Privacy Preserving Back-Propagation Neural Network Learning Over Arbitrarily Partitioned Data

Ankur Bansal Tingting Chen Sheng Zhong
Computer Science and Engineering Department
State University of New york at Buffalo
Amherst, NY 14260, U. S. A
Email : {abansal5, tchen9, szhong}@cse.buffalo.edu

***Abstract*—Neural Networks have been an active research area for decades. However, privacy bothers many when the training dataset for the neural networks is distributed between two parties, which is quite common nowadays. Existing cryptographic approaches such as secure scalar product protocol provide a secure way for neural network learning when the training dataset is vertically partitioned. In this paper we present a privacy preserving algorithm for the neural network learning when the dataset is arbitrarily partitioned between the two parties. We show that our algorithm is very secure and leaks no knowledge (except the final weights learned by both parties) about other party's data. We demonstrate the efficiency of our algorithm by experiments on real world data.**

***Index Terms*—Privacy, Arbitrary Partitioned Data, Neural Network**

I. Introduction

Neural Networks have been an active research area for decades. Trained neural networks can predict efficient outputs which might be difficult to obtain in the real world. The expansion of internet and world wide web has made it easier to gather data from many sources [5], [10]. Training neural network from the distributed data is common: for example, making use of data from many hospitals to train the neural network to predict a certain disease, collecting datasets of purchased items from different grocery stores and training neural network from those to predict a certain pattern of purchased items. When training neural network from distributed data, privacy is a major concern.

With the invention of new technologies, whether it is data mining, in databases or in any networks, resolving privacy problems has become very important. Because all sorts of data is collected from many sources, the field of machine learning is equally growing and so are the concerns regarding the privacy. Data providers for machine learning are not willing to train the neural network with their data at the expense of privacy and even if they do participate in the training they might either remove some information from their data or can provide false information. Recent surveys [5] from web users conclude that huge percentage of people are concerned about releasing their private data in any form to the outside world. HIPAA, Health Insurance Portability and Accountability Act rule [13] prohibits to use individuals' medical records and other personal health related information for personal or distribution uses. Even the insurance companies have to take permission to disclose anyone's health related data [11]. So whenever there is distributed data for machine learning, privacy measures are must.

Correspondence Author: Sheng Zhong, Department of Computer Science and Engineering, State University of New York at Buffalo, Amherst, NY 14260, U. S. A. Phone: +1-716-645-3180 ext. 107. Fax: +1-716-645-3464. Email: szhong@cse.buffalo.edu

The datasets used for neural network training can be collectively seen as a virtual database. In the distributed data scenario this database can be partitioned in many ways. When some rows of the database are with one party and the other party holds the rest of the rows of the database, this is called horizontal partitioned database. In such a case for neural network training this does not pose a significant privacy threat since each data holder can train the network in turns. When some columns of the database are with one party and other party holds the rest of the columns, this is called vertical partitioning of the datasets for training. Chen and Zhong [6] propose privacy preserving algorithm in the neural networks when the training data is vertically partitioned. Their algorithm is efficient and provides strong privacy guarantees. There is yet another category for partitioned data (arbitrary partitioning) which is studied in this paper. To the best of our knowledge the problem of privacy preserving neural network learning over arbitrarily partitioned data has not been solved.

In arbitrary partitioning of data between two parties, there is no specific order of how the data is divided between two parties. Combined data of two parties can be collectively seen as a database. Specifically if we have database D, consisting of n rows $\{DB_1, DB_2, \cdots, DB_n\}$, and each row DB_i (i goes from 1 to n) contains m attributes, then in each row, A holds a subset DB_i^A of j attributes and B holds a subset DB_i^B of k attributes (where k=m-j) such that $DB_i = DB_i^A \cup DB_i^B$ and $DB_i^A \cap DB_i^B = \emptyset$. In each row, the number of attributes in two subsets can be equal (j=k) but does not have to be equal that is,(j $\neq$ k). It might happen that j=m which means that A completely holds that row, in rows where j=0, B completely holds that row.

In this paper we propose a privacy preserving algorithm for back-propagation neural network learning when the data is arbitrarily partitioned. Our contributions can be summarized as follows. (1) To the best of our knowledge we are the first to propose privacy preserving algorithm for the neural networks when the data is arbitrarily partitioned. (2) Our algorithm is

quite efficient in terms of computational and communication overheads. (3) In terms of privacy, our algorithm leaks no knowledge about other's party data except the final weights learned by the network at the end of training.

The rest of the paper is organized as follows. Section II describes the related work. In Section III, we introduce the technical preliminaries including definitions, notations and problem statement. In Section IV, we present the privacy preserving algorithm for the back propagation neural network learning when the data is arbitrarily partitioned. We show computation and communication overhead analysis of our algorithm in Section V. In Section VI, we verify the accuracy and efficiency of our algorithm by experiments on real world data. In the end we conclude our paper.

II. Related Work

Privacy preserving neural network learning has been studied in [6], [3], [26]. Barni et al. [3] proposed security algorithms for three scenarios in neural networks. (a) When the data is being held by one party and network parameters (weights) are being held by the other (b) When in addition to the weights, the other party wants to preserve activation function also (c) When the other party wants to preserve the network topology. Their work is limited to the extent that only one party holds the data and the other holds the parameters of the network. Distributed data scenario is not discussed in their paper. Chen and Zhong [6] propose privacy preserving back-propagation neural network learning algorithm when training data is vertically partitioned. Their algorithm provides strong privacy guaranty to the participants. The solution when the training data is horizontally partitioned data is much easier since all the data holders can train the neural network in turns. In this paper we address the problem when the training data for the neural network is arbitrarily partitioned (defined below) between two parties. We will use secure scalar product algorithm [27] and algorithm 3 of [6] in our algorithm so that both parties just have random shares after each round of training without each party knowing the other's party data.

Privacy preserving algorithms have also been investigated in data mining when the data to be mined is distributed among different parties. For data mining, Agrawal et al. [1] proposed randomization to preserve sensitive data at the cost of accuracy. In order to preserve data accuracy, Lindell and Pinkas [19] introduced cryptographic tools for privacy preservation but the computation complexity increases with huge data. Clifton et al. used commutative encryption property to preserve privacy for the associative rule mining when the data is either horizontally [23] or vertically partitioned [17]. Jagannathan and Wright introduced privacy preserving k-means clustering algorithm to cluster datasets when the data is arbitrarily partitioned[14]. There are many more privacy preserving data mining algorithms. However, none of these algorithms can be used directly in the problem of privacy preserving neural network learning when the training data is arbitrarily partitioned between two parties. In this paper we present privacy preserving back-propagation neural network learning algorithms when the data is arbitrarily partitioned between two parties so that no party is able to learn anything about other's party data except the final weights learned by the network.

There is also a general-purpose technique in cryptography, called secure multi-party computation that can be applied to privacy preserving neural network learning problems. In particular, the protocol proposed by Yao in [28] can privately compute any probabilistic polynomial function. Secure multi-party computation can theoretically solve all problems of privacy-preserving computation. However, it is very costly to be applied when it comes to practical problems [12]. Furthermore, in scenarios in which nerual networks is applied, usually parties hold huge amounts of data. Therefore, this general solution is especially infeasible to our problem.

III. Technical Preliminaries

In this section we present the problem definition, notations used, and an overview of the algorithm we propose to preserve privacy in neural network training from arbitrarily partitioned data between two parties.

A. Definitions

In this section we briefly describe the concept of arbitrarily partitioned data and an overview of problem statement.

- **Arbitrary Partitioned Data**: We consider arbitrary partitioning of data between two parties in this paper. In arbitrary partitioning of data between two parties, there is no specific order of how the data is divided between two parties. Combined data of two parties can be seen as a database. Specifically if we have a database D, consisting of n rows $\{DB_1, DB_2, \cdots, DB_n\}$, and each row DB_i (i goes from 1 to n) contains m attributes, then in each row, DB_i^A is the subset of attributes held by A (say j is the number of attributes in the subset DB_i^A) and DB_i^B is the subset of attributes held by B (say k is the number of attributes in the subset DB_i^B) such that DB_i = $DB_i^A \cup DB_i^B$ and $DB_i^A \cap DB_i^B = \emptyset$. In each row the number of attributes in two subsets can be equal (j=k) but does not have to be equal that is, (j $\neq$ k). It might happen that j=m which means that A completely holds that row or j=0 which means B completely holds that row. In general, arbitrary partitioning is a more general case of combinations of many horizontal and vertical partitions of a database.
- **Problem Definition**: When the training data for the neural networks is arbitrarily partitioned between two parties, both parties want to train the network but at the same time they do not want that the other party should learn anything about its data except the final weights learned by the network. *NOTE*:(We will not talk about the rows whose all attributes are completely owned by one party. This is trivial, since the party who holds this row can independently train the network with its data without revealing others anything about the data). So we propose a privacy preserving back-propagation neural network learning algorithm for the arbitrarily partitioned data between two parties.

B. Notations

We consider a 3-layer (a-b-c configuration) neural network in the paper but our work can easily be extended to any N-layer neural network.

- The input vector is denoted as $\{x_1, x_2, \cdots, x_n\}$ where any x_i (i goes from 1 to n) is an input to the input node of the neural network. In the paper we consider that two parties (A and B) hold arbitrary partitioned data. This can be extended to n-party partitioning but we leave this for our future work. As discussed, the two parties share the arbitrarily partitioned data such that for every object[1], if an object vector of n-dimensions (n attributes) is denoted as $x_1, x_2, \cdots, x_n$ Party A holds $x_{1_A}, x_{2_A}, \cdots, x_{n_A}$ and Party B $x_{1_B}, x_{2_B}, \cdots, x_{n_B}$ such that *for every object of a virtual database*

$$x_{1_A} + x_{1_B} = x_1$$

$$x_{2_A} + x_{2_B} = x_2$$

  and so on ..
  We require that for every attribute in $\{x_1, x_2, \cdots, x_n\}$ for every object, either x_{i_A} or x_{i_B} (i goes from 1 to n) is 0. This means that, that attribute is being completely held by the other party.
- We assume the values of hidden nodes to be $h_1, h_2, \cdots, h_n$and the values of output nodes to be $o_1, o_2, \cdots, o_n$.
- **Network Parameters**: $w^h{}_{jk}$ 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≤k≤a; 1≤j≤b; 1≤i≤c. Here a denotes the number of input nodes, b the hidden nodes and c denotes the output nodes.

C. Algorithm Overview

It is highly important that not only the data but the intermediate weights also should not be revealed to the other party because intermediate weights contain partial knowledge about the data. We propose an algorithm in which both parties modify the weights and hold random shares of the weights during the training. Both the parties use the secure 2-party computation [27] and algorithm 3 of [6] to calculate the random shares of the weights between the training rounds. Specifically, if $x_{1_A}, x_{2_A}, \cdots, x_{n_A}$ is an object held by A where x_{i_A} (i varies from 1 to n) is an attribute in the row held by A and $x_{1_B}, x_{2_B}, \cdots, x_{n_B}$ is an object held by B where x_{i_B} (i varies from 1 to n) is an attribute in the row held by B, the algorithm starts modifying weights till they reach a target value t(x), where $x = \sum_{i=1}^{n} x_{i_A} + x_{i_B}$ and t(x) is any function. Both the parties calculate the activation function using [6] and they use secure 2-party computation algorithm [27] and algorithm 3 of [6] to calculate random shares of the weights in the proposed algorithm.

[1]object corresponds to a row in database in this paper

D. Security Model

In this paper, we assume semi-honest model which is a standard security model in many privacy preserving papers [19], [30]. Semi-honest model requires that all parties follow the protocol but any party might try to learn some information from the intermediate results. So our aim is that no knowledge about each party's data (except the final weights learned by the network) is leaked in this model.

E. ElGamal scheme and Homomorphic Encryption

ElGamal Encryption scheme [8] and homomorphic property of the scheme is used in our algorithm. Homomorpic property is a property of certain encryption algorithms where specific algebraic operations (multiplication) can be performed on plaintext by performing the operations on encryption messages without actually decrypting them. For example say we have two messages m_1 and m_2, the encryption of message is denoted by $E(m_1)$ and $E(m_2)$ then operation $m1m2$ can be performed using $E(m_1)$ and $E(m_2)$ only without actually decrypting the two messages. Specifically, for ElGamal scheme, we have

$$E(m_1 \cdot m_2) = E(m_1) \cdot E(m_2). \tag{1}$$

This property of encryption is being used in secure scalar product algorithm [27].

IV. Privacy Preserving Neural Network Learning

In this section we present the privacy preserving back-propagation neural network learning algorithm over arbitrary partitioned data between two parties.

NOTE: In this algorithm after each round of training both the parties just hold the random shares of weights and not the exact weights, this guarantees more security and privacy against the intrusion by the other party. It is only at the end of the training that both the parties know the actual weights in the neural networks.

The error function which is used to calculate whether the output is desired or not is given by:

$$e = \frac{1}{2}\sum_i (t_i - o_i)^2$$

where i varies from 1 to n (number of outputs) .

If the value of this error function does not satisfy the output requirements we have some more rounds of training , repeating the algorithm again. This error is propagated backwards and require change in weights according to the equations:

$$\frac{\partial e}{\partial w^o_{ij}} = -(t_i - o_i)h_j \tag{2}$$

$$\frac{\partial e}{\partial w^h_{jk}} = -h_j(1 - h_j)x_k \sum_{i=1}^{N}[(t_i - o_i)w^o_{ij}] \tag{3}$$

The owner of the network assigns random weights to the neural networks in the beginning of training. We will just explain for one object, for rest of the objects it is same and self explanatory. Let us assume party A holds $x_{1_A}, x_{2_A}, \cdots, x_{n_A}$

where any x_{i_A} is an attribute in the row held by A and party B holds $x_{1_B}, x_{2_B}, \cdots, x_{n_B}$ where any x_{i_B} is an attribute in the row held by B. For each input node i (i varies from 1 to n) of the neural network, party A holds x_{i_A} and party B holds x_{i_B} such that

$$x_{i_A} + x_{i_B} = x_i$$

in which either x_{i_A} or x_{i_B} is 0 because only one of the two parties can contain an input corresponding to that input node of the input layer in the neural network.

The target value t(x) is known to both the parties. The aim of the algorithm is to train the network, so as to modify the weights $w^h{}_{jk}$ and $w^o{}_{ij}$ ($w^h{}_{jk}$ denotes the weight connecting the input layer node k and the hidden layer node j and $w^o{}_{ij}$ denotes the weight connecting j and the output layer node i) so that, given the above input distributed dataset between A and B, the output corresponds to nearly the target value.

During training, for each training sample, party A and party B randomly share weights w^h_{jk} and w^o_{ij} after each training round where $w^h_{jk} = w^{h_A}_{jk} + w^{h_B}_{jk}$($w^{h_A}_{jk}$ is the share of party A and $w^{h_B}_{jk}$ is the share of party B) and $w^o_{ij} = w^{o_A}_{ij} + w^{o_B}_{ij}$ ($w^{o_A}_{ij}$ is the share of party A and $w^{o_B}_{ij}$ is the share of party B). At the end of each round of training, each party holds only a random share of each weight. Algorithm 1 describes the feed forward stage.

Algorithm 1 Privacy preserving back-propagation learning algorithm – feed forward stage

→ Party A holds $(x_{1_A}, \cdots, x_{n_A})$, $w^{h_A}_{jk}$ and $w^{o_A}_{ij}$.
→ Party B holds $(x_{1_B}, \cdots, x_{n_B})$, $w^{h_B}_{jk}$ and $w^{o_B}_{ij}$. weight, $w^h_{jk} = w^{h_A}_{jk} + w^{h_B}_{jk}$, $w^o_{ij} = w^{o_A}_{ij} + w^{o_B}_{ij}$.
For each hidden layer node h_j,

1) Using Algorithm 3, party A and B respectively obtain random shares φ_A and φ_B for $\sum_{k=1}^{a}(w^{h_A}_{jk} + w^{h_B}_{jk})(x_{k_A} + x_{k_B})$.
2) Party A and B jointly compute the sigmoid function for each hidden layer node h_j, obtaining the random shares h_{j_A} and h_{j_B} respectively s.t. $h_{j_A} + h_{j_B} = f(\varphi_A + \varphi_B)$ using [6].

For each output layer node o_i,

1) Using Algorithm 3, party A and B respectively obtain random shares o_{i_A} and o_{i_B} for $\sum_{j=1}^{b}(w^{o_A}_{ij} + w^{o_B}_{ij})(h_{j_A} + h_{j_B})$.

In *Algorithm 1* party A and party B compute their random shares φ_A and φ_B from $\sum_{k=1}^{a}(w^{h_A}_{jk} + w^{h_B}_{jk})(x_{k_A} + x_{k_B})$ using secure scalar product algorithm [27] and algorithm 3 of [6]. With the help of [6], they calculate the approximation of sigmoid function for each hidden layer node h_j, obtaining h_{jA} and h_{jB} as their random shares (where h_{jA} is the share held by A and h_{jB} is the share held by B). Then with the help of secure scalar product algorithm again party A and party B calculate $\sum_{j=1}^{b}(w^{o_A}_{ij} + w^{o_B}_{ij})(h_{j_A} + h_{j_B})$ and obtain o_{i_A} and o_{i_B} as their random shares where i depends on the number of output nodes in the neural network. After obtaining the random shares o_{i_A} and o_{i_B}, they follow Algorithm 2 which is the back-error propagation stage. *NOTE:*We do not calculate the error function after every round where both the parties might have to exchange the random shares o_{i_A} and o_{i_B} to calculate the error function. Rather we could fix a certain number of rounds after which they can exchange the output shares to calculate the error function.

Algorithm 2 Privacy preserving back-propagation learning algorithm – back-propagation stage

→ Party A holds $(x_{1_A}, \cdots, x_{n_A})$, t_i, h_{j_A}, o_{i_A}, $w^{h_A}_{jk}$ and $w^{o_A}_{ij}$;
→ party B holds $(x_{1_B}, \cdots, x_{n_B})$, t_i, h_{j_B}, o_{i_B}, $w^{h_B}_{jk}$ and $w^{o_B}_{ij}$.
For each output layer weight w^o_{ij},

1) Using Algorithm 3, Party A and B respectively obtain random shares $\Delta_A w^o_{ij}$ and $\Delta_B w^o_{ij}$ for $(o_{i_A} + o_{i_B} - t_i)(h_{j_A} + h_{j_B})$.

For each hidden layer weight w^h_{jk},

1) Using Algorithm 3, party A and B respectively obtain random shares μ_A and μ_B for $\sum_{i=1}^{c}[o_{i_A} + o_{i_B} - t_i)(w^{o_A}_{ij} + w^{o_B}_{ij})]$.
2) Party A and B respectively obtain random shares κ_A and κ_B, such that $\kappa_A + \kappa_B = (x_{k_A} + x_{k_B})(\mu_A + \mu_B)$, using Algorithm 3.
3) Party A and B securely compute $(h_{j_A} + h_{j_B})(1 - h_{j_A} - h_{j_B})$ by applying Algorithm 3, respectively obtaining random shares ϑ_A and ϑ_B.
4) Using Algorithm 3, party A and B respectively obtain random shares $\Delta_A w^h_{jk}$ and $\Delta_B w^h_{jk}$, for $(\vartheta_A + \vartheta_B)(\kappa_A + \kappa_B)$.

A computes $w^{o_A}_{ij} \leftarrow w^{o_A}_{ij} - \eta(\Delta_A w^o_{ij})$; $w^{h_A}_{jk} \leftarrow w^{h_A}_{jk} - \eta(\Delta_A w^h_{jk})$.
B computes $w^{o_B}_{ij} \leftarrow w^{o_B}_{ij} - \eta(\Delta_B w^o_{ij})$; $w^{h_B}_{jk} \leftarrow w^{h_B}_{jk} - \eta(\Delta_B w^h_{jk})$.

Algorithm 2 is the back-error propagation stage. This stage helps to modify the weights so as to achieve correct weights in the neural network. Both A and B modify their weights according to equation 1 and 2. After some rounds of training both A and B share their outputs to calculate the error function $e = \frac{1}{2}\sum_i (t_i - o_i)^2$, if the error is more, then the two parties A and B have more rounds of training to achieve the target function. Error propagation means that we are trying to modify the values of weights so as to achieve the correct values of weights. In this algorithm, for each output layer weight w^o_{ij}, both parties obtain the random shares of the changes in weights $\Delta_A w^o_{ij}$ and $\Delta_B w^o_{ij}$ (where $\Delta_A w^o_{ij}$ is the share held by A and $\Delta_B w^o_{ij}$ is the share held by B) from equation 1 using the secure scalar product protocol [27] and algorithm 3 of [6] where t_i is the target value of the i^{th} output node of the neural network.

For hidden layer weights w^h_{jk}, we break the equation 2 above into three parts. First of all the two parties calculate random shares μ_A and μ_B from $\sum_{i=1}^{c}[o_{i_A} + o_{i_B} - t_i)(w^{o_A}_{ij} + w^{o_B}_{ij})]$ using [27] and algorithm 3 of [6] (where μ_A is the share held by A and μ_B is the share held by B). With the help of these shares they

calculate random shares κ_A and κ_B using secure scalar product algorithm [27] and algorithm 3 of [6], such that $\kappa_A + \kappa_B = (x_{k_A} + x_{k_B})(\mu_A + \mu_B)$ (where κ_A is the share held by A and κ_B is the share held by B). Then the two parties calculate $(h_{j_A} + h_{j_B})(1 - h_{j_A} - h_{j_B})$ to obtain the random shares ϑ_A and ϑ_B (where ϑ_A is the share held by A and ϑ_B is the share held by B) . Finally they calculate $(\vartheta_A + \vartheta_B)(\kappa_A + \kappa_B)$ obtaining $\Delta_A w^h_{jk}$ and $\Delta_B w^h_{jk}$ as the random shares.

After obtaining random shares

- A computes $w^{o_A}_{ij} \leftarrow w^{o_A}_{ij} - \eta(\Delta_A w^o_{ij})$ and $w^{h_A}_{jk} \leftarrow w^{h_A}_{jk} - \eta(\Delta_A w^h_{jk})$.
- B computes $w^{o_B}_{ij} \leftarrow w^{o_B}_{ij} - \eta(\Delta_B w^o_{ij})$ and $w^{h_B}_{jk} \leftarrow w^{h_B}_{jk} - \eta(\Delta_B w^h_{jk})$.

where η is the network learning rate.

Algorithm 3 Securely computing $(R_A + R_B)(S_A + S_B)$

$\rightarrow$ Party A holds R_A and S_A $\rightarrow$ Party B holds R_B and S_B.

1) Using secure scalar product algorithm [27] and algorithm 3 of [6], party A and B respectively obtain random shares λ_A and λ_B for $R_A S_B$, and random shares γ_A, γ_B for $R_B S_A$.
2) Party A computes $R_A S_A + \lambda_A + \gamma_A$. Party B computes $R_B S_B - \lambda_B - \gamma_B$, such that $(R_A + R_B)(S_A + S_B) = R_A S_A + \lambda_A + \gamma_A + R_B S_B - \lambda_B - \gamma_B$.

Algorithm 3 describes how party A and party B calculate any equation of the form $(R_A + R_B)(S_A + S_B)$. They both use secure scalar party algorithm [27] and algorithm 3 of [6] to calculate $R_A S_B$ and $R_B S_A$. The secure scalar party algorithm is used to calculate the product of two vectors such that at the end of the calculation each party holds a random share of the result so that no party is able to predict the other party's vector.

V. Computation Complexity And Communication Overhead Analysis

In this section we present the computation and communication overhead analysis of our privacy preserving algorithm.

1) **Computation Complexity Analysis**:
   - *Securely computing the scalar product of two vectors* – Let us assume that $t_1$be the time taken by A to generate public-private key pair and to send public key to B. A does n encryptions where n is the number of dimensions in the vector. B does n+1 encryptions. Then B performs (2n+1) multiplications. So the total time taken by the algorithm 4 is $T_1 = t_1 + (2n+1)E + (2n+1)M$ where E is the time taken to encrypt one message and M is the time taken for 1 multiplication.
   - *Securely computing equation of the form* $(R_A + R_B)(S_A + S_B)$ – Since A and B can run the algorithm in parallel, so they obtain random shares for $R_A S_B$ and $R_B S_A$ in T_1time. Party A does 1 multiplication and 2 additions and B does 1 multiplication and 2 subtractions. We assume the time taken for addition and subtraction to be negligible in comparison to multiplication and neglect them . So the total time taken to compute equation of the form $R_A S_B$ and $R_B S_A$ is $T_2 = T_1 + 2M$.
   - *Privacy Preserving Back Propagation learning Algorithm -Feed Forward Stage* – Step 1 of the algorithm 1 takes $aT_2 + bz$ time where $z = (2p+1)C + 2D$ [6] where p is the system approximation parameter, C is the cost of encryption and D is the cost of partial decryption. Step 2 of the algorithm takes bT_2 time where N is the number of hidden and output layers whose number is same in our case. So the total time taken in feed forward stage is $aT_2 + bz + bT_2$.
   - *Privacy Preserving Back propagation learning Algorithm -back propagation stage* – The equation in step 1 of the Algorithm can be rewritten as $(o_{i_A} + o_{i_B} - t_i)(h_{j_A} + h_{j_B}) = (o_{i_A} + o_{i_B})(h_{j_A} + h_{j_B}) - t_i h_{j_A} - t_i h_{j_B}$ So Step 1 of the algorithm for back propagation stage takes $c(T_2 + 2M)$ where M again is the time taken for 1 multiplication. Step 2.1 of the algorithm takes $bc(T_2 + 2M)$ time. Step 2.2 consumes bT_2 time. Step 2.3 can be broken and it takes 2 Multiplications an T_1 time. So step 2.3 takes $b(2M + T_1)$ time. Step 4 takes bT_2 time. So total time taken in back propagation stage is $c(T_2 + 2M) + bc(T_2 + 2M) + 3bT_2$.
2) **Communication Overhead Analysis**:
   - *Communication overhead*– We know, to calculate securely the product of two integers it takes 2n+2 messages between Party A and Party B [6]. For each message being s bits long, each communication takes (2n+2)s bits. In the feed forward stage, for each hidden layer node it needs $b[a(t_1) + t_2]$ bits and for each output layer node it takes $c(bT_1)$ bits where $T_1 = 2s(2n+2)$ and $T_2 = s(2n+2)$. In the back propagation stage of the algorithm, for the first part of the algorithm it needs $c(bT_1)$ bits and for the second part of the algorithm it needs $b[c(bT_1) + 3T_1]$ bits.

VI. Evaluation

In this section we perform our experiments to compare our results of privacy preserving version of the algorithm with non-privacy version of the algorithm to calculate the accuracy losses (defined below). The experiments are carried out on the data from UCI dataset repository [2].

A. Set Up

We have used C++ to implement our algorithm with g++ version 2.8. The experiments are carried out on a Linux operating system (Ubuntu) with 1.9 GHz Intel processors and 2 GB of memory.

The experiments are performed on the real world data from UCI dataset repository [2]. Table 1 shows the training parameters, number of epochs (Number of training rounds), architecture (Number of input nodes, hidden nodes, output

nodes) used in the neural network model. The weights are initialized randomly in the range [-0.1,0.1]. We have trained the neural network using Iris, Dermatalogy, Sonar and Landsat datasets. The attributes from each row of the datasets are randomly divided between two parties (A and B) so that the datasets can be modeled as arbitrarily partitioned between A and B. The network learning rate for each dataset is assumed as 0.2. The number of input nodes for the neural network depends on each dataset and the hidden nodes are chosen such that there are atleast 3 hidden nodes for each output.

The test samples (for each dataset) for the experiments are taken randomly from the datasets only. Specifically, 20 test samples are taken randomly each for Iris and Sonar and 30 each for Dermatalogy and Landsat. The number of epochs are kept small for large datasets like Landsat and large for other datasets. After training (completion of epochs for the respective dataset) each test sample is run against the network to observe whether it is misclassified (belongs to different class) or it belongs to the same class.

B. Experimental Results

The main objective of our experiment is to measure the accuracy loss of our algorithm as a cost of protecting privacy. Accuracy loss is a loss which occurs while applying cryptographic schemes [8] on the non-privacy version of the algorithm to protect each party's data and random shares of the intermediate computations. As our algorithm uses two approximations (describe below), so when we perform our experiments on the non-privacy (When both the parties are not worried revealing their data to the outside world) versus privacy version (When both the parties are worried revealing their data to the outside world) of the algorithm (which uses approximations) accuracy loss takes place. The accuracy loss for each dataset is calculated using the equation

$$AccuracyLoss = T_1 - T_2$$

where T_1 is the Test error rate for Privacy version of the algorithm and T_2 is the Test error rate for Non-Privacy version of the algorithm. Test Error rates for privacy as well as non-privacy version of the algorithm are calculated using the equation given by

$$TestErrorRate = \frac{No.ofTestSamplesMisclassified}{TotalNo.ofTrainingSamples}$$

Cryptographic operations are required whenever there are privacy issues, so accuracy loss is inevitable.

The accuracy loss of a privacy-preserving learning algorithm like ours comes from two approximations. One is the approximation of sigmoid function, and the other is the approximation of real numbers. To calculate the sigmoid function we use [6], but [6] uses piecewise linear approximation of the sigmoid function given by

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

We also map real numbers to the finite fields when applying cryptographic algorithms for ElGamal scheme [8] because cryptographic operations are on discrete finite fields. These are the two approximations introduced in our privacy preserving version of the algorithm.

Table I
DATASETS AND PARAMETERS

Dataset	Sample	Class	Architecture	Epochs	Learning Rate
Iris	150	3	$4-5-3$	80	0.2
Dermatalogy	366	6	$34-3-6$	80	0.2
Sonar	208	2	$60-6-2$	125	0.2
Landsat	6435	6	$36-3-6$	8	0.2

Table II
TEST ERROR RATES COMPARISON

Dataset	Non-privacy-preserving Version	Privacy-preserving Algorithm
Iris	20.00%	25.00%
Dermatalogy	36.66%	43.33%
Sonar	35.00%	40.00%
Landsat	23.33%	26.66%

Table 2 shows the results carried out on non-privacy versus privacy version of the algorithm. Accuracy loss is unavoidable since cryptographic operations are on discrete finite fields. Because we have fixed the number of epochs in the beginning of training, the minimum testing error might not be achieved. But as can be seen, the accuracy loss varies between 3.33% for Landsat to 6.67% for Dermatalogy. Since the accuracy loss is within limits, our algorithm is quite effective in learning these real world datasets.

VII. CONCLUSION

In this paper, we present a privacy preservation back propagation neural network training algorithm when the training data is arbitrarily partitioned between two parties. We assume a semi-honest model and our algorithm is quite secured as the intermediate results are randomly shared between the two parties. The experiments we perform on the real world data show that the amount of accuracy losses are within limits.

REFERENCES

[1] Agrawal, D., & Srikant, R. (2000). Privacy preserving data mining, *In Proc. ACM SIGMOD*, 439-450.

[2] Blake, C.L., & Merz, C.J. (1998). 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.

[3] Barni, M., Orlandi, C., & Piva, A. (2006). A Privacy-Preserving Protocol for Neural-Network-Based Computation, in *Proceeding of the 8th workshop on Multimedia and security.* 146-151.

[4] Chang, Y. C., & Lu. C. J. (2001). Oblivious polynomial evaluation and oblivious neural learning, In *Proceedings of Asiacrypt*, 369-384.

[5] Cranor, L. F., Reagle. J., & Ackerman. M. S. (1999). Beyond concern: Understanding net Users attitudes about online privacy. Technical report TR 99.4.3, AT&T Labs-Research, Available from *http://www.research.att.com/library/trs/TRs/99/99.4/99.4.3/report.htm*.

[6] Chen, T., & Zhong, S., (2009). Privacy Preserving Back-Propagation Neural Network Learning, *IEEE Transactions on Neural Networks*, 20(10) 1554 - 1564.

[7] Du, W., Han, Y. S. & Chen, S., (2004). Privacy-preserving multivariate Statistical Analysis :Linear Regression and Classification. *Proceedings of the SIAM International Conference on Data Mining*.

[8] ElGamal, T., (1985). A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, *IEEE Trans.Information Theory*, 31(4). 469-472.

[9] Editorial. (2000). Whose scans are they, anyway?, *Nature*, 406, 443.

[10] Lorrie faith cranor, editor. (1999). Special Issue on *Internet Private.Comm.ACM*. 42(2).

[11] (2001). Standard for privacy of individually identifiable health information. *Federal Register*, 66(40).

[12] Goldreich, O.,Micali, S., & Wigderson, A. (1987). How to play ANY mental game, In *Proceedings of Annual ACM Conference on Theory of Computing*, 218-229.

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

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

[15] Kantarcioglu, M., & Clifton, C. (2002). Privacy-preserving distributed mining of association rules on horizontally partitioned data. In *The ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'02)*.

[16] Kantarcioglu, M., & Vaidya, J. (2003). Privacy preserving naive Bayes classifier for Horizontally partitioned data. In *IEEE Workshop on Privacy Preserving Data Mining.*

[17] Kantarcioglu, M,, & Clifton, C. (2004). Privacy-preserving distributed mining of association rules on horizontally partitioned data. *IEEE Trans. Knowledge Data Eng.*, 16(4).

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

[19] Lindell, Y., & Pinkas, B. (2000). Privacy preserving data mining, in *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, 1880. 36-4.

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

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

[22] Vaidya, J. & C. Clifton. (2002). Privacy Preserving Association Rule Mining in Vertically Partitioned Data, in *Proc. of SIGKDD'02*, 639-644.

[23] Vaidya, J. & C. Clifton. (2002). Privacy preserving association rule mining in vertically partitioned data, in The *Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 639-644.

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

[25] Westin, A. F. (1999). Freebies and privacy: What netusers think. Technical Report, Opinion Research Corporation, July 1999. Available from *http://www.privacyexchange.org/iss/surveys/sr99014.html*.

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

[27] Wright, R., & Yang, Z. (2004). Privacy Preserving Bayesian Network Structure Computation on Distributed Heterogeneous Data, In *Proc. of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 713-718.

[28] Yao, A. C. (1982). Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, 160-64.

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

[30] Yang, Z., Zhong, S., & Wright. R. (2005). Privacy-preserving classification of customer data without loss of accuracy. In *Proc. 5th SIAM International Conference on Data Mining (SDM)*.

Ankur Bansal received his B.Tech degree in Electronics and Communication Engineering from Ambala College of Engineering and Applied Research, India, in 2007. He is currently a Masters candidate at the department of computer science and engineering, The State University of New York at Buffalo, U. S. A. His research interests include data privacy issues and economic incentives in wireless networks.

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.