Privacy Preserving Back-Propagation Learning Made Practical with Cloud Computing

Jiawei Yuan and Shucheng Yu

University of Arkansas at Little Rock, USA
{jxyuan, sxyu1}@ualr.edu

Abstract. Back-propagation is an effective method for neural network learning. To improve the accuracy of the learning result, in practice multiple parties may want to collaborate by jointly executing the back-propagation algorithm on the union of their respective data sets. During this process no party wants to disclose her/his private data to others for privacy concerns. Existing schemes supporting this kind of collaborative learning just partially solve the problem by limiting the way of data partition or considering only two parties. There still lacks a solution for more general and practical settings wherein two or more parties, each with an arbitrarily partitioned data set, collaboratively conduct learning.

In this paper, by utilizing the power of cloud computing, we solve this open problem with our proposed privacy preserving back-propagation algorithm, which is tailored for the setting of multiparty and arbitrarily partitioned data. 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 with ciphertexts but learns nothing about the original private data. By securely offloading the expensive operations to the cloud, we keep the local 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 multiparty setting. Thorough analysis shows that our proposed scheme is secure, efficient and scalable.

Keywords: privacy reserving, learning, neural network, back-propagation, cloud computing, computation outsource.

1 Introduction

Back-propagation[17] 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[19,11]: 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[12,13].

Despite the potential benefits, one crucial issue pertaining to the Internet-wide collaborative 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 learning on neural networks 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.

Related Work. Theoretically, secure multiparty computation (SMC)[20] 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 network (BPN) learning, several schemes have been proposed recently. Schlitter[18] introduces a privacy preserving BPN learning scheme that enables two or more parties to jointly perform BPN 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 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 n, 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. To our best knowledge, there is no efficient and scalable solution that supports collaborative BPN learning with privacy preservation in the multiparty setting over arbitrarily partitioned data.

Our Scheme. 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. 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 into simple operations such as addition, multiplication, and scalar product. To support these operations over ciphertexts, we adopt the BGN '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. Performance evaluation shows that our scheme is efficient and highly scalable.

Contribution. Our contribution can be summarized as follows:

- To our best knowledge, this paper is the first that provides privacy preservation for multiparty (more than two parties) collaborative back-propagation network learning over arbitrarily partitioned data;
- Thorough analysis shows that our proposed scheme is secure and efficient;
- We tailor [5] to support multiparty secure scalar product and introduc 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* 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 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.

2.3 Design Goals

– The multiple (two or more) participating parties can jointly perform a BPN learning over arbitrarily partitioned data. Specifically, the parties shall be able to jointly execute all the learning steps as defined by the BPN algorithm [17], which mainly includes a feed forward stage and a back-propagation stage.
– Confidentiality of private data shall be protected during the joint learning process. Specifically, we want to protect confidentiality of each party's private data set as well as all the intermediate results during the learning process, which means each party learns nothing but the final learned neural network.
– The system shall be efficient and scalable. In particular, the cost introduced on each party shall not grow with the number of participating parties. The computation tasks can be securely offloaded to the cloud without compromising data privacy. But the processing time on the cloud shall be less than or comparable to that on each participant. The overall execution time of the learning algorithm shall be practically acceptable.

3 Technique Preliminaries

3.1 Arbitrarily Partitioned Data

In this paper, we consider arbitrarily partitioned data as Bansal et al. did in[4] among multi-parties, say Z parties($Z > 2$). For arbitrarily partitioned data, each party $P_s, 1 \leq s \leq Z$, 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 $DB^v, 1 \leq v \leq N$ 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 $t_s^v, 1 \leq v \leq N$ 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 partitioning becomes vertical partitioning. Similarly, it is horizontal partitioning if each P_s completely holds some DB^v.

3.2 Back-Propagation Network Learning

Back-Propagation Network[17] is one of the most widely used model in neural network learning. The multi-layer BPN can approximate any nonlinear function.

BPN learning algorithm is mainly composed of two stages: $feed\ forward$ and $error\ signal\ back-propagation$. As shown in Figure.1, there is a configuration for a three layer(a-b-c) BP network: vector$\{x_1, x_2, \cdots, x_a\}$ contains the values of input nodes, vector$\{h_1, h_2, \cdots, h_b\}$ represents values of hidden nodes and the values of output nodes are $\{o_1, o_2, \cdots, o_c\}$. w_{jk}^h denotes 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 $1 \leq k \leq a, 1 \leq j \leq b, 1 \leq i \leq c$.

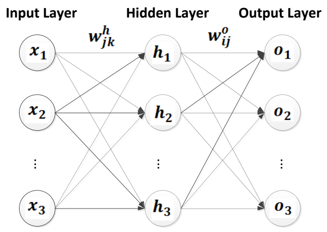

Fig. 1. Configuration of BP Network

During the BPN learning process, the goal is to model a given function by modifying internal weights of input signals to generate an expected output signal. As described in Algorithm 1, all the weights are initialized as small random numbers[10,7,14]. In the $FeedForwardStage$, values at each layer are calculated using the weights, the sigmoid function, and the values at the previous layer. In the $signalBack-Propagationstage$, 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 the used to generate the learned network. Ref.[17] describes details of the BPN algorithm.

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

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

Algorithm 1. Back-Propagation Learning Algorithm

```
Input: $N$ input sample vectors $\boldsymbol{V_i}, 1 \leq i \leq N$, 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: $w^h_{jk}, w^o_{ij}, 1 \leq k \leq a, 1 \leq j \leq b, 1 \leq i \leq c$
begin
    Randomly Initialize all $w^h_{jk}, w^o_{ij}$.
    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
                $h_j = f(\sum_{k=1}^{a} x_k * w^h_{jk})$
            for $i = 1, 2 \cdots, c$ do
                $o_i = f(\sum_{j=1}^{a} h_j * w^o_{ij})$
            //Back-Propagation Stage:
            if $Error = \frac{1}{2}\sum_{i=1}^{c}(ti - oi)^2 > threshold$ then
                $\Delta w^o_{ij} = -(t_i - o_i) * h_j$
                $\Delta w^h_{jk} = -h_j 1 - h_j x_k \sum_{i=1}^{c}[(t_i - o_i) * w^o_{ij})]$
                $w_{ij} = w_{ij} - \eta \Delta w_{ij}$
                $w^h_{jk} = w^h_{jk} - \eta \Delta w^h_{jk}$
            else
                //Learning Finish
                break
```

3.3 BGN Homomorphic Encryption

Homomorphic encryption is a form of encryption that enables operations on plaintexts to be performed on correspondingly 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 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 encrypted by the system's public key. Specifically, this scheme can be described as follows.

- **KeyGen:** Generate two cyclic groups G and G_1 of order $n = q_1q_2$ as well as a bilinear map $e : G \times G \to G_1$, where q_1 and q_2 are large primes. Randomly pick two generators $g, u \leftarrow G$ and set $h = u^{q_2}$. The public key is published as $PK = (n, G, G_1, e, g, h)$ and the private key is $SK = q_1$.
- **Encrypt:** Pick a random number $r \leftarrow Z_n$ and encrypt the message M as: $C = g^M h^r$, where C is the ciphertext.
- **Decrypt:** Obtain q_1. Compute $C^{q_1} = g^{Mq_1}h^{rq_1}$. As $h^{rq_1} = 1$ and g^{q_1} can be easily computed, the message M can be decrypted using Pollard's lambda method[15] as long as the message is not large.

Apparently, this scheme is homomorphic under the addition operation. It is easy to verify that one multiplication operation over the message can still be applied using bilinear map, after which unlimited number of addition operations can be applied. Details of this scheme is described in [5].

We shall point out that this scheme is designed for two parties. Moreover, due to message decryption involves solving discrete logarithm of the ciphertext using Pollard's lambda method, this algorithm just works with small numbers. While it is easy to extend the work to decrypt long messages (in which the message is treated as a bit string) using a mode of operation, it remains open to 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

In this paper, we aim at enabling multiple parties to jointly conduct back-propagation 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$\{x_1^m, x_2^m, \ldots, x_a^m\}$, $1 \leq m \leq N$), is arbitrary partitioned into $Z(Z \geq 2)$ 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 $\{x_1^m, x_2^m, \ldots, x_a^m\}, 1 \leq m \leq N$, is possessed by only one party - if P_s possesses $x_k^m, 1 \leq k \leq a$, then $x_{ks}^m = x_k^m$; otherwise $x_{ks}^m = 0$. In this paper, we use w_{jk}^h to denote the weight used to connect the input layer node k and the hidden layer node j; w_{ij}^o to denote the weight used to connect the hidden layer node j and the output layer node i, where $1 \leq k \leq a, 1 \leq j \leq b, 1 \leq i \leq c$ and a, b, c are the number of nodes of each layer as we describe 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 signal 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(weights, value of hidden layer node, value of output layer node, etc) generated can be revealed to anybody else other than TA.

4.2 Privacy Preserving Multi-party Neural Network Learning

In this section, we introduce our cloud based privacy preserving multi-party BPN 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

Algorithm 2. Privacy Preserving Multi-Party BPN Learning Algorithm

```
begin
    Input: each $P_s$'s data set for $N$ data samples,$x^v_{1s}, x^v_{2s}, \cdots, x^v_{as}, 1 \leq v \leq N$, $w^{hv}_{jks}$ and $w^{ov}_{ijs}$
           for $N$ samples, $iteration_{max}$, $\eta$, target value $t_i$
    Output: Network with final weights: $w^h_{jk}, w^o_{ij}, 1 \leq k \leq a, 1 \leq j \leq b, 1 \leq i \leq c$
    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
                $\sum_{k=1}^{a}(x^v_{k1} + x^v_{k2} + \cdots + x^v_{kZ}) * (w^{hv}_{jk1} + w^{hv}_{jk2} + \cdots + w^{hv}_{jkZ})$
                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 $f(\sum_{j=1}^{b}(h_{vj1} + h_{vj2} + \cdots + h_{vjZ}) * (w^{ov}_{ij1} + w^{ov}_{ij2} + \cdots + w^{ov}_{ijZ}))$
            //Back-Propagation Stage: 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
                for $i = 1, 2, \cdots, c$ do
                    //(step 1)
                    Using Algorithm 4 and Algorithm 3, each $P_s$ obtains random share $\Delta w^{ov}_{ijs}$ for
                    $\Delta w^{ov}_{ij} = (-(t_{vi} - \sum_{s=1}^{Z} o_{vis}) * (\sum_{s=1}^{Z} h_{vjs})$
                for $j = 1, 2, \cdots, b$ do
                    //(step 2)
                    Using Algorithm 4 and Algorithm 3, each $P_s$ obtains random share $\mu^v_s$ for
                    $\sum_{i=1}^{c}[(\sum_{s=1}^{Z} o_{vis} - t_{vi}) * (\sum_{s=1}^{Z} w^{ov}_{ijs})]$
                    //(step 3)
                    Using Algorithm 4 and Algorithm 3, each $p_s$ obtains random share $\kappa^v_s$ for
                    $\sum_{s=1}^{Z} x^v_{ks} * \sum_{s=1}^{Z} \mu^v_s$
                    //(step 4)
                    Using Algorithm 4 and Algorithm 3, each $P_s$ obtains random share $\vartheta^v_s$ 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 $\Delta w^{hv}_{jks}$ for
                    $\Delta w^{hv}_{jk} = \sum_{s=1}^{Z} \kappa^v_s * \sum_{s=1}^{Z} \vartheta^v_s$
                Each $P_s$ updates $w^{ov}_{ijs} = w^{ov}_{ijs} - \eta * \Delta w^{ov}_{ijs}$ and $w^{hv}_{jks} = w^{hv}_{jks} - \eta * \Delta w^{hv}_{jks}$
            else
                Learning Finished;
```

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 go into 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^o_{ij}, each P_s obtains random shares of the changes in weights, denoted as Δw^o_{ijs}, for Δw^o_{ij} from Eq. (1) by using the cloud based Algorithm 3 and Algorithm 4. For the weights of each hidden layer node w^h_{jk}, instead of directly computing the changes in weights according to Eq. (2), our proposed scheme divided it into four step computation: $\sum_{i=1}^{c}[(t_i - o_i) * w_{ij}^o)]$, $x_k \sum_{i=1}^{c}[(t_i - o_i) * w_{ij}^o)]$,$-h_j(1 - h_j)$ and Δw_{jk}^h, and let each P_s obtains the random shares(μ_s,κ_s,ϑ_s,Δw_{jks}^h) for each computation step respectively by using Algorithm 3 and Algorithm 4. Finally, each P_s updates its own weights with their shares and 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 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. 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.
Decryption of Large Numbers: Message decryption in the BGN algorithm involves solving the discrete log using Pollard's lambda method[15]. 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. W.l.o.g., 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 vector 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:

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

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 $e : G \times G \to G_1$. 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:** W.l.o.g., 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[15] given g^{q_1}. The encrypted scalar product can be decrypted jointly in the similar way.

The scalar product of V_A and V_B can be calculated as follows:

$$\begin{aligned}
&\sum_{i=1}^{k}(A_i * B_i) \qquad (5)\\
&= 2^{4d}\sum_{i=1}^{k}(A_{i2} * B_{i2}) + 2^{3d}(\sum_{i=1}^{k}(A_{i2} * B_{i1}) + \sum_{i=1}^{k}(A_{i1} * B_{i2}))\\
&+2^{2d}(\sum_{i=1}^{k}(A_{i2} * B_{i0}) + \sum_{i=1}^{k}(A_{i0} * B_{i2}) + \sum_{i=1}^{k}(A_{i1} * B_{i1}))\\
&+2^{d}(\sum_{i=1}^{k}(A_{i1} * B_{i0}) + \sum_{i=1}^{k}(A_{i0} * B_{i1})) + \sum_{i=1}^{k}(A_{i0} * B_{i0})
\end{aligned}$$

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 $\sum_{i=1}^{k}(A_i * B_i)$ using Eq. (5). 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

As the intermediate results generated during the BPN learning process may be used to derive some privacy information, the actual intermediate results cannot be known to each party as well as the cloud server. However, the 'BGN' algorithm[5] only supports one step multiplication over ciphertext and need to decrypt the intermediate results first, which will disclose some privacy data, for further privacy-preserving learning operations. 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 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 uploads the ciphertexts of L_s to the cloud and the cloud securely calculates the ciphertext of $sumL = \sum_{s=1}^{Z} L_s$ as:

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

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 is not very large. Finally, all each P_s get its secure share ϵ_s of ϵ. For P_1, $\epsilon_1 = L_1 + \hat{L}$ and for other parties, $\epsilon_s = L_s$.

To ensure the efficiency for computing $\hat{L}$, we consider about 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 $\epsilon < \sum_{s=1}^{Z} L_s$. At the same time, $\sum_{s=1}^{Z} L_s$ will not be much larger than ϵ(in 10 party scenarios, $\sum_{s=1}^{Z} L_s$ is at most 4bits larger than ϵ). Therefore, the cloud and all the parties can start decryption from $Case2$. If the successfully decryption of $\hat{L}$ cannot be achieved in empirical time using Pollard's lambda method, we can change to decrypt $\hat{L}$ process in $Case1$.

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

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 $L_s \xleftarrow{R} (0, u)$
        $C(L_s) = g_1^{L_s} h_1^{r_s q_2}$
        //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:
    $\epsilon_1 = L_1 + \hat{L}$
    for $i = 2, 3 \cdots, Z$ do
        $\epsilon_s = L_s$
    end
```

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. Since 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. Based on the aforementioned Algorithm 3 and 4, Z parties are allowed to securely calculate and share x. With these properties, we proposed Algorithm 5 to securely share x^k. Using 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 each P_s. With the ciphertext of x and x_s, each P_s then calculates $\hat{C}_s(x) = C(x)^{x_s}$ 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. Due to spcae limitation, we provide the correctness of this algorithm in Appendix.

4.6 Security Analysis

In this section, we sketch the prove that our scheme is semantically secure under the subgroup decision assumption. As stated in section 4.1, our scheme is

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 $x_s^j$
```

composed of four sub-algorithm and we first analyze the security of Algorithm 4, since the other three algorithm are based on it:

Theorem 1. *Algorithm 3 is semantically secure assuming the group G satisfies the subgroup decision assumption.*

Proof. First, in our scheme, the secret key q_1 is randomly split into Z parts, and each part is distributed to a data holder. Therefore, unless all the data holders work together, they cannot recover the secret key q_1 with their own parts. Suppose a polynomial time adversary B, which is also a involved data holder in the collaborative computation and can collude with less than other $Z-2$ data holders and the cloud server, is able to break the semantic security of the scheme with non-negligible advantage. We can construct an algorithm A that breaks the subgroup decision assumption with the same advantage. The construction of the algorithm A is the same as that in [5].

Security of Algorithm 4: To share one number, each party P_s independently chooses a random number L_s and encrypts it locally before he uploads to the cloud. Since Algorithm 3 is secure according to the above theorem, the random number chosen by each party is well protected. The decrypted data, i.e., the difference between ϵ and the sum of all the local random numbers, is indistinguishable from a random number as long as at least one of the local random numbers is not disclosed. This means that the data confidentiality of ϵ, which can be an intermediate value, can be well protected under the random oracle model as long as there is at least one non-malicious party.

Security of Algorithm 5: To share the result of approximation of activation function, we utilize the Algorithm 3 and Algorithm 4. Since we do not introduce any other steps to Algorithm 5 beside Algorithm 3 and Algorithm 4, we can achieve the same data confidentiality in Algorithm 5 as Algorithm 3 and Algorithm 4 did.

Security of Algorithm 2: In Algorithm 2, all the data exchange for parties are during the the secure computation of *step*$1, 2, 3, 4, 5$. All these steps are based on Algorithm 3, Algorithm 4 and Algorithm 5, which has been proved secure in terms of data confidentiality. Thus, we can prove that the same security for Algorithm 5 as Algorithm 3, Algorithm 4 and Algorithm 5 according to the composable security model.

5 Performance Analysis

5.1 Complexity 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. Since our scheme is composed of four sub-algorithms, we first give the analysis of each sub-algorithm and then provide the complexity of whole scheme. For expression simplicity, in the following part of this section, we denote time complexity of one multiplication operation on Group G as MUL[1] and that of one exponentiations operation on Group G as EXP.

Complexity of Algorithm 3: In multi-party scenarios, when all the parties need to jointly perform a secure scalar product or addition computation, each party P_s first needs to encrypt all his data attributes once, which costs $2n_s$ EXP and n_s MUL, where n_s is the number of data attributes owned by P_s. Then cloud would calculate the ciphertext of scalar product or sum based on encrypted data uploaded by each party and send back the ciphertext of result to each party. After receiving the ciphertext of result, each P_s just needs to perform one EXP and upload it into cloud for generate the final result. Therefore, the total computation cost for each party P_s for the secure computation for scalar product or sum is $2n_s$ EXP and $(n_s + 1)$ MUL. Note, $2n_s$ EXP and n_s MUL in our scheme is one time cost, it does not need to be performed in each secure computation round. For communication cost, each P_s needs to exchange $(n_s + 2)$ messages with cloud, where $|G|$ bits for each of n_s messages and $|G_1|$ bits for the other two. Note, n_s messages are also one time cost.

Complexity of Algorithm 4: To securely get the random share of the result of scalar product or addition, each P_s first needs to perform 2 EXP and one MUL to encrypt its chosen random number. After the cloud calculates the ciphertext of difference between the result the sum of all the local random number based on the uploaded ciphertexts, another EXP is needed for each P_s to decrypt the difference. Therefore, the total cost for each P_s during the secure sharing process is 3 EXP and one MUL. For communication cost, only 3 messages exchange are needed for each party P_s and cause $3|G_1|$ bits cost.

Complexity of Algorithm 5: To jointly perform the approximation of activation function(here we choose 5 terms for our approximation to achieve acceptable accuracy as in [21]), each party P_s needs to perform 8 EXP and 2 MUL besides the cost in Algorithm 3 and 4. For communication cost, 9 more messages, which have $9|G_1|$ bits are needed for each party besides cost in Algorithm 3 and 4.

Complexity of the Whole Scheme: In this part, we analyze the computation cost and communication cost of our whole privacy preserving multi-party neural network learning scheme. Considering the neural network configuration(a-b-c) as described in section 3.2, each party P_s first 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 holed by P_s, a,b and c represent the number

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

Table 1. Computation/Communication Cost of Privacy-Preserving BPN 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

	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)	$n_s * a * \|G\| + (24b + 5c) * \|G_1\|$	$Z^2 * (ab + 3bc + 4b + 2)\|G\|$ $+2n_s\|G\|$	$Z^2 * (b + 2bc + 4ab + 2)\|G\|$ $+2n_s\|G\|$

of input layer nodes, hidden layer nodes and output layer nodes respectively. Note: this is the one time cost and do not need to be performed again during the whole learning process. In the Feed Forward Stage, by using Algorithm 4 and Algorithm 5, each P_s 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, $step1, 3$ cost each P_s $5c$ EXP+$2c$ MUL and $5b$ EXP+$2b$ MUL respectively; $step1, 3$ 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.

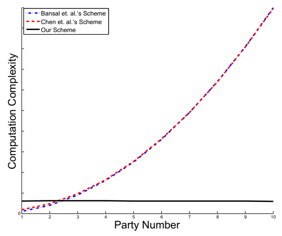

Fig. 2. Cost Influence of Party Number

Combining the cost for the two stages, the computation complexity for each party P_s of one round privacy preserving back-propagation neural 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 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, which is a significant difference with the excising scheme[4,6] as we shown in Figure.2. 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)+$Z^2 * (12b\text{EXP}+8b\text{MUL})$ are needed for each party P_s for computation. When compared with scheme in Ref.[6], which can support two party privacy preserving back-propagation neural network 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 a computation/communication complexity quadratic in Z for Z parties scenario and make their scheme unpractical. 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 Accuracy Analysis

In our proposed scheme, the only place that introduces accuracy loss is the approximation for the activation function. As described in section 4.5, we achieve the approximation by using Maclaurin series expansion, whose accuracy can be adjusted by modifying the number of series terms according to the system requirement. Due to the property of Maclaurin series, our scheme can achieve any higher accuracy by adding more series terms in approximation. Similar method of approximation with Maclaurin series expansion is used in [21], but it just supports two party setting. Moreover, the cost brought by the increasing accuracy requirement in our scheme is lightweight. Taking the a-b-c configuration BPN for an example, it will cause $8(b+c)$ EXP $2(b+c)$ MUL for each party if we extend 5 series terms to 9 series terms for more accuracy. Compared with the existing schemes in [4,6], which use the piecewise linear approximation[16] for the activation function and introduce about only $3\% - 6\%$ more error rate than none privacy-preserving scheme, our approximation can achieve at least the same accuracy as these works. Furthermore, due to limitation of finite fields for secure computation, both schemes in [4] and [6] need to map the real numbers in sigmod function to fixed-point representations in every step of Feed Forward Stage and Back-Propagation Stage, which will lead to further loss in accuracy. However, our proposed scheme can omit this limitation and be efficiency performed on the sigmod function without any accuracy loss during the secure computation process by utilizing the cloud server.

6 Conclusion

In this work, we proposed the first secure and practical multi-party BPN learning scheme over arbitrary 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 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. As a future work, we will study the feasibility of performing secure multiparty learning without the help of any trusted authority.

References

1. The health insurance portability and accountability act of privacy and security rules, `http://www.hhs.gov/ocr/privacy`
2. National standards to protect the privacy of personal health information, `http://www.hhs.gov/ocr/hipaa/finalreg.html`
3. Abramowitz, M., Stegun, I.A.: Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables. Dover Books on Mathematics. Dover, New York (1964)
4. Bansal, A., Chen, T., Zhong, S.: Privacy preserving back-propagation neural network learning over arbitrarily partitioned data. Neural Comput. Appl. 20(1), 143–150 (2011)
5. Boneh, D., Goh, E.-J., Nissim, K.: Evaluating 2-DNF Formulas on Ciphertexts. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 325–341. Springer, Heidelberg (2005)
6. Chen, T., Zhong, S.: Privacy-preserving backpropagation neural network learning. Trans. Neur. Netw. 20(10), 1554–1564 (2009)
7. Cun, L., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Handwritten digit recognition with a back-propagation network. In: Advances in Neural Information Processing Systems, pp. 396–404. Morgan Kaufmann (1990)
8. di Vimercati, S.D.C., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Over-encryption: management of access control evolution on outsourced data. In: Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB 2007, pp. 123–134. VLDB Endowment (2007)
9. El Gamal, T.: A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 10–18. Springer, Heidelberg (1985)
10. Fahlman, S.E.: Faster-learning variations on Back-propagation: An empirical study, pp. 38–51. Morgan Kaufmann (1988)
11. Flouri, K., Beferull-lozano, B., Tsakalides, P.: Training a svm-based classifier in distributed sensor networks. In: Proceedings of 14nd European Signal Processing Conference, pp. 1–5 (2006)
12. Grossman, R., Gu, Y.: 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 2008, New York, USA, pp. 920–927 (2008)
13. Grossman, R.L.: The case for cloud computing. IT Professional 11(2), 23–27 (2009)

14. Law, R.: Back-propagation learning in improving the accuracy of neural network-based tourism demand forecasting. Tourism Management 21(4), 331–340 (2000)
15. Menezes, A.J., Oorschot, P.C.V., Vanstone, S.A., Rivest, R.L.: Handbook of applied cryptography (1997)
16. Myers, D., Hutchinson, R.: Efficient implementation of piecewise linear activation function for digital vlsi neural networks. Electronics Letters 25(24), 1662–1663 (1989)
17. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. In: Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1, pp. 318–362. MIT Press, Cambridge (1986)
18. Schlitter, N.: A protocol for privacy preserving neural network learning on horizontal partitioned data. In: Proceedings of the Privacy Statistics in Databases (PSD) (September 2008)
19. Yang, B., Wang, Y.-D., Su, X.-H.: Research and Design of Distributed Neural Networks with Chip Training Algorithm. In: Wang, L., Chen, K., S. Ong, Y. (eds.) ICNC 2005, Part I. LNCS, vol. 3610, pp. 213–216. Springer, Heidelberg (2005)
20. Yao, A.C.: Protocols for secure computations. In: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, SFCS 1982, Washington, DC, USA, pp. 160–164 (1982)
21. Zang, S., Zhong, S.: A privacy-preserving algorithm for distributed training of neural network ensembles. To appear in Neural Computing and Applications

Appendix

Correctness of Algorithm 5

This section proves the correctness of Algorithm 5 and shows how to compute any $x^k, k \in Z_R$ for the approximation of activation function.

Proof. 1. When $k = 1$, $x^k = x$, as aforementioned in *Section* 4.3, it can be securely calculated without disclosing any privacy information.
2. If when $k = n$, multi-party can securely compute x^n without revealing any privacy information. For $k = n+1$, we have: $C(x^n) = g_1^{x^n} h_1^{r_n}$, $\hat{C}(x^n)_s = C(x^n)^{x_s} = g_1^{x^n * x_s} h_1^{r_n * x_s}$, where $C(x^n)$ is the ciphertext of x^n based on Algorithm 3 and $x_s$is the random share of $x(\sum_{s=1}^{Z} = x)$ for party P_s by using Algorithm 4. After each P_s uploading his $\hat{C}(x^n)_s$ to cloud, cloud can calculates as:

$$\begin{aligned}
&\prod_{s=1}^{Z} \hat{C}(x^n)_s \qquad (7)\\
&= g_1^{\sum_{s=1}^{Z} x^n * x_s} h_1^{\sum_{s=1}^{Z} r_n * x_s} = g_1^{x^n * \sum_{s=1}^{Z} x_s} h_1^{\sum_{s=1}^{Z} r_n * x_s}\\
&= g_1^{x^n * x} h_1^{x * \sum_{s=1}^{Z} r_n} = g_1^{x^{n+1}} h_1^{x * r_{n+1}} = C(x^{n+1})
\end{aligned}$$

With the ciphertext of x^{n+1}, all the parties can utilize Algorithm 3 and Algorithm 4 to securely get the random share of $x^n + 1$.