

Method and System for Secure Training of Neural Network

Technical Field

5

The present invention relates, in general terms, to a method and system for secure training of a neural network. In particular, the present invention provides a system and method for performing neural network evaluations on encrypted data while preserving data privacy.

10

Background

This background is provided for generally presenting the context of the disclosure. The reference in this specification to any prior publication (or information derived from it), or to any matter which is known, is not, and should not be taken as an acknowledgment or admission or any form of suggestion that that prior publication (or information derived from it) or known matter forms part of the prior art base or common general knowledge in the field of endeavour to which this specification relates.

20

Machine learning as a service has shown numerous potential applications in the real world. Implementation of machine learning services typically involves training and inference phases performed on a machine learning model, often a neural network. Neural network training and inferences are usually considered computationally expensive, yet many users do not have sufficiently powerful computing resources to make training convenient or time and cost effective. Therefore, commercial machine learning service providers prefer to put neural network training into the cloud.

30 Issues arise when training a neural network in the cloud, particularly that data privacy is often compromised. Traditional neural networks train and infer on raw, unencrypted data. For some data types, such as medical records, it is not permissible to send raw data and thus such traditional training methods cannot be used.

35

More recently, privacy-preserving neural networks have been utilised to address privacy issues in 'artificial intelligence (AI) as a service' products. With such networks, a user will encrypt data before sending it to the cloud server. The cloud server then performs neural network evaluations on encrypted data and
5 outputs an encrypted prediction result.

While inference may be privacy preserving, the training phase still typically employs an optimization algorithm for network parameter optimization, that uses a training data comprising users' raw, unencrypted data.

10

It is desirable therefore that there be provided a system and method for neural network training and inference, that is computationally efficient while preserving privacy.

15 **Summary**

In view of the abovementioned privacy issues, proposed herein is a practical privacy-preserving training scheme, that enables fast and accurate training over encrypted data. The training scheme employs homomorphic encryption, and a
20 combination of encryption and decryption of user data, based on the sensitivity of that data. The present training scheme significantly reduces training latency while protecting privacy of sensitive data.

Disclosed herein is a computer implemented method for secure training of a
25 neural network. The method involves receiving a label and an encrypted input from a client. Thus, privacy in the first instance is preserved over the entire input. Homomorphic computations are then performed on the encrypted input using the neural network to obtain an encrypted intermediate ciphertext. The encrypted intermediate ciphertext is returned to the client. The client then
30 supplies a non-sensitive decrypted input corresponding to a decryption of the encrypted intermediate ciphertext. This provides non-sensitive raw data that can be used to update network parameters, while sensitive data remains encrypted. Computations are then performed on the non-sensitive decrypted input using the neural network to obtain a decrypted output, and the weights
35 of the neural network are updated based on the decrypted output.

As used herein, the term "homomorphic computations" refers to inference (i.e., operation on an input) by a neural network layer or layers on a homomorphically encrypted input. Similarly, the term "computations" refers to inference by a neural network layer or layers on a plaintext or decrypted input.

5

Also disclosed herein is a computer-implemented method for secure training of a neural network. Being performed on a computer, steps of the method are thus implemented in software, firmware or hardware, or a combination thereof. To that end, the computer-implemented method involves generating, at a client terminal, an encrypted input from a decrypted input. The encrypted input and a corresponding label are then transmitted to a server. Homomorphic computations are then performed, at the server, on the encrypted input using the neural network to obtain an encrypted intermediate ciphertext. The client is provided the encrypted intermediate ciphertext and generates a non-sensitive decrypted input corresponding to a decryption of the encrypted intermediate ciphertext. The client then transmits the non-sensitive decrypted input to the server that then performs computations on the non-sensitive decrypted input using the neural network to obtain a decrypted output. Weights of the neural network are then updated based on the decrypted output.

20

The non-sensitive decrypted input may be a decryption of encrypted gradients derived from the encrypted intermediate ciphertext. To that end, the client derives encrypted gradients from the encrypted intermediate ciphertext, and generates decrypted gradients from the encrypted gradients. The non-sensitive decrypted input comprises the decrypted gradients.

25

Homomorphic computations may be performed on the encrypted input by applying a first linear function to the encrypted input to produce the encrypted intermediate ciphertext. In such embodiments, the non-sensitive decrypted input may be an output of an activation function applied to a decryption of the encrypted intermediate ciphertext. Thus, the client decrypts the encrypted intermediate ciphertext to produce a decrypted intermediated ciphertext, and applies the activation function to the decrypted intermediated ciphertext to produce an output. The non-sensitive decrypted input comprises the output.

30

The activation function may include multiple sign functions or multiple step functions.

Weights of the neural network can be updated based on the decrypted output
5 by evaluating, at the server, a loss function based on the decrypted output and the label. This can involve, for example, evaluating gradients of the loss function and weights of the neural network at both the server and client, and updating the weights based on the evaluation.

10 Also disclosed herein is a system for secure training of a neural network, comprising:

- a processor;

- a memory storing a plurality of modules, the modules comprising:

- 15 a pre-processing module configured to receive a label and an encrypted input from a client;

- a homomorphic module configured to perform homomorphic computations on the encrypted input using the neural network to obtain an encrypted intermediate ciphertext and provide the encrypted intermediate ciphertext to the client;

- 20 a client level computation module configured to receive a non-sensitive decrypted input from the client, wherein the non-sensitive decrypted input corresponds to a decryption of the encrypted intermediate ciphertext and perform computations on the non-sensitive decrypted input using the neural network to
25 obtain a decrypted output; and

- an output module configured to update weights of the neural network based on the decrypted output.

Also disclosed herein is a system for secure training of a neural network,
30 comprising:

- a processor;

- a memory storing a plurality of modules, the modules comprising:

- 35 a client level pre-processing module configured to generate an encrypted input from a decrypted input at a client terminal and to transmit the encrypted input and a corresponding label to a server;

- a homomorphic module configured to perform homomorphic computations on the encrypted input at the server using the neural network to obtain an encrypted intermediate ciphertext, and providing the encrypted intermediate ciphertext to the client;
- 5 a client level computation module configured to generate a non-sensitive decrypted input corresponding to a decryption of the encrypted intermediate ciphertext, and transmitting the non-sensitive decrypted input to the server;
- 10 performing, at the server, computations on the non-sensitive decrypted input using the neural network to obtain a decrypted output; and
- an output module configured to update weights of the neural network based on the decrypted output.
- 15 Advantageously, embodiments of the invention provide a hybrid training protocol, whereby encrypted sensitive and non-sensitive information and decrypted, non-sensitive information, are used for training a cloud-based neural network with low latency while preserving privacy of sensitive information.

20 **Brief description of the drawings**

- Some embodiments of the present invention will now be described, by way of non-limiting example, with reference to the drawings in which:
- Figure 1 is a flowchart of a method for secure training of neural network,
- 25 according to an embodiment of the present invention;
- Figure 2 depicts a schematic block diagram of a system for secure training of neural network, according to an embodiment of the present invention;
- 30 Figures 3a to 3d depict schematic diagrams of a system for secure training of neural network, according to an embodiment of the present invention;
- Figure 4 is a bar chart illustrating number of useful equations after activation function in an experiment, according to an embodiment of the present
- 35 invention;

Figures 5a and 5b are line graphs illustrating examples of a 3-step function and a 5-step function, according to an embodiment of the present invention;

Figure 6 illustrates the ten classes in CIFAR-10 dataset and random images from each class, as used in an embodiment of the present invention;

Figure 7 illustrates a convolutional neural network with 3 convolution layers, according to an embodiment of the present invention; and

Figure 8 illustrates how transfer learning is applied in privacy-preserving facial recognition network training.

Detailed description

Disclosed herein are methods for training a neural network, using homomorphic encryption. When adopting homomorphic encryption (HE) is applied in the field of neural network training, efficiency is a significant concern. Homomorphic computations over encrypted data are much slower than computations on non-encrypted data. For this reason, traditional training methods over large datasets have been primarily conducted using raw (unencrypted) data.

With current HE techniques, if the whole network is trained in ciphertext, which means that an encrypted input is uploaded to the cloud server at the outset, and the server performs both forward propagation and backward propagation on the encrypted data without decryption, training is too slow to be practical.

The methods disclosed herein employ a hybrid HE-based training model. Deep neural network training is split between a ciphertext training phase (encrypted input) and a plaintext training phase (unencrypted or decrypted input). In this model, when the cloud server receives the encrypted sensitive input, it first performs some expensive homomorphic computations on the encrypted input and obtains an encrypted non-sensitive intermediate result. The cloud server asks the client to decrypt the intermediate result, and the remaining training is performed in non-sensitive plaintext.

The hybrid-HE method combines the privacy preservation of homomorphic encryption and training, with the speed of training on unencrypted data. Figure 1 provides the workflow of an example training method 100 for secure training of a neural network. As with all neural network training methods, method 100
5 is necessarily computer-implemented.

According to step 102 of the method 100, a client terminal generates an encrypted input from a decrypted input. The client terminal may also obtain or generate a label for the decrypted input – the label is also necessarily true for
10 the encrypted input. The label specifies the ground truth. For example, in a training dataset for a neural network being trained to categorise or label images based on objects in those images, the label corresponding to an input image (decrypted input) may identify the object shown in that image.

15 The client then transmits the encrypted input and a corresponding label to a server, per step 104, the server then applying the neural network to the encrypted input, per step 106. The neural network applies homomorphic computations on the encrypted data, to obtain an encrypted intermediate ciphertext. Since the computations are homomorphic, the intermediate
20 ciphertext is an encryption of the output of the same computations when applied to the corresponding plaintext – i.e., the input prior to encryption.

The client receives the encrypted intermediate ciphertext per step 108 and generates a non-sensitive decrypted input at step 110. The non-sensitive,
25 decrypted input is, or corresponds to, a decryption of the encrypted intermediate ciphertext. The non-sensitive decrypted input is then transmitted to the server per step 112.

Per step 114, the server performs computations on the non-sensitive, decrypted
30 input (i.e., non-sensitive plaintext input) to obtain a decrypted output. The decrypted output is then used to update weights of the neural network, per step 116.

In general, a single party will operate the server and interact with multiple
35 clients. To that end, from the perspective of the server, the method 100 involves

- the server receiving a label and an encrypted input from a client (per step 104) and performing homomorphic computations on the encrypted input using the neural network to obtain the encrypted intermediate ciphertext (per step 106). The server then provides the encrypted intermediate ciphertext to the client
- 5 (per step 108) and receives the non-sensitive decrypted input from the client (per step 112). The server subsequently performs computations per step 114, to obtain the decrypted output and the weights of the neural network are updated per step 116.
- 10 The training algorithm corresponding to method 100 is set out and discussed in greater detail, below (see Algorithm 1). The training procedure for one sample (or one batch) is divided into two parts ($S_1(\cdot)$, $S_2(\cdot)$).

Algorithm 1 Hybrid HE-based training model (informal).

- 15
- 1: **Input:** Encrypted training sample $\text{Enc}(x)$.
 - 2: The cloud server performs homomorphic computations on encrypted input $\text{Enc}(x) := S_1(\text{Enc}(x))$.
 - 3: The cloud server sends ciphertext $\text{Enc}(x_1)$ to the client.
 - 20 4: The client decrypts the ciphertext $x_1 = \text{Dec}(\text{Enc}(x_1))$ and checks its sensitivity.
 - 5: The client send x_1 to the cloud server if it is non-sensitive.
 - 6: The cloud server performs the remaining training computations $S_2(x_1)$, and updates the neural network weights.
 - 7: Move to next training sample (or batch).
-

- 25
- In Algorithm 1 $\text{Enc}(\text{data})$ and $\text{Dec}(\text{data})$ refer to encryption and decryption of "data", respectively, x is the raw training data, x_1 is the intermediate result, and S_1 and S_2 are functions for evaluating the homomorphically encrypted input and plaintext input, respectively.

- 30
- In this training model, the client knows no information of the neural network. This helps to preserve the privacy of the cloud server. Conversely, the cloud server only knows the non-sensitive intermediate result x_1 at any stage during training. The server never knows the sensitive raw training data x . This helps to
- 35 preserve the privacy of the client's data.

Because the training is in the cloud server, using plaintext computations after the intermediate result means that the cloud server is able to learn some information about the training dataset. Therefore, in this model, careful design
5 of the system and analysis of privacy leakage can be useful to ensure protection of the client's input.

Further, the hybrid HE-based training model is able to train deeper models. While existing works are able to train shallow networks if the data is encrypted,
10 this model can apply training to deeper models in a much shorter time period. This is due to the first function S_1 being applied to the homomorphically encrypted input, and all other functions (only S_2 is mentioned above, though similar functions for subsequent neural network layers can be used) operate on decrypted plaintext or the result of processing of decrypted plaintext. Thus, all
15 but the first function train in a standard manner, per traditional neural network training on unencrypted data.

A neural network can be considered as population of artificial neurons arranged in layers. Each layer receives the data generated by its previous layer and
20 outputs the processed data for the next layer. Such a structure allows higher-level, abstract features to be represented as lower-level features through non-linear function computation at each layer.

The raw data is encoded properly and fed into the input layer of a neural network.
25 Then these features from the input data are gradually mapped to higher-level abstractions via iterative update (forward propagation and backward propagation) in the hidden layers until the convergence condition is achieved. These mapping abstractions, also known as learned neural network model or neural weights, can be used to perform inference tasks.

30 Training of a neural network includes forward propagation and backward propagation. In forward propagation, the input data goes through all layers consecutively in the forward direction. At each neuron layer l , the processing happens in two steps:

35

- Linear function: It is a weighted sum of inputs, which can be described as $\gamma_l = W_l x_l + \beta_l$.
- Activation function: The calculated weighted sum is passed to the activation function. An activation function is a mathematical function which adds non-linearity to the network. There are some commonly used activation functions: Sigmoid, ReLu and Softmax. It can be described as $x_l = f_l(\gamma_l)$.

W_l is the edge weight matrix connecting layer $l - 1$ and l , β_l is the bias vector of layer l . Let x_l be the output vector of layer l , and let f_l be the activation function in layer l . The input layer is set to be layer 0 and let $W_0, f_0, \beta_0 = \emptyset$. Let x_0 be the input to the neural network.

The process of moving from output layer to the input layer is called backward propagation. Backward propagation is the preferable method to adjusting or correcting the weights to minimize the loss function – i.e., the function describing the distance or error between the output of the neural network and the true value that the neural network is trying to achieve. This is a non-linear optimization problem, which is typically solved using variants of gradient descent. The processing happens in two steps:

- Derivative computation. It computes the partial derivatives of the loss function L at each layer matrix W_l , which can be described as $\frac{\delta L}{\delta W_l}$, and for simplicity notation $\delta W_l := \frac{\delta L}{\delta W_l}$ is defined. Similarly, $\delta \beta_l := \frac{\delta L}{\delta \beta_l}$ is defined.
- Parameters update. It updates the weights according to the derivatives and learning rate α , which can be described as $W'_l = W_l - \alpha \delta W_l$, where W'_l is the updated weights for layer l . Similarly, bias vector is updated by $\beta'_l = \beta_l - \alpha \delta \beta_l$.

Framework

In this section, three schemes are proposed according to various embodiments of the present invention, each offering varying levels of efficiency and privacy preservation. The proposed scheme can be applied to neural networks of any depth, including convolutional neural networks, as the evaluation of

convolutional layers consists of linear computations that can be represented as linear transformations.

For the sake of clarity, the schemes are illustrated using a fully connected neural network with one hidden layer, consisting of an input layer, a hidden layer, and an output layer. The input layer is composed of L nodes, the hidden layer contains H hidden nodes, and the output layer consists of C nodes. The weight matrices and bias vectors in this neural network are $(W_1, W_2, \beta_1, \beta_2)$. The activation function in hidden nodes is $f_1(\cdot)$, e.g., $ReLU(\cdot)$. The activation function in output nodes is $f_2(\cdot)$, e.g., $sigmoid(\cdot)$. Let $L(\cdot, \cdot)$ be the loss function of the neural network. It takes the output of neural network and the true label as input. Its input includes the neural network output vector and the goal vector.

For simplicity of notations, encrypted ciphertext are underlined in these schemes. Their decrypted counterparts use the same notation but without the underline: for example, where $\underline{x}$ refers to encrypted data, x refers to the decrypted version of that data. In the schemes, all training samples are labelled and stored in set $\mathcal{X} = \{(x^{(t)}, y^{(t)})\}$. Training samples are encrypted, therefore in the model the user holds $\{(x^{(t)}, y^{(t)})\}$ and the server holds $\{(\underline{x}^{(t)}, y^{(t)})\}$ or $\{(\underline{x}^{(t)}, \underline{y}^{(t)})\}$ depending on different schemes.

20

Scheme 1: Decrypt after evaluating gradients

To compare the present hybrid training model with other models, set out below are three schemes, each with different efficiency and privacy-preserving level. In Scheme 1 (Algorithm 2), the server homomorphically performs the forward propagation, and evaluates the loss function. After homomorphically computing gradients, the server will return the ciphertexts to the user. User decrypts the gradients and returns them to the server. Finally, the server updates the neural network parameters.

30 **Algorithm 2** Scheme 1: Decrypt after evaluating gradients

1. **Input:** Current neural network parameters $(W_1, W_2, \beta_1, \beta_2)$, activation functions $f_1(\cdot), f_2(\cdot)$, loss function $\mathcal{L}(\cdot, \cdot)$, batch size m , m encrypted input $\{\underline{x}^{(t)}\}_{t \in [m]}$ and their labels $\{\underline{y}^{(t)}\}_{t \in [m]}$, learning rate α .
2. **Output:** Updated neural networks parameters $(W'_1, W'_2, \beta'_1, \beta'_2)$.

3. **for** $t = 1, \dots, m$ **do**
4. Set $\underline{x}_0 = \underline{x}^{(t)}, \underline{y} = \underline{y}^{(t)}$.
5. The server homomorphically evaluates $\underline{y}_1 = W_1 \underline{x}_0 + \beta_1$.
6. The server homomorphically evaluates activation function $\underline{x}_1 = f_1(\underline{y}_1)$.
- 5 7. The server homomorphically evaluates $\underline{y}_2 = W_2 \underline{x}_1 + \beta_2$.
8. The server homomorphically evaluates activation function $\underline{x}_2 = f_2(\underline{y}_2)$.
- For n -layer neural networks, repeating steps 7 and 8 with input $W_3, \dots, W_n, \beta_3, \dots, \beta_n$ and $f_3, \dots, f_n$.
9. The server homomorphically evaluates loss function $\underline{L}^{(t)} = \mathcal{L}(\underline{x}_2, \underline{y})$
- 10 10. The server homomorphically evaluates gradients:

$$\begin{aligned} \sum_{t=1}^m \underline{grad}^{(t)} &:= \sum_{t=1}^m \frac{\partial \underline{L}^{(t)}}{\partial W_1}, \sum_{t=1}^m \frac{\partial \underline{L}^{(t)}}{\partial W_2}, \sum_{t=1}^m \frac{\partial \underline{L}^{(t)}}{\partial \beta_1}, \sum_{t=1}^m \frac{\partial \underline{L}^{(t)}}{\partial \beta_2}, \\ &:= \sum_{t=1}^m \frac{\partial W_1^{(t)}}{\partial W_1}, \sum_{t=1}^m \frac{\partial W_2^{(t)}}{\partial W_2}, \sum_{t=1}^m \frac{\partial \beta_1^{(t)}}{\partial \beta_1}, \sum_{t=1}^m \frac{\partial \beta_2^{(t)}}{\partial \beta_2} \end{aligned}$$

- 15 11. **end for**
12. Server sends $\{\sum_{t=1}^m \underline{grad}^{(t)}\}$ to user.
13. User decrypts and returns $\{\sum_{t=1}^m \underline{grad}^{(t)}\}$ to server.
14. Server updates neural network parameters and returns $(W'_1, W'_2, \beta'_1, \beta'_2)$.

$$20 \quad W'_l = W_l - \alpha \sum_{t=1}^m \frac{\partial W_l^{(t)}}{\partial W_l}, \beta'_l = \beta_l - \alpha \sum_{t=1}^m \frac{\partial \beta_l^{(t)}}{\partial \beta_l}, l = 1, 2.$$

In this scheme, the user does not know the training methods, such as choice of loss function, activation function and learning rate. The server does not know the output of each layer and the true label of input data. However, there are

25 lots of homomorphic non-linear function computations and ciphertext x ciphertext computations, both of which are expensive and inefficient. Therefore, this scheme is good at privacy-preserving but it is inefficient.

Scheme 2: Decrypt after evaluating the first linear function.

In Scheme 2, the server homomorphically evaluates the first linear function $\underline{\gamma}_l = W_1 \underline{x}_0 + \beta_1$ in the forward propagation. Then the server sends $\underline{\gamma}_l$ to the user, and user decrypts it, then returns γ_l to the server. Finally, the server performs subsequent steps on plaintext input.

Algorithm 3 Scheme 2: Decrypt after evaluating the first linear function $\underline{x}_1$.

1. **Input:** Current neural network parameters $(W_1, W_2, \beta_1, \beta_2)$, activation functions $f_1(\cdot), f_2(\cdot)$, loss function $\mathcal{L}(\cdot, \cdot)$, batch size m , m encrypted input $\{\underline{x}^{(t)}\}_{t \in [m]}$ and their labels $\{y^{(t)}\}_{t \in [m]}$, learning rate α .
2. **Output:** Updated neural networks parameters $(W'_1, W'_2, \beta'_1, \beta'_2)$.
3. **for** $t = 1, \dots, m$ **do**
4. Set $\underline{x}_0 = \underline{x}^{(t)}$, $y = y^{(t)}$.
5. The server homomorphically evaluate $\underline{\gamma}_1 = W_1 \underline{x}_0 + \beta_1$.
6. The server sends $\underline{\gamma}_1$ to the user.
7. The user decrypts $\underline{\gamma}_1$ and sends γ_1 to the server.
8. The server evaluates activation function $\underline{x}_1 = f_1(\gamma_1)$.
9. The server evaluates $\gamma_1 = W_2 \underline{x}_1 + \beta_2$.
10. The server evaluates activation function $\underline{x}_2 = f_2(\gamma_2)$.
11. For n -layer neural networks, repeating steps 8 and 9 with input $W_3, \dots, W_n, \beta_3, \dots, \beta_n$ and $f_3, \dots, f_n$.
12. The server evaluates loss function $\mathcal{L}^{(t)} = \mathcal{L}(\underline{x}_2, y)$.
13. The server evaluates gradients:

$$\begin{aligned} & \left(\cup_{t=1}^m \left\{ \frac{\partial \mathcal{L}^{(t)}}{\partial W_1 \underline{x}^{(t)}} \right\}, \sum_{t=1}^m \frac{\partial \mathcal{L}^{(t)}}{\partial W_2}, \sum_{t=1}^m \frac{\partial \mathcal{L}^{(t)}}{\partial \beta_1}, \sum_{t=1}^m \frac{\partial \mathcal{L}^{(t)}}{\partial \beta_2} \right) \\ & := \left(\cup_{t=1}^m \{ \partial W_1 \underline{x}^{(t)} \}, \sum_{t=1}^m \partial W_2^{(t)}, \sum_{t=1}^m \partial \beta_1^{(t)}, \sum_{t=1}^m \partial \beta_2^{(t)} \right), \end{aligned}$$

Unlike in Scheme 1, here the evaluation of

$$\sum_{t=1}^m \partial W_1^{(t)} = \sum_{t=1}^m \frac{\partial \mathcal{L}^{(t)}}{\partial W_1 \underline{x}^{(t)}} \frac{\partial W_1 \underline{x}^{(t)}}{\partial W_1}$$

needs to know $\{\underline{x}^{(t)}\}_{t \in [m]}$, while server only knows its encryption $\{\underline{x}^{(t)}\}_{t \in [m]}$. So, server

computes the other parts first and asks user's help to get $\sum_{t=1}^m \partial W_1^{(t)}$.

13. **end for**

14. Server sends $\cup_{t=1}^m \{\partial W_1 \mathbf{x}^{(t)}\}$ to user.

15. User computes and returns $\{\sum_{t=1}^m \partial W_1^{(t)}\}$ to server.

16. Server updates neural network parameters and returns $(W'_1, W'_2, \beta'_1, \beta'_2)$.

$$5 \quad W'_l = W_l - \alpha \sum_{t=1}^m \partial W_l^{(t)}, \beta'_l = \beta_l - \alpha \sum_{t=1}^m \partial \beta_l^{(t)}, l = 1, 2.$$

In this scheme, the user does not know the training methods, such as the choice of loss function, activation function and learning rate. The server will know the output of the first inner-product layer and the true label of input data.

10 Homomorphic evaluations only include linear computation, which is much cheaper than non-linear evaluations. Therefore, this scheme is efficient but not very good at privacy preserving.

Scheme 3: User evaluate first activation function

15 Scheme 3 is an example of an embodiment of the present invention. In Scheme 3 (Algorithm 4), the server homomorphically evaluates the first linear function $\gamma_l = W_1 \underline{x}_0 + \beta_1$ in the forward propagation process (step 106). Then, the server sends γ_l to the client (step 108), and client decrypts it, then returns $f_1(\gamma_1)$ to the server (as per steps 110 and 112). Scheme 3 is very similar to Scheme 2, 20 except that the client returns $f_1(\gamma_1)$ instead of γ_1 to the server. In this scheme, the user knows the activation function $f_1(\cdot)$.

Algorithm 4 Scheme 3: User evaluate the first activation function $f_1(\gamma_1)$.

1. **Input:** Current neural network parameters $(W_1, W_2, \beta_1, \beta_2)$, activation functions $f_1(\cdot), f_2(\cdot)$,
25 loss function $\mathcal{L}(\cdot, \cdot)$, batch size m , m encrypted input $\{\underline{x}^{(t)}\}_{t \in [m]}$ and their labels $\{y^{(t)}\}_{t \in [m]}$,
learning rate α .
2. **Output:** Updated neural networks parameters $(W'_1, W'_2, \beta'_1, \beta'_2)$.
3. **for** $t = 1, \dots, m$ **do**
4. Set $\underline{x}_0 = \underline{x}^{(t)}$, $y = y^{(t)}$.
- 30 5. The server homomorphically evaluates $\gamma_1 = W_1 \underline{x}_0 + \beta_1$.
6. The server sends γ_1 to the user.
7. The user decrypts γ_1 and sends $\mathbf{x}_1 = f_1(\gamma_1)$ to the server.

8. The server evaluates $\gamma_2 = W_2 x_1 + \beta_2$.
9. The server evaluates activation function $x_2 = f_2(\gamma_2)$.
For n -layer neural networks, repeating steps 8 and 9 with input $W_3, \dots, W_n, \beta_3, \dots, \beta_n$ and $f_3, \dots, f_n$.
- 5 10. The server evaluates loss function $\mathcal{L}^{(t)} = \mathcal{L}(x_2, y)$.
11. The server evaluates gradients:

$$\begin{aligned} & \left(\cup_{t=1}^m \left\{ \frac{\partial \mathcal{L}^{(t)}}{\partial W_1 x^{(t)}} \right\}, \sum_{t=1}^m \frac{\partial \mathcal{L}^{(t)}}{\partial W_2}, \sum_{t=1}^m \frac{\partial \mathcal{L}^{(t)}}{\partial \beta_1}, \sum_{t=1}^m \frac{\partial \mathcal{L}^{(t)}}{\partial \beta_2} \right) \\ & := \left(\cup_{t=1}^m \left\{ \partial W_1 x^{(t)} \right\}, \sum_{t=1}^m \partial W_2^{(t)}, \sum_{t=1}^m \partial \beta_1^{(t)}, \sum_{t=1}^m \partial \beta_2^{(t)} \right) \end{aligned}$$

10

Unlike in Scheme 1, here the evaluation of

$$\sum_{t=1}^m \partial W_1^{(t)} = \sum_{t=1}^m \frac{\partial \mathcal{L}^{(t)}}{\partial W_1 x^{(t)}} \frac{\partial W_1 x^{(t)}}{\partial W_1}$$

- needs to know $\{x^{(t)}\}_{t \in [m]}$, while server only knows its encryption $\{\underline{x}^{(t)}\}_{t \in [m]}$. So, server
 15 computes the other parts first and asks user's help to get $\sum_{t=1}^m \partial W_1^{(t)}$.

12. **end for**

13. Server sends $\cup_{t=1}^m \{\partial W_1 x^{(t)}\}$ to user.
14. User computes and returns $\{\sum_{t=1}^m \partial W_1^{(t)}\}$ to server.
15. Server updates neural network parameters and returns $(W'_1, W'_2, \beta'_1, \beta'_2)$.

20

$$W'_l = W_l - \alpha \sum_{t=1}^m \partial W_l^{(t)}, \beta'_l = \beta_l - \alpha \sum_{t=1}^m \partial \beta_l^{(t)}, l = 1, 2.$$

As shown in Scheme 3 (Algorithm 4), the server receives encrypted input $\{\underline{x}^{(t)}\}_{t \in [m]}$ and their labels $\{y^{(t)}\}_{t \in [m]}$ from a client as per step 104 of method 100.

- 25 Prior to that, the client had generated the encrypted input $\{\underline{x}^{(t)}\}_{t \in [m]}$ at client terminal or at client level as per step 102. Then, as per step 106, the server performs homomorphic computations on the encrypted input using the neural network in its forward propagation step. This step obtains an encrypted intermediate ciphertext $\underline{\gamma}_1$, which is the weighted sum of inputs, and can be

described as $\underline{\gamma}_1 = W_1 x_0 + \beta_1$. Performing homomorphic computations on the encrypted input comprises applying a first linear function to the encrypted input to produce the encrypted intermediate ciphertext $\underline{\gamma}_1$. Moreover, the neural network of the present embodiment comprises two trainable layers. Other
 5 embodiments include a greater number of layers. The homomorphic computations are performed using a first set of layers of the neural network – in general, the first set of layers will be a single, first trainable layer (i.e., a layer with trainable weights) of the neural network.

10 The server then provides the encrypted intermediate ciphertext $\underline{\gamma}_1$ to the client as per step 108. Then, as per step 110, the client decrypts the ciphertext $\underline{\gamma}_1$ and generates a non-sensitive decrypted input x_1 . The non-sensitive decrypted input is produced by applying an activation function to the non-sensitive decrypted, described by $x_1 = f_1(\underline{\gamma}_1)$. In some embodiments, the output of the activation
 15 function, non-sensitive decrypted input x_1 , comprises a decryption of encrypted gradients derived from the encrypted intermediate ciphertext $\underline{\gamma}_1$.

The client then sends the non-sensitive decrypted input x_1 to the server as per step 112. As per step 114, the server computes a decrypted output from the
 20 non-sensitive decrypted input x_1 by evaluating $\gamma_2 = W_2 x_1 + \beta_2$ and evaluates the output of the activation function of the second layer, $x_2 = f_2(\gamma_2)$. For n -layer neural networks, the server repeatedly evaluates inputs $\gamma_3, \dots, \gamma_n$ and the outputs of the activation functions, $x_3, \dots, x_n$. In other words, the neural network includes the first set of layers followed by (i.e., sequentially, where no
 25 layer of one set is included in the other set) the second set of layers, such that the homomorphic computations occur first (during training and, later, during normal runtime on new data), in the first set of layers, and the computations on decrypted data occur thereafter, in the second set of layers. In some embodiments, the neural network includes more than two sets of layers, or
 30 more than two layers where each set includes a single layer only. In such embodiments, each further set of layers receives the output of the second set of layers or a further set of layers that is earlier in the sequence. To illustrate, where there is one layer per set of layers, the first layer operates on encrypted data, the second layer operates on decrypted data, the first further layer

operates on the output of the second layer and, if there are any additional further layers, they each operate in sequence on the output of the immediately preceding further layer. Thus, all layers but for the first layer operate on decrypted data or the result of one or more operations on decrypted data. The server then proceeds to evaluate loss function, gradients, and updating weights of the neural network based on the decrypted output (step 116).

In an embodiment of the present invention, as disclosed in scheme 3, the server will know the output of first layer and the true labels of input data. During the training phase, true labels are not considered to be a big privacy problem, since in the training dataset the input is usually uniformly distributed over all classes. So, knowledge of the ground truth does not generally give away much, or any, information about the input. Similar to scheme 2, scheme 3 is very efficient since all ciphertext evaluations are linear, and training on all but one layer of the neural network is conducted over plaintext. Moreover, the activation function in Scheme 3 helps protect the information in the input data. Therefore, Scheme 3 can achieve better privacy-preservation than scheme 2, while having high efficiency.

The flow of the 3 schemes is summarised in Table 1 below. Parameters in handwriting number recognition on MNIST dataset as an example. Size of training dataset is 60000. Length of input is 784. Number of hidden nodes is 30. Number of classes is 10. Number of training epochs is 5.

	Scheme 1 Decrypt after evaluating gradients	Scheme 2 Decrypt after evaluating the first linear function	Scheme 3 (present invention) User evaluate the first activation function
Input x_0	<u>Encrypted</u>	<u>Encrypted</u>	<u>Encrypted</u>
The 1 st linear function y_l	<u>Encrypted</u>	Decrypted	<u>Encrypted</u>
The 1 st activation function $x_l = f_l(y_l)$	<u>Encrypted</u>	Decrypted	Decrypted

Gradients	Encrypted	Decrypted	Decrypted
Updated weights	Decrypted	Decrypted	Decrypted

Table 1: Flow comparison of 3 schemes

In many practical cases, clients do not have very powerful computing resources. Therefore, computations at client level should be minimised. The computations

5 in Scheme 3 are slightly more than Scheme 2 but the extra computations are performed on plaintext, which makes the extra computations economical.

	Scheme 1 Decrypt after evaluating gradients	Scheme 2 Decrypt after evaluating the first linear function	Scheme 3 (present invention) User evaluate the first activation function
Computations in client side	Most Decrypting $\sum_{t=1}^m grad^{(t)}$	Least Decrypting y_1 Computing $\sum_{t=1}^m \delta W_1^{(t)}$	Small Decrypting y_1 Computing x_1 and $\sum_{t=1}^m \delta W_1^{(t)}$
Computations in server side	All training computations	All training computations	All training computations excepts activation function.
Efficiency	Slow Lots of non-linear evaluations in ciphertext. Time per input: 5.4 min (estimated) Total time: 3 years (estimated)	Fastest No non-linear evaluations in ciphertext. Time per input: 0.006s Total time: 1710 seconds	Fastest No non-linear evaluations in ciphertext. Time per input: 0.006s Total time: 1714 seconds
Privacy preserving	Best	Worst When training epochs increase, the server may	Good Level of privacy preserving depends on choice

		recover the input vector	of activation function
Comments	Too slow	Dominated by Scheme 3	Most appropriate

Table 2: Performance comparison of 3 schemes

Figure 2 depicts a schematic block diagram of a system 200 for secure training of a neural network according to an embodiment of the present invention, corresponding to the above-mentioned method 100. The system 200 comprises: at least one memory 202; and at least one processor 204 communicatively coupled to the at least one memory 202 and configured to perform the method 100.

It will be appreciated by a person skilled in the art that the at least one processor 204 may be configured to perform various functions or operations through set(s) of instructions (e.g., software modules) executable by the at least one processor 204 to perform various functions or operations. Accordingly, as shown in Figure 2, the system 200 comprises a memory 202 for storing a plurality of modules, the modules comprising: a pre-processing module 206 configured to receive a label and an encrypted input from a client; a homomorphic module 208 configured to perform homomorphic computations on the encrypted input using the neural network to obtain an encrypted intermediate ciphertext and provide the encrypted intermediate ciphertext to the client; a client level computation module 210 configured to receive a non-sensitive decrypted input from the client, wherein the non-sensitive decrypted input corresponds to a decryption of the encrypted intermediate ciphertext and perform computations on the non-sensitive decrypted input using the neural network to obtain a decrypted output; and an output module 210 configured to update weights of the neural network based on the decrypted output.

It will be appreciated by a person skilled in the art that the above-mentioned modules are not necessarily separate modules, and two or more modules may be realized by or implemented as one functional module (e.g., a circuit or a software program) as desired or as appropriate without deviating from the scope of the present invention. For example, two or more of the pre-processing

module 206, the homomorphic module 208, the client level computation module 210, and the output module 212, may be realized (e.g., compiled together) as one executable software program (e.g., software application), which for example may be stored in the at least one memory 202 and executable by the
5 at least one processor 204 to perform the corresponding functions or operations as described herein according to various embodiments.

Various embodiments described herein in context of methods (e.g., the method 100 of secure training of neural network) are analogously valid for the
10 corresponding systems or devices (e.g., the system 200 for secure training of neural network), and vice versa. For example, in various embodiments, the at least one memory 202 may have stored therein the pre-processing module 206, the homomorphic module 208, the client level computation module 210, and the output module 212, which respectively perform the various operations,
15 functions or steps of the method 100 described above, under operation of the processor.

Figure 3a to 3d depict schematic diagrams of a system for secure training of neural network, according to an embodiment of the present invention. According
20 to step 102 of the method 100, a client terminal generates an encrypted input 302 from a decrypted input. The client terminal may also obtain or generate a label for the decrypted input – the label is also necessarily true for the encrypted input. The label specifies the ground truth. For example, in a training dataset for a neural network being trained to categorise or label images based on
25 objects in those images, the label corresponding to an input image (decrypted input) may identify the object shown in that image.

As shown in Figure 3b, the client then transmits the encrypted input 302 and a corresponding label to a server, per step 104, the server then applying the
30 neural network to the encrypted input, per step 106. The neural network applies homomorphic computations on the encrypted data, to obtain an encrypted intermediate ciphertext 304. Since the computations are homomorphic, the intermediate ciphertext is an encryption of the output of the same computations when applied to the corresponding plaintext – i.e., the input prior to encryption.

35

The client receives the encrypted intermediate ciphertext per step 108 and generates a non-sensitive decrypted input (306) at step 110. The non-sensitive, decrypted input is, or corresponds to, a decryption of the encrypted intermediate ciphertext. The non-sensitive decrypted input is then transmitted
 5 to the server per step 112.

Per step 114, as shown in Figure 3d, the server performs computations on the non-sensitive, decrypted input (i.e., non-sensitive plaintext input) to obtain a decrypted output 308. The decrypted output 308 is then used to update weights
 10 of the neural network, per step 116.

Homomorphic encryption scheme

According to an embodiment of the present invention, the method 100 uses
 15 learning with error (LWE)-based secret key encryption scheme (Algorithm 5) and related computations (Algorithm 6 and Algorithm 7).

The learning with errors (LWE) problem is a generalization of learning parity with noise. For positive integers n and $q \geq 2$, a vector $s \in \mathbb{Z}_q^n$, and a probability
 20 distribution $\chi = \chi^n$ over $\mathbb{Z}_q$, let $\mathcal{A}_{s,\chi}$ be the distribution obtained by choosing a vector $a \xleftarrow{\$} \mathbb{Z}_q^n$ uniformly at random and a noise term $e \xleftarrow{\$} \chi$, and outputting $(a, \langle a, s \rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$. The LWE problem is defined as follows.

For an integer $q = q(n)$ and an error distribution $\chi = \chi(n)$ over $\mathbb{Z}_q$, the LWE
 25 problem $LWE_{n,m,q,\chi}$ is defined as: Given m independent samples from $\mathcal{A}_{s,\chi}$, output s with non-negligible probability.

The decisional version is to distinguish between m samples chosen according to $\mathcal{A}_{s,\chi}$ for some uniformly random s and m samples from the uniform distribution
 30 over $\mathbb{Z}_q^n \times \mathbb{Z}_q$.

Algorithm 5 LWE Encryption Scheme $LWE_s^{n,q}(m)$

Input: plaintext m , modulus q , dimension n , secret key $s \in \{-1,0,1\}^n$, error distribution χ .**Output:** LWE ciphertext $(a, b) \in LWE_s^{n,q}(m)$.Sample error $e \xleftarrow{\$} \chi$.5 Sample vector $a \xleftarrow{\$} \mathbb{Z}_n^q$.Define $b := m + e - \langle a, s \rangle \bmod q$.Return (a, b) .

Algorithm 6 Scale multiplication $c \cdot LWE_s^{n,q}(m)$

10 **Input:** LWE ciphertext $(a, b) \in LWE_s^{n,q}(m)$, plaintext $c \in \mathbb{Z}_n^q$.**Output:** LWE ciphertext $(a', b') \in LWE_s^{n,q}(m)$.Compute $a' := c \cdot a \bmod q$.Compute $b' := c \cdot b \bmod q$.Return (a', b') .

15

Algorithm 7 Homomorphic addition $LWE_s^{n,q}(m_1) + LWE_s^{n,q}(m_2)$

Input: LWE ciphertexts $(a_1, b_1) \in LWE_s^{n,q}(m)$ and $(a_2, b_2) \in LWE_s^{n,q}(m)$.**Output:** LWE ciphertext $(a', b') \in LWE_s^{n,q}(m)$.Compute $a' := a_1 + a_2 \bmod q$.20 Compute $b' := b_1 + b_2 \bmod q$.Return (a', b') .

Algorithm 5 is extended to encrypt a L dimension vector m :

$$LWE_s^{n,q}(m) := (LWE_s^{n,q}(m_1), LWE_s^{n,q}(m_2), \dots, LWE_s^{n,q}(m_L)).$$

25

Homomorphic addition and scale multiplication can be extended in a similar way.

Using these algorithms, it has become feasible to homomorphically evaluate the inner product of a plaintext vector and ciphertext vector (Algorithm 8).

Algorithm 8 Homomorphic inner product HE-IP $(c, ct \in LWE_s^{n,q}(m))$

30 **Input:** Plaintext vector c , LWE ciphertext ct of vector m : $ct \in LWE_s^{n,q}(m)$ **Output:** LWE ciphertext $(a', b') \in LWE_s^{n,q}(\langle c, m \rangle)$.Let $L = \dim(c) = \dim(m)$, $a' = 0, b' = 0$.

```

for  $i = 1, \dots, L$  do
    Parse  $(a_i, b_i) \in LWE_s^{n,q}(m_i)$  from ct.
     $\mathbf{a}' = \mathbf{a}' + c_i \mathbf{a}_i$ 
     $b' = b' + c_i b'_i$ .
5 end for
 $\mathbf{a}' = \mathbf{a}' \bmod q$ .
 $b' = b' \bmod q$ .
Return  $(\mathbf{a}', b')$ .

```

10 Privacy analysis

In the interactive training scheme, it has been discussed that the server aims to recover the input vector x from the information obtained by the server during training. It is assumed that the cloud server is *honest-but-curious*, and the analysis focuses on the cryptographic aspect.

A fully connected neural network with one hidden layer is again used as an illustrative example: $(W_1, W_2, \beta_1, \beta_2, f_1(\cdot), f_2(\cdot), L(\cdot))$. The input layer has L nodes, the hidden layer has H nodes and the output layer has C classes (nodes). The problem is considered to be in $\mathbb{R}$. In Scheme 2 and Algorithm 3, the information obtained by the server is a set of linear equations; while in Scheme 3 and Algorithm 4, the information obtained by the server is a set of equations composed by inner-product and activation $f_1(\cdot)$.

In forward propagation, for a fixed input x , the number of equations the server can obtain from one forward propagation is H , i.e., the number of hidden nodes. This is because in each hidden node, an inner product is calculated between a weight vector from W_1 and the input x , then the activation function $f_1(\cdot)$. Suppose there are a total of E training epochs. Since each input is used only once in one epoch, it is concluded that the total number of equations the server can obtain is $T := H \times E$.

Privacy analysis of Scheme 2

In Scheme 2, the T equations the server can obtain from training are all linear equations. For a fixed input $x \in \mathbb{R}^L$, the T equations together is combined to be $Ax = b$, where $A \in \mathbb{R}^{T \times L}$, $b \in \mathbb{R}^T$.

- 5 Case 1: $T = H \times E \geq L$. In this case, the server is able to solve these linear systems and recover the input x . This is the case not considered by existing prior arts, and this case is not privacy-preserving. Therefore, the total training epochs E and the number of hidden nodes H have to be limited.
- 10 Case 2: $T = H \times E < L$. In this case, we know that $Ax = b$ is feasible by referring to linear algebra because the existence of x is clear.

Lemma 1. Let $x^* \in \mathbb{R}^L$ be one of the solutions of $Ax = b$. Define set $Null(A) := \{x \in \mathbb{R}^L \mid Ax = 0\}$. Then the solution set of $Ax = b$ is:

$$15 \quad \{x^* + \zeta \mid \zeta \in Null(A)\}.$$

Proof of Lemma 1. It has been shown that x' is a solution of $Ax = b$ if and only if x' is in this solution set.

- " $\Rightarrow$ ": Assume x' is a solution of $Ax = b$. Then $A(x' - x^*) = 0$, which implies $x' - x^* \in Null(A)$ i.e.,

$$x' \in \{x^* + \zeta \mid \zeta \in Null(A)\}.$$

- " $\Leftarrow$ ": Assume $x' \in \{x^* + \zeta \mid \zeta \in Null(A)\}$. Then we have $A(x' - x^*) = 0$, i.e., $Ax' = Ax^* = b$.

- 25 From Lemma 1 we know that the solution of $Ax = b$ is not unique when $T < L$, so the server cannot recover the input from linear system $Ax = b$. Further, we are able to show that, when $T < L$ the solution set in Lemma 1 is large enough that the server is not able to recover any element in x , unless A satisfies a very strong property.

- 30 Lemma 2. $\forall x \in \{x^* + \zeta \mid \zeta \in Null(A)\}$, the i -th element $x_i \in x$ is fixed if and only if:
- $$\forall \zeta \in Null(A), \zeta_i = 0.$$

Proof of Lemma 2.

- " $\Rightarrow$ ": $\forall x \in \{x * + \zeta | \zeta \in \text{Null}(A)\}$, let $\zeta = x - x^*$. By condition $x_i \in x$ being fixed, we have ζ_i is fixed. Finally note that $0 \in \text{Null}(A)$, it is known that $\zeta_i = 0$ is fixed.
- " $\Leftarrow$ ": The proof is straightforward according to definition $x \in \{x * + \zeta | \zeta \in \text{Null}(A)\}$.

- 5 In a preferred embodiment, Lemma 2 can be extended when the server knows extra information of input x . Particularly, the server knows that $x \in S$ for some set $S \subseteq \mathbb{R}^L$. Then the conclusion will vary from case to case. For example, if the input is the feature vector output by Facenet, then $x \in S = \{\|x\|_2 = 1\}$, i.e., the unit sphere in $\mathbb{R}^L$. Here, $\|\cdot\|_2$ is the l_2 -norm of a vector. In this case, the result is similar to that of Lemma 2.
- 10

Lemma 3. $\forall x \in \{x * + \zeta | \zeta \in \text{Null}(A)\} \cap \{\|x\|_2 = 1\}$, the i -th element $x_i \in x$ is fixed if and only if $\forall \zeta \in \text{Null}(A), \zeta_i = 0$, or $\{x * + \zeta | \zeta \in \text{Null}(A)\}$ has only one element.

- 15 Proof of Lemma 3.

Let $S := \{\|x\|_2 = 1\}$, $P := \{x * + \zeta | \zeta \in \text{Null}(A)\}$. When $|S \cap P| = 0, 1$, the proof is trivial. The case is discussed for when $|S \cap P| > 1$.

Let $\bar{x} := \operatorname{argmin}_{x \in S} \|x\|_2$. It is well-defined because $S \cap P \neq \emptyset$. By definition, $\bar{x}$ has the property that $\bar{x} \perp \text{Null}(A)$. Further, $|S \cap P| > 1$ implies $\|\bar{x}\|_2 < 1$. First, it is

- 20 observed that the set $P = \{x * + \zeta | \zeta \in \text{Null}(A)\} = \{\bar{x} + \zeta | \zeta \in \text{Null}(A)\}$. Then $\forall x \in S \cap P$, $x = \bar{x} + \zeta$, and prove $\bar{x} - \zeta \in S \cap P$. On the other hand, $\bar{x} - \zeta \in P$ is by the definition of P , noting that $-\zeta \in \text{Null}(A)$. On the other hand, $\|\bar{x} - \zeta\|_2^2 = \|\bar{x} + \zeta\|_2^2 - 4 \langle \bar{x}, \zeta \rangle$. By $\bar{x} \perp \zeta$, it can be known that $\bar{x} - \zeta \in S$. Therefore, the i -th element $x_i \in x \in S \cap P$ is fixed if and only if: for any $x = \bar{x} + \zeta$ from $S \cap P$, $\bar{x}_i + \zeta_i = \bar{x}_i - \zeta_i$, i.e., $\zeta_i = 0$.
- 25

Finally, it is proven that given $|S \cap P| > 1$, $\forall \zeta \in \text{Null}(A), \exists c > 0$ s.t. $\bar{x} + c\zeta \in S$. This is because $\bar{x} + c\zeta$ in continues in c . This implies that $\forall \bar{x} + \zeta \in S \cap P$, $\zeta_i = 0$ if and only if $\forall \zeta \in \text{Null}(A), \zeta_i = 0$.

- 30

When $T = L$ and $\text{rank}(A) = L$, we have $\text{Null}(A) = \{0\}$, and Lemma 2 is satisfied. When the assumed dimension of $T = L$ and $\text{rank}(A) = L$, we have $\text{Null}(A) = \{0\}$, Lemma 2 is almost impossible to satisfy. On the one hand, all the matrices satisfying this condition form a set with measure 0 in $\mathbb{R}^{T \times L}$ where $T < L$. On the

other hand, the initial matrix for the training procedure is generated randomly from $\mathbb{R}^{T \times L}$, and thus the probability it will satisfy Lemma 2 during training is negligible.

5 Privacy analysis of Scheme 3

When compared with Scheme 2, the server operation in Scheme 3 knows equation set $f(Ax) = b = (b_1, \dots, b_L)^T$, where $f(\cdot)$ is the activation function. When $f(x)$ is strictly increasing in x , the analysis is the same as in the previous section, because given $f(\cdot)$ and $f(Ax) = b$, the server can recover Ax . An example of such an activation function is the sigmoid function $f(x) = \frac{1}{1+e^{-x}}$.

When $f(x)$ is not strictly increasing in x , e.g., the ReLU function $f(x) = \max\{0, x\}$, the number of linear equations obtained by the server decreases compared with the case in the previous section. $Ax[i]$ is used to denote the i^{th} slot of vector Ax . More concretely, when $f(\cdot)$ is the ReLU function, when $Ax[i] \geq 0$, $Ax[i]$ can be recovered from $b_i = f(Ax[i])$, and when $Ax[i] < 0$, $f(Ax[i])$ is always 0 and it is impossible to obtain any information of $Ax[i]$ except that $Ax[i] < 0$. Then, the server cannot use this to recover the input x .

The number of useful equations relies on the input data, and the neural network weights. Analysis has to be done on case-by-case basis. Figure 4 is a bar chart illustrating the number of useful equations in an experiment on facial recognition neural network, in 10 training epochs. About 1/3 equations are useless and 2/3 equations are useful. This means that the number effective linear equations is $\frac{2}{3}H \times E$, while in Scheme 2 it is $H \times E$. Less effective linear equations usually mean less knowledge server can learn from training phase.

Activation functions

In some cases, the number of effective linear equations obtained by the server may exceed the input vector dimension n . The server may recover the input vector using these linear equations and causing the model unable to preserve privacy. The present methodology provides a solution to this problem by

reducing the number of effective linear equations obtained by the server. In the early years neural networks often used the sign function $\text{sgn}(x) := 1_{x>0}$ as activation function. The sign function can perfectly hide an input, compared with commonly used activation functions like $\text{ReLU}(x) = \max\{x, 0\}$ and $\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}}$.

- 5 If a sign function is applied as the activation function, the server cannot obtain any of the linear equations, and thus the server is not able to recover the input vector. In some embodiments, for additional privacy preservation, the activation function includes multiple sign functions.
- 10 However, the sign function has derivative value 0 almost everywhere, so the common gradient descent algorithm based on backward propagation (BP) cannot be applied directly. In other words, neural network weights cannot be updated unless the training algorithm is modified.
- 15 The present methodology provides a technical solution to these issues by applying a binary neural network to privacy-preserving neural network training, to inhibit the server from recovering the input vector using the collected equations because of the sign function's definition.
- 20 As discussed above, the common gradient descent algorithm based on backward propagation (BP) cannot be applied. In this section, an algorithm is instead proposed to approximate the partial derivative of the sign function. Again, for the sake of clarity, but without an express intention to limit the scope of the present disclosure, the present algorithm is described with reference to a fully
- 25 connected neural network with one hidden layer, whose parameters are $(W_1, W_2, \beta_1, \beta_2)$, and with activation functions $f_1(\cdot) = \text{sgn}(\cdot)$, $f_2(\cdot)$, and *loss function* $L(\cdot, \cdot)$.

In the backward propagation, the server has to evaluate $\frac{\partial L}{\partial w_1}$, which is given by

- 30 $\frac{\partial L}{\partial x_1}$, $\frac{\partial x_1}{\partial y_1}$, and $\frac{\partial y_1}{\partial w_1}$. Note that $x_1 = f_1(y_1) = \text{sgn}(y_1)$, to evaluate the approximated partial derivative of $\frac{\partial x_1}{\partial y_1}$, the following is defined:

$$\tilde{f}_1(x) = \begin{cases} 0, & x \leq 0 \\ x, & 0 < x \leq c, \\ c, & x \geq c \end{cases}$$

for some integer c . It is one kind of hard sigmoid function. Then $\frac{\partial \tilde{f}_1(x)}{\partial x}$ is used to approximate $\frac{\partial f_1(x)}{\partial x}$. This is to define:

$$\frac{\partial \tilde{x}_{1l}}{\partial \gamma_{1j}} := \frac{\partial \tilde{f}_1(\gamma_{1l})}{\partial \gamma_{1j}} = 1_{l=j} \times 1_{\gamma_{1j} \in (0,1]}, \forall l, j \sim,$$

5

which together give the approximated partial derivative of $\frac{\partial x_1}{\partial \gamma_1}$. In training algorithm 9, $\left(\frac{\partial \tilde{x}_{1l}}{\partial \gamma_{1j}}\right)_{l,j}$ is used to evaluate the approximated partial derivative of

$\frac{\partial L}{\partial W_1}$ and $\frac{\partial L}{\partial W_1}$ is used to denote it. Similarly, $\frac{\partial L}{\partial \beta_1}$ is defined.

-
- 10 **Algorithm 9** Training algorithm, setting the first activation function $f_1(\cdot) = \text{sgn}(\cdot)$
-
1. **Input:** Current neural network parameters $(W_1, W_2, \beta_1, \beta_2)$, activation functions $f_1(\cdot) = \text{sgn}(\cdot), f_2(\cdot)$, loss function $\mathcal{L}(\cdot, \cdot)$, batch size m , m encrypted input $\{\underline{x}^{(t)}\}_{t \in [m]}$ and their labels $\{y^{(t)}\}_{t \in [m]}$, learning rate α .
 2. **Output:** Updated neural networks parameters $(W'_1, W'_2, \beta'_1, \beta'_2)$.
 - 15 3. **for** $t = 1, \dots, m$ **do**
 4. Set $\underline{x}_0 = \underline{x}^{(t)}$, $y = y^{(t)}$.
 5. The server homomorphically evaluate $\gamma_1 = W_1 \underline{x}_0 + \beta_1$.
 6. The server sends γ_1 to the user.
 7. The user decrypts γ_1 and sends $x_1 = f_1(\gamma_1)$ to the server.
 - 20 8. The server evaluate activation function $x_1 = f_1(\gamma_1)$.
 9. The server evaluate $\gamma_2 = W_2 x_1 + \beta_2$.
 10. The server evaluate activation function $x_2 = f_2(\gamma_2)$.
 - For n -layer neural networks, repeating step 8 and 9 with input $W_3, \dots, W_n, \beta_3, \dots, \beta_n$ and $f_3, \dots, f_n$.
 - 25 11. The server evaluate loss function $\mathcal{L}^{(t)} = \mathcal{L}(x_2, y)$
 12. The server evaluate gradients:

$$\begin{aligned} & \left(\cup_{t=1}^m \left\{ \frac{\partial \widetilde{\mathcal{L}}^{(t)}}{\partial W_1 \mathbf{x}^{(t)}} \right\}, \sum_{t=1}^m \frac{\partial \mathcal{L}^{(t)}}{\partial W_2}, \sum_{t=1}^m \frac{\partial \widetilde{\mathcal{L}}^{(t)}}{\partial \beta_1}, \sum_{t=1}^m \frac{\partial \mathcal{L}^{(t)}}{\partial \beta_2} \right) \\ & := \left(\cup_{t=1}^m \left\{ \partial \widetilde{W_1 \mathbf{x}^{(t)}} \right\}, \sum_{t=1}^m \partial W_2^{(t)}, \sum_{t=1}^m \partial \widetilde{\beta_1^{(t)}}, \sum_{t=1}^m \partial \beta_2^{(t)} \right) \end{aligned}$$

Unlike in Scheme 1, here the evaluation of

$$\sum_{t=1}^m \partial \widetilde{W_1 \mathbf{x}^{(t)}} = \sum_{t=1}^m \frac{\partial \widetilde{\mathcal{L}}^{(t)}}{\partial W_1 \mathbf{x}^{(t)}} \frac{\partial W_1 \mathbf{x}^{(t)}}{\partial W_1}$$

needs to know $\{\mathbf{x}^{(t)}\}_{t \in [m]}$, while server only know its encryption $\{\mathbf{x}^{(t)}\}_{t \in [m]}$. So server computes the other parts first and asks user's help to get $\sum_{t=1}^m \partial W_1^{(t)}$.

13. **end for**

10 14. Server sends $\cup_{t=1}^m \{\partial W_1 \mathbf{x}^{(t)}\}$ to user.

15. User computes and returns $\{\sum_{t=1}^m \partial W_1^{(t)}\}$ to server.

16. Server updates neural network parameters and returns $(W'_1, W'_2, \beta'_1, \beta'_2)$.

$$W'_1 = W_1 - \alpha \sum_{t=1}^m \partial \widetilde{W_1 \mathbf{x}^{(t)}}, \beta'_1 = \beta_1 - \alpha \sum_{t=1}^m \partial \widetilde{\beta_1^{(t)}}, W'_2 = W_2 - \alpha \sum_{t=1}^m \partial W_2^{(t)}, \beta'_2 = \beta_2 - \alpha \sum_{t=1}^m \partial \beta_2^{(t)}.$$

15

Although using the sign function as activation function can help to preserve privacy when compared with the ReLU function, there is a loss of accuracy because of the approximation of gradients. In this section, a family of functions is described, called the n-step function, to serve as an activation function. A function $f(x)$ is a n-step function if it is non-decreasing, and is a piecewise constant function having n pieces.

Figure 5a and 5b show the examples of a 3-step function and a 5-step function, though any other suitable number of steps can be used. $f(x) = 0$ for $x \leq 0$ and $f(x) = 4$ for $x > 4$. For n-step function, it can be observed that:

$$n \rightarrow +\infty, \quad n\text{-step function} \rightarrow \text{hard sigmoid function} = \begin{cases} 0, & x \leq 0 \\ x, & 0 < x \leq 4 \\ 4, & x > 4 \end{cases}$$

30

When n is large, it can have better approximation, but privacy-preservation in the training scheme decreases; when n is small, it is less accurate, but has better privacy-preservation.

- 5 For 5-step function is used as shown in Figure 5b, the accuracy loss is below 1% and it is acceptable. The upper bound of the n -step function is not fixed. It can be changed according to the data range. For example, a better choice of the upper bound such as 8 may be chosen during experiments.

10 **Communication overhead**

To analyse the communication overhead of the present methodology, for each batch of training, there are two rounds of communications between the server and the user. Let L_{input} be the input size, L_i be the number of nodes in the i -th layer, and m be the batch size. In the first communication, the server sends $m \times L_1$ LWE ciphertexts to the user. The size of each ciphertext is around 2KB, in experimental settings. So, the communication overhead from server to user is $2mL_1$ KB. The user returns mL_1 integers to the server, which are the evaluation results of the activation function applied to the decrypted ciphertexts. The communication overhead from user to server is $4mL_1$ B.

In the second communication, the server sends $m \times L_2 \times L_1$ floating numbers to the user. The communication overhead from server to user is $8mL_1L_2$ B. The user returns $L_{\text{input}}L_1$ floating numbers to the server, which is the sum of m gradients. The communication overhead from user to server is $8L_{\text{input}}L_1$ B. If the gradient matrix contains lots of 0, i.e., it is a sparse matrix, the non-zero value can be recorded and its position indicated instead of sending the whole matrix out. It could help to reduce the communication overhead.

30 The parameters of training a BP network are then entered for MNIST dataset into the calculations, where $L_{\text{input}} = 784$, $L_1 = 30$, $L_2 = 10$, $m = 50$. In the first communication, the size of server-to-user message is $2 \times 50 \times 30 = 3000$ KB = 3 MB. The size of user-to-server message is $4 \times 50 \times 30 = 6000$ B = 6 KB. In the second communication, the size of server-to-user message is $8 \times 50 \times 30 \times$

$10 = 120000 \text{ B} = 117 \text{ KB}$. The size of user-to-server message is $8 \times 784 \times 30 = 188160 \text{ B} = 184 \text{ KB}$.

Evaluation Results

5

All schemes are evaluated on an Intel(R) Xeon(R) Platinum 8480+ CPU which has 56 cores. The security level is at least 80 bits.

The experiments include one of the most popular benchmark datasets, MNIST and a facial recognition dataset collected independently. For MNIST dataset, a BP network with one hidden layer and 30 hidden nodes is trained, which is a commonly used network structure in the area of inference on encrypted data. The average time for training one image is 0.006s and the accuracy loss compared to train in plaintext is only 0.5%.

15

Besides demonstrating that the system is highly efficient on the standard benchmark MNIST dataset, it is also shown that the system can be applied to practical problems, such as facial recognition. As facial recognition is more challenging and requires a deep neural network, transfer learning is applied during the training phase.

20

A pre-trained FaceNet (Schroff et. al., 2015) is chosen as feature extractor by removing the fully-connected layers. A small dataset of 30 people is independently collected as the training dataset and two fully-connected layers are added behind FaceNet for classification. It takes 0.008s per input and achieves 100 % accuracy.

25

MNIST

There are 60000 training examples and 10000 test examples in MNIST dataset for handwritten digit recognition. Each example is a 28×28 gray-level image, which is represented by a vector with length 784. For each point of the image, the value is set to be 1 if the original value is > 0 and set it to be 0 otherwise. The architecture of the neural network is a BP network with one hidden layer and 30 hidden nodes, whose input is 784-dim vector and output has 10 classes.

30

Both the hidden layer and output layer contain a bias. So, the number of parameters to be learned is $(784 \times 30) + 30 + (30 \times 10) + 10 = 23860$.

For the choice of activation functions, a 5-step function is used in the hidden layer. The activation function in the output layer is the Sigmoid function and uses the L^2 norm loss function $\mathcal{L} = \frac{1}{2} \|x_2 - y\|_2^2$, where x_2 is the output of Sigmoid function and y is the goal vector indicated by the standard label. The results are presented in Table 3 below:

Number of trainable parameters			23860
Size of each batch			50
Traditional Machine Learning	Accuracy		94.30% after 5 epochs
	Privacy preserving level		No privacy preserving
Present invention	Accuracy		93.95% after 5 epochs
	Time		1714s in total (5 epochs)
	Communication per batch	1 st time	Server to user: 3MB. User to server: 6KB
		2 nd time	Server to user: 117KB. User to server: 184KB
	Number of communications		2 times per batch, 2400 times per epoch. 12000 times in total.
	Privacy preserving level		High privacy preserving

Table 3: Training on MNIST (1 hidden layer, 30 nodes)

CIFAR-10

CIFAR-10 are labelled subsets of the 80 million tiny images dataset collected by Krizhevsky et. al., 2009. The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. Some samples from CIFAR-10 are included in Figure 6 which illustrates 10 classes in CIFAR-10, as well as random images from each class.

A convolutional neural network with three convolution layers, as shown in Figure 7, was chosen for experiments. The input is an RGB image with a size of 32×32 . The final output layer consists of 10 classes. For the activation functions, a 5-step function is used in the first convolution layer, while ReLU is used for all other activation functions.

With the hybrid HE-based training model, all parameters are made trainable, even when the training data is encrypted. As a concrete implementation of Algorithm 1, the training procedure for one batch is detailed in Algorithm 10.

The convolutional neural network shown in Figure 7 is divided into two parts, $(S_1(\cdot), S_2(\cdot))$. The first convolution layer is included in $S_1(\cdot)$, while all other components are contained in $S_2(\cdot)$.

Algorithm 10 Hybrid HE-based training model of CNN of Figure 7

1. **Input:** Encrypted training sample $\text{Enc}(\mathbf{x})$
 2. The cloud server performs homomorphic computations on encrypted input $\text{Enc}(\mathbf{x}_1) := S_1(\text{Enc}(\mathbf{x}))$, i.e. homomorphically compute the first convolution layer.
 3. The cloud server sends ciphertext $\text{Enc}(\mathbf{x}_1)$ to the client.
 4. The client decrypts the ciphertext $\mathbf{x}_1 = \text{Dec}(\text{Enc}(\mathbf{x}_1))$.
 5. The client computes the activation and Max-Pool of $\mathbf{x}_1$. Let $\mathbf{x}_2 = \text{MaxPool}(5 - \text{step}(\mathbf{x}_1))$.
 6. The client send $\mathbf{x}$ to the cloud server.
 7. The cloud server performs the remaining training computations in $S_2(\mathbf{x}_2)$, and updates the neural network weights.
 8. Move to next training batch.
-

The results are presented in Table 4. A total of 49,000 images are constructed as the training set, 1,000 images as the validation set, and 10,000 images as the test set. The batch size is set to 64 images, resulting in 766 batches per epoch. The training procedure is conducted over 9 epochs.

Number of trainable parameters			81290
Size of each batch			64
Traditional Machine Learning	Accuracy		72.19% after 9 epochs
	Privacy preserving level		No privacy preserving

Present invention	Accuracy		70.47% after 9 epochs
	Time		6.3s per batch, 1.34 hour per epoch. 12 hours in total
	Communication per batch	1 st time	Server to user: 2GB. User to server: 2MB
		2 nd time	Server to user: 2MB. User to server: 3.4KB
	Number of communications		2 times per batch, 1532 times per epoch. 13788 times in total.
	Privacy preserving level		High privacy preserving

Table 4: Training on CIFAR-10 (CNN with 3 convolution layers)

In the line Traditional Machine Learning of Table 4, the library Pytorch is used to finish the training experiment. It may be perceived that an accuracy of 72.19% is insufficient. Better accuracy could be obtained by several optimizations of the neural network structure, the training procedure, or pre-process of training data.

- A more complex deep neural network structure such as ResNet50 or inceptionV3 will have an accuracy above 85%. These models usually have millions of parameters to be trained.
- With transfer learning, one can train a deep neural network on a huge dataset (e.g., ImageNet, which has 1.4 million images). Then one can use the smaller dataset (CIFAR-10) to further fine-tune the pre-trained model.
- Pre-process techniques such as data augmentation and data normalization can also help to obtain faster convergence and thus the outcome can have better accuracy.

The above experiments demonstrate that the additional accuracy loss caused by homomorphic encryption computations and approximated gradients is sufficiently small to remain practical (as shown in Table 4, below 2%). The

experiment is conducted using a simpler CNN with three convolution layers. Pre-training is not included, and the initial parameters of the neural network are randomly generated. Data preprocessing techniques, such as data augmentation, are also excluded from the experiments. Additionally, it is
5 ensured that the security level of the encryption schemes used in this work is at least 80 bits.

Facial recognition

10 Besides the commonly used benchmark datasets, it is demonstrated that the system can be applied to solve practical problems with the help of transfer learning. Photos are collected, and a custom training dataset is established, containing 30 individuals and 835 photos. Although many studies have improved the efficiency of homomorphic convolutions, homomorphic
15 computation of a deep convolutional neural network, which is a widely used structure in facial recognition, remains very slow. To reduce the training latency of privacy-preserving deep convolutional neural network training, transfer learning (Pan et al., 2009) is proposed.

20 *Transfer learning for private neural network training.*

Since the first several convolutional layers of a deep CNN are used to extract features, which are independent of the datasets, the parameters of the convolutional layers of a pre-trained FaceNet (trained on VGGFace2) (Schroff et al., 2015) can be frozen, and the fully-connected layers are re-trained to classify
25 the individuals in the dataset.

Applying transfer learning can:

- 1) Transfer learning reduces the number of trainable layers. i.e., the weights in convolutional layers are fixed, so the training latency can be reduced.
- 30 2) With the help of a pre-trained DNN, we only need a small dataset to train the fully-connected layers.
- 3) If the pre-trained network is open source, i.e., the parameters are open and fixed, then the AI model solution providers can pass it to client. So the expensive ciphertext convolution computations can be converted to cheaper
35 plaintext computations.

- 4) Transfer learning does not hurt the protection of privacy of HE-based neural network training, since the extracted feature will be encrypted before send to the cloud server.
- 5 It is shown how transfer learning is applied in privacy-preserving facial recognition network training in Figure 8. The convolutional layers from the pre-trained FaceNet, which is trained on an unencrypted public dataset, are reused, and the last fully-connected layers are discarded. During the privacy-preserving network training, only the weights in the last two fully-connected layers are updated. Since FaceNet is an open-source network with publicly available parameters, convolution computations can be performed in clear, which is much faster than homomorphic convolution computations in ciphertext. Before the client submits the feature vector to train the fully-connected layers, the extracted feature is encrypted, ensuring that the server does not have access to the feature vector.

The results are presented in Table 5. For the choice of activation function in AC1, the first two lines correspond to training with the ReLU function, while the last line corresponds to training with the 5-step function. The output of the convolutional layers is a vector of length 512. The first fully-connected layer contains 30 nodes, and the output layer (second fully-connected layer) also contains 30 nodes. Both layers include biases. Therefore, the number of trainable parameters is $(512 \times 30) + 30 + (30 \times 30) + 30 = 16320$. Regarding the evaluation of FaceNet, the advancements in GPU technology allow the photos to be batched and processed very efficiently.

Number of trainable parameters			16320
Size of each batch			8
Traditional Machine Learning	Accuracy		100% after 3 epochs
	Privacy preserving level		No privacy preserving
Present invention	Accuracy		100% after 3 epochs
	Time		20s in total (3 epochs)
	Communication per batch	1 st time	Server to user: 480KB. User to server: 1KB

		2 nd time	Server to user: 62KB. User to server: 120KB
	Number of communications		2 times per batch, 210 times per epoch. 630 times in total.
	Privacy preserving level		High privacy preserving

Table 5: Training on facial recognition (Facenet + 2 fully-connected layers)

The privacy-preserving training scheme as disclosed herein is capable of training a deep neural network (DNN) in a short time, enabling a wide range of applications such as facial verification and object classification. With the assistance of this training scheme, machine learning service providers can offer privacy-preserving machine learning on cloud servers. On one hand, computationally expensive training can be offloaded to the cloud, allowing users without powerful machines to benefit from it. On the other hand, user data can be protected by encrypting it before uploading to the server.

Additionally, it increases the possibility of collaboration between different institutions whose data cannot be shared in clear. Such institutions can encrypt their data locally and send the ciphertexts to a third-party machine learning service provider. With the help of the privacy-preserving training scheme, the machine learning service provider can process encrypted data from various institutions as input and train a neural network without the need for decryption. It will be appreciated that many further modifications and permutations of various aspects of the described embodiments are possible. Accordingly, the described aspects are intended to embrace all such alterations, modifications, and variations that fall within the spirit and scope of the appended claims.

Throughout this specification and the claims which follow, unless the context requires otherwise, the word “comprise”, and variations such as “comprises” and “comprising”, will be understood to imply the inclusion of a stated integer or step or group of integers or steps but not the exclusion of any other integer or step or group of integers or steps.

CLAIMS:

1. A computer-implemented method for secure training of a neural network comprising a first set of layers and a second set of layers, the first set of layers and second set of layers being non-overlapping, comprising:
 - 5 receiving a label and an encrypted input from a client;
 - performing homomorphic computations on the encrypted input using the first set of layers to obtain an encrypted intermediate ciphertext;
 - 10 providing the encrypted intermediate ciphertext to the client;
 - receiving a non-sensitive decrypted input from the client, wherein the non-sensitive decrypted input corresponds to an output of an activation function applied to a decryption of the encrypted intermediate ciphertext;
 - 15 performing computations on the non-sensitive decrypted input using the second set of layers to obtain a decrypted output; and
 - updating weights of the neural network based on the decrypted output.
- 20 2. The computer-implemented method of claim 1, wherein receiving the non-sensitive decrypted input comprises receiving a decryption of encrypted gradients derived from the encrypted intermediate ciphertext.
- 25 3. The computer-implemented method of claim 1 or 2, wherein performing homomorphic computations on the encrypted input comprises applying a first linear function, utilising weights of the first set of layers, to the encrypted input to produce the encrypted intermediate ciphertext.
- 30 4. The computer-implemented method of any one of claim 1 to 3, wherein the activation function includes multiple sign functions or multiple step functions.
5. The method of any one of claims 1 to 4, wherein performing computations on the non-sensitive decrypted input comprises applying an activation

function, utilising weights of the second set of layers, to the non-sensitive decrypted input.

- 5 6. The method of any one of claims 1 to 5, wherein the neural network comprises at least one further set of layers, the first set of layers, second set of layers and each further set of layers being non-overlapping, wherein performing computations on the non-sensitive decrypted input comprises sequentially applying the second set of layers, to the non-sensitive decrypted input, and then each further set of layers to an output of the second set of layers or an output of a previous said further set of layers, the method comprising evaluating a loss function over an output of the neural network and the label.
- 10
- 15 7. The method of any one of claims 1 to 6, wherein each said set of layers comprises a single layer.
- 20 8. A computer-implemented method for secure training of a neural network comprising a first set of layers and a second set of layers, the first set of layers and second set of layers being non-overlapping, comprising:
- 25 generating, at a client terminal, an encrypted input from a decrypted input;
- transmitting the encrypted input and a corresponding label to a server;
- performing homomorphic computations, at the server, on the encrypted input using the first set of layers to obtain an encrypted intermediate ciphertext;
- 30 providing the encrypted intermediate ciphertext to the client;
- generating, at the client, a non-sensitive decrypted input corresponding to an output of an activation function applied to a decryption of the encrypted intermediate ciphertext, and transmitting the non-sensitive decrypted input to the server;
- performing, at the server, computations on the non-sensitive decrypted input using the second set of layers to obtain a decrypted output; and

updating weights of the neural network based on the decrypted output.

- 5 9. The computer-implemented method of claim 8, wherein the client derives encrypted gradients from the encrypted intermediate ciphertext, and generates decrypted gradients from the encrypted gradients, the non-sensitive decrypted input comprising the decrypted gradients.
- 10 10. The computer-implemented method of claim 8 or 9, wherein performing the homomorphic computations on the encrypted input comprises applying a first linear function, utilising weights of the first set of layers, to the encrypted input to produce the encrypted intermediate ciphertext.
- 15 11. The computer-implemented method of any one of claims 8 to 10, wherein the activation function includes multiple sign functions or multiple step functions.
- 20 12. The computer-implemented method of any one of claims 8 to 11, wherein updating weights of the neural network based on the decrypted output comprises evaluating, at the server, a loss function based on the decrypted output and the label.
- 25 13. The computer-implemented method of claim 12, wherein updating weights of the neural network based on the decrypted output comprises evaluating gradients of the loss function and weights of the neural network at both the server and client, and updating the weights based on the evaluation.
- 30 14. The computer-implemented method of any one of claims 8 to 13, wherein performing computations on the non-sensitive decrypted input comprises applying an activation function, utilising weights of the second set of layers, to the non-sensitive decrypted input.
- 35 15. The computer-implemented method of any one of claims 8 to 14, wherein the neural network comprises at least one further set of layers, the first

5 set of layers, second set of layers and each further set of layers being non-overlapping, wherein performing computations on the non-sensitive decrypted input comprises sequentially applying the second set of layers, to the non-sensitive decrypted input, and then each further set of layers to an output of the second set of layers or an output of a previous said further set of layers, the method comprising evaluating a loss function over an output of the neural network and the label.

10 16. The computer-implemented method of any one of claims 8 to 15, wherein each said set of layers comprises a single layer.

15 17. A system for secure training of a neural network comprising a first set of layers and a second set of layers, the first set of layers and second set of layers being non-overlapping, comprising:

a processor;

a memory storing a plurality of modules, the modules comprising:

a pre-processing module configured to receive a label and an encrypted input from a client;

20 a homomorphic module configured to perform homomorphic computations on the encrypted input using the first set of layers to obtain an encrypted intermediate ciphertext and provide the encrypted intermediate ciphertext to the client;

25 a server level computation module configured to receive a non-sensitive decrypted input from the client, wherein the non-sensitive decrypted input corresponds to an output of an activation function applied to a decryption of the encrypted intermediate ciphertext and perform computations on the non-sensitive decrypted input using the second set of layers to obtain a decrypted output; and

30 an output module configured to update weights of the neural network based on the decrypted output.

35 18. The system according to claim 19, wherein receiving a non-sensitive decrypted input comprises receiving a decryption of encrypted gradients derived from the encrypted intermediate ciphertext.

19. The system according to claim 19 or 20, wherein performing homomorphic computations on the encrypted input comprises applying a first linear function, utilising weights of the first set of layers, to the encrypted input to produce the encrypted intermediate ciphertext.

20. The system according to claim 19, wherein the activation function includes multiple sign functions or multiple step functions.

21. The system according to any one of claims 17 to 20, wherein the homomorphic module performs computations on the non-sensitive decrypted input comprises applying an activation function, utilising weights of the second set of layers, to the non-sensitive decrypted input.

22. The system of any one of claims 17 to 21, wherein the neural network comprises at least one further set of layers, the first set of layers, second set of layers and each further set of layers being non-overlapping, wherein server level computation module performs computations on the non-sensitive decrypted input by sequentially applying the second set of layers, to the non-sensitive decrypted input, and then each further set of layers to an output of the second set of layers or an output of a previous said further set of layers, and evaluates a loss function over an output of the neural network and the label.

23. The system of any one of claims 17 to 22, wherein each said set of layers comprises a single layer.

24. A system for secure training of a neural network comprising a first set of layers and a second set of layers, the first set of layers and second set of layers being non-overlapping, comprising:
a processor;
a memory storing a plurality of modules, the modules comprising:
a client level pre-processing module configured to generate an encrypted input from a decrypted input at a client terminal and to transmit the encrypted input and a corresponding label to a server;

- a homomorphic module configured to perform homomorphic computations on the encrypted input at the server using the first set of layers to obtain an encrypted intermediate ciphertext, and providing the encrypted intermediate ciphertext to the client;
- 5 a client level computation module configured to generate a non-sensitive decrypted input corresponding to an output of an activation function applied to a decryption of the encrypted intermediate ciphertext, and transmitting the non-sensitive decrypted input to the server;
- 10 a server level computation module configured to perform computations on the non-sensitive decrypted input using the second set of layers to obtain a decrypted output; and
- an output module configured to update weights of the neural network based on the decrypted output.
- 15
25. The system according to claim 24, wherein the client derives encrypted gradients from the encrypted intermediate ciphertext, and generates decrypted gradients from the encrypted gradients, the non-sensitive decrypted input comprising the decrypted gradients.
- 20
26. The system according to claim 24 or 25, wherein performing homomorphic computations on the encrypted input comprises applying a first linear function, utilising weights of the first set of layers, to the encrypted input to produce the encrypted intermediate ciphertext.
- 25
27. The system according to claim 26, wherein the activation function includes multiple sign functions or multiple step functions.
- 30
28. The system according to any one of claims 25 to 27, wherein updating weights of the neural network based on the decrypted output comprises evaluating, at the server, a loss function based on the decrypted output and the label.
- 35
29. The system according to claim 28, wherein updating weights of the neural network based on the decrypted output comprises evaluating gradients

of the loss function and weights of the neural network at both the server and client, and updating the weights based on the evaluation.

5 30. The system of any one of claims 25 to 29, wherein the homomorphic module performs computations on the non-sensitive decrypted input comprises applying an activation function, utilising weights of the second set of layers, to the non-sensitive decrypted input.

10 31. The system of any one of claims 25 to 30, wherein the neural network comprises at least one further set of layers, the first set of layers, second set of layers and each further set of layers being non-overlapping, wherein the server level computation module performs computations on the non-sensitive decrypted input by sequentially applying the second set of layers, to the non-sensitive decrypted input, and then each further set
15 of layers to an output of the second set of layers or an output of a previous said further set of layers, and evaluates a loss function over an output of the neural network and the label.

20 32. The system of any one of claims 25 to 31, wherein each said set of layers comprises a single layer.