Secure Multilayer Perceptron Based on Homomorphic Encryption

Reda Bellafqira[1,2(✉)], Gouenou Coatrieux[1,2], Emmanuelle Genin[2], and Michel Cozic[3]

[1] IMT Atlantique, 655 Avenue du Technopole, 29200 Plouzane, France
{reda.bellafqira,gouenou.coatrieux}@imt-atlantique.com
[2] Unit INSERM 1101 Latim, 29238 Brest Cedex, France
emmanuelle.genin@inserm.fr
[3] MED.e.COM, 29470 Plougastel Daoulas, France
mcozic@wanadoo.fr

Abstract. In this work, we propose an outsourced Secure Multilayer Perceptron (SMLP) scheme where privacy and confidentiality of the data and the model are ensured during its training and the classification phases. More clearly, this SMLP: (i) can be trained by a cloud server based on data previously outsourced by a user in an homomorphically encrypted form; its parameters are homomorphically encrypted giving thus no clues about them to the cloud; and (ii) can also be used for classifying new encrypted data sent by the user while returning him the encrypted classification result. The originality of this scheme is threefold: To the best of our knowledge, it is the first multilayer perceptron (MLP) secured homomorphically in its training phase with no problem of convergence. It does not require extra-communications with the user. And, is based on the Rectified Linear Unit (ReLU) activation function that we secure with no approximation contrarily to actual SMLP solutions. To do so, we take advantage of two semi-honest non-colluding servers. Experimental results carried out on a binary database encrypted with the Paillier cryptosystem demonstrate the overall performance of our scheme and its convergence.

Keywords: Secure neural network · Multilayer perceptron · Homomorphic encryption · Cloud computing

1 Introduction

Nowadays, cloud technology allows outsourcing the processing and the storage of huge volume of data, these ones being personal data or data issued from

This work has received a French government support granted to the CominLabs excellence laboratory and managed by the National Research Agency in the "Investing for the Future" program under reference ANR0LABX0701, and to the ANR project INSHARE, ANR15CE1002402.

C. D. Yoo et al. (Eds.): IWDW 2018, LNCS 11378, pp. 322–336, 2019.
https://doi.org/10.1007/978-3-030-11389-6_24

many sources for big data analysis purposes. In healthcare for example, different initiatives aim at sharing medical images and Personal Health Records (PHR) between either health professionals or hospitals with the help of cloud [9]. They take advantage of the medical knowledge, volume of data represents so as to develop new decision making tools based on machine learning techniques. Among such techniques, there is the multilayer perceptron (MLP) method which belongs to Neural Network (NN) family and which is a core element of deep learning methods; methods that are broadly studied and used nowadays. A MLP consists of multiple layers of interconnected perceptrons (see Fig. 2). A perceptron is a classifier that maps its inputs with a vector of weights followed by an activation function. The output of a perceptron is the input of the next perceptron layer. As all machine learning algorithms, MLP works in two distinct ways: the training phase and the classification of new data. In a supervised mode, the training phase aims at inferring the network parameters from a labeled database by optimizing some objective function. Once trained, a MLP scheme is used so as to classify new data.

Despite the attractive benefits provided by MLP, one of the actual limits of its outsourcing in a cloud environment stands on the security of the data used for the training phase or for classification purposes as well as of the MLP parameters. At the same time, the parameter of a process, like those of a trained MLP, may have some important added value for a company. There is thus an interest to develop secured MLP (SMLP) methods that can be trained remotely using outsourced data while respecting data privacy and confidentiality.

Different approaches have been proposed to secure neural network methods. Some of them are based on additive secret sharing that allows several parties to jointly compute the value of a target function $f(.)$ without compromising the privacy of its input data. For instance, [17] presents a privacy preserving NN learning protocol where each participant performs most of the learning computations in the clear domain except the NN weight update which is performed with secret additive sharing (e.g. secure sum and secure matrix addition). One limit of this solution is that, at each iteration, all updated weights are revealed to all participants which may leak information about the training data. To reduce such information leakage, [18] proposes to share through a server only a small fraction of the parameters; parameters on which the NN weights update can be performed using a synchronous stochastic gradient descent (ASGD) instead of a stochastic gradient descent. This method consequently establishes a compromise between accuracy and privacy. Higher the number of shared parameters, better is the classification accuracy but lower is privacy. Recently, in [1], the authors demonstrate that in [18], even a small portion of shared gradients of weights (gradients that are used to update the weight values) can leak useful information about training data. To overcome this problem, they suggest a solution based on homomorphic encryption to secure ASGD. The interest of homomorphic encryption is that it allows performing operations (e.g. $+$, $\times$) onto encrypted data with the guarantee that the decrypted result equals the one carried out with unencrypted data [2–4,6]. In [1], all users: (i) share the same homomorphic cryptosystem pub-

lic and secret keys (ii) train locally a NN over their data (iii) computes their weight gradients and send them in homomorphic encrypted form to the server. When the server receives the encrypted gradients, it updates the weights in the encrypted domain and (iv) sends them back to the users. Since each user has the secret key, they decrypt the weights and train again their NN based on the new weights. In this scheme, homomorphic encryption is just used for the NN update, one computation of the NN learning process. Beyond, due to the fact this solution is based on AGSD [1,17], it suffers of the delayed gradient problem, i.e. a converge issue of the training phase problem [22].

An alternative to these approaches is to train neural networks in the clear domain and by next use them with encrypted data. Most solutions make use of homomorphic encryption. In this work, we are focusing on conducting all the computations of the training phase in the homomorphic encrypted domain. To the best of our knowledge, such an issue has only been theoretically studied in [21] where is shown that NN can be trained using fully homomorphic encryption data and by approximating activation functions with polynomials as homomorphic encryption only allows linear operations.

However, due to the fact fully homomorphic cryptosystems add noise to the data after each multiplication or addition operation. Both the computational complexity and the length of cipher-texts increase with the number of desired operations to guarantee the correct polynomial evaluation. In order to maintain a fixed cipher-text length, a practical implementation requires a de-noising process so as to be feasible or restrict the computation just on low degree polynomials. Moreover, the efficiency of the homomorphic computations depends on the multiplicative depth. To avoid a multiplicative depth too big which increases the computation complexity, after a certain number of iteration, the encrypted updated weights are sent to the parties to be decrypted and re-encrypted. Nevertheless, this solution leads to a higher communication complexity of the scheme.

Beyond this theoretical work, all other proposals [5,8] focus on securing the NN classification phase. For instance, [5] proposes three privacy homomorphic encryption based classifiers: the linear model and two low degree models. In [8] a fully homomorphic convolutional neural networks classifier (CNN) is proposed.

In this paper, we propose a secure multilayer perceptron (SMLP) method, the training and classification procedures of which do not suffer of convergence issues. To do so, we take advantage of the rectified linear unit (ReLU). Beyond its accuracy and its contribution to MLP efficiency [11], ReLU can be secured with homomorphic encryption and two non-colluding semi-honest servers avoiding thus the need to use of an approximation procedure of the perceptron's output as proposed by the above methods. Another originality of our SMLP, is that its output is also encrypted. That is not the case of actual solutions that provide unencrypted output. Furthermore, our SMLP is entirely outsourced in the sense it does not require extra-communications overhead in-between the servers and the user to conduct the training and classification phases. The user just has to send his data homomorphically encrypted to the cloud server that will train the

SMLP or classify data, without the cloud being able to infer information about the SMLP model parameters, the data or the classification result.

The rest of this paper is organized as follows. Section 2 regroups preliminaries related to Multilayer Perceptron and the Paillier cryptosystem on which relies the implementation of our SMLP. We also provide the basic properties and the operations one can implement over Paillier encrypted data when using two non-colluding semi-honest servers. In Sect. 3, we detail our secure multilayer perceptron. Section 4 provides some experimental results conducted to model the "AND" logic function on a binary database, and the security analysis of our proposal. Section 5 concludes this paper.

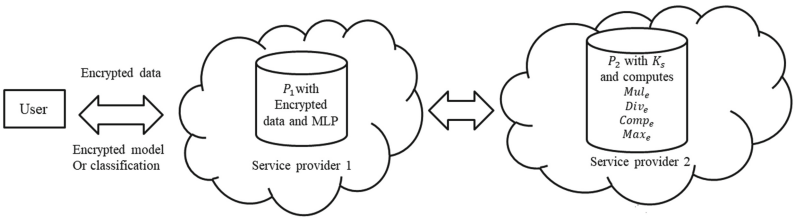

Fig. 1. Secure neural network architecture in a cloud environment

2 Preliminaries on the Paillier Cryptosystem and Multilayer Perceptron

2.1 The Paillier Cryptosystem

Being asymmetric, the Paillier cryptosystem [16] makes use of a pair of private and public keys. Let p and q be two large primes. Let also $K_p = pq$, $\mathbb{Z}_{K_p} = \{0, 1, ..., K_p - 1\}$, $\mathbb{Z}^*_{K_p}$ denotes the set of integers that have multiplicative inverses modulo K_p in $\mathbb{Z}_{K_p}$. We select also $g \in \mathbb{Z}^*_{K_p^2}$ such that

$$gcd(L(g^{\lambda} \mod K_p^2), K_p) = 1 \tag{1}$$

where: $gcd(.)$ is the greatest common divisor function; $\lambda = lcm(p-1, q-1)$ is the private key (K_s), with $lcm(.)$ the least common multiple function; the pair (K_p, g) defines the public key; and, $L(s) = \frac{s-1}{K_p}$. Let $m \in \mathbb{Z}_{K_p}$ the message to be encrypted. Its cipher-text is derived as

$$c = E[m, r] = g^m r^{K_p} \mod K_p^2 \tag{2}$$

where $E[.]$ is the encryption function, $r \in \mathbb{Z}^*_{K_p}$ is a random integer. Since r is not fixed, the Paillier cryptosystem satisfies the so-called "semantic security". More clearly, depending on the value of r, the encryption of the same plain-text message yields to different cipher-texts even though the public encryption key

is the same. The plain-text m is recovered using the decryption function $D[.]$ defined as follow

$$m = D[c, \lambda] = \frac{L(c^{\lambda} mod K_p^2)}{L(g^{\lambda} mod K_p^2)} \mod K_p \tag{3}$$

The Paillier cryptosystem has an additive homomorphic property. Considering two plain-texts m_1 and m_2, then

$$E[m_1, r_1].E[m_2, r_2] = E[m_1 + m_2, r_1.r_2] \tag{4}$$

$$E[m_1, r_1]^{m_2} = E[m_1.m_2, r_1^{m_2}] \tag{5}$$

For the sake of simplicity, in the sequel we denote $E[m, r]$ simply by $E[m]$.

2.2 Operations over Paillier Encrypted Data

As stated above, the Paillier cryptosystem allows implementing linear operations. It can however be used so as to compute multiplications, divisions and comparisons with the help of two non-colluding semi-honest servers P_1 and P_2.

– Multiplication operator in Paillier encrypted domain $Mul_e^{P_1,P_2}(.;.)$
Let us consider two messages a and b and their respective Paillier encrypted versions $E[a]$ and $E[b]$ obtained with the user public key K_p. In order to compute $E[a \times b]$ without revealing any information about a and b, one can take advantage of blinding and of two servers P_1 and P_2. Assuming that P_1 possesses $(E[a], E[b])$, the objective is that P_2 returns $E[a \times b]$ to P_1 while ensuring that no clues about a and b are revealed to P_1 and P_2. Under the hypothesis P_2 knows the user secret key K_s and that it does not collude with P_1, this objective can be reach according to the following procedure we will refer as $Mul_e^{P_1,P_2}(a; b)$:

1. *Data randomization* - P_1 firstly randomizes $E[a]$ and $E[b]$ such that

$$a' = E[a] \times E[r_a] = E[a + r_a] \tag{6}$$

$$b' = E[b] \times E[r_b] = E[b + r_b] \tag{7}$$

where r_a and r_b are two random numbers only known from P_1 and uniformly chosen in $\mathbb{Z}_{K_p}$. Then P_1 sends a' and b' to P_2.

2. *Multiplication computation phase* - On its side, using the user private key K_s, P_2 decrypts a' and b' and multiplies the result

$$M = (a + r_a)(b + r_b) \tag{8}$$

P_2 next encrypts M into $E[M]$ using the user public key K_p and sends it to P_1.

3. *Multiplication denoising* - In order to get $E[a \times b]$, P_1 just has to remove the extra-random factors as follow

$$E[a \times b] = E[M] \times E[b]^{-r_a} \times E[a]^{-r_b} \times E[-r_a \times r_b] \tag{9}$$

– Division operator in Paillier encrypted domain: $Div_e^{P_1,P_2}(.;.)$
Different ways, based on two servers, have been proposed so as to compute the division. The one used in this paper works as follows [19]. Let us consider P_1 has an encrypted message $E[a]$ and that it wants to divide a by d. At this time d can be encrypted or not, that is to say known or unknown from P_1. Again, we don't want P_1 and P_2 to learn details about a. The computation of $E[a/d]$ from $E[a]$ and d is also based on blinding. As above, it is assumed that P_2 possesses the decryption key K_s. Our division operation $Div_e^{P_1,P_2}(a;d)$ is thus a procedure defined as
  1. *Data blinding* - P_1 randomly chooses a number $r \in \mathbb{Z}_{K_p}$ and computes $E[z] = E[a+r] = E[a]E[r]$. P_1 then sends $E[z]$ to P_2.
  2. *Division computation* - P_2 decrypts $E[z]$ with the user private key K_s and computes $c = z/d$. P_2 encrypts the division result $E[c]$ and sends it to P_1.
  3. *Division denoising* - P_1 computes $E[a/d]$ such as:

$$E[a/d] = E[c] \times E[-r/d]. \tag{10}$$

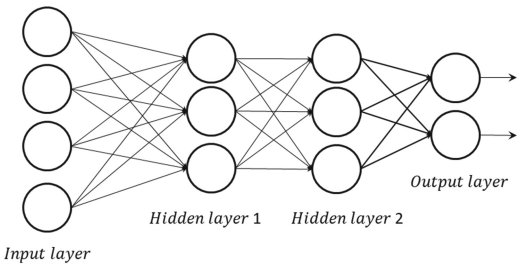

Fig. 2. Example of MultiLayer perceptron (MLP)

2.3 Multilayer Perceptron (MLP)

The common architecture of a MLP is given in Fig. 2. It is constituted of perceptrons organized in different layers: the input and the output layers and, in-between them, a given number of hidden layers (two in the given example of Fig. 2). The first layer takes as input the user data set $X = \{x_n\}_{n=1...N}$ where each vector $x_n = \{x_n(1), x_n(2), ..., x_n(q)\}$ represents the n^{th} training vector with q the number of perceptron inputs. The output layer provides the class of the input data $t_n^{'}$ (i.e. $MLP(x_n) = t_n^{'}$ where $MLP(.)$ refers to the function that computes the output of the network).

A perceptron is a classifier that maps its input to an output value, for example the output of the i^{th} perceptron of the l^{th} layer is given as Z:

$$Z = A_i^l(N_i^l) = A_i^l(\sum_j W_{j,i}^{l-1}.A_j^{l-1}) \tag{11}$$

where $W_{j,i}^{l-1}$ denotes the weight between the j^{th} perceptron in layer $l-1$ and the i^{th} perceptron in layer l. $A_i^l()$ is the activation function of the i^{th} perceptron of the l^{th} layer. Notice that many activation functions have been proposed (e.g. Sigmoid, Tanh, ReLU). In this work, we opted for the rectified linear unit (ReLU) activation function, one of the most used function due to its accuracy and its efficiency [11]. Another reason is that it can be secured by means of homomorphic operators. We will come back in details on this point in Sect. 3. ReLU is defined such as

$$A(y) = \begin{cases} y & \text{if} \quad y \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

To make such a MLP scheme operational, it should be trained so as to find the perceptron weight values. This training can be supervised or unsupervised. In the former case, the classes of data the MLP should distinguished are *a priori* known. Thus to train a MLP scheme, the user provides labeled data $T = \{t_n\}_{n=1...N}$ where N is the size of the training set and t_n indicates the label of the n^{th} training input data x_n. In the second case, the perceptron identifies by itself the different classes of data. The solution we proposed in this work is trained on labeled data.

The supervised training of NN relies on two phases that are iteratively applied: the feed-forward and the back-propagation phases. Before starting the first feed-forward phase, all perceprons' weights are initialized with random values, for instance. Then training data are provided as input to the MLP. By next, the error between the original label and the ones computed by the MLP is calculated using an objective function (called also cost function) (e.g. cross entropy, Mean Square Error, Minkowski distance). This error is then used in the back-propagation phase, so as to update all perceptrons' weights applying gradient descent. Once weights updated, a new feed-forward starts using the same labeled data.

Many solutions have been proposed so as to decide when to stop the training phase [7]. Among these conditions, one can fix a number of iterations ("aka epochs"): the MLP will stop once a number of epochs have elapsed. An alternative stands in thresholding the training set Mean Squared Error (MSE) (i.e. MSE between the training set labels and the MLP outputs). The smaller MSE, the network better performs. Thus the training stops when MSE is smaller than a given threshold value. Instead, it has been proposed to use the Training Set Accuracy; that is the number of correctly classified data over the total number of data in the training set for an epoch. In this work, we opted for a fix number of iterations.

Once a MLP model or scheme trained, i.e. once the perceptrons' weights known, it can be used so as to classify new data. This classification process simply

consists in applying the feed-forward phase with new data as input, considering that the MLP output will give the data class.

3 Secure Multilayer Perceptron

3.1 General Framework and System Architecture

The general framework we consider in this work is given in Fig. 1, where a user outsourced into the cloud Paillier encrypted data; data on which the user wants the cloud service provider to train our secure multilayer perceptron (SMLP). Once trained, this SMLP will be used by the user so as to classify new data. The user will also send it encrypted to the cloud. In our view, the data, the classification data result as well as all the parameters of the SMLP should be unknown from the cloud. As it can be seen in Fig. 1 and as we will see in the sequel, the computations of both the SMLP training and classification phases are distributed over two servers, P_1 and P_2, of two distinct cloud service providers. We consider them as honest but curious [14]. More clearly, they will try to infer information about the data, the classification results as well as about the SMLP parameters. In our scenario, P_1 interacts with the user, stores and handles his data. P_2 cooperates with P_1 so as to conduct some operations (division, multiplication, etc. ...) involved into the training or classification phases of ours MLP.

3.2 Proposed Secure Multilayer Perceptron

Securing a multilayer perceptron consists in implementing the feed-forward and back-propagation phases over encrypted data. The MLP that we propose to secure, both in its learning and classification phase, is based on: (i) perceptrons, the activation function of which ReLU, (ii) the mean squared error (MSE) as cost function. The secure version of this MLP, we describe it in the following, works with the Paillier cryptosystem and takes advantage of the above two servers based system architecture so as to exploit the secure multiplication and division operators depicted in Sect. 2. As we will see, different issues have to be overcome in order to ensure the convergence of such a Secure MLP. In particular, we propose a new "Max" function operator so as to secure ReLU.

Secure MLP Feed-Forward Phase. The feed-forward phase consists in calculating the MLP output for a given input. Based on the fact a MLP is constituted of different layers of perceptrons, securing the feed-forward phase relies on securing each perceptron independently. As seen in Sect. 2, the i^{th} perceptron in the l^{th} layer performs a weighted sum of the input vector $\{A_j^{l-1}(N_j^{l-1})\}_{j=1...S}$ where S is the number of perceptron in the $l-1$ layer (see Eq. (11)), the result of which is provided to an activation function (see Eq. (12)). Considering that all pieces of information provided by the user are Paillier encrypted, i.e. $\{E[A_j^{l-1}(N_j^{l-1})]\}_{i=1...S}$, the weighted sum in the encrypted domain becomes:

$$E[N_i^l] = \prod_{i=0}^{n} Mul_e^{P_1,P_2}(E[W_{j,i}^{l-1}], E[A_j^{l-1}(N_j^{l-1})]) \tag{13}$$

where P_1 and P_2 are the two independent servers (see Fig. 1), $E[N_i^l]$ is the secure weighted sum and $\{E[W_{j,i}^{l-1}]\}_{j=1...n}$ the encrypted perceptron weights which are also confidential.

In this computation as well as in all others, one important constraint to consider stands on the fact that the Paillier cryptosystem only works with plaintext and cipher-text constituted of positive integers in $\mathbb{Z}_{K_p}$. More clearly, all data and parameters of the SMLP should be represented by integers. To overcome this issue, taking as example the input data, these ones are turned into integer values by scaling and quantizing them as follow

$$X = [Qx] \tag{14}$$

where [.] is the rounding function and Q is an expansion or scaling factor. Beyond, even if the SMLP parameters and inputs are integers, their processing may lead to negative values. In order to represent such values in $\mathbb{Z}_{K_p}$, integer values greater than $(K_p+1)/2$ will correspond to negative values and the others to positive values.

By next, the secure perceptron's output is computed by applying a secure version of the ReLU activation function to the encrypted weighted sum $E[N_i^l]$. One key issue to overcome in securing ReLU (see Eq. (12)) stands in the calculation of the function $Max(a, b)$ in-between two integer values a and b in the Paillier encrypted domain. Different solutions have been proposed so as to securely compare encrypted data [4,10,13,20]. Most of them are based on blinding and two non-colluding parties. However, with all these approaches, the comparison result is provided in a clear form. More clearly, if P_1 asks P_2 to compare $E[a]$ and $E[b]$, P_1 will know if $E[a]$ is or not greater than $E[b]$. In our framework, this leads to an information leak. Indeed, P_1 is not authorized to get some information about the SMLP parameters. To solve this problem, the authors of [15] propose a protocol so as to compare two input values in a secure distributed fashion between two or more participants using the Paillier cryptosystem and secret sharing schemes. However, this scheme has several issues: (i) they consider the public key of the Paillier cyrptosystem as a secret key, a concept in contradiction with the one of public key, (ii) they assume that the computation of the multiplicative inverse of a ciphertext is possible without the knowledge of the public key, which mathematically speaking is not possible. Indeed the inverse is defined in the multiplicative group $\mathbb{Z}_{K_p}^*$ where K_p is the public key of Paillier cryptosystem; (iii) they compute the multiplication in the Paillier cryptosystem domain without explaining "how" the Paillier cryptosystem is homomorphically additive. In this work, we propose a novel comparison operator $Comp_e^{P_1,P_2}$ which overcomes these issues. Its output is encrypted and it will be used so as to compute $Max_e^{P_1,P_2}$ operator in our secure version of ReLu. $Comp_e^{P_1,P_2}$ works accordingly two steps:

– *Data randomization.* The objective of this step is to apply blinding to data P_1 will send to P_2. To do so, and as given in [15], P_1 selects two random values r and r' from $\mathbb{Z}_{K_p}$ such that r is significantly greater than r' ($r >> r'$) and that it verifies the constraint

$$log_2(K_p) > log_2(r) + l + 2 \tag{15}$$

where l is the number of bits used to encode the input. Then, P_1 computes

$$E[r(a-b)-r'] = (E[a]E[b]^{-1})^r \times E[r']^{-1} \tag{16}$$

and sends the result to P_2.

– *Secure comparison.* P_2 decrypts the data and compare them to 0 and sends an encrypted bit i such as:

$$Comp_e^{P_1,P_2}(E[a],E[b]) = E[i] = \begin{cases} E[1] & if \quad r(a-b)-r' > 0 \\ E[0] & if \quad \text{else} \end{cases} \tag{17}$$

Then P_2 sends $E[i]$ to P_1.
Based on $Comp_e^{P_1,P_2}(E[a],E[b])$, P_1 can compute the $Max_e^{P_1,P_2}$ operator $Max_e^{P_1,P_2}(E[a],E[b])$:

$$\begin{aligned} Max_e^{P_1,P_2}(E[a],E[b]) &= E[max(a,b)] \\ &= Mul_e^{P_1,P_2}(E[a]E[b]^{-1},E[i]) \times E[b] \\ = E[i(a-b)+b] &= \begin{cases} E[a] & if \quad i=1 \\ E[b] & if \quad i=0 \end{cases} \end{aligned} \tag{18}$$

With $Comp_e^{P_1,P_2}(E[a],E[b])$, P_1 accesses to the encrypted version of the maximum value between two integers without knowing which value is greater than the other one. Such a security level is achieved based on fact the Paillier cryptosystem uses random values which multiply after each multiplication (i.e. $E[a,r_1]E[b,r_2] = E[a+b,r_1r_2]$ - see Sect. 2). Finally, based on the $Max_e^{P_1,P_2}$ operator, the output of our secure ReLU based perceptron is given by:

$$E[max(0,y)] = Max_e^{P_1,P_2}(E[0],E[y]) \tag{19}$$

Based on this results, a secure MLP is based on secure perceptron layers.

Secure Back-Propagation Phase. As stated in Sect. 3, the objective of the back-propagation phase is to update the MLP weights of each perceptron. In the supervised mode, for a given input, one computes the error between the MLP output and the input data label according to an objective or cost function. In this work, the Mean Square Error (MSE) is used. Then, the MLP weights are updated so as to minimize this function.

Let us consider a MLP network composed of M_L layers (see Fig. 2) and an *a priori* known vector input data x_n along with its label t_n. If $t_n^{'}$ corresponds to the MLP output (see Fig. 2), then the error e_n in the clear domain is such that

$$e_n = MSE(t_n^{'},t_n) = ||t_n^{'} - t_n||_2^2. \tag{20}$$

This cost function can be expressed in the Paillier encrypted domain by

$$E[e_n] = Mul_e^{P_1,P_2}(E[t_n^{'} - t_n], E[t_n^{'} - t_n]) \tag{21}$$

Let us recall, that in our framework, P_1 holds $E[t_n^{'}]$ and $E[t_n]$. It computes $E[t_n^{'} - t_n]$ thanks to the homomorphic Paillier properties, and interacts with P_2 so as to compute $E[e_n] = Mul_e^{P_1,P_2}(E[t_n^{'} - t_n], E[t_n^{'} - t_n])$.

Once the error computed e_n, the next step stands in back propagating it so as to update the MLP weights of each perceptron. This update is based on the descent gradient. The updated value of a MLP weight $w_{i,j}^l$ is given by

$$w_{i,j}^l = w_{i,j}^l - \frac{1}{\lambda^{-1}} \frac{\partial e_n}{\partial w_{i,j}^l} \tag{22}$$

where: $w_{i,j}^l$ represents the weight between the i^{th} perceptron in layer l and the j^{th} perceptron in layer $l+1$; λ is the learning rate factor.

The descent gradient can be computed with the help of the chain rule algorithm so as to calculate all partial derivatives, even those of intermediary layers. According to this algorithm, the gradient is given as

$$\frac{\partial e_n}{\partial w_{i,j}^l} = N_i^l . \delta_j^{l+1} \tag{23}$$

δ_j^{l+1} is the fraction of the network error that is caused by the j^{th} perceptron in the layer $l+1$. The computation of the error depends on the location of the perceptron in the network and is such as

$$\delta_i^l = \begin{cases} A_i^{'l}(t_n^{'}(i))(t_n(i) - t_n^{'}(i)) & if \quad l = M_L \\ A_i^{'l}(N_i^l) \sum_j w_{i,j}^l \delta_j^{l+1} & if \quad l = M_L \end{cases} \tag{24}$$

where $A_i(.)$ is the activation function of the i^{th} perceptron and t_n denotes the label of the data placed at the input of the MLP. Notice that, derivate of the ReLU function is $A_i^{'}(y) = 1_{(y>0)}$, where $1_{(.)}$ represents the unit step function the value of which is zero for negative input or one, otherwise. The same update operation in the encrypted domain becomes

$$E[w_{i,j}^l] = E[w_{i,j}^l] Div_e^{P_1,P_2}(E[\frac{\partial e_n}{\partial w_{i,j}^l}], \lambda) \tag{25}$$

The back-propagation phase in the encrypted domain can be easily derived

$$E[\frac{\partial e_n}{\partial w_{i,j}^l}] = Mul_e^{P_1,P_2}(E[N_i^l], E[\delta_j^{l+1}]) \tag{26}$$

where

$$E[\delta_i^l] = \begin{cases} Mul_e^{P_1,P_2}(E[A_i^{'l}(t_n^{'}(i))] E[(t_n(i) - t_n^{'}(i))] & if \quad l = M_L \\ Mul_e^{P_1,P_2}(E[A_i^{'l}(N_i^l)], \prod_j Mul_e^{P_1,P_2}(E[w_{i,j}^l], E[\delta_j^{l+1}])) & if \quad l = M_L \end{cases} \tag{27}$$

It is important to underline that the encrypted version of the unit step function $E[1_{(y>0)}]$ is equivalent to $E[1_{(y>0)}] = Comp_e^{P_1,P_2}(E[0], E[y])$.

By iteratively applying the secure feed-forward and back-propagation phases, it is possible to train our SMLP without compromising the security of the input data, of the SMLP parameters and of its output. It is the same when classifying new data.

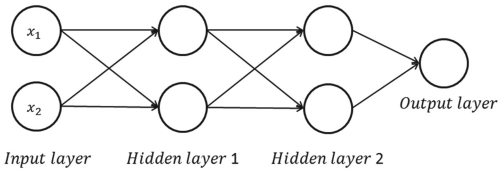

Fig. 3. SMLP architecture used to learning the AND function

4 Experimental Results and Security Analysis

The proposed SMLP solution has been implemented so as to learn the $And - Logic$ function in a supervised training mode. This function takes as input two real numbers x_1 and x_2 in the interval $[0, 1]$ and its output is a binary value such that:

$$y = [x_1]AND[x_2] \tag{28}$$

where [.] denotes the rounding function.

4.1 Dataset and MLP Architecture

The training data set is constituted of 10000 lines of three columns each, where each line represents a training input sample. The first two columns contain two real values between 0.0 and 1.0, while the last column contains their AND value.

Figure 3 provides the architecture of the implemented SMLP. It is composed of an input layer and of two hidden layers, both containing two perceptrons, followed by an output layer of one perceptron. As stated above, the network is based on our secure ReLU activation function (see Sect. 3). In all following tests, the expansion factor Q was fixed to 10^6 so as to ensure the SMLP works with integer values with a training phase limited to 100 epochs and 8000 samples of the training data set are used for the learning phase and the 2000 other for the testing phase. The expected result is that, upon the entry of two values contained between 0.0 and 1.0, the activation of the output layer after feed-forward contains the value of the AND between the two inputs, that is, either 0 or 1.

4.2 Secure MLP Performance

The performance of our secure MLP, which is expressed in terms of classification accuracy and also convergence, depends on the learning rate. Precision is the number of correct predictions made divided by the total number of predictions made. We recall that the learning rate factor λ plays a critical role in the convergence of the network. Indeed, it influences the modification step in the weights update in the back-propagation phase (see Sect. 3). We tested several λ values in the range $[10^{-12}; 10^{-4}]$. We give in Table 1, the precision of our SMLP in average after 10 tries, and if yes or no it has converged after the training. It can be seen that SMLP converges for values of λ smaller than $\lambda = 10^{-8}$. We thus recommend taking initial weights distributed in the range $[10^{-5}, 10^{5}]$ and a learning rate factor $\lambda = 10^{-10}$.

Table 1. Convergence and precision of our SMLP for different learning rate factor values

λ	10^{-12}	10^{-10}	10^{-8}	10^{-6}	10^{-4}
Convergence (Y/N)/Accuracy	Yes/72%	Yes/83%	Yes/93.1%	No	No

We also have trained the equivalent MLP in the clear domain under the same conditions with a learning rate factor of 0.005 and 100 epochs. The obtained MLP precision is about 98.3% with of course no convergence issues. It can be seen based on Table 1, that SMLP always provides lower performance. This can be explained by the use of an expansion factor so as to convert real values into integer values. Anyway, these results show that it is possible to train a MLP in a secure outsourced way.

4.3 Security Analysis Under the Semi-honest Model

The following analysis considers the semi-honest cloud adversary model as presented in Sect. 3. Due to the fact all data (i.e. input data and SMLP parameters) are encrypted with the Paillier cryptosystem the security of which has been demonstrated in [16], the security of the feed-forward and back-propagation phases stand on the security of the operators $Mul_e^{P_1,P_2}(.)$, $Div_e^{P_1,P_2}(.)$ and $Comp_e^{P_1,P_2}(.)$.

- Security of $Mul_e^{P_1,P_2}(.)$ - As shown in Sect. 2, $Mul_e^{P_1,P_2}(E[a], E[b])$ relies on a data blinding operation. P_1 applies on $E[a]$ and $E[b]$ to compute $E[a \times b]$. To do so, P_1 generates two random values r_a and r_b from $\mathbb{Z}_{K_p}$ and computes $E[a + r_a]$ and $E[b + r_b]$. P_2 decrypts by next these values. Since r_a and r_b are randomly chosen in $\mathbb{Z}_{K_p}$ and only known from P_1, they give no clues to P_2 regarding a and b.
- Security of $Div_e^{P_1,P_2}(.)$ - We let the reader refer to [19], where Thijs Veugen proved the security of the operator $Div_e^{P_1,P_2}(.)$.

– Security of $Comp_e^{P_1,P_2}(.)$ and consequently of $Max_e^{P_1,P_2}(.,.)$– As explained in Sect. 3, $Max_e^{P_1,P_2}(E[a], E[b]) = E[max(a,b)]$ depends on the $Comp_e^{P_1,P_2}(;)$ operator. Let us consider P_1 possesses the couple $(E[a], E[b])$ and that he wants to compute $Max_e^{P_1,P_2}(E[a], E[b])$. To do so, it computes $E[r(a-b)-r']$ where r and r' are chosen uniformly from $\mathbb{Z}_{K_p}$ under the constraint that $r >> r'$. P_2 accesses to $r(a-b)-r'$ from which it cannot deduce any information about a and b nor about $a-b$ since it does not know r and r'. P_2 compares this value to zero. This comparison gives not more information to P_2. By next, in order to avoid that P_1 knows the comparison result, P_2 encrypts using the user public key the bit 0 or 1 (see Sect. 3) and it sends it to P_1. P_1 can derive the results of the function $Max_e^{P_1,P_2}(E[a], E[b])$, because all these computations are conducted over encrypted data, P_1 has no idea about a, b and $Max(a,b)$.
The rest of the computations (e.g. MSE, error derivatives) are based on either encrypted or randomized data. As consequence, if P_1 and P_2 do not collude, no information related to the user data or to the SMLP model is disclosed. Since all operations involved in the computation of the feed-forward and back-propagation phases are in cascade, then according to the sequential Composition theorem [12], SMLP is completely secure under the semi-honest model.

5 Conclusion

In this paper, we have proposed a new Secure Multilayer Perceptron (SMLP) which can be deployed in the cloud. Its main originality, compared to actual homomorphic encryption based SMLP schemes, is that it can be trained with homomorphically encrypted data with no extra communications between the user and the servers. With this scheme, all data; input data and SMLP output and parameters, are encrypted. Our SMLP is based on: an original secure version of the $Max(.,.)$ function we propose, the result of which is encrypted and a ReLU activation function secured with no linear approximation. Such a SMLP has been implemented so as to learn or model the AND function in-between real values. Experimental results demonstrate that SMLP converges in its training phase under some parameter initialization constraints. Beyond the complexity of our SMLP, which is based on homomorphic encryption, these preliminary results are very encouraging.

References

1. Aono, Y., Hayashi, T., Wang, L., Moriai, S., et al.: Privacy-preserving deep learning via additively homomorphic encryption. IEEE Trans. Inf. Forensics Secur. **13**(5), 1333–1345 (2018)
2. Bellafqira, R., Coatrieux, G., Bouslimi, D., Quellec, G.: Content-based image retrieval in homomorphic encryption domain. In: 2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), pp. 2944–2947. IEEE (2015)
3. Bellafqira, R., Coatrieux, G., Bouslimi, D., Quellec, G.: An end to end secure CBIR over encrypted medical database. In: 2016 IEEE 38th Annual International Conference of the Engineering in Medicine and Biology Society (EMBC), pp. 2537–2540. IEEE (2016)

4. Bellafqira, R., Coatrieux, G., Bouslimi, D., Quellec, G., Cozic, M.: Proxy re-encryption based on homomorphic encryption. In: Proceedings of the 33rd Annual Computer Security Applications Conference, pp. 154–161. ACM (2017)
5. Bost, R., Popa, R.A., Tu, S., Goldwasser, S.: Machine learning classification over encrypted data. In: NDSS (2015)
6. Bouslimi, D., Bellafqira, R., Coatrieux, G.: Data hiding in homomorphically encrypted medical images for verifying their reliability in both encrypted and spatial domains. In: 2016 IEEE 38th Annual International Conference of the Engineering in Medicine and Biology Society (EMBC), pp. 2496–2499. IEEE (2016)
7. Castellano, G., Fanelli, A.M.: Variable selection using neural-network models. Neurocomputing **31**(1), 1–13 (2000)
8. Chabanne, H., de Wargny, A., Milgram, J., Morel, C., Prouff, E.: Privacy-preserving classification on deep neural network. IACR Cryptol. ePrint Archive **2017**, 35 (2017)
9. Decencière, E., et al.: TeleOphta: machine learning and image processing methods for teleophthalmology. IRBM **34**(2), 196–203 (2013)
10. Ding, W., Yan, Z., Deng, R.H.: Encrypted data processing with homomorphic re-encryption. Inf. Sci. **409**, 35–55 (2017)
11. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pp. 315–323 (2011)
12. Goldreich, O.: Foundations of Cryptography. Basic Applications, vol. 2. Cambridge University Press, New York (2009)
13. Hsu, C.Y., Lu, C.S., Pei, S.C.: Image feature extraction in encrypted domain with privacy-preserving sift. IEEE Trans. Image Process. **21**(11), 4593–4607 (2012)
14. Huang, Y., Evans, D., Katz, J., Malka, L.: Faster secure two-party computation using garbled circuits. In: USENIX Security Symposium, vol. 201 (2011)
15. Kerschbaum, F., Biswas, D., de Hoogh, S.: Performance comparison of secure comparison protocols. In: 20th International Workshop on Database and Expert Systems Application, pp. 133–136. IEEE (2009)
16. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_16
17. Schlitter, N.: A protocol for privacy preserving neural network learning on horizontal partitioned data. In: PSD (2008)
18. Shokri, R., Shmatikov, V.: Privacy-preserving deep learning. In: Proceedings of the 22nd ACM SIGSAC conference on computer and communications security, pp. 1310–1321. ACM (2015)
19. Veugen, T.: Encrypted integer division. In: 2010 IEEE International Workshop on Information Forensics and Security (WIFS), pp. 1–6. IEEE (2010)
20. Wu, F., Zhong, H., Shi, R., Huang, H.: Secure two-party computation of the quadratic function's extreme minimal value. In: 2012 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), pp. 2975–2978. IEEE (2012)
21. Xie, P., Bilenko, M., Finley, T., Gilad-Bachrach, R., Lauter, K., Naehrig, M.: Crypto-nets: Neural networks over encrypted data. arXiv preprint arXiv:1412.6181 (2014)
22. Zheng, S., et al.: Asynchronous stochastic gradient descent with delay compensation. arXiv preprint arXiv:1609.08326 (2016)