GELU-Net: A Globally Encrypted, Locally Unencrypted Deep Neural Network for Privacy-Preserved Learning

Abstract

Privacy is a fundamental challenge in many smart applications that depend on data aggregation and collaborative learning across different entities. In this paper, we propose a novel privacy-preserved architecture where clients can collaboratively train a deep model while preserving the privacy of each client's data. Our main strategy is to carefully partition a deep neural network onto two non-colluding parties. One party performs linear computations on encrypted data utilizing a less complex homomorphic cryptosystem, while the other executes non-polynomial computations in plaintext but in a privacy-preserved manner. We analyze security and compare the communication/computation complexity with the existing approaches. Our extensive experiments on different datasets demonstrate not only stable training without accuracy loss, but also 14 to 35 times speedup compared to the state-of-the-art system, CryptoNets.

1 Introduction

Privacy is a fundamental challenge in many smart applications that depend on data aggregation and collaborative learning across different entities. Existing endeavors take different directions to address the privacy issue. Two major directions are differential privacy [Shokri and Shmatikov, 2015] and fully homomorphic encryption [Gilad-Bachrach *et al.*, 2016]. Differential privacy injects noise into query results to avoid inferring information about any specific record. However, it needs careful calibration to balance privacy and model usability. Further, private attributes still remain in plaintext so users may still have security concerns. A more promising solution comes from the recent advance in fully homomorphic encryption (FHE) [Gentry, 2009]. It allows users to encrypt data with the public key and offload computation to the cloud. The cloud computes on the encrypted data and generates encrypted results. Without the secret key, cloud simply serves as a computation platform but cannot access any user information. This powerful technique has been integrated with deep learning in the pioneering work of [Gilad-Bachrach *et al.*, 2016], known as CryptoNets, which built a convolutional neural network on FHE to process inference queries. However, it has three fundamental problems: P1) FHE is extremely costly in computation, thus unsuitable for large-scale neural networks; P2) the (nonlinear) activation functions are not cryptographically computable, hence have to be approximated by polynomials, leading to degraded model accuracy; P3) only inference is supported, but training is unstable due to approximated polynomial activation functions. Privacy-preserved training is considered in [Yuan and Yu, 2014], which also utilizes polynomial approximation (e.g., Taylor expansion) to circumvent the hardness of activations. Thus, it suffers from the same problems of CryptoNets such as accuracy loss and training instability.

In this paper, we propose a novel privacy-preserved learning architecture that resolves the three problems of existing FHE-based approaches such as CryptoNets. It is dubbed *Globally Encrypted, Locally Unencrypted Deep Neural Network* (GELU-Net). The intrinsic strategy is to split each neuron into linear and nonlinear components and implement them separately on non-colluding parties. Linear computations are conducted based on a partially homomorphic cryptosystem, i.e., Paillier [Paillier, 1999]. It offers sufficient security strength to keep data *globally encrypted*, and at the same time is significantly more efficient than FHE used in CryptoNets. As such, it solves P1. Note that it would be impossible to use Paillier without the novel design of separating the two components, because Paillier does not support nonlinear polynomials. The cryptographically incomputable activations are resolved in a *locally non-encrypted* yet still privacy-preserved manner to retain the original accuracy, which solves P2 and P3. GELU-Net can effectively perform model training without the stability and accuracy loss issues. We apply techniques such as random masking to surgically inject privacy-preserving components into the backpropagation algorithm, at minimal computation/communication cost while ensuring loss-free model accuracy.

Our contributions are summarized as follows: 1) we propose a novel privacy-preserved, computationally efficient, homomorphic encryption based learning architecture, GELU-Net, which successfully resolves the three major problems of CryptoNets and other similar existing approaches; 2) we carry out security analysis and compare the complexity of GELU-Net with existing approaches; 3) we conduct extensive experiments on common datasets and demonstrate that GELU-Net achieves 14 to 35 times speed-up compared to

CryptoNets in different environments.

2 Preliminaries

2.1 DNN Training

We first revisit the backpropagation algorithm in DNN. For simplicity, the following discussion is based on fully connected network. It can be readily extended to Convolutional Neural Networks as shown in our experiments. The problem is to classify data samples $\boldsymbol{x}$ into multiple classes. For an n-layer network, $\boldsymbol{w}_i$ and $\boldsymbol{b}_i$ are the weights and biases corresponding to the i-th layer. The forward propagation calculates weighted-sums $\boldsymbol{w}_i\boldsymbol{a}_{i-1} + \boldsymbol{b}_i$ for the i-th layer, where $\boldsymbol{a}_{i-1}$ is the activation from the $(i-1)$-th layer and $\boldsymbol{a}_0 = \boldsymbol{x}$. The output layer adopts the *softmax* function to map high-dimensional vectors into prediction probabilities, i.e., $\boldsymbol{y}$. Once the forward propagation generates $\boldsymbol{y}$, the distance between the prediction and the true label $\boldsymbol{t}$ is calculated as the cost. The weight and bias are then updated based on the backpropagation $\boldsymbol{w}_i = \boldsymbol{w}_i - \eta\Delta\boldsymbol{w}_i$ and $\boldsymbol{b}_i = \boldsymbol{b}_i - \eta\Delta\boldsymbol{b}_i$, where η is the learning rate. The gradients $\Delta\boldsymbol{w}_i$ and $\Delta\boldsymbol{b}_i$ are calculated as follows:

$$\Delta\boldsymbol{w}_i = \boldsymbol{a}_{i-1}\boldsymbol{\delta}_i, \Delta\boldsymbol{b}_i = \boldsymbol{\delta}_i, \tag{1}$$

where $\boldsymbol{\delta}_i = \boldsymbol{\delta}_{i+1}\boldsymbol{w}_{i+1}\boldsymbol{a}_i\,(1-\boldsymbol{a}_i)$ for sigmoid activation and $\boldsymbol{\delta}_{n-1} = \boldsymbol{y} - \boldsymbol{t}$ for cross-entropy loss.

2.2 Paillier Homomorphic Encryption

Homomorphic encryption allows secure computation over encrypted data. To cope with operations in DNN (mainly multiplication and addition), we adopt a well-known partially homomorphic encryption system called *Paillier* [Paillier, 1999]. *Paillier* supports unlimited number of additions between ciphertext, and multiplication between a ciphertext and a scalar constant. Given ciphertext $\{[\![M_1]\!], [\![M_2]\!], \cdots, [\![M_g]\!]\}$ and scalar constants $\{s_1, s_2, \cdots, s_g\}$, we can calculate $([\![M_1]\!] \otimes s_1) \oplus ([\![M_2]\!] \otimes s_2) \oplus \cdots \oplus ([\![M_g]\!] \otimes s_g)$ without knowing the plaintext message. Here, $[\![M_g]\!]$ represents the ciphertext. $\oplus$ is the homomorphic addition with ciphertext and $\otimes$ is the homomorphic multiplication between a ciphertext and a scalar constant. Cryptosystems are defined over integers $\mathbb{Z}_N$ whereas model parameters in machine learning are typically implemented in floating point (FP) numbers for high accuracy. To bridge such gap, FP is encoded into integers before encryption and decoded to FP after decryption.

3 Overview of GELU-Net Architecture

In this research, we adopt the semi-honest model, which is a standard adversary model that many state-of-the-art privacy-preserving designs have built upon [Chen and Zhong, 2009; Zhang *et al.*, 2017]. In such model, each participant follows the protocol but is free to use what it sees to learn private information from others. It is a natural fit for many smart applications since each participant wants to benefit from each other's data so they would follow the protocol in order to obtain correct results. We assume the participants are authenticated and do not attempt to maliciously disrupt the learning process, e.g. add adversarial samples [Papernot *et al.*, 2017].

Consider a centralized *server* and multiple *clients* with private data. We assume the clients also have certain computation power. The server runs a neural network model that is shared by all clients for collaborative learning. Similar to previous work including CryptoNets, the model parameters are usually kept in plaintext on the server for efficiency.

In CryptoNets, the entire neural network is implemented on the server based on FHE operations. The private data are encrypted by the clients using FHE. Each client sends the encrypted data to the server, which runs the model and returns encrypted inference result to the client.

In GELU-Net, the overall neural network model is still implemented on the server. However, the nonlinear activation is securely outsourced and resolved in a non-encrypted form. More specifically, each client uses *Paillier* to encrypt its private data. Similar to CryptoNets, the encrypted data are sent to the neural network model on the server. The server is able to perform most computation based on the partially homomorphic encrypted data. However, it cannot compute the activation function, which is non-polynomial and thus unsupported by the Paillier cryptosystem. To this end, the input for the activation (i.e., the intermediate weighted-sum in encrypted form) is sent back to the *client*, which, as the corresponding data owner, has the key and thus can decrypt the inputs, execute the activation, re-encrypt the result, and send it to the *server* for the next layer.

The proposed GELU-Net has two prominent advantages as summarized below. The first advantage of this design is to enable activation without approximation, because it is now computed by the client in plaintext form. This ensures free of accuracy loss and the desired stability in training, thus addressing problems P2 and P3 introduced in Sec. 1. The second advantage is the significantly improved computation efficiency. The neural network runs much faster than CryptoNets, solving problem P1.

While the first advantage is obvious, the second seems anti-intuitive at the first glance. Given the proposed GELU-Net requires communication between server and client as well as decryption and encryption for computing each activation, would it become a performance bottleneck? Surprisingly, it not only is not the performance bottleneck, but also contributes significant performance gain. To fully understand such potential, we conduct a set of initial experiments on a commodity desktop using the Paillier package[1], and compare it with FHE implemented by Microsoft's SEAL Library[2]. Table 1 shows the different computation times of an activation function, approximated by square (that is used in CryptoNets and involves FHE multiplication between two ciphertext), 5-th order Taylor expansion (a better approximation using FHE), and our proposed approach where the activation is securely outsourced. The results show that, despite the cost paid for communication and encryption/decryption, the computation of an activation in GELU-Net is 10 times faster than

[1] Paillier Cryptosystem (in Python), https://github.com/n1analytics/python-paillier

[2] Simple Encrypted Arithmetic Library, https://www.microsoft.com/en-us/research/project/simple-encrypted-arithmetic-library

Algorithm 1: Privacy Preserved Forward Propagation

Input: *Client*: Data $\boldsymbol{x}$, set gradient/random accumulator $\Delta \boldsymbol{w}_i^c, \Delta \boldsymbol{b}_i^c, \boldsymbol{r}_i^{wc}, \boldsymbol{r}_i^{bc}$ to $\mathbf{0}$. *Server*: Initialize model, training bound d_{max}. Record initial parameters $\boldsymbol{w}_i^{\mathrm{I}}$ and $\boldsymbol{b}_i^{\mathrm{I}}$ (Section 4.4)

Output: Softmax output $\boldsymbol{y}$

for $d = 1, 2, 3, \cdots, d_{max}$ **do**
  Client: $\boldsymbol{a}_0 \leftarrow \boldsymbol{x}_d$
  for $i = 1, 2, \cdots, n-1$ **do**
    Client: Encrypt $\boldsymbol{a}_{i-1}$ with PK_c as $[\![\boldsymbol{a}_{i-1}]\!]_c$ and send it to server
    Server: Compute $[\![\widetilde{\boldsymbol{z}}_i]\!]_c \leftarrow (\widetilde{\boldsymbol{w}}_i \otimes [\![\boldsymbol{a}_{i-1}]\!]_c) \oplus \widetilde{\boldsymbol{b}}_i$ and send $[\![\widetilde{\boldsymbol{z}}_i]\!]_c$ to client
    Client: Decrypt $[\![\widetilde{\boldsymbol{z}}_i]\!]_c$ with SK_c, call Algorithm 4 to remove randomness in $\widetilde{\boldsymbol{z}}_i$
    if $i = n-1$ **then**
      Client: $\boldsymbol{y} \leftarrow e^{\boldsymbol{z}_i} / \sum_j e^{\boldsymbol{z}_i}$ (softmax)
    else
      Client: $\boldsymbol{a}_i \leftarrow f(\boldsymbol{z}_i)$ (next layer)
  Call Algorithm 2 for backpropagation

Algorithm 2: Privacy Preserved Backpropagation

Server: Encrypt $\frac{1}{\eta}$ with PK_s as $[\![\frac{1}{\eta}]\!]_s$ and send it to client
Client: For the last layer $(n-1)$, compute $\boldsymbol{\delta}_{n-1} \leftarrow \boldsymbol{y} - \boldsymbol{t}$, $\Delta \boldsymbol{w}_{n-1} \leftarrow \boldsymbol{a}_{n-2}\boldsymbol{\delta}_{n-1}$, $\Delta \boldsymbol{b}_{n-1} \leftarrow \boldsymbol{\delta}_{n-1}$,
$\Delta \boldsymbol{w}_{n-1}^c \leftarrow \Delta \boldsymbol{w}_{n-1}^c + \Delta \boldsymbol{w}_{n-1}$, $\Delta \boldsymbol{b}_{n-1}^c \leftarrow \Delta \boldsymbol{b}_{n-1}^c + \Delta \boldsymbol{b}_{n-1}$
for $i = n-2, n-3, \cdots, 1$ **do**
  Client: Encrypt $\boldsymbol{\delta}_{i+1}$ with PK_c as $[\![\boldsymbol{\delta}_{i+1}]\!]_c$ and send it to server
  Server: Compute $[\![\widetilde{\boldsymbol{q}}_{i+1}]\!]_c \leftarrow [\![\boldsymbol{\delta}_{i+1}]\!]_c \otimes \widetilde{\boldsymbol{w}}_{i+1}$, and send it to client
  Client: Decrypt $[\![\widetilde{\boldsymbol{q}}_{i+1}]\!]_c$ with SK_c, call Algorithm 4 to remove randomness in $\widetilde{\boldsymbol{q}}_{i+1}$,and calculate $\boldsymbol{\delta}_i \leftarrow \boldsymbol{\delta}_{i+1}\boldsymbol{w}_{i+1}\boldsymbol{a}_i(1-\boldsymbol{a}_i)$, $\Delta \boldsymbol{w}_i \leftarrow \boldsymbol{a}_{i-1}\boldsymbol{\delta}_i$, $\Delta \boldsymbol{b}_i \leftarrow \boldsymbol{\delta}_i$.
  Update $\Delta \boldsymbol{w}_i^c \leftarrow \Delta \boldsymbol{w}_i^c + \Delta \boldsymbol{w}_i$, $\Delta \boldsymbol{b}_i^c \leftarrow \Delta \boldsymbol{b}_i^c + \Delta \boldsymbol{b}_i$
  if $d < d_{max}$ **then**
    Client: Call Algorithm 3 to mask $\Delta \boldsymbol{w}_i$ and $\Delta \boldsymbol{b}_i$ as $[\![\Delta \widetilde{\boldsymbol{w}}_i]\!]_s$ and $[\![\Delta \widetilde{\boldsymbol{b}}_i]\!]_s$, send to server
    Server: Decrypt $[\![\Delta \widetilde{\boldsymbol{w}}_i]\!]_s$ and $[\![\Delta \widetilde{\boldsymbol{b}}_i]\!]_s$ with SK_s, and update $\widetilde{\boldsymbol{w}}_i = \widetilde{\boldsymbol{w}}_i - \eta \Delta \widetilde{\boldsymbol{w}}_i$ and $\widetilde{\boldsymbol{b}}_i \leftarrow \widetilde{\boldsymbol{b}}_i - \eta \Delta \widetilde{\boldsymbol{b}}_i$
Call Algorithm 1 for the next iteration or call Algorithm 5 to update model parameter on server when finish

the square approximation used in CryptoNets and about 180 times faster than the 5-th order approximation.

Scheme	Communication	Crypto	Activation	Total
Square	0	0	90.6	90.6
5-th order	0	0	1619.6	1619.6
GELU-Net	5	3.7	0.2	**8.9**

Table 1: Comparison of activation under different schemes (ms).

The above discussion is based on activation only. As to be discussed in Sec. 5, the overall performance gain is even higher, because other functions of GELU-Net are also implemented by *Paillier*, which enjoys significantly lower complexity than FHE as shown in [Morris, 2013; Hu, 2013]. To this end, our motivation is to avoid FHE as long as *Paillier* is sufficient to meet privacy requirement. This would significantly improve computation efficiency and accordingly boost the overall performance. It is worth mentioning that running the entire neural network model on a client is not an option since we aim to perform collaborative learning, i.e., building a model utilizing the data from all clients.

4 Privacy-Preserved Learning Algorithms

In this section, we elaborate the proposed privacy-preserved learning algorithms. For lucid presentation, the following description is based on training between a client and a server. The same process repeats for all clients. In Paillier, given a public key pair (PK_u, SK_u), a vector of ciphertext is denoted as $[\![\boldsymbol{x}_i]\!]_u$ encrypted by public key PK_u. Initially, the client and server generate key pairs (PK_c, SK_c) and (PK_s, SK_s) respectively and publish their public keys. The proposed scheme consists of privacy-preserved forward propagation (Algorithm 1) and backpropagation (Algorithm 2) as described below.

4.1 Privacy-Preserved Forward Propagation

The forward propagation is summarized in Algorithm 1. The client first encrypts the data with PK_c and sends it to the server. The weighted sum is homomorphically calculated by the server, $[\![\widetilde{\boldsymbol{z}}_i]\!]_c = (\widetilde{\boldsymbol{w}}_i \otimes [\![\boldsymbol{a}_{i-1}]\!]_c) \oplus \widetilde{\boldsymbol{b}}_i$, *which can be carried out by Paillier, since only one quantity is in the encrypted form*. To prevent the server from inferring activations and data during the backpropagation (which will be discussed next), random masks are applied on $\boldsymbol{w}_i$ and $\boldsymbol{b}_i$ (denote by $\widetilde{\boldsymbol{w}}_i$ and $\widetilde{\boldsymbol{b}}_i$, respectively). The encrypted weighted-sum $[\![\widetilde{\boldsymbol{z}}_i]\!]_c$ with random masks is sent back to the client for computing activation. The client calls Algorithm 4 to remove randomness in $\widetilde{\boldsymbol{z}}_i$ and compute the activation for the next layer. The process repeats until the final layer is reached.

Note that in each layer i, the client can accumulate $\boldsymbol{a}_{i-1}$ and $\boldsymbol{z}_i$ over several iterations to solve the linear equation $\boldsymbol{w}_i\boldsymbol{a}_{i-1} + \boldsymbol{b}_i = \boldsymbol{z}_i$ for $\boldsymbol{w}_i$ and $\boldsymbol{b}_i$. In an iteration, a number of m linear equations can be established (where m is the number of neuron in the layer). There are $m^2 + m$ unknowns including m^2 weighted connections and m biases. In the next iteration, an additional m equations are established while there is only one more unknown, i.e., the learning rate η. This is because $\boldsymbol{w}_i = \boldsymbol{w}_i - \eta \Delta \boldsymbol{w}_i$ and $\Delta \boldsymbol{w}_i$ is known by the client during backpropagation. Thus, the client can extract the model after $m + 2$ iterations, which can accordingly cause leakage of the training data [Tramèr *et al.*, 2016]. To address this problem, the server imposes a bound d_{max} randomly selected between $(1, m + 2)$ that a client can be continuously trained. If d_{max} is reached, the next client is selected. The server can always return to the same client for training at a later time, but not continuously exceeding d_{max} (see Section 4.5 for detailed analysis).

4.2 Privacy-Preserved Backpropagation

As illustrated in Algorithm 2, backpropagation starts from the last layer $i = n - 1$ to compute the error $\boldsymbol{\delta}_i$ between softmax prediction $\boldsymbol{y}$ and true label $\boldsymbol{t}$. Note that Paillier can be used as all computations involve at most one quantity in the encrypted

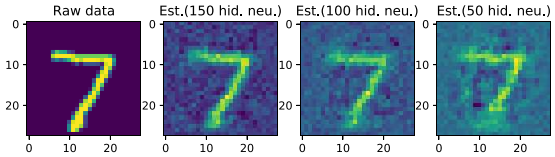

Figure 1: Server reconstructs training data via activations.

form. Then the error is propagated backward throughout the network via gradients $\Delta \boldsymbol{w}_i$ and $\Delta \boldsymbol{b}_i$ for all the layers. In order to correctly update weights on the server, the client must sent private gradients to the server. Revealing such private gradients to the server can cause privacy leaks. To this end, the client calls Algorithm 3 to protect the gradients by random masking before sending them to the server.

On the other hand, the random mask should be removed by the client before the nonlinear activation; otherwise, it would be difficult to recover the original value after activation. To achieve this, the client calls Algorithm 4 to recover $\boldsymbol{q}_i$ from masked $\widetilde{\boldsymbol{q}}_i = \boldsymbol{\delta}_i \widetilde{\boldsymbol{w}}_i$. The client does this in each iteration so the error does not accumulate and there is no need to keep track of it. The client only needs to track the sum of the correct gradients $\Delta \boldsymbol{w}_i^c$ and $\Delta \boldsymbol{b}_i^c$, as well as the sum of injected randomness $\boldsymbol{r}_i^{wc}$ and $\boldsymbol{r}_i^{bc}$ in each gradient update for final model update in Algorithm 5 when training finishes.

4.3 Secure Gradient Updates

The gradients should be protected during backpropagation. Otherwise, the server can similarly establish $\Delta \boldsymbol{w}_i = \boldsymbol{a}_{i-1}\boldsymbol{\delta}_i$, $\Delta \boldsymbol{b}_i = \boldsymbol{\delta}_i$ from the received gradients and quickly derive the activations. From those private activations, the server can further invert the neural network to reconstruct user data [Mahendran and Vedaldi, 2015]. Fig. 1 shows an example to reconstruct private data of handwritten digits. For fully connected networks, the server can simply utilize the Moore−Penrose inverse [Campbell and Meyer, 2009] to estimate data $\boldsymbol{x}$ by, $\widehat{\boldsymbol{x}} = \boldsymbol{w}_1^{\mathrm{T}}(\boldsymbol{w}_1\boldsymbol{w}_1^{\mathrm{T}})^{-1}(\boldsymbol{z}_1 - \boldsymbol{b}_1)$, where $\boldsymbol{z_1} = f^{-1}(\boldsymbol{a_1})$ is the inverse of the activation function from the first layer. To protect the gradients, random vectors are introduced to prevent the server from deriving activation and user data. For layer i, random vectors $\boldsymbol{r}_i^w$ and $\boldsymbol{r}_i^b$ (uniformly distributed over $\mathbb{Z}_N$) are generated by the client. Using the learning rate encrypted by the server $[\![\frac{1}{\eta}]\!]_s$, the client injects the randomness into the encrypted gradients by homomorphically computing $[\![\Delta\widetilde{\boldsymbol{w}}_i]\!]_s = \Delta \boldsymbol{w}_i \oplus ([\![\frac{1}{\eta}]\!]_s \otimes \boldsymbol{r}_i^w)$ and $[\![\Delta\widetilde{\boldsymbol{b}}_i]\!]_s = \Delta \boldsymbol{b}_i \oplus ([\![\frac{1}{\eta}]\!]_s \otimes \boldsymbol{r}_i^b)$ for weights and biases. The server decrypts the masked gradients by SK_s and blindly updates the parameters as,

$$\begin{aligned} \widetilde{\boldsymbol{w}}_i &= \widetilde{\boldsymbol{w}}_i - \eta(\Delta \boldsymbol{w}_i + \boldsymbol{r}_i^w/\eta) = \widetilde{\boldsymbol{w}}_i - \eta\Delta \boldsymbol{w}_i - \boldsymbol{r}_i^w, \\ \widetilde{\boldsymbol{b}}_i &= \widetilde{\boldsymbol{b}}_i - \eta(\Delta \boldsymbol{b}_i + \boldsymbol{r}_i^b/\eta) = \widetilde{\boldsymbol{b}}_i - \eta\Delta \boldsymbol{b}_i - \boldsymbol{r}_i^b. \end{aligned} \quad (2)$$

In this way, the server is oblivious of the actual weights so has no way to figure out the activations (detailed proofs in Section 4.5). Note that random errors are accumulated at the server in each iteration. To perform activation on the actual weighted-sum, the client needs to remove those randomness in $[\![\widetilde{\boldsymbol{z}}_i]\!]_c$ during forward propagation and $[\![\boldsymbol{\delta}_{i+1}]\!]_c \otimes \widetilde{\boldsymbol{w}}_{i+1}$ in backpropagation. Eq. (2) shows that the actual weights/biases on server are $\boldsymbol{w}_i - \boldsymbol{r}_i^{wc}$ and $\boldsymbol{b}_i - \boldsymbol{r}_i^{bc}$ after each update. In forward propagation, to recover $\boldsymbol{z}_i$ from $\widetilde{\boldsymbol{z}}_i = (\boldsymbol{w}_i - \boldsymbol{r}_i^{wc})\boldsymbol{a}_{i-1} + \boldsymbol{b}_i - \boldsymbol{r}_i^{bc}$, the client adds $\boldsymbol{r}_i^{wc}\boldsymbol{a}_{i-1} + \boldsymbol{r}_i^{bc}$ to $\widetilde{\boldsymbol{z}}_i$. Similarly, in backpropagation, it adds $\boldsymbol{\delta}_{i+1}\boldsymbol{r}_{i+1}^{wc}$ to $[\![\widetilde{\boldsymbol{q}}_{i+1}]\!]_c$. These steps are summarized in Algorithm 4.

Algorithm 3: Protect Gradients on Server

Input: $\Delta \boldsymbol{w}_i$, $\Delta \boldsymbol{b}_i$, $[\![\frac{1}{\eta}]\!]_s$, $\boldsymbol{r}_i^w$ and $\boldsymbol{r}_i^b \in \mathbb{Z}_N$
Output: $[\![\Delta\widetilde{\boldsymbol{w}}_i]\!]_s$, $[\![\Delta\widetilde{\boldsymbol{b}}_i]\!]_s$, $\boldsymbol{r}_i^{wc}$, $\boldsymbol{r}_i^{bc}$
$[\![\Delta\widetilde{\boldsymbol{w}}_i]\!]_s \leftarrow \Delta \boldsymbol{w}_i \oplus ([\![\frac{1}{\eta}]\!]_s \otimes \boldsymbol{r}_i^w)$
$[\![\Delta\widetilde{\boldsymbol{b}}_i]\!]_s \leftarrow \Delta \boldsymbol{b}_i \oplus ([\![\frac{1}{\eta}]\!]_s \otimes \boldsymbol{r}_i^b)$
$\boldsymbol{r}_i^{wc} \leftarrow \boldsymbol{r}_i^{wc} + \boldsymbol{r}_i^w$, $\boldsymbol{r}_i^{bc} \leftarrow \boldsymbol{r}_i^{bc} + \boldsymbol{r}_i^b$

Algorithm 4: Randomness Cancellation

if *Forward propagation* **then**
  Input: $\widetilde{\boldsymbol{z}}_i$, $\boldsymbol{a}_{i-1}$, $\boldsymbol{r}_i^{wc}$, and $\boldsymbol{r}_i^{bc}$
  Output: $\boldsymbol{z}_i$
  $\boldsymbol{z}_i \leftarrow \widetilde{\boldsymbol{z}}_i + \boldsymbol{r}_i^{wc}\boldsymbol{a}_{i-1} + \boldsymbol{r}_i^{bc}$
  return $\boldsymbol{z}_i$
if *Backpropagation* **then**
  Input: $\widetilde{\boldsymbol{q}}_{i+1}$, $\boldsymbol{\delta}_{i+1}$, and $\boldsymbol{r}_{i+1}^{wc}$
  Output: $\boldsymbol{q}_{i+1}$
  $\boldsymbol{q}_{i+1} \leftarrow \widetilde{\boldsymbol{q}}_{i+1} + \boldsymbol{\delta}_{i+1}\boldsymbol{r}_{i+1}^{wc}$
  return $\boldsymbol{q}_{i+1}$

4.4 Final Parameter Update

Once the training is completed, the final weights are updated on the server in one shot by subtracting the cumulative sum of actual gradients as shown in Algorithm 5.

While the above discussion is based on fully connected networks, a convolutional neural network (CNN)-based GELU-Net can be implemented in a similar way, since convolution is a linear operation and thus can be computed homomorphically. Max pooling can be adapted by mean pooling, thus handled by the server. Feature activations are returned to the clients and gradients are securely updated. Due to space limit, we skip the details but present its results in Sec. 5.

4.5 Security Analysis

In this section, we perform security analysis of GELU-Net in the semi-honest model.

Proposition 1. (Gradients protection in backpropagation) *The server cannot learn true values of $\Delta \boldsymbol{w}_i$ and $\Delta \boldsymbol{b}_i$ in order to reconstruct activations and private user data.*

Algorithm 5: Final Parameter Updates

Input: Final values of $\Delta \boldsymbol{w}_i^c, \Delta \boldsymbol{b}_i^c$, initial weights $\boldsymbol{w}_i^{\mathrm{I}}$ and $\boldsymbol{b}_i^{\mathrm{I}}$
Output: Final weights $\boldsymbol{w}_i^{\mathrm{F}}$ and $\boldsymbol{b}_i^{\mathrm{F}}$
for $i = 1, 2, 3, \cdots, n-1$ **do**
  Client: Encrypt $\Delta \boldsymbol{w}_i^c, \Delta \boldsymbol{b}_i^c$ with PK_s as $[\![\Delta \boldsymbol{w}_i^c]\!]_s$ and $[\![\Delta \boldsymbol{b}_i^c]\!]_s$ and send them to server
  Server: Decrypt $[\![\Delta \boldsymbol{w}_i^c]\!]_s$ and $[\![\Delta \boldsymbol{b}_i^c]\!]_s$ with SK_s, update $\boldsymbol{w}_i^{\mathrm{F}} = \boldsymbol{w}_i^{\mathrm{I}} - \eta\Delta \boldsymbol{w}_i^c$ and $\boldsymbol{b}_i^{\mathrm{F}} = \boldsymbol{b}_i^{\mathrm{I}} - \eta\Delta \boldsymbol{b}_i^c$

Proof. The prove follows the simulation method [Goldreich, 2009]. The basic idea is to construct simulators given the input to a party and global output, and show that it learns nothing except the final result. During training, the server attempts to remove the randomness from the received gradients. Given a value r_j selected by the client and an attempt r_k from the server, both in the space of $\mathbb{Z}_N$, the probability that r_j equals r_k is $Pr\{r_j = r_k\} \leq 1 - e^{-2/|\mathbb{Z}_N|}$ [Schneier, 2007]. $|\mathbb{Z}_N|$ is the size of a finite field identical to the cipher space of Paillier. Since the elements of the random mask is independent, the server can correctly yield matrices of $\boldsymbol{r}_i^w$ and $\boldsymbol{r}_i^b$ with probabilities $Pr\{\boldsymbol{r} = \boldsymbol{r}_i^w\} \leq (1 - e^{-2/|\mathbb{Z}_N|})^{m^2}$ and $Pr\{\boldsymbol{r} = \boldsymbol{r}_i^b\} \leq (1 - e^{-2/|\mathbb{Z}_N|})^{m}$. Because $|\mathbb{Z}_N|$ is a large number, the probability that server can successfully derive the gradients is close to zero. □

Proposition 2. (Model protection in forward propagation) *The accumulated function groups* $\{\boldsymbol{z}_i = \boldsymbol{w}_i \boldsymbol{a}_{i-1} + \boldsymbol{b}_i\}_d$ *reveal nothing but the subspaces of weights and bias from which the matrices* $\boldsymbol{w}_i$ *and* $\boldsymbol{b}_i$ *cannot be reconstructed by client.*

Proof. Let $\boldsymbol{z}^{(i)}_{m\times 1} = \boldsymbol{w}^{(i)}_{m\times m}\boldsymbol{a}^{(i-1)}_{m\times 1} + \boldsymbol{b}^{(i)}_{m\times 1}$ denote the function group obtained by client after one forward propagation. Since a client is continuously trained for $d = d_{max}$ (less than the bound of $m + 2$ in Section 4.1), the function group does not reveal any information regarding the actual values of the matrices $\boldsymbol{w}_i$ and $\boldsymbol{b}_i$ but the subspaces linearly combined by infinitely many possible matrices solutions. Hence, model weights $\boldsymbol{w}_i$ and $\boldsymbol{b}_i$ cannot be successfully reconstructed by the client with $d = d_{max}$. □

Proposition 3. (Gradients protection in final model update) *The accumulated parameter update groups* $\{\boldsymbol{w}_i^{\mathrm{F}} = \boldsymbol{w}_i^{\mathrm{I}} - \eta\Delta\boldsymbol{w}_i^c, \boldsymbol{b}_i^{\mathrm{F}} = \boldsymbol{b}_i^{\mathrm{I}} - \eta\Delta\boldsymbol{b}_i^c\}_i$ *reveal nothing but the subspaces of gradients from which the matrices* $\Delta\boldsymbol{w}_i$ *and* $\Delta\boldsymbol{b}_i$ *in the previous backpropagations cannot be reconstructed.*

Proof. In the final model update, the client sends $\Delta\boldsymbol{w}_i^c$ and $\Delta\boldsymbol{b}_i^c$ to server. Since the client is allowed for d_{max} training iterations, the server ultimately obtains $d_{max} - 1$ pairs of randomized gradients $\Delta\widetilde{\boldsymbol{w}}_i$ and $\Delta\widetilde{\boldsymbol{b}}_i$. For each element in weight/bias matrices, there are totally d_{max} linear equations with $2d_{max} - 1$ unknown parameters ($d_{max} - 1$ random numbers and d_{max} gradients for each backward propagation). Since $d_{max} > 1$ and there is no way the server can add extra equations, the function group does not reveal any information regarding the actual values of $\Delta\boldsymbol{w}_i$ and $\Delta\boldsymbol{b}_i$ but the subspaces linearly combined by infinitely many possible matrices solutions. Therefore, the intermediate gradients cannot be reconstructed by the server. □

Fig. 2 shows an example when gradient protection is in place. We can see that the server can no longer reconstruct training data during backpropagation.

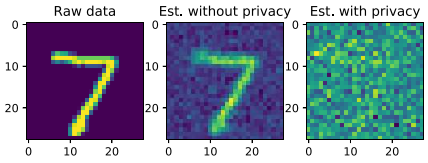

Figure 2: With/without gradient protection.

Table 2: Complexity comparison (per iteration).

Approach	Computation	Communication
GELU-Net	$\mathcal{O}(nm^2p)$	$\mathcal{O}(nm^2)$
BGN-Net	$\mathcal{O}(Cnmb)$	$\mathcal{O}(nm^2)$
ElGamal-Net	$\mathcal{O}(Z^2m^3ne)$	$\mathcal{O}(Z^2m^2n)$

4.6 Complexity Analysis

Communication Cost: The communication cost is analyzed as the total number of messages transmitted between the client and server. We assume a unit message size for encrypted data. For an n-layer network, in the forward propagation, the communication cost is $2m(n-1)$, where m is the number of activations in a layer. This is because total m ciphertexts need to be transmitted by the client and the server, for the sum of inputs $\boldsymbol{z}_i$ and activations $\boldsymbol{a}_i$, respectively, at each layer. In the backpropagation, the client needs $m^2 + m + 1$ messages for model updates between two consecutive layers. Except the final layer, the client interacts with the server to calculate the gradients, which requires transmitting encrypted error $[\![\boldsymbol{\delta}_{i+1}]\!]_c$ in m messages between client and server for each layer. Summing up cost from the forward and backpropagation, the entire network requires $\mathcal{O}(nm^2)$ communication messages for an iteration.

Computation Cost: Arithmetic multiplications and additions are mapped to modular exponentiations and modular multiplications over ciphertext, respectively. Here, we denote such cost of conducting homomorphic arithemtics in Paillier by p. For n layers, both forward and backpropagations take $\mathcal{O}(nm^2p)$ so the total computation cost is $\mathcal{O}(nm^2p)$.

Numerical Comparison: Two previous studies have considered privacy-preserved training for DNN. [Yuan and Yu, 2014] uses a doubly homomorphic encryption called BGN [Boneh *et al.*, 2005] that supports one multiplication between ciphertext and unlimited additions. We call this scheme *BGN-Net* henceforth. [Chen and Zhong, 2009] adopts ElGamal for homomorphic encryption [ElGamal, 1985], which supports either additive or multiplicative computation but not both. The arithmetic costs of BGN and ElGamal are denoted by b and e, respectively. Our tests show that the running time is $e \leq p \ll b$. BGN is at least 15 times slower than Paillier and Paillier is comparable with ElGamal. Note that CryptoNets do not support training, and thus is not comparable here.

Table 2 compares the complexity between different schemes in terms of computation and communication for an n-layer fully connected network. GELU-Net achieves over an order of magnitude improvements in the computation cost compared to ElGamal-Net, which requires all Z parties to participate in each iteration. In BGN-Net, since the 5-th order polynomial is employed to approximate activation, a large number of C homomorphic computations ($C > 30$) is needed using BGN. This actually gives GELU-Net a leverage. As long as the number of neurons per layer (m) is less than 450, GELU-Net is faster. Furthermore, a drawback of both schemes is that they are built on vertically divided data among users where the partial update of plaintext parameters will involve global activation values, from which the data distribution of other users can be derived.

5 Experiments

To evaluate the performance of GELU-Net, we use commodity workstations to implement the clients and server. The workstations have 2.8 GHz Intel Core i7 CPU and 8GB RAM connected by 1 Gbps LAN. The Paillier package is integrated with *Numpy* and *Theano* to build the neural network. We run experiments based on Iris, Diabetes, kr-vs-kp and MNIST datasets to compare with CryptoNets (implemented in Microsoft's SEAL library) and BGN-Net in terms of training stability, accuracy and computation speed.

5.1 Training Stability and Accuracy

Inspired by [Livni *et al.*, 2014], polynomial activation is adopted by cryptography experts. It proposes square activation to train a convex objective and injects hidden neurons in a gradual manner. CryptoNets uses square function and BGN-Net uses the 5-th order polynomial (Taylor expansion) to approximate the activation. However, polynomials may incur instability on non-convex objectives. Our results indicate that they make the network hard to train. In contrast, GELU-Net leaves the activation function unscathed so model parameters are still learnable in a privacy-preserved manner.

To compare the training stability of the three approaches, we adopt the BGN-Net network architecture of 1 densely connected hidden layer with $5, 12, 15, 300$ neurons for the four datasets respectively. As shown in Fig. 3, the square activation of CryptoNets fails immediately. A higher order approximation (e.g., 3rd or 5th order) used by BGN-Net is better, but still unsuccessful as training terminates prematurely.

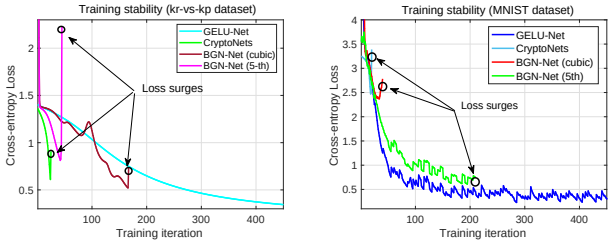

Figure 3: Training stability of different schemes.

Next we compare the accuracy of these approaches in Table 3. Since CryptoNets and BGN-Net are unstable in training, they are pre-trained with plaintext data. Encrypted data are used for inference only. GELU-Net is able to retain the original model accuracy while other two approaches suffer an accuracy loss from $2\%\sim7\%$. This makes GELU-Net especially appealing in many smart applications on large datasets when model accuracy is the key consideration.

5.2 Computation Speed

We compare the computation speed between GELU-Net and CryptoNets on MNIST. Note that our testing shows that BGN encryption is even slower than FHE so we only compare GELU-Net with CryptoNets here. First, we adopt the CNN architecture used in CryptoNets (denoted by Conv-1) and then stack more convolutional layers to form an architecture identical to LeNet-5.

- Conv-1: Conv(5×5, stride 2, 5 filters)-ReLU (square)-Mean Pooling-ReLU (square)- Softmax.
- LeNet-5: Conv(5×5, stride 1, 6 filters)-Mean Pooling-ReLU (square)-Conv(5×5, stride 1, 16 filters)-Mean Pooling-ReLU (square)-Dense(120)-Dense(84)- Softmax.

Since CryptoNets only supports inference, the model is loaded with pre-trained weights on MNIST. Table 4 shows the computation time (for one inference) and the accuracy. We observe that GELU-Net achieves 18 to 35x speed-up over CryptoNet with no accuracy loss. The performance gain is more obvious when the network gets deeper because more expensive homomorphic multiplications over ciphertext (for square activations) are required in CryptoNets.

Table 3: Comparison of accuracy.

Datasets	GELU-Net	CryptoNets	BGN-Net
Iris	**0.986±0.004**	0.966±0.012	0.96±0.007
Diabetes	**0.760±0.011**	0.741±0.023	0.723±0.028
kr-vs-kp	**0.967±0.008**	0.948±0.015	0.944±0.014
MNIST	**0.969±0.004**	0.919±0.0009	0.901±0.006

Table 4: Computation speed in different networks (s).

Architecture	Time (s)	Accuracy
GELU-Net (Conv-1)	**67.5±2.8**	0.936±0.006
CryptoNets (Conv-1)	1271.8±1.9	0.909±0.002
GELU-Net (LeNet-5)	**85.5±2.1**	0.989±0.001
CryptoNets (LeNet-5)	3009.6±1.7	0.967±0.003

Since the network communication is an integral part of GELU-Net, we further evaluate its performance in two different environments. We first deploy the server in a cloud computing infrastructure with local workstations in a different network domain. This scenario is denoted as Cloud-Local. Then we put both the server and clients inside the cloud on virtual machines, denoted as Cloud-Cloud. Table 5 shows the computation time for one inference. We observe that GELU-Net achieves optimal performance in the data center since the propagation delay is minimal. The communication cost increases once the client and server reside at different network domains. In the worst case, GELU-Net still achieves 14x speed-up compared to CryptoNets.

Table 5: GELU-Net speed in different environments (s).

Architecture	Cloud-Local	Cloud-Cloud	Local-Local
Dense(12)	0.381±0.002	**0.156±0.005**	0.281±0.01
Dense(300)	147.5±9.8	**59.7±1.9**	107.4±3.5
Conv1	91.3±5.4	**37.5±1.6**	67.5±2.8
LeNet-5	126.7±6.3	**47.5±1.2**	85.5±2.1

6 Conclusion

We have proposed a privacy-preserving deep neural network architecture based on a partially homomorphic cryptosystem. The novel design has ensured the neural network to be free of accuracy loss and achieve the desired stability in training. We have analyzed its security and complexity, and carried out extensive experiments that demonstrate 14 to 35 times speed-up compared to the state-of-the-art solutions.

References

[Boneh *et al.*, 2005] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *TCC*, volume 3378, pages 325–341. Springer, 2005.

[Campbell and Meyer, 2009] Stephen L Campbell and Carl D Meyer. *Generalized inverses of linear transformations*. SIAM, 2009.

[Chen and Zhong, 2009] Tingting Chen and Sheng Zhong. Privacy-preserving backpropagation neural network learning. *IEEE Transactions on Neural Networks*, 20(10):1554–1564, 2009.

[ElGamal, 1985] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[Gentry, 2009] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.

[Gilad-Bachrach *et al.*, 2016] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210, 2016.

[Goldreich, 2009] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[Hu, 2013] Yin Hu. *Improving the efficiency of homomorphic encryption schemes*. PhD thesis, Worcester Polytechnic Institute, 2013.

[Livni *et al.*, 2014] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, pages 855–863, Cambridge, MA, USA, 2014. MIT Press.

[Mahendran and Vedaldi, 2015] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5188–5196, 2015.

[Morris, 2013] Liam Morris. Analysis of partially and fully homomorphic encryption. *Rochester Institute of Technology, New York*, page 5, 2013.

[Paillier, 1999] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt*, volume 99, pages 223–238. Springer, 1999.

[Papernot *et al.*, 2017] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, pages 506–519, New York, NY, USA, 2017. ACM.

[Schneier, 2007] Bruce Schneier. *Applied cryptography: protocols, algorithms, and source code in C*. john wiley & sons, 2007.

[Shokri and Shmatikov, 2015] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1310–1321. ACM, 2015.

[Tramèr *et al.*, 2016] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 601–618, Austin, TX, 2016. USENIX Association.

[Yuan and Yu, 2014] Jiawei Yuan and Shucheng Yu. Privacy preserving back-propagation neural network learning made practical with cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, 25(1):212–221, 2014.

[Zhang *et al.*, 2017] Yuan Zhang, Qingjun Chen, and Sheng Zhong. Efficient and privacy-preserving min and k th min computations in mobile sensing systems. *IEEE Transactions on Dependable and Secure Computing*, 14(1):9–21, 2017.