Privacy Preserving Back-Propagation Neural Network Learning

Tingting Chen, Sheng Zhong

***Abstract*— With the development of distributed computing environment, many learning problems now have to deal with distributed input data. To enhance coorperations in learning, it is important to address the privacy concern of each data holder by extending the privacy preservation notion to original learning algorithms. In this paper, we focus on preserving the privacy in an important learning model, multi-layer neural networks. We present a privacy preserving two-party distributed algorithm of back-propagation which allows a neural network to be trained without requiring either party to reveal her data to the other. We provide complete correctness and security analysis of our algorithms. The effectiveness of our algorithms is verified by experiments on various real world datasets.**

***Index Terms*— Privacy, Learning, Neural Network, Back-Propagation**

I. Introduction

With the development of distributed computing environment, many learning problems now have distributed input data. In such distributed scenarios, privacy concerns often become a big issue. For example, if medical researchers want to apply machine learning to study health care problems, they need to collect the raw data from hospitals and the follow-up information from patients. Then the privacy of the patients must be protected, according to the privacy rules in Health Insurance Portability and Accountability Act (HIPAA), which establishes the regulations for the use and disclosure of Protected Health Information [1].

A natural question is why the researchers would want to build a learning model (e.g, neural networks) without first collecting all the training data on one computer. If there is a learner trusted by all the data holders, then the trusted learner can collect data first and build a learning model. However, in many real-world cases it is rather difficult to find such a trusted learner, since some data holders will always have concerns like "What will you do to my data?" and "Will you discover private information beyond the scope of research?". On the other hand, given the distributed and networked computing environments nowadays, collaborations will greatly benefit the scientific advances. The researchers have the interest to obtain the result of cooperative learning even before they see the data from other parties. As a concrete example, [2] stated that the progress in neuroscience could be boosted by making links between data from labs around the world, but some researchers are reluctant to release their data to be exploited by others because of privacy and security concerns. More specifically, the neuroscientist in Dartmouth College found it difficult to encourage the sharing of brain-imaging data because there was possibility that the raw data could be misused or misinterpreted [3]. Therefore, there is a strong motivation for learners to develop cooperative learning procedures with privacy preservation.

In this paper, we focus on one of the most popular techniques in machine learning, multi-layer neural networks [4], [5], in which the privacy preservation problem is far from being practically solved. In [6] a preliminary approach is proposed to enable privacy preservation for gradient descent methods in general. However, in terms of multi-layer neural networks, their protocol is limited as it is only for one simple neural network configuration with one node in the output layer and no hidden layers. Although their protocol is elegant in its generality, it may be very restricted in practice for privacy preserving multi-layer neural networks.

We propose a light-weight two-party distributed algorithm for privacy preserving back-propagation training with vertically partitioned data[1] (i.e., when each party has a subset of features). Our contributions can be summarized as follows. (1) Our paper is the first to investigate the problem of training multi-layered neural networks over vertically partitioned databases with privacy constraints. (2) Our algorithms are provably private in the semi-honest model [7] and light-weight in terms of computational efficiency. (3) Our algorithms include a solution to a challenging technical problem, namely privacy preserving computation of activation function. This problem is highly challenging because most of the frequently used activation functions are infinite and continuous while cryptographic tools are defined in finite fields. To overcome this difficulty, we propose the first cryptographic method to securely compute sigmoid function, in which an existing piecewise linear approximation of the sigmoid function [8] is used. In order to make our algorithms practical, we do not adopt the costly circuit evaluation based approaches [9]. Instead, we use a homomorphic encryption based approach and the cryptographic scheme we choose is ElGamal [10]. (4) Both analytical and experimental results show that our algorithms are light-weight in terms of computation and communication overheads.

The rest of the paper is organized as follows. In Section

T. Chen and S. Zhong are with the Computer Science and Engineering Department, The State University of New York at Buffalo, Buffalo, NY 14260, U.S.A. (email:{tchen9, szhong}@cse.buffalo.edu)

[1]For horizontally partitioned scenario (i.e., when each party holds a subset of data objects with the same feature set), there is a much simpler solution that one party trains the neural network first and passes the training result to another party so that she can further train it with her data, and so on. So in this paper we only focus on the vertical partition case, which is much more technically challenging.

II, we introduce the technical preliminaries including notations, definitions and problem statement. In Section III, we present the novel privacy preserving back-propagation learning algorithm and two key component secure algorithms. Then we provide security analysis of the algorithm as well as the computation and communication overhead. In Section VI, with comprehensive experiments on various datasets, we verify the effectiveness and efficiency of our algorithm. After that, we conclude our paper.

A. Related Work

The notion of privacy preserving data mining was proposed by two different papers ([11] and [12]) in the year 2000. Both of the two papers addressed the problem of performing data analysis on distributed data sources with privacy constraints. In [11], Agrawal et al. presented a solution by adding noise to the source data, while in [12] Lindell and Pinkas used cryptographic tools to efficiently and securely build a decision tree classifier.After these two papers, a good number of data mining tasks have been studied with the consideration of privacy protection, for example classification [13], clustering [14], [15], association rule mining [16], etc.

In particular, privacy preserving solutions have been proposed for the following classification algorithms (to name a few): decision trees [12], [17], [18], Naive Bayes classifier [19], [20], and SVM [21], [22], [23]. Generally speaking, the existing works have taken either randomization based approaches (e.g., [11], [21]) or cryptography based approaches (e.g., [17], [24], [19], [23]). Randomization based approaches, by perturbing data, only guarantee a limited degree of privacy. In contrast, cryptography based approaches provide better guarantee on privacy than randomized based approaches, but most of the cryptography based approaches are difficult to be applied with very large databases, because they are resource demanding. For example, although Laur et al. proposed an elegant solution for privacy preserving SVM in [23], their protocols are based on circuit evaluation which is considered very costly in practice.

In cryptography, there is also a general-purpose technique called secure multi-party computation. The works of secure multi-party computation originate from the solution to the millionaire problem proposed by Yao [25], in which two millionaires can find out who is richer without revealing the amount of their wealth. In [9], a protocol is presented which can privately compute any probabilistic polynomial function. Although secure multi-party computation can theoretically solve all problems of privacy-preserving computation, it is too expensive to be applied to practical problems [26]. This general solution is especially infeasible for cases in which parties hold huge amounts of data.

There are few works on the problem of privacy preserving neural networks learning (limited to [27], [28], [6]). The most recent one is [6]. As discussed above, the difference between our work and their is that we focus on privacy preserving neural networks and provide a light-weight algorithm applicable to more complex neural networks configurations, while their protocol is for gradient descent methods in general and thus loses some power for neural networks in particular.

Protocols in [27], [28] are also designed for privacy-preserving neural-network-based computations. In particular, Chang and Lu [27] proposed a cryptographic protocol for non-linear classification with feed-forward neural networks. Barni et al. in [28] presented three algorithms with different levels of privacy protections, i.e., protecting private weight vectors, protecting private activation functions, and preventing data providers from injecting fake cells to the system.

The fundamental difference between our work and the protocols in [27] and [28] is that they work in different learning scenarios. In [27] and [28], it is assumed that there is a neural network owner; this neural network owner owns a neural network, but does not have any data to train it. In addition, there are some data providers who have data that can be used to train the neural network. The goal of [27] and [28] is to ensure that the neural network owner does not get any knowledge about the data, and at the same time, the data providers do not get the knowledge embedded in the neural network (e.g., the node activation functions). In constrast, in this paper we consider a totally different scenario in which there are at least two neural network owners, each having his own set of data. The two parties want to jointly build one neural network based on all the data, but each party does not want to reveal his own data. As a result, protocols in [27] and [28] cannot be applied in the scenario that we are studying in this paper.

Moreover, [27] and [28] are of theoretical interests only; they have not implemented their protocols, neither have they tested their protocols in any experiments. In contrast, we have implemented our algorithm and carried out extensive experiments. The results of our experiments show that our algorithm is practical.

II. Technical Preliminaries

In this section we give a brief review of the version of the Back-Propagation Network (BPN) algorithm we consider [29] and introduce the piecewise linear approximation we use for the activation function. We also give a formal statement of problem with a rigorous definition of security. Then we briefly explain the main cryptographic tool we use, ElGamal [10].

A. Notations for back-propagation learning

For ease of presentation, in this paper we consider a neural network of three layers, where the hidden layer activation function is sigmoid and the output layer is linear. Note that it is trivial to extend our work to more layers.

Given a neural network with a-b-c configuration, one input vector is denoted as $(x_1, x_2, \cdots, x_a)$. The values of hidden layer nodes are denoted as $\{h_1, h_2, \cdots, h_b\}$, and the values of output layer nodes are $\{o_1, o_2, \cdots, o_c\}$. w^h_{jk} denotes the weight connecting the input layer node k and the hidden layer node j. w^o_{ij} denotes the weight connecting j and the output layer node i, where $1 \leq k \leq a, 1 \leq j \leq b, 1 \leq i \leq c$.

We use Mean Square Error (MSE) as the error function in the back-propagation algorithm, $e = \frac{1}{2}\sum_i (t_i - o_i)^2$. For the neural networks described above, the partial derivatives are listed as (1) and (2), for future reference.

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

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

B. The piecewise linear approximation of activation function

In this subsection, we introduce the piecewise linear approximation of activation function. The major reason of introducing the approximation is that cryptographic tools work in finite fields and thus can not be directly applied to the secure computation of functions like sigmoid. Approximating the activation function in a piecewise way offers us an opportunity to apply cryptographic tools to make the computation of sigmoid function privacy-preserving.

Equation (3) is a piecewise linear approximation [8] of the sigmoid function $\frac{1}{1+e^{-x}}$. Our privacy preserving algorithm for back-propagation network learning is based on this approximation.[2]

$$y(x) = \begin{cases} 1 & x > 8 \\ 0.015625x + 0.875 & 4 < x \leq 8 \\ 0.03125x + 0.8125 & 2 < x \leq 4 \\ 0.125x + 0.625 & 1 < x \leq 2 \\ 0.25x + 0.5 & -1 < x \leq 1 \\ 0.125x + 0.375 & -2 < x \leq -1 \\ 0.03125x + 0.1875 & -4 < x \leq -2 \\ 0.015625x + 0.125 & -8 < x \leq -4 \\ 0 & x \leq -8 \end{cases} \tag{3}$$

C. Security definition and problem statement

• **Semi-honest model.** As many existing privacy preserving data mining algorithms (e.g., [17], [30]), we adopt semi-honest model in this paper. Semi-honest model is a standard adversary model in cryptography [7]. In this paper the security of our algorithm is guaranteed in this model. When computing function f in a distributed fashion, semi-honest model requires that each party that participates in the computation follow the algorithm, but a party may try to learn additional information by analyzing the messages that she receives during the execution. In order to guarantee the security of distributed algorithm of computing f, it must be ensured that each party can learn nothing beyond what can be implied by her own input and output.

Semi-honest model is a right fit for our setting, because normally participants want to learn the neural network learning results and thus they are willing to follow the algorithm to guarantee the results correctness. The security guaranteed in semi-honest model can relieve the concerns about their data privacy. Of course, in reality there may be scenarios in which there are malicious adversaries. It has been shown (see [7]) that a distributed algorithm that is secure in the semi-honest model can be converted to one that is secure in the malicious model, with some additional costs in computation and communications for zero knowledge proofs. (It is a highly challenging task to reduce these additional costs to achieve security in the malicious model. We leave it as one of our future research topics.)

Based on semi-honest model, the problem of privacy preserving back-propagation neural networks learning in this paper, is stated below.

• **Privacy preserving two-party distributed neural network training.** Suppose that a set of training samples are vertically partitioned between two parties A and B. A holds a dataset D_1 with m_A attributes for each data entry. B holds a dataset D_2, with m_B attributes for each data entry. We denote one data entry in D_1 as $x_A = (x_1, x_2, \cdots, x_{m_A})$ and in D_2 $x_B = (x_{m_A+1}, \cdots, x_{m_A+m_B})$.

Privacy preserving two-party distributed neural network training is that in each round of neural network learning, two parties jointly compute the additive values of connection weights without compromising their privacy of input data.[3] Formally, with training samples x_A and x_B from party A and B respectively and a target value $t(x)$, our goal is to let each party get her own share of the additive value of each weight Δw, without revealing the original training data, x_A or x_B, to each other.

Note that, in this paper, we restrict our attention to the privacy concerns brought by insiders (i.e., participants of the distributed neural network training) only. The security and privacy issues associated with outsider attacks (i.e., attacks by non-participating parties and attacks in the communication channels) are orthogonal issues beyond the scope of this paper. In practice, if our algorithms are to be used, appropriate access control and security communication techniques must also be used, to guarantee that all sensitive information is transmitted over secure channels and unauthorized access to the system is prevented.

D. ElGamal scheme

In our privacy preserving algorithms for neural network learning, a homomorphic cryptographic scheme, ElGamal [10], is utilized. In this subsection, we review the basics and properties of ElGamal scheme.

ElGamal is a public-key encryption scheme which can be defined on any cyclic group. Let G be a cyclic group of prime order q with generator g. We assume that decisional Diffie-Hellman assumption (DDH) [31] holds in G such that ElGamal is semantically secure.

Components. ElGamal scheme consists of three components, i.e., key generation, encryption and decryption.

- Key Generation.

[2]It is easy to extend our work to other piecewise linear approximations of activation function. Here we choose this specific approximation as an example to demonstrate in detail how our algorithms work.

[3]In this paper, we provide an algorithm in which two parties learn all the weights after each round of training. Note that this algorithm can be extended to a more secure fashion by making each party hold only a random additive share of each weight at the end of each round and continue to the next round with the partitioned weights [7]. But in this case much more computational overhead will be added. So for efficiency reasons, we keep the algorithm as it is.

A value $x \in Z_p$ is randomly chosen as the private key. The corresponding public key is (G, q, g, h), where $h = g^x$.

- Encryption.
  A message $m \in G$ is encrypted as follows. A value $r \in Z_p$ is chosen as random. Then the ciphertext is constructed as $(C_1, C_2) = (g^r, m \cdot h^r)$.
- Decryption.
  The plain text is computed as

$$\frac{C_2}{C_1^x} = \frac{m \cdot h^r}{g^{x \cdot r}} = \frac{m \cdot h^r}{h^r} = m.$$

Homomorphic Property. ElGamal scheme is homomorphic in that for two messages, m_1 and m_2, an encryption of $m_1 + m_2$ can be obtained by an operation on $E(m_1, r)$ and $E(m_2, r)$ without decrypting any of the two encrypted messages.

Probabilistic Property. ElGamal scheme is also probabilistic, which means that besides clear texts, the encryption operation also needs a random number as input. Let an encryption of message m using public key (G, q, g, h) and a random number r be denoted as $E_{(G,q,g,h)}(m, r)$. For simplicity we use notation $E(m, r)$ instead of $E_{(G,q,g,h)}(m, r)$ in the rest of this paper.

In probabilistic encryption schemes, there are many encryptions for each message. ElGamal allows an operation that takes one encrypted message as input and outputs another encrypted message of the same clear message. This is called rerandomization operation. For instance, taking the encrypted message $(C_1, C_2) = (g^r, m \cdot h^r)$ as input, one can do the rerandomization and obtain another cyphertext of m as,

$$(C_1', C_2') = (C_1 \cdot g^s, C_2 \cdot h^s) = (g^{r+s}, m \cdot h^{r+s}).$$

In this paper, messages are randomized by both parties. Therefore, no individual party can decrypt any message by itself. However, each party can *partially* decrypt a message [32]. When the message is partially decrypted by both parties, it is fully decrypted.

III. Privacy preserving neural network learning

In this section, we present a privacy-preserving distributed algorithm for training the neural networks with back-propagation algorithm. A privacy preserving testing algorithm can be easily derived from the feed-forward part of the privacy-preserving training algorithm.

Our algorithm is composed of many smaller private computations. We will look into them in detail after first giving an overview.

A. Privacy preserving neural network training algorithm

Here we build the privacy preserving distributed algorithm for the neural network training process under the assumption that we already have the algorithm to securely compute the piecewise linear function (Algorithm 2) and the algorithm to securely compute the product of two numbers held by two parties (Algorithm 3). We will explain the two component algorithms in detail later.

For each training iteration, the input of the privacy-preserving back-propagation training algorithm is $\langle \{x_A, x_B\}, t(x) \rangle$, where x_A is held by party A, while x_B is held by party B. $t(x)$ is the target vector of the current training data and it is known to both parties. The output of algorithm is the connection weights of output layer and hidden layer, i.e., $\{w_{ij}^o, w_{jk}^h | \forall k \in \{1, 2, \cdots, a\}, \forall j \in \{1, 2, \cdots, b\}, \forall i \in \{1, 2, \cdots, c\}\}$.

The main idea of the algorithm is to secure each step in the non-privacy-preserving back-propagation algorithm, with two stages, feeding forward and back-propagation. In each step, neither the input data from the other party nor the intermediate results can be revealed. In particular, we apply Algorithm 2 to securely compute the sigmoid function, and Algorithm 3 is used to guarantee privacy preserving product computation.

To hide the intermediate results such as the values of hidden layer nodes, the two parties randomly share each result so that neither of the two parities can imply the original data information from the intermediate results. Here by "randomly share", we mean that each party holds a random number and the sum of the two random numbers equals to the intermediate result. Note that with intermediate results randomly shared among two parties, the learning process can still securely carry on to produce the correct learning result (see correctness analysis in Section III-A.1).

After the entire process of private training, without revealing any raw data to each other, the two parties jointly establish a neural network representing the properties of the union dataset.

Our training algorithm for back-propagation neural networks can be summarized as in Algorithm 1.

For clarity of presentation, in Algorithm 1, we separate the procedure to compute $\sum_i [-(t_i - o_i) w_{ij}^o] h_j (1 - h_j)$ and explain it in Algorithm 1.1.

1) Correctness: We show that if party A and party B follow Algorithm 1, they will jointly derive the correct weight update result in each learning round, with each of them holding a random share.

For any output layer weight w_{ij}^o, in step (2.1) of Algorithm 1 we have

$$\begin{aligned}
& \Delta_1 w_{ij}^o + \Delta_2 w_{ij}^o \\
= \; & (o_{i1} - t_i) h_{j1} + r_{11} + r_{21} + (o_{i2} - t_i) h_{j2} + r_{12} + r_{22} \\
= \; & (o_{i1} - t_i) h_{j1} + (o_{i2} - t_i) h_{j2} + h_{j1} o_{i2} + h_{j2} o_{i1} \\
= \; & -(t_i - o_{i1} - o_{i2})(h_{j1} + h_{j2}) \\
= \; & -(t_i - o_i) h_j \\
= \; & \frac{\partial e}{\partial w_{ij}^o} \\
= \; & \Delta w_{ij}^o.
\end{aligned}$$

Therefore the additive random shares of the two parties can add up to the correct value for hidden layer connection weights.

Now we show that the correctness is also guaranteed for hidden layer weights.

In step (2.2) of Algorithm 1, it is easy to get that no matter which party holds the attribute x_k, we always have

Algorithm 1 Privacy preserving distributed algorithm for back-propagation training

Initialize all weights to small random numbers, and make them known to both parties.

Repeat

for all training sample $\langle\{x_A, x_B\}, t(x)\rangle$ **do**

Step1: feed forward stage

(1.1) **For each** hidden layer node h_j, party A computes $\sum_{k\le m_A} w^h_{jk}x_k$, and party B computes $\sum_{m_A<k\le m_A+m_B} w^h_{jk}x_k$.

(1.2) Using Algorithm 2, party A and B jointly compute the sigmoid function for each hidden layer node h_j, obtaining the random shares h_{j1} and h_{j2} respectively s.t. $h_{j1} + h_{j2} = f(\sum_k w^h_{jk}x_k)$.

(1.3) **For each** output layer node o_i, Party A computes $o_{i1} = \sum_i w^o_{ij}h_{j1}$ and party B computes $o_{i2} = \sum_i w^o_{ij}h_{j2}$, s.t. $o_i = o_{i1} + o_{i2} = \sum_i w^o_{ij}h_{j1} + \sum_i w^o_{ij}h_{j2}$.

Step2: back-propagation stage

(2.1) **For each** output layer weight w^o_{ij}

Party A and B apply Algorithm 3 to securely compute the product $h_{j1}o_{i2}$, obtaining random shares r_{11} and r_{12} respectively, s.t. $r_{11} + r_{12} = h_{j1}o_{i2}$. Similarly they compute the random partitions of $h_{j2}o_{i1}$, r_{21} and r_{22}, s.t. $r_{21} + r22 = h_{j2}o_{i1}$.

Party A computes $\Delta_1 w^o_{ij} = (o_{i1} - t_i)h_{j1} + r_{11} + r_{21}$ and B computes $\Delta_2 w^o_{ij} = (o_{i2} - t_i)h_{j2} + r_{12} + r_{22}$.

(2.2) **For each** hidden layer weight w^h_{jk}

Using Algorithm 1.1, party A and B jointly compute $\sum_i[-(t_i - o_i)w^o_{ij}]h_j(1 - h_j)$, obtaining random shares q_1 and q_2 respectively, s.t. $q_1 + q_2 = \sum_i[-(t_i - o_i)w^o_{ij}]h_j(1 - h_j)$.

If $k \le m_A$, that is A holds the input attribute x_k, applying Algorithm 3 to securely compute x_kq_2, A and B respectively get r_{61} and r_{62} s.t. $x_kq_2 = r_{61} + r_{62}$. Then $\Delta_1 w^h_{jk} = q_1x_k + r_{61}$ and $\Delta_2 w^h_{jk} = r_{62}$.

If $m_A < x_k \le m_A + m_B$, A and B apply Algorithm 3 to get r_{61} and r_{62} s.t $x_kq_1 = r_{61} + r_{62}$. In this case, $\Delta_1 w^h_{jk} = r_{61}$, $\Delta_2 w^h_{jk} = q_2x_k + r_{62}$. A and B respectively compute $\Delta_1 w^h_{jk}$, $\Delta_2 w^h_{jk}$.

Step3:

A (B, resp.) sends $\Delta_1 w^o_{ij}$, $\Delta_1 w^h_{jk}$($\Delta_2 w^o_{ij}$, $\Delta_2 w^h_{jk}$, resp.) to B(A, resp.). A and B compute $w^o_{ij} \leftarrow w^o_{ij} - \eta(\Delta_1 w^o_{ij} + \Delta_2 w^o_{ij})$ for each hidden layer weight, and $w^h_{jk} \leftarrow w^h_{jk} - \eta(\Delta_1 w^h_{jk} + \Delta_2 w^h_{jk})$ for each output layer weight.

end for

Until (termination condition)

Algorithm 1.1 Securely computing

$$\sum_i[-(t_i - o_i)w^o_{ij}]h_j(1 - h_j).$$

Input: h_{j1}, o_{i1} (h_{j2}, o_{i2} resp.) for party A (party B resp.) abtained inside Algorithm 1.

Output: random shares q_1, q_2 for A and B resp.

1. Using Algorithm 3, party A and B get random shares of $h_{j1}h_{j2}$, r_{31} and r_{32} respectively, s.t. $h_{j1}h_{j2} = r_{31} + r_{32}$.
2. For clarity, we name intermediate results as $h_{j1} - h^2_{j1} - 2r_{31} = p_1$, $\sum_i(-t_i + o_{i1})w^o_{ij} = s_1$, $h_{j2} - h^2_{j2} - 2r_{32} = p_2$ and $\sum_i o_{i2}w^o_{ij} = s_2$.
3. Party A computes p_1 and s_1; B computes p_2 and s_2.
4. Applying Algorithm 3, party A gets r_{41} and r_{51}, party B gets r_{42} and r_{52}, s.t. $r_{41} + r_{42} = s_1p_2$, $r_{51} + r_{52} = s_2p_1$.
5. Name $q_1 = s_1p_1 + r_{41} + r_{51}$ and $q_2 = s_2p_2 + r_{42} + r_{52}$. Party A computes q_1 and party B computes q_2 locally.

$$\begin{aligned}
& \Delta_1 w^h_{jk} + \Delta_2 w^h_{jk} \\
= \;& x_kq_1 + x_kq_2 \\
= \;& (s_1p_1 + r_{41} + r_{51} + s_2p_2 + r_{42} + r_{52})x_k \\
= \;& (s_1 + s_2)(p_1 + p_2)x_k \\
= \;& (\sum_i(-t_i + o_{i1})w^o_{ij} + \sum_i o_{i2}w^o_{ij}) \\
& (h_{j1} - h^2_{j1} - 2r_{31} + h_{j2} - h^2_{j2} - 2r_{32})x_k \\
= \;& \sum_k[-(t_i - o_i)w^o_{ij}](h_{j1} + h_{j2})(1 - h_{j1} - h_{j2})x_k \\
= \;& \sum_i[-(t_i - o_i)w^o_{ij}]h_j(1 - h_j)x_k \\
= \;& \frac{\partial e}{\partial w^h_{jk}} \\
= \;& \Delta w^h_{jk}
\end{aligned}$$

Hence using Algorithm 1, the additive update of hidden layer weights can be correctly computed by the two parties without compromising their data privacy.

B. Securely computing the piecewise linear sigmoid function

In this subsection we present a secure distributed algorithm for computing piecewise linear approximated sigmoid function $y(x)$ (as shown in Algorithm 2).

Although each party only holds one part of the input to the sigmoid function, this algorithm enables them to compute the approximate value of the function without knowing the part of input from the other party. Actually, in this algorithm there is no way for each party to explore the input of the other, but the function value can still be computed.

Formally, the input of the algorithm is x_1 held by party A, x_2 held by party B. The output of function y, $y(x_1 + x_2)$, is also randomly shared by the two parties. Note that the parties can always exchange their random shares of result at the end of the algorithm, so that they can learn the complete value of

sigmoid function, but since in our paper the result of sigmoid function is only an intermediate result for the whole learning process, here we keep it randomly shared.

For ease of presentation, we write the algorithm under the assumption that x_1 and x_2 are integers. Note that we can easily rewrite the algorithm to allow real numbers with precision of a few digits after the dot. Actually the algorithm for factorial numbers is the same in essence, given that we can shift the float point to the right to get integer numbers, or in other words, integers and factorials are easy to exchange to each other in binary representation.

After the piecewise linear approximation, as the input is splitted between two parties, both parties do not even know which linear function to use without the knowledge of which range the input falls in. Our main idea is to let one party compute all possible values according to her own input and among those values, the other party picks the result they are looking for.

As shown in Algorithm 2, party A first computes the sigmoid function values for different possible inputs of B. After subtracting a same random number R from each of the function values, A encrypts each of them (step 1). Actually the random number generated by A is the final output for A, the random share of the sigmoid function value. The random share for B is one of the clear texts hidden behind ElGamal scheme. So the remaining task of this algorithm is to let B obtain that clear text, $y(x_1 + x_2) - R$, which corresponds to her input without revealing the input of any party to each other. As A does not know the input of B, she sends all the encrypted results to B and lets B pick the one based on her own input. Here since all the results are encrypted, B cannot get any information about A's original data either. Because A can get to know the value of x_2 by comparing the encrypted message chosen by B with all those generated by herself, a rerandomization is conducted by B, before B sends back her choice to A to protect B's data, x_2. After A and B sequentially decrypt, B gets her share of the final sigmoid function value (step 3 and step 4).

Algorithm 2 Securely computing the piecewise linear sigmoid function

Step 1: Party A generates a random number R and computes $y(x_1 + i) - R$ for each i, s.t. $-n < i \leq n$. Define $m_i = y(x_1 + i) - R$. Party A encrypts each m_i using ElGamal scheme and gets $E(m_i, r_i)$, where each r_i is a new random number. Party A sends each $E(m_i, r_i)$ in the increasing order of i.

Step 2: Party B picks $E(m_{x_2}, r_{x_2})$. She rerandomizes it and sends $E(m_{x_2}, r')$ back to A, where $r' = r_{x_2} + s$, and s is only known to party B.

Step 3: Party A partially decrypts $E(m_{x_2}, r')$ and sends the partially decrypted message to B.

Step 4: Party B finally decrypts the message (by doing partial decryption on the already partially decrypted message) to get $m_{x_2} = y(x_1 + x_2) - R$. Note R is only known to A and m_{x_2} is only known to B. Furthermore, $m_{x_2} + R = y(x_1 + x_2) = f(x)$.

Note that in Step 1 of Algorithm 2, party A must generate a *new* random number r_i for encrypting each message. Although party B can get $h^{r_{x_2}}$ at the end of the algorithm from $E(m_{x_2}, r_{x_2})$ and the clear message m_{x_2}, he has no way to get other numbers of h^{r_i}. On the other hand, party A can always conduct the partial decryption without knowing which encrypted message B has chosen, because the decryption of ElGamal scheme is independent from random number r_i (please see details in Section II-D).

C. Privacy-preserving distributed algorithm for computing product

To securely compute product, some existing secure multi-party computation protocols for dot product can be utilized (e.g. [33], [34]) by taking integer numbers as a special form of vector input. Here we provide another option for privacy preserving product computation, stated as Algorithm 3. The advantage of Algorithm 3 is that it can be very efficient for some applications, when the finite field of input is small.

Before applying Algorithm 2, a pre-processing step is needed to convert the input of each party into a small integer.

Assume party A holds integer M and party B holds integer N, both M and N are in the same range $-n < M \leq n$, $-n < N \leq n$ (Recall that n is a small integer). Basically it follows a similar idea as of Algorithm 2. After running Algorithm 3, A and B get random numbers respectively which are summed to $M \cdot N$.

Algorithm 3 Securely computing the product of two integers

Step 1: Party A generates a random number R and computes $M \cdot i - R$ for each i, s.t. $-n < i \leq n$. Note $m_i = M \cdot i - R$. A encrypts each m_i using ElGamal scheme and gets $E(m_i, r_i)$, where each r_i is a new random number. Then party A sends each $E(m_i, r_i)$ to party B in the increasing order of i.

Step 2: Party B picks $E(m_N, r_N)$. She rerandomizes it and sends $E(m_N, r')$ back to A, where $r' = r_N + s$, and s is only known to party B.

Step 3: Party A partially decrypts $E(m_N, r')$ and sends the partially decrypted message to B.

Step 4: Party B finally decrypts the message (by doing partial decryption on the already partially decrypted message) to get $m_N = M \cdot N - R$. Note R is only known to A and m_N is only known to B. Furthermore, $m_N + R = M \cdot N$.

IV. Security analysis

In this section, we explain why our algorithms are secure in the semi-honest model. Recall that in semi-honest model, the parties follow the protocol and may try to analyze what she can see during the protocol execution. To guarantee security in the semi-honest model, we must show that parties can learn nothing beyond their outputs from the information they get

throughout the protocol process[4]. A standard way to show this is to construct a simulator which can simulate what the party can see in the protocol given only the input and output of the protocol for this party.

Since the privacy preserving back-propagation training (Algorithm 1) uses Algorithm 2 and Algorithm 3 as building blocks, we first analyze the security of Algorithm 2 and Algorithm 3, and then we conduct a security analysis of Algorithm 1 in the viewpoint of overall security level.

A. Security of Algorithm 2 and Algorithm 3

Before we discuss the security of computing piecewise linear sigmoid function (Algorithm 2), recall that the encryption scheme, ElGamal, which we are using here is semantically secure [31] and satisfies the properties discussed in the earlier section. Since ElGamal is semantically secure, each ciphertext can be simulated by a random ciphertext. Hence, we can construct two simulators, for A and B in Algorithm 2, respectively.

In step 1, the simulator for A does what A should do in the protocol. In step 2, the simulator for A generates a random ciphertext to simulate the ciphertext A should receive. In step 3, again the simulator for A does what A should do in the protocol. Since A does not observe anything in step 4, the simulator for A does not need to simulate this step.

In step 1, the simulator for B generates random ciphertexts to simulate the ciphertexts B should receive. In step 2, the simulator for B does what B should do in the protocol. In step 3, the simulator for B generates a random encryption of the output of B in this algorithm to simulate the ciphertext B should receive. In step 4, again the simulator for B does what B should do in the protocol.

The simulators for Algorithm 3 can be constructed in the same way as simulators for Algorithm 2.

B. Security of Algorithm 1

In Algorithm 1, the termination condition of training is known to both parties, so the simulators can set up the training loop for both parties. In each training round, most of the message transmissions are taking place inside the calls of Algorithm 2 and 3, except in step 3, where B and A respectively receive $\Delta_1 w_{ij}^o, \Delta_2 w_{ij}^o$ and $\Delta_1 w_{jk}^h, \Delta_2 w_{jk}^h$. We will first show that $\Delta_1 w_{ij}^o$ and $\Delta_1 w_{jk}^h$ can be simulated by the simulator for party B, and then likewise $\Delta_2 w_{ij}^o$ and $\Delta_2 w_{jk}^h$ can be simulated for party A.

In Section 2.1, $\Delta_2 w_{ij}^o$ is defined as $(o_{i2} - t_i)h_{j2} + r_{12} + r_{22}$. The variable h_{j2} can be simulated because of the fact that Algorithm 2 is secure (as shown above). The variables r_{12} and r_{22} can also be simulated based on the security of Algorithm 3. Since o_{i2} and t_i are part of the input of party B, $\Delta_2 w_{ij}^o$ can be simulated for B by only looking at its own input. Meanwhile, since weights are output of the training algorithm and η is known to both parties as input, by the weight update rule, $w_{ij}^o \leftarrow w_{ij}^o - \eta(\Delta_1 w_{ij}^o + \Delta_2 w_{ij}^o)$, $\Delta_1 w_{ij}^o$ can be simulated for party B. The simulation of $\Delta_1 w_{jk}^h$ for party B is likewise. Similarly, we can construct the simulator for party A to simulate $\Delta_2 w_{ij}^o$ and $\Delta_2 w_{jk}^h$.

Note that, in the above, the simulators for Algorithm 1 integrates the simulators for Algorithm 2 and 3 when the two component algorithms are called together with the simulations of $\alpha_1, \alpha_2, \beta_1, \beta_2$ in step 3. This completes the construction of simulators for Algorithm 1.

V. Analysis of algorithm complexity and accuracy loss

A. Complexity analysis

In this subsection, we analyze the computation and communication complexity of our privacy preserving back-propagation algorithms. First we present the computation and communication cost in the two component algorithms(i.e., for computing piecewise linear sigmoid function and product computing) and then use the result to further analyze the running time of Algorithm 1.

1) Securely computing the piecewise linear sigmoid function: In step 1 of Algorithm 2, there are $2 \times n$ encryptions by party A, where n is the parameter in piecewise linear sigmoid function definition. In step 2, 1 rerandomization is conducted. In step 3 and step 4, party A and party B perform one partial decryption respectively. So the total computation cost in Algorithm 2 is $T = (2n + 1)C + 2D$, where C is the cost of encryption and D is the cost of partial description.

Similarly, the total computation cost in Algorithm 3 is also $(2n + 1)C + 2D$.

2) Execution time of one round training: The running time of one round of back-propagation training consists of two parts, time for feedforward stage and for back-propagation stage.

We first consider the execution time of non-privacy preserving back-propagation algorithm. When executing a non-privacy-preserving fully-connected neural network with $a-b-c$ configuration defined earlier in this paper, one multiplication operation and one addition operation are needed for each connecting weight. Besides, we also need to call activation function one time for each hidden layer node. Therefore the running time for feeding forward is $(ab+bc)S+b\times G$, where S is the cost for one multiplication and one addition, G is the cost for computing activation function, and a, b, c represent the number of units in each layer, respectively.

In step 1 of Algorithm 1, the major difference from the non-privacy-preserving version is the execution of Algorithm2 in (1.2) instead of calling the sigmoid function. In (1.3), one more addition and multiplication is needed for each connection since the value of each hidden layer unit of activation is splitted by the two parties. But since the party A and B can run the algorithm in parallel, this does not increase the execution time of the whole algorithm. With the time cost for Algorithm 2 is T, we get the running time in feedforward step is $(ab + bc)S + b \times T$.

[4]The definition of security we use here is actually standard for a *distributed algorithm* in the semi-honest model (see [35], in which a distributed algorithm is called a "protocol"). Note that we do not use the definition of semantic security, because it is for an *encryption scheme* (see [35], in which an encryption scheme is called a "cryptosystem"), not for a distributed algorithm. In this paper, we build a distributed learning algorithm and study its security. Hence, we have to use this definition, not the definition of semantic security.

Now we consider the back-propagation stage. In step (2.1) there are two calls of the Algorithm 3 for each hidden-output connection. Time for one multiplication and three additions is also needed. Because multiplication time is much more significant than addition time, for simplicity, we also use S to denote the time for one multiplication and three additions. In step (2.2), Algorithm 3 are called 4 times, and thus the time for multiplications and additions is $(c+4)S$. The total execution time for back-propagation is $bc(2\times T+S)+ab(4\times T+4\times S+c\times S)=(2bc+4ab)T+(bc+4ab+abc)S$.

Combining the time for the two stages, we obtain the running time of one round privacy preserving back-propagation learning, $(5ab+2bc+abc)S+(2bc+4ab+b)T$.

3) Communication overhead: In Algorithm 2 and 3, there are $2n+2$ messages have been passed between party A and B. With each message being s bits long, the communication overhead is $(2n+2)\times s$. The total communication overhead of one round learning in Algorithm 1 is the overhead caused by calling algorithm 2 and 3, plus $2\times s$ in step 3, which is $(b+2bc+4ab)(2n+2)s+2s$.

B. Analysis of accuracy loss

There are two places in our algorithms where we introduce approximation for the goal of privacy. One is that the sigmoid function used in the neuron computing is replaced by a piecewise linear function. The other approximation is introduced by mapping the real numbers to fixed-point representations to enable the cryptographic operations in Algorithm 2 and Algorithm 3. This is necessary in that intermediate results, for example the values of neurons, are represented as real numbers in normal neural network learning, but cryptographic operations are on discrete finite fields. We will empirically evaluate the impact of these two sources of approximation on the accuracy loss of our neural network learning algorithm in Section VI. Below we give a brief theoretical analysis of the accuracy loss caused by the fixed-point representations.

1) Error in truncation: Suppose that the system, in which neural network is implemented, uses μ bits for representation of real numbers. Recall that before applying Algorithm 2 and Algorithm 3, we preprocess the input numbers into finite field that is suitable for crytographic operations. Assume that we truncate the μ-bit numbers by chopping off the lowest ν bits and leaves the new lowest order bit unchanged. The precision error ratio can be bounded by $\epsilon=2^{\mu-\nu}$.

2) Error in Feeding-forward stage: In the feed forward stage of Algorithm 1, since only Algorithm2 is applied once, the error ratio bound introduced by number conversion for cryptographic operations is ϵ.

3) Error in Output-layer Delta: In Step (2.1) of Algorithm 1, $\Delta_1 w_{ij}^o=(o_{i1}-t_i)h_{j1}+r_{11}+r_{21}$, in which o_{i1} and h_{j1} are already approximated in preceding operations. Therefore, the error ratio bound for $\Delta_1 w_{ij}^o$ is $(1+\epsilon)^2-1$. We can obtain the same result for $\Delta_2 w_{ij}^o$.

4) Error in Hidden-layer Delta: In Step (2.2) of Algorithm 1, Algorithm 3 is applied 4 times sequentially. The error ratio bound for $\Delta_1 w_{jk}^h$ and $\Delta_2 w_{jk}^h$ is $(1+\epsilon)^4-1$.

5) Error in Weight update: In Step (3) of Algorithm 1, the update of weights introduces no successive error.

TABLE I
DATASETS AND PARAMETERS

Dataset	Sample	Class	Architecture	Epochs	Learning Rate
kr-vs-kp	3196	2	$36-15-1$	20	0.1
Iris	150	3	$4-5-3$	80	0.1
diabetes	768	2	$8-12-1$	40	0.2
Sonar	104	2	$60-6-2$	150	0.1
Landsat	6435	6	$36-3-6$	12	0.1

VI. EVALUATION

In this section, we perform experiments to measure the accuracy and overheads of our algorithms. We have two sets of experiments on the accuracy. In the first set, we compare the testing error rates in privacy-preserving and non-privacy-preserving cases. In the second set, we distinguish two types of approximation introduced by our algorithms: piece-wise linear approximation of sigmoid function and conversion of real numbers to fixed-point numbers when applying cryptographic algorithms, and analyze how they affect the accuracy of the back-propagation neural networks model. Our experiments on overheads cover the computation and communication costs as well as comparisons with an alternative solution using general purpose secure computation.

A. Setup

The algorithms are implemented in C++ and compiled with g++ version 3.2.3. The experiments were executed on a Linux (Red Hat 7.1) workstation with dual 1.6 GHz Intel processors and 1 Gb of memory. The programs used GNU Multiple Precision Arithmetic Library in implementation of ElGamal scheme. The results shown below are the average of 100 runs.

The testing datasets are from UCI data set repository [36]. We choose a variety of datasets, kr-vs-kp, Iris, Pima-indian-diabetes (diabetes), Sonar and Landsat with different characteristics, in the number of features, the number of labeled classes, the size of datasets and data distributions. Different neural network models are chosen for varying datasets. Table I shows the architecture and training parameters used in our neural network model. We choose the number of hidden nodes based on the number of input and output nodes. This choice is based on the criteria of having at least one hidden unit per output, at least one hidden unit for every ten inputs, and five hidden units being a minimum. Weights are initialized as uniformly random values in the range [-0.1, 0.1]. Feature values in each dataset are normalized between 0 and 1. The privacy preserving back-propagation neural networks have the same parameters as the non-privacy-preserving version. For privacy preserving version, we use the key length of 512 bits.

B. Accuracy loss

First we measure the accuracy loss of our privacy preserving algorithms when the neural network is trained with a fixed number of epochs (shown in Table I). The number of epochs set is based both on the number of examples and on the parameters (i.e., topology) of the network. Specifically, we use 80 epochs for small problems involving fewer than 250 examples; 40 epochs for the mid-sized problems containing

TABLE II
TEST ERROR RATES COMPARISON

Dataset	Non-privacy-preserving Version	Privacy-preserving Algorithm 1
kr-vs-kp	12.5%	15.5%
Iris	14.17%	19.34%
Pima-indian-diabetes	34.71%	38.43%
Sonar	18.26%	21.42%
Landsat	4.12%	5.48%

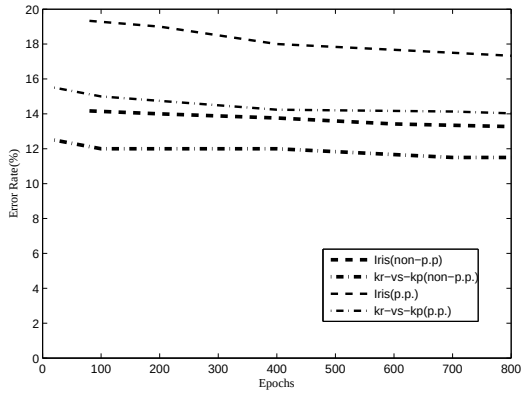

Fig. 1. Error Rates on Training Epochs

between 250 to 500 examples; and 20 epochs for larger problems.

Table II shows testing set error rates for both non-privacy-preserving back-propagation neural network and privacy preserving back-propagation neural networks.

Since the numbers of training epochs are fixed, the global error minimum may not be achieved when the training ends. That explains the relatively high error rates for both privacy-preserving training and the non-privacy-preserving case. From Table II, we can see that for experiments on different datasets, the increase of test error rates by privacy-preserving algorithm remain in small range, 1.26% for Landsat to 5.17% for Iris.

We extend the experiments by varying the number of epochs and evaluate the accuracy of privacy preserving back-propagation neural networks on different training epochs. For clarity of presentation, we only show the results of dataset Iris and kr-vs-kp in Figure 1. The result of other datasets have the similar characteristics but with different epochs scales. From Figure 1, we can see clear that the error rates are decreasing when the number of epochs increases. Furthermore, the error rates of privacy preserving back-propagation network decrease faster than the standard algorithm, which means increasing the number of epochs can help to reduce the accuracy loss.

C. Effects of two types of approximation on accuracy

In this set of experiments, we aim to analyze the causes of accuracy loss in our privacy preserving neural network learning algorithms.

Recall that we have two types of accuracy loss, introduced by sigmoid function approximation and mapping real numbers to fixed point representation respectively. We distinguish and evaluate the effects of these two approximation types by performing a back-propagation learning on approximated piecewise linear sigmoid function, without cryptographic operations (we call it sigmoid approximation test). This will eliminate the security of the learning, but note that the purpose of this modification is to measure the two kinds of accuracy loss, and thus we have to separate them.

TABLE III
ERROR RATES BY DIFFERENT APPROXIMATIONS

Dataset	Non-P.P.	Sigmoid Apx.	Algorithm 1
kr-vs-kp	11.3%	12.2%	13.98%
Iris	14.10%	14.80%	17.04%
Pima-indian-diabetes	33.11%	34.98%	37.28%
Sonar	18.01%	18.85%	21.02%
Landsat	4.10%	4.56%	5.48%

Table III displays the training error rates and testing error rates comparison of backpropatation learning without privacy concern, versus sigmoid approximation test and privacy preserving learning with two types of approximations. In this set of experiments, we make the training process stop when the error is less than the error tolerance threshold 0.1 and if the the number of epochs reaches the maximum number of 1000 before converging, the training also stops. We call the latter case a failure case of training.

From Table III, we can observe that both of the approximation types cause a certain amount of accuracy loss. The approximation brought by piecewise linear sigmoid function is less significant than by real numbers conversion to fixed-point numbers. For example, in testing of dataset Iris, sigmoid function approximation only contributes 0.70% out of 2.96% in the total accuracy loss while, while conversion of real numbers to fixed-point numbers causes the remaining accuracy loss, which is 2.26%.

D. Computation and Communication overhead

We now examine the computation and communication overhead of our algorithm to verify that it is light-weight and practical. In particular, we measure and record the overall time for privacy computation and the time for communication between party A and B in the entire training process.

Since we are the first to study privacy preserving neural network learning with vertically partitioned data, no other complete algorithm is now available for this problem. Consequently, in order to demonstrate that our algorithm is efficient among possible cryptographic privacy preserving solutions, we consider a modification of Algorithm 1, in which we replace the calls to Algorithms 2 and 3 with executions of Yao's general-purpose two party secure computation protocol [9]. (We utilize the Fairplay secure function evaluation system [37] in our implementation of Yao's protocol.) In the setting described in Section VI-A, we compare our original Algorithm 1 with this modified algorithm in terms of computation and communication overheads.

Table IV shows the computation and communication overhead measurements (in minutes) of our algorithm and the modified algorithm. It is clear that, in experiments on the

TABLE IV
COMPUTATION OVERHEAD AND COMMUNICATION OVERHEAD

Dataset	Our Algorithm		Modified Algorithm	
	Comp.	Comm.	Comp.	Comm.
kr-vs-kp	63.49	39.20	1778.29	443.43
Iris	10.94	6.23	310.62	75.19
diabetes	24.19	14.24	628.32	169.33
Sonar	317.43	197.37	8868.04	2193.13
Landsat	8.14	4.92	211.92	56.98

TABLE V
COMPUTATION OVERHEAD AND COMMUNICATION OVERHEAD COMPARISON WITH [27] AND [28]

Overhead	Our Algoirthm	Protocol in [27]	Protocol in [28]
Computation	63.49	> 1057.10	> 111.65
Communication	39.20	> 219.24	> 71.38

five different datasets, our algorithm is significantly more efficient than the algorithm integrated with Yao's protocol. More precisely, our algorithm is $25.97 - 28.39$ times more efficient in terms of computation overhead, and $11.11 - 12.06$ times more efficient in terms of communication overhead.

Now we compare the computation and communication overhead of our work with [27] and [28]. As we have mentioned, the objectives of [27] and [28] are different from ours; the authors of [27] and [28] have not implemented their protocols; neither have they performed any experiment. Consequently, in order to compare their overhead with ours, we have to implement these protocols by ourselves and measure their overheads.

Nevertheless, for practical purposes, we do not implement and measure the complete protocols of [27] and [28]. The reason is that these protocols are so slow in practice that even *parts of them* have significantly more overheads than ours. So measuring the overheads of such parts is sufficient to demonstrate the better efficiency of our algorithm. Furthermore, if we measure the overheads of the complete protocols, [27] and [28] (especially [27]) will take more time than we can afford.

Hence, we have implemented a key compnent of both [27] and [28], namely secure evaluation of the activation function. When we measure the overheads of [27] and [28], we count the accumulative overhead of secure activation function evaluations only, and ignore the overhead of all other operations. We compare the measured partial overhead of [27] and [28] with the total overhead of our own algorithm, which includes the computation and communication overheads of *all* components. Table V shows the comparison results when we train the neural network using dataset kr-vs-kp. We put the symbol "$>$" before the measured partial overheads of [27] and [28], to emphasize that these are not the total overheads. As we can see, for both [27] and [28], the measured partial overheads are already significantly more than the total overhead of our protocol, either in terms of computation or in terms of communication. Therefore, we can safely claim that our algorithm is more efficient than the protocols in [27] and [28].

VII. CONCLUSION AND FUTURE WORK

In this paper, we present a privacy preserving algorithm for back-propagation neural network learning. The algorithm guarantees privacy in a standard cryptographic model, the semi-honest model. Although approximations are introduced in the algorithm, the experiments on real world data show that the amount of accuracy loss is reasonable.

Using our techniques, it should not be difficult to develop the privacy preserving algorithms for back-propagation network learning with three or more participants. In this paper we have considered only the back-propagation neural network. A future research topic is to extend our work to other types of neural networks training.

REFERENCES

[1] HIPPA, National Standards to Protect the Privacy of Personal Health Information, http://www.hhs.gov/ocr/hipaa/finalreg.html.

[2] M. Chicurel, Databasing the Brain, *Nature*, 406, pp. 822-825, 24 August 2000.

[3] Editorial. Whose scans are they, anyway?, *Nature*, 406, 443, 3 August 2000.

[4] D. E. Rumelhart, B. Widrow, and M. A Lehr, The basic ideas in neural networks, *Communications of the ACM,* 37, 87-92, 1994.

[5] R. P. Lippmann, An introduction to computing with neural networks, *IEEE Acoustics Speech and Signal Processing Magazine*, vol. 4, pp. 4-22, 1987.

[6] L. Wan, W. K. Ng, S. Han, V. C. S. Lee, Privacy-preservation for gradient descent methods, in *Proc. of ACM SIGKDD*, 2007, pp. 775-783.

[7] O. Goldreich, Foundations of Cryptography, Volumes 1 and 2, Cambridge University Press, 2001-2004.

[8] D.J. Myers and R.A. Hutchinson, Efficient implementation of piecewise linear activation function for digital VLSI neural networks, *Electron. Letter*, 1989, pp. 1662-1663.

[9] A. Yao. How to Generate and Exchange Secrtes, in *Proc. of the 27th IEEE Symposium on Foundations of Computer Science*, 1986, pp. 162-167.

[10] T. ElGamal, A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, *IEEE Trans.Information Theory*, vol. IT-31, pp. 469-472, no. 4, 1985.

[11] D. Agrawal and R. Srikant. Privacy preserving data mining, In *Proc. ACM SIGMOD*, 2000, pp. 439-450.

[12] Y. Lindell and B. Pinkas. Privacy preserving data mining, in *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO 00)*, vol. 1880 of Lecture Notes in Computer Science, pp. 36-44, Santa Barbara, Calif, USA, August 2000.

[13] N. Zhang, S. Wang, W. Zhao, A New Scheme on Privacy-Preserving Data Classification, In *Proc. of the ACM SIGKDD*, 2005, pp. 374-383.

[14] G. Jagannathan and R. N. Wright. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In *Proc. ACM SIGKDD*, 2005, pp. 593-599.

[15] J. Vaidya and C. Clifton. Privacy-preserving k-means clustering over vertically partitioned data, In *Proc. ACM SIGKDD*, 2003, pp. 206-215.

[16] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke, Privacy preserving mining of association rules. In *Proc. ACM SIGKDD*, 2002, pp. 217-228.

[17] Y. Lindell and B. Pinkas, Privacy preserving data mining, *Journal of Cryptology*, vol. 15(3):177-206, 2002.

[18] Wenliang Du and Zhijun Zhan. Using randomized response techniques for privacypreserving data mining. In *Proceeding of SIGKDD'03*, pages 505?10. ACM, 2003.

[19] J. Vaidya and C. Clifton. Privacy preserving naive Bayes classifier for vertically partitioned data, In *Proc. SIAM International Conference on Data Mining*, 2004.

[20] R. Wright and Z. Yang. Privacy-preserving Bayesian network structure computation on distributed heterogeneous data. In *Proc. 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 713? 718. ACM, 2004.

[21] Keke Chen and Ling Liu. Privacy preserving data classification with rotation perturbation. In *Proceeding of ICDM'05*, pages 589-592. IEEE Computer Society, 2005.

[22] Hwanjo Yu, Xiaoqian Jiang, and Jaideep Vaidya. Privacy-preserving SVM using nonlinear kernels on horizontally partitioned data. In *Proceeding of SAC'06*, 2006.
[23] S. Laur, H. Lipmaa, and T. Mielikaihen, Cryptographically private support vector machines, in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD 06)*, pp. 618-624, ACM Press, Philadelphia, Pa, USA, August 2006.
[24] J. Vaidya and C. Clifton. Privacy-preserving k-means clustering over vertically partitioned data, In *Proc. ACM SIGKDD*, 2003, pp. 206-215.
[25] A. C. Yao. Protocols for secure computations. in *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science,* pp. 160?64, Chicago, Ill, USA, November 1982.
[26] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game, In *Proc. Annual ACM Conference on Theory of Computing*, 1987, pp. 218-229.
[27] Y. C. Chang and C. J. Lu, Oblivious polynomial evaluation and oblivious neural learning, In *Proc. Asiacrypt*, 2001, pp. 369-384.
[28] M. Barni, C. Orlandi, and A. Piva, A Privacy-Preserving Protocol for Neural-Network-Based Computation, in *Proceeding of the 8th workshop on Multimedia and security*. New York, NY, USA: ACM Press, 2006, pp. 146?51.
[29] D. E. Rumelhart, G. E. Hinton and R. J. Williams, Learning internal representations by error propagation, *Parallel Distributed Processing* Cambridge, MA: The MIT Press, 1986, pp. 318-362.
[30] Z. Yang, S. Zhong, and R. Wright. Privacy-preserving classification of customer data without loss of accuracy. In *Proc. 5th SIAM International Conference on Data Mining (SDM)*. SIAM, 2005
[31] Dan Boneh, The Decision Diffie-Hellman Problem, ANTS 1998, pp. 48-63.
[32] Y. Desmedt and Y. Frankel, Threshold Cryptosystems In *Proc. CRYPTO*, 1989, pp. 307–315.
[33] J. Vaidya and C. Clifton, Privacy Preserving Association Rule Mining in Vertically Partitioned Data," in *Proc. of SIGKDD'02*, 2002, pp. 639-644.
[34] B. Goethals, S. Laur, H. Lipmaa, and T. Mielik, On Private Scalar Product Computation for Privacy-Preserving Data Mining, in *Proc. the 7th Annual International Conference on Information Security and Cryptology*, 2004, pp. 104-120.
[35] O. Goldreich, Foundations of Cryptography, Volume 2, Cambridge University Press, 2003.
[36] C.L. Blake and C.J. Merz, UCI Repository of machine learning databases, http://www.ics.uci.edu/ mlearn/MLRepository.html, Irvine, CA: University of California, Department of Information and Computer Science, 1998.
[37] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella, Fairplaya secure two-party computation system. in *Proceedings of the 13th conference on USENIX Security Symposium*, pp. 20-20, San Diego, CA, 2004.
[38] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In 31th Annual Symposium on Theory of Computer Science (STOC), pages 245-254, Atlanta, GA, May 1-4 1999.

Tingting Chen received her B.S. and M.S. degrees in computer science from the department of computer science and technology, Harbin Institute of Technology, China, in 2004 and 2006 respectively. She is currently a Ph.D. candidate at the department of computer science and engineering, the State University of New York at Buffalo, U. S. A. Her research interests include data privacy and economic incentives in wireless networks.

Sheng Zhong is an assistant professor at the computer science and engineering department of the State University of New York at Buffalo. He received his BS (1996), ME (1999) from Nanjing University, and PhD (2004) from Yale University, all in computer science. His research interests include privacy and incentives in data mining and databases, economic incentives in wireless networks.