A_k)$ and $V_B = (B_1, B_2, \cdots, B_k)$ be two vectors, where A_i and B_i are $3d$-bit numbers for $1 \leq i \leq k$. Each number can be represented as:

$$A_i = A_{i2} * 2^{2d} + A_{i1} * 2^d + A_{i0}$$
$$B_i = B_{i2} * 2^{2d} + B_{i1} * 2^d + B_{i0}$$

We can compute the product of $A_i * B_i$ as follows:

$$A_i * B_i = 2^{4d}(A_{i2} * B_{i2}) + 2^{3d}(A_{i2} * B_{i1} + A_{i1} * B_{i2}) + 2^{2d}(A_{i2} * B_{i0} + A_{i0} * B_{i2} + A_{i1} * B_{i1}) + 2^d(A_{i1} * B_{i0} + A_{i0} * B_{i1}) + A_{i0} * B_{i0}$$

Therefore, instead of directly calculating $\sum_{i=1}^{k}(A_i * B_i)$, the participants can first compute $\sum_{i=1}^{k}(A_{i0} * B_{i0}), \cdots, \sum_{i=1}^{k}(A_{i2} * B_{i2})$ separately and finally recover $\sum_{i=1}^{k}(A_i * B_i)$. For this purpose, the data holders need to split A_i and B_i and encrypt A_{i0}, A_{i1}, A_{i2} and B_{i0}, B_{i1}, B_{i2}, which are d-bit numbers. By doing this, the encryption cost on each data holder increases by x times, where x is the number of smaller numbers that each large number is broken into. In the above example $x = 3$. The cloud needs to perform x^2 more time operations on ciphertexts than for a single scalar product. x^2 more decryptions will be performed by participants and the cloud. If x is fixed, the expansion of computation/communication cost is of constant time. Usually, x will not be large. For example, if 30-bit numbers can be efficiently decrypted, our technique can efficiently decrypt 90-bit numbers with $x = 3$.

4.4 Secure Sharing of Scalar Product and Sum

For privacy protection, the actual intermediate results shall also be protected and cannot be known to each party or the cloud server. While collaboratively running the BPN network learning algorithm, the parties need to execute consecutive operations such as addition and multiplication. However, the 'BGN' algorithm [5] only supports one step multiplication over ciphertext. To support consecutive multiplications, the parties need to decrypt the intermediate results first. However, straightforward decryption of the intermediate values will reveal these values to the parties, which may have privacy implications and shall be avoided [6]. To protect these intermediate results(scalar products or sum), we introduce a secure sharing algorithm in Algorithm 4, which enables each participating party to get a random share of the intermediate result without knowing its actual value. As described in Algorithm 3, Z parties can efficiently perform secure scalar product and addition computation with the help of cloud. To securely share the result, say ϵ, each party P_s first generates a random number $L_s \leftarrow (0, u)$, where u is the upper bound of ϵ, and encrypts it as: $C(L_s) = g_1^{L_s} h_1^{r_s q_2}$,where $g_1 = e(g, g), h_1 = e(g, h), r_s \overset{R}{\leftarrow} Z_p$. Then all the parties upload the ciphertexts $C(L_s)$ to the cloud and the cloud securely calculates the ciphertext of $sumL = \sum_{s=1}^{Z} L_s$ as:

$$C(sumL) = \prod_{s=1}^{Z} C(L_s) = g_1^{L_1 + L_2 + \cdots + L_Z} h_1^{q_2 \hat{r_s}} \quad (5)$$

where $\hat{r_s} \overset{R}{\leftarrow} Z_p$. All the parties work together to decrypt the difference between ϵ and $sumL$ as $\hat{L} = |\epsilon - sumL|$ and send it to P_1. Note that $0 < \sum_{s=1}^{Z} L_s < Z * u, 0 < \epsilon < u$, we have $-u < \sum_{s=1}^{Z} L_s - \epsilon < Z * u$. As the cloud is able to efficiently decrypt numbers as large as u, it can decrypt $\sum_{s=1}^{Z} L_s - \epsilon$ efficiently as long as Z (i.e., the number of parties) is not very large. Finally, each P_s get its secure share ϵ_s of ϵ, where $\epsilon = \sum_{s=1}^{Z} \epsilon_s$. For party $P_s, (s > 2)$, $\epsilon_s = L_s$. For P_1, if $\epsilon < \sum_{s=1}^{Z} L_s$, $\epsilon_1 = L_1 - \hat{L}$, otherwise, $\epsilon_1 = L_1 + \hat{L}$.

As mentioned before, the sign of $\hat{L}$ cannot be decided before decryption. If $\epsilon - sumL$ is negative, the direct decryption of $\hat{L}$ using Pollard's lambda method will be extremely slow. This is because the negative value on Group will result in decryption of $n - \hat{L}$ which is a very large number given that the order n of group is large, and cannot be efficiently handled by Pollard's lambda method. To ensure efficient computation of $\hat{L}$, we consider the following two possible cases: $Case1$: $\epsilon > \sum_{s=1}^{Z} L_s$; and $Case2$: $\epsilon < \sum_{s=1}^{Z} L_s$). In multi-party scenarios($Z \geq 2$), as $L_s \overset{R}{\leftarrow} (0, u)$, there is a high possibility that $\epsilon < \sum_{s=1}^{Z} L_s$. Therefore, the cloud and all the parties can start decryption from $Case2$ as shown in Algorithm 4. If the successful decryption of $\hat{L}$ cannot be achieved (i.e., the decryption algorithm does not return) in an empirical time using the Pollard's lambda method, we can change to decrypt $\hat{L}$ using method of $Case1$.

Algorithm 4: Secure Share of Scalar Product and Sum

```
Input: Ciphertext of $\epsilon$
Output: Shares of $\epsilon$: $\epsilon_s$ for $P_s, 1 \leq s \leq Z$
begin
    for $s = 1, 2 \cdots, Z$ do
        Choose $L_s \overset{R}{\leftarrow} (0, u)$
        $C(L_s) = g_1^{L_s} h_1^{r_s q_2}$
        //where $u$ is the upper bound of $\epsilon$
    //Cloud Calculates:
    $C(sumL) = \prod_{s=1}^{Z} C(L_s)$
    case $1. \epsilon > \sum_{s=1}^{Z} L_s$
        $C(\hat{L}) = C(\epsilon) * C(sumL)^{-1}$
    case $2. \epsilon < \sum_{s=1}^{Z} L_s$
        $C(\hat{L}) = C(sumL) * C(\epsilon)^{-1}$
    Decrypt $C(\hat{L})$ with Algorithm 3 and send $\hat{L}$ to $P_1$
    //Output Shares:
    $\epsilon_1 = L_1 + \hat{L}$ (Case 1) or $\epsilon_1 = L_1 - \hat{L}$ (Case 2)
    for $i = 2, 3 \cdots, Z$ do
        $\epsilon_s = L_s$
    end
```

4.5 Approximation of Activation Function

In this subsection, we introduce the approximation of activation function using Maclaurin series expansion[3] and its secure sharing based on Algorithm 3 and Algorithm 4. Since the 'BGN' encryption does not support exponentiation operations over ciphertext(i.e. calculating $C(e^x)$ given $C(x)$) and cannot directly support the secure computation of sigmoid function $\frac{1}{1+e^{-x}}$. We utilize Maclaurin series expansion to approximate the sigmoid function and make it suitable for our proposed Algorithm 3 and Algorithm 4. For the sigmoid function $\frac{1}{1+e^{-x}} \in (0,1)$, we can guarantee the converge of its Maclaurin series and approximate it as:

$$\frac{1}{1+e^{-x}} = \frac{1}{2} + \frac{x}{4} - \frac{x^3}{48} + \frac{x^5}{480} + O(x^6) \tag{6}$$

Due to the property of Maclaurin series, the terms in the expansion can be decided depends on the accuracy requirement. As shown in the approximation of sigmoid function in Eq.6, the major challenge of secure computation for the equation becomes how to compute $x^k, 2 \leq k \leq n$ and share it without disclosing any privacy data. Using Algorithm 3 and 4, Z parties are able to securely calculate and share x. Based on these properties, we propose Algorithm 5 to securely share x^k. Taking x^2 for instance, Z parties first work together to get the secure shares of x using Algorithm 3 and 4, denoted as x_s for party P_s. With the ciphertext of x and x_s, P_s then calculates $\hat{C}_s(x) = C(xx_s) = C(x)^{x_s}$ and uploads it to the cloud. Cloud computes $C(x^2)$ using secure addition in Algorithm 3 and finally all the parties securely get the shares of x^2 with Algorithm 4. The scenarios of $x^k, k > 2$ can be easily extended as x^2. The correctness of this algorithm is proved in Appendix 1.2.

Algorithm 5: Secure Share of Activation Function

```
Input: $x_s$,ciphertext of $x$: $C(x)$
Output: Shares of $x^k$, $x_s^k$ for $P_s, 1 \leq s \leq Z$
begin
    for $j = 2, 3 \cdots, k$ do
        //each $P_s$ calculate:
        $\hat{C}_s(x^{j-1}) = C(x^{j-1})^{x_s}$
        Cloud Calculate $C(x^j)$ using Algorithm 3.
        $C(x^j) = \prod_{s=1}^{Z} \hat{C}_s(x^{j-1})$
        Call Algorithm 4, generate secure shares of $x_s^j$
```

4.6 Security Analysis

Lemma 4.1. *Algorithm 3 is semantically secure assuming the group G satisfies the subgroup decision assumption.*

Lemma 4.2. *Algorithm4, Algorithm5 and Algorithm2 are secure if Algorithm 3 is semantically secure.*

Due to the space limitation, the detail security analysis is provided in Appendix 1.1.

5 Performance Evaluation

To measure the impact of our improvements from existing schemes and evaluate the practicality of our privacy-preserving BPN network learning scheme, we numerically analyze it and fully implemented it on Amazon EC2 cloud.

5.1 Numerically analysis

In this section, we numerically evaluate the performance of our proposed scheme in terms of computation cost and communication cost and compare it with the existing techniques. For expression simplicity, in the following part of this section, we denote time cost of one multiplication operation on Group G as MUL[1] and that of one exponentiations operation on Group G as EXP.

For the neural network configuration (a-b-c) (cf. section 3.2), first each party P_s needs to encrypt all its privacy data once using Algorithm 3 with $2(n_s + b + c)$ EXP and (n_s+b+c) MUL, where n_s is the number of data attributes held by P_s, a,b and c represent the number of input layer nodes, hidden layer nodes and output layer nodes respectively. Note that this is the one-time cost performed before learning process starts. In the Feed Forward Stage, by using Algorithm 4 and Algorithm 5, each party performs $11(b + c)$ EXP and $3(b + c)$ MUL to get the random shares of every hidden layer node value and output layer node value. In the Back-Propagation Stage, to get the random share of changes for each output layer nodes, each P_s performs $5c$ EXP+$2c$ MUL and $5b$ EXP+$2b$ MUL respectively for $step1, 3$; $step2, 4$ both need $3b$ EXP and b MUL and $step5$ needs $7b$ EXP and $3b$ MUL. Thus P_s needs to perform $(18b+5c)$ EXP+$(4b+2c)$ MUL using Algorithm 3 and Algorithm 4. Our scheme is composed of four sub-algorithms. We give the detail analysis of each sub-algorithm in Appendix 1.3 due to space limitation.

Combining the cost for the two stages, the computation cost for each party P_s for one round privacy preserving BPN network learning in multi-party scenarios is $31b + 18c + 2n_s$ EXP and $8b + 6c + n_s$ MUL. For cloud side, it needs to perform $4 + a + b + c$ pairing operations on Group G, $Z * (8b + 14c)$ MUL and 11 decryption, where the complexity of each decryption is $O(\sqrt{K})$ and K is the size of message for decryption. Although the computation cost on cloud side will linearly increase with the party number, cloud can handle it in parallel efficiently. For communication cost, each party P_s needs to exchange $n_s * a + 24b + 5c$ messages with $(n_s * a * |G| + (24b + 5c) * |G_1|)$ bits during the one round privacy preserving BPN network learning process. By securely outsource most computation tasks to the cloud server, our scheme makes the cost of each party independent to the number of participating parties. To compare our scheme with existing ones[4], [6], we summarize the cost of our scheme and Ref.[4], [6] in Table.1. Considering the same neural network configuration(a-b-c), when extending scheme in Ref.[4] to Z parties scenarios, which utilized ElGamal[9] for secure computation, $Z^2 * (4n_s(a + 4b + c + bc))*$(EXP+MUL)$+Z^2 *$ (12bEXP+8bMUL) are needed for each party P_s for computation. When compared to scheme in Ref.[6], which can support two party privacy preserving BPN network

1. When the operation is on the elliptic curve, EXP means scalar multiplication operation and MUL means one point addition operation

	Our Scheme	Bansal's scheme	Chen's Scheme
Comp.	$(31b + 18c + 2n_s)$EXP $+(8b + 6c + n_s)$MUL	$Z^2 * (4n_s(a + 4b + c + bc))$ $*$(EXP+MUL) $+Z^2 * (12b$EXP$+8b$MUL$)$	$Z^2 * (5ab + 2bc$ $+abc)$(EXP+MUL) $+Z^2 * (4n_s(2bc + 4ab$ $+b))$(EXP+MUL)
Comm.(bit)	$n_s * a * \|G\| + (24b + 5c) * \|G_1\|$	$Z^2 * (ab + 3bc + 4b + 2)\|G\|$ $+2n_s\|G\|$	$Z^2 * (b + 2bc + 4ab + 2)\|G\|$ $+2n_s\|G\|$

TABLE 1
Computation/Communication Cost of each data owner in Privacy-Preserving BPN network Learning Schemes. n_s: number of data attributes owned by party P_s; Z: number of participating party; G and G_1: size of messages

learning over vertical partitioned data, Z parties scenario will introduce $Z^2 * (5ab + 2bc + abc + 4n_s(2bc + 4ab + b))$(EXP+MUL) computation cost to the each party. For communication cost, schemes in Ref.[4], [6] will cause $Z^2 * (ab + 3bc + 4b + 2) * |G_1| + 2n_s * |G_1|$ bits and $Z^2 * (b+2bc+4ab+2) * |G_1| + 2n_s * |G_1|$ bits respectively for Z parties scenarios. Different form our scheme, both [4] and [6] will introduce computation/communication cost quadratic in Z where Z is the number of participating parties. As a result, by offloading most computation cost to the cloud, our proposed scheme significantly outperforms the existing works in multi-party scenarios without any limitation on the type of data partition.

5.2 Experimental Evaluation

5.2.1 Experiment Setup

To evaluate the practical performance of our cloud based privacy-preserving BPN network learning scheme, we fully implemented it using C language under Linux environment. The experiments were executed on Amazon EC2 cloud, including 10 nodes with 8-core 2.93G Hz Intel Xeon CPU and 8GB memory. The testing datasets(Iris, kr-vs-kp, diabetes) are from UCI Machine Learning Repository[12]. We randomly distribute the datasets to each participate(i.e. arbitrarily partitioned). The learning parameters and architecture used for our neural network model are shown in Table 2. The number of hidden nodes are chosen based on the number of input and output nodes, which follows the same criteria utilized in [6]. Feature values in each dataset are normalized between $[0, 1]$. The number of participants in our experiments varies from 2 to 8.

Dataset	Sample	Architecture	Class	Epochs	Learning Rate
Iris	150	4-5-3	3	80	0.1
Diabetes	768	8-12-1	2	40	0.1
kr-vs-kp	3196	36-15-1	2	20	0.1

TABLE 2
Experiment Datasets and Parameters

5.2.2 Experimental Result

In this section, we provide the experimental results of our scheme in terms of computational and communicational costs. At the same time, we compare it with the existing privacy-preserving BPN network scheme [6].

Dataset Preparation: Before outsourcing his/her private data to cloud for BPN network learning, every party P_s needs to encrypt the data with Algorithm 3. Fig. 2 shows that the data encryption time for each party is greatly affected by the number of samples and features in the dataset, which varies from $4.12s$ to $120.21s$ for different datasets. However, this is a one-time cost and can be pre-computed off-line.

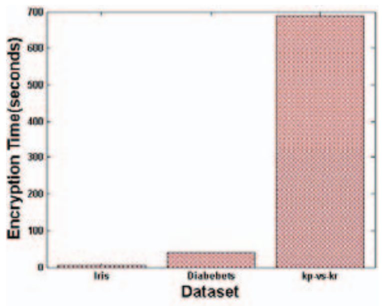

Fig. 2. Encryption Cost for Different Dataset

Collaborative Learning: For multi-party privacy-preserving BPN network learning with fixed neural network architecture, there are two factors - the number of party and the size of dataset - that mainly affect the system performance according to the experimental results of existing schemes [6]. As the number of participants increases, the operations for each party to share intermediate results and decrypt the final learning result remain the same in our proposed scheme.

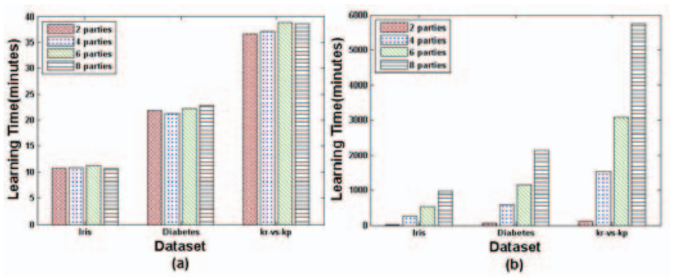

Fig. 3. Learning time for different party number and dataset: (a)Learning time of Our Scheme; (b)Learning time of Chen's Scheme

As shown in Fig.3(a), when the number of parties varies form 2 to 8, the overall learning time stays stable in our scheme, which are about 10.68 minutes, 21.86 minutes and 36.69 minutes for Iris, Diabetes and kr-vs-kp respectively. The learning time does not change with the number of parties because the cloud performs most learning operations in parallel without learning the private data. However, as demonstrated in Fig3(b), Ref.[6], when extended to the multi-party setting, introduces a quadratically increasing cost in the number of parties for each P_s, which can be up to 961.52 minutes, 2152.12 minutes and 5750.08 minutes for Iris, Diabetes and kr-vs-kp respectively and makes the learning scheme less practical. Notably, even in the two-party setting, for which [6] was designed, our scheme can still save at least 37% learning time as shown in Fig.4.

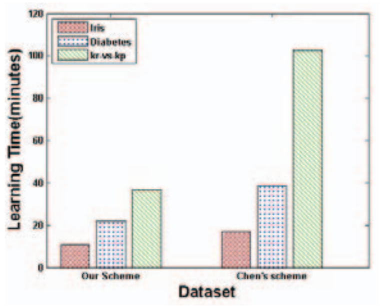

Fig. 4. two-Party scenario learning efficiency Comparison

To measure the influence of dataset size, we use 1/4, 1/2, 3/4 and all the samples in the dataset respectively for learning. As demonstrated in Fig.5(a), the increase of dataset size has slightly influence on the performance of our scheme. The growth of dataset size will not cause

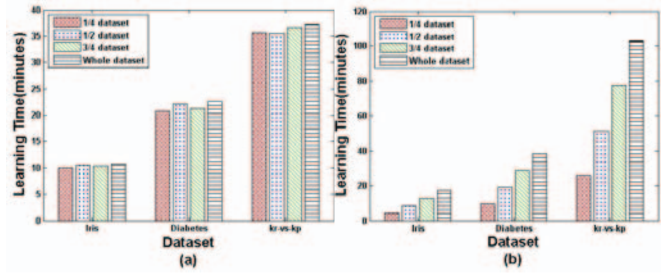

Fig. 5. Learning time for different dataset size: (a)Learning time of Our Scheme; (b)Learning time of Chen's Scheme

extra computation to the participants. This is because most operations needed by the learning algorithm have been offloaded to the cloud which can conduct parallel computation. The overall processing time can be kept stable by employing more cloud servers as the dataset size increases. Nevertheless, in [6] the computational cost on each party increases linearly to the dataset size as shown in Fig.5(b). This is because in [6] each party has to perform all the operations pertaining to the learning process by himself/herself. Similarly, based on our previous numerical analysis, [4] has the comparable computation complexity as [6], which increases quadratically with the number parties and changes linearly to the size of dataset. Therefore, [4] cannot support privacy-preserving BPN network learning efficiently in the setting of multi-party (i.e., more than two parties) or in case of large datasets as our proposed scheme does.

Communication Cost: In our scheme, the only party that each participating party P_s communicates with is the cloud server during the collaborative learning process. For dataset preparation, each party P_s needs to upload his/her encrypted dataset to the cloud. As shown in Table 3, in our scheme the bandwidths introduced to each party are 7.72KB, 79.09KB and 1481.27KB for Iris, Diabetes and kr-vs-kp respectively. Although the cost grows linearly to the size of dataset, it is a one-time cost and will not influence the real-time performance of the learning process. For collaborative learning, each party P_s only exchanges data with cloud for decryption or securely sharing operations. As shown in Table 3, the bandwidth consumption for each party P_s during the learning process varies from 3.44KB to 7.84KB for different datasets. As the number of parties increases,

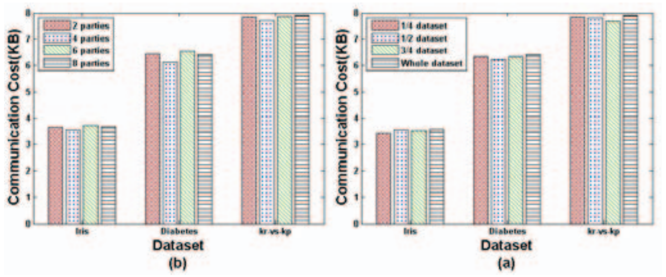

Fig. 6. Communication Cost of Our Scheme: (a)Communication cost for increasing party number; (b)Communication cost for increasing dataset size

the communication cost remains stable for each party during the learning process as shown in Fig.6(a). This is because the volume of data for each party to exchange stays constant as the total number of parties changes. Furthermore, Fig.6(b) demonstrates that the growth of dataset size has slight influence on each party's communication cost in our scheme during the collaborative learning process. As our previous complexity analysis indicates, the communication complexity of both [6] and [4] grows quadratically to the number of parties and linearly to the dataset size, which limits their practical usage in scenarios of multiple parties or large datasets ([6] or [4] did not provide their communication costs in their experiments.).

From our experimental results, we clearly note that: 1) our scheme can efficiently deal with privacy-preserving BPN network learning for multi-party scenarios thanks to the high scalability of cloud; 2) Our scheme can efficiently handle the relative large dataset for learning, which is compared to the linearly increasing computation/communication cost of existing works. 3) Our scheme can be implemented with easy communication protocol design since each party only needs to communicate with the cloud server.

	Our Scheme	
	Data Preparation(KB)	Collaborative Learning(KB)
Iris	7.72	3.44
Diabetes	79.09	6.33
kr-vs-kp	1481.27	7.84

TABLE 3
Communication Cost for Our Scheme

	Our Scheme 5-terms	9terms	Chen's Scheme	Bansal's Scheme	Non Privacy Preserving Scheme
Iris	18.32%	16.21%	19.34%	20%	14.17%
Diabetes	36.98%	36.01%	38.43%	NA	34.71%
kr-vs-kp	14.67%	13.81%	15.50%	NA	12.50%

TABLE 4
Test Error Rate Comparison

5.3 Accuracy Analysis

In this section, we analyze the accuracy loss in our privacy-preserving BPN network learning scheme and compare it with existing schemes [6], [4]. Recall that in our proposed scheme (cf. Section.4) the only place that introduces accuracy loss is the approximation of the activation function. We utilize the Maclaurin series expansion to approximate the function, whose accuracy can be adjusted by modifying the number of series terms according to the system requirement. Similar method of approximation with Maclaurin series expansion is also used in [24] though it just supports the two-party setting. As shown in Table.4, our scheme can achieve a similar accuracy performance as the existing schemes [6], [4] when we set the number of Maclaurin series terms as 5. By extending the number of series terms to 9, we can reduce the accuracy loss by about 1% which outperforms the existing schemes [6], [4]. This is because [6], [4] introduce accuracy losses not only in the approximation of the activation function, but also during the mapping of real numbers in sigmod function to fixed-point representations in every step of Feed Forward Stage and Back-Propagation Stage. Differently, our proposed scheme omits this limitation and thus can be efficiency performed on the sigmod function without any accuracy loss during the secure computation process. Compared to the non-privacy-preserving BPN network learning algorithm, our scheme introduces about only 1.3%-2% more error rate in 9 series terms setting, as shown in Table.4, which are acceptable in practical use and can be further improved by adding more series terms.

6 Conclusion

In this work, we proposed the first secure and practical multi-party BPN network learning scheme over arbitrarily partitioned data. In our proposed approach, the parties encrypt their arbitrarily partitioned data and upload the ciphertexts to the cloud. The cloud can execute most operations pertaining to the BPN network learning algorithm without knowing any private information. The cost of each party in our scheme is independent to the number of parties. This work tailors the BGN homomorphic encryption algorithm to support the multi-party scenario, which can be used as an independent solution for other related applications. Complexity and security analysis shows that our proposed scheme is scalable, efficient and secure. One interesting future work is to enable multiparty collaborative learning without the help of TA.

References

[1] The health insurance portability and accountability act of privacy and security rules. url: http://www.hhs.gov/ocr/privacy.

[2] National standards to protect the privacy of personal health information. url: http://www.hhs.gov/ocr/hipaa/finalreg.html.

[3] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables*. Dover books on mathematics. Dover, New York, 1964.

[4] A. Bansal, T. Chen, and S. Zhong. Privacy preserving back-propagation neural network learning over arbitrarily partitioned data. *Neural Comput. Appl.*, 20(1):143–150, Feb. 2011.

[5] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Proceedings of the Second international conference on Theory of Cryptography*, TCC'05, pages 325–341, Berlin, Heidelberg, 2005.

[6] T. Chen and S. Zhong. Privacy-preserving backpropagation neural network learning. *Trans. Neur. Netw.*, 20(10):1554–1564, Oct. 2009.

[7] L. Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems*, pages 396–404. Morgan Kaufmann, 1990.

[8] S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Over-encryption: management of access control evolution on outsourced data. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 123–134. VLDB Endowment, 2007.

[9] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA, 1985.

[10] S. E. Fahlman. *Faster-learning variations on Back-propagation: An empirical study*, pages 38–51. Morgan Kaufmann, 1988.

[11] K. Flouri, B. Beferull-lozano, and P. Tsakalides. Training a svm-based classifier in distributed sensor networks. In *Proceedings of 14nd European Signal Processing Conference*, pages 1–5, 2006.

[12] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[13] R. Grossman and Y. Gu. Data mining using high performance data clouds: experimental studies using sector and sphere. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 920–927, New York, NY, USA, 2008.

[14] R. L. Grossman. The case for cloud computing. *IT Professional*, 11(2):23–27, Mar. 2009.

[15] A. Inc. *Amazon Elastic Compute Cloud (Amazon EC2)*. Amazon Inc., http://aws.amazon.com/ec2/#pricing, 2008.

[16] R. Law. Back-propagation learning in improving the accuracy of neural network-based tourism demand forecasting. *Tourism Management*, 21(4):331–340, 2000.

[17] A. J. Menezes, P. C. V. Oorschot, S. A. Vanstone, and R. L. Rivest. Handbook of applied cryptography, 1997.

[18] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: explorations in the microstructure of cognition, vol. 1. chapter Learning internal representations by error propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[19] N. Schlitter. A protocol for privacy preserving neural network learning on horizontal partitioned data. In *Proceedings of the Privacy Statistics in Databases (PSD)*, Sep. 2008.

[20] S. Stolfo, A. L. P. S. Tselepis, A. L. Prodromidis, S. Tselepis, W. Lee, D. W. Fan, and P. K. Chan. Jam: Java agents for meta-learning over distributed databases. In *In Proc. 3rd Intl. Conf. Knowledge Discovery and Data Mining*, pages 74–81. AAAI Press, 1997.

[21] B. Yang, Y.-d. Wang, and X.-h. Su. Research and design of distributed neural networks with chip training algorithm. In *Proceedings of the First international conference on Advances in Natural Computation - Volume Part I*, ICNC'05, pages 213–216, Berlin, Heidelberg, 2005.

[22] A. C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164, Washington, DC, USA, 1982.

[23] J. Yuan and S. Yu. Privacy preserving back-propagation learning made practical with cloud computing. Securecomm'12, Pauda, Italy, Sep. 2012.

[24] S. Zang and S. Zhong. A privacy-preserving algorithm for distributed training of neural network ensembles. *To appear in Neural Computing and Applications*.

Jiawei Yuan (S'12) is a Ph.D. student at University of Arkansas at Little Rock. He received his B.S from University of Electronic Science and Technology of China in 2011. He worked as an internship at Research Programs of Information Technology at University of Arkansas for Medical Sciences since Aug 2011. His research interests are in the areas of cloud computing and network security, with current focus on securing the data and computation outsourced into the cloud. He is a student member of IEEE.

Shucheng Yu (S'07-M'10) received his Ph.D in Electrical and Computer Engineering from Worcester Polytechnic Institute, a MS in Computer Science from Tsinghua University and a BS in Computer Science from Nanjing University of Post & Telecommunication in China. He joined the Computer Science department at the University of Arkansas at Little Rock as an assistant professor in 2010. His research interests are in the general areas of Network Security and Applied Cryptography. His current research interests include Secure Data Services in Cloud Computing, Attribute Based Cryptography, and Security and Privacy Protection in Cyber Physical Systems. He is a member of IEEE.