SecureNets: Secure Inference of Deep Neural Networks on an Untrusted Cloud

Xuhui Chen xxc296@case.edu
Jinlong Ji jxj405@case.edu
Lixing Yu lxy257@case.edu
Changqing Luo[†] cxl881@case.edu
Pan Li pxl288@case.edu
Dept. of EECS, Case Western Reserve University, Cleveland, USA
[†]*Dept. of Computer Science, Virginia Commonwealth University, Richmond, USA*

Editors: Jun Zhu and Ichiro Takeuchi

Abstract

Inference using deep neural networks may be outsourced to the cloud due to its high computational cost, which, however, raises security concerns. Particularly, the data involved in deep neural networks can be highly sensitive, such as in medical, financial, commercial applications, and hence should be kept private. Besides, the deep neural network models owned by research institutions or commercial companies are their valuable intellectual properties and can contain proprietary information, which should be protected as well. Moreover, an untrusted cloud service provider may return inaccurate and even erroneous computing results. To address above issues, we propose a secure outsourcing framework for deep neural network inference called SecureNets, which can preserve both a user's data privacy and his/her neural network model privacy, and also verify the computation results returned by the cloud. Specifically, we employ a secure matrix transformation scheme in SecureNets to avoid privacy leakage of the data and the model. Meanwhile, we propose a verification method that can efficiently verify the correctness of cloud computing results. Our simulation results on four- and five-layer deep neural networks demonstrate that SecureNets can reduce the processing runtime by up to 64%. Compared with CryptoNets, one of the previous schemes, SecureNets can increase the throughput by 104.45% while reducing the data transmission size by 69.78% per instance.

Keywords: Inference, deep neural networks, secure outsourcing, chosen-plaintext attack

1. Introduction

Deep learning techniques have been widely applied to various areas, including biomedical engineering (Park et al. (2018); Ji et al. (2018a)), computer vision (Goodfellow et al. (2016)), natural language processing (Socher et al. (2012); Ji et al. (2018b)). However, both obtaining a well-trained neural network (the training process) and utilizing the trained model for predictions (the inference process) can be extremely computationally expensive, which hinders the large-scale implementation of deep neural networks for many practical applications. As a new computing paradigm that enables resource-limited users to conduct intensive computing, cloud computing can potentially relieve users from expensive computations in deep learning. However, directly outsourcing data and computations to the cloud raises critical security concerns (Papernot et al. (2016)). In particular, in many application such as medicine, finance, and marketing, the data are typically too sensitive to be exposed to any untrusted third-party including the cloud. Besides, the architecture of a well-trained deep model, such as the network structure, each layer's information, and the model weights, is valuable intellectual property and can contain proprietary information of a company or a designer, which should be protected as well. No matter training a deep model in the cloud or deploying a well-trained model to the cloud can lead to models being compromised. In addition, cloud service providers have financial incentives to reduce its computational cost, which thus may return inaccurate or even erroneous results to users.

The privacy and security issues in cloud computing have spurred technical advances in the area of secure outsourcing (Salinas et al. (2016a); Luo et al. (2018); Liao et al. (2017); Luo et al. (2017); Salinas et al. (2018)). Most previous works utilize traditional cryptographic techniques, such as homomorphic encryption (Gennaro et al. (2010); Liu et al. (2015); Urs (2013)). Researchers have also harnessed similar techniques to securely outsourcing deep learning networks. (Chen and Zhong (2009); Orlandi et al. (2007); Piva et al. (2008); Gilad-Bachrach et al. (2016)). In particular, (Barni et al. (2006)) and (Orlandi et al. (2007)) propose interactive algorithms for privately using deep models in cloud computing. For each layer of the neural networks, a user encrypts the data and the model's weights, and sends them to the cloud. The cloud computes an inner product between the data and the weights, and send the result back to the customer. Then, the user decrypts the result, conducts element-wise non-linear transformation, encrypts the new data and the next layer's weights and sends them to the cloud. The process continues until the user gets the inference results. In 2016, (Gilad-Bachrach et al. (2016)) proposed the CryptoNets, a high-throughput outsourcing method that applies neural networks to encrypted data with a non-interactive cloud computing paradigm. The data owner encrypts the data and sends it to the cloud. The cloud uses deep networks to process the data and sends back the results. However, the proposed CryptoNets framework only works on a specific class of neural networks that can be transformed into arithmetic circuits (Shpilka et al. (2010)). The major concern about the above traditional cryptographic techniques based outsourcing schemes is that they require the data owner to perform computationally expensive operations due to data encryptions and decryptions. Moreover, the data owner forces the cloud to conduct computations on ciphertexts, which has to be handled with specialized software and hence adds significant overhead to the cost of using cloud service. In addition, although the above works can provide privacy-preserving outsourcing for neural network inference, they cannot verify the correctness of returned results (Lai et al. (2014)). (Ghodsi et al. (2017)) attempt to address this issue, but their scheme only works for neural networks that can be transformed into arithmetic circuits.

In this paper, we propose a new secure outsourcing framework for deep neural network inference called SecureNets. We consider that a user has a well-trained deep neural network and aims to accelerate the inference process by utilizing cloud computing, while preserving the privacy of both the data and the model and assuring the correctness of computations performed by the cloud. Specifically, to protect the privacy of both the data and the deep neural model, we multiply data and weights matrices by carefully designed pseudorandom sparse matrices respectively. The multiplications can be proved to be chosen-plaintext attack (CPA) secure in both value and structure. Then the cloud performs matrix multiplications on the transformed data and link weights and sends the result to the user, who can efficiently verify the correctness of the computation and find out the true result. Locally, the user performs a non-linear function on the obtained result and gets the output of the current layer. Therefore, iteratively the user can efficiently and securely obtain the prediction results with the help of the cloud. The main contributions of this work can be summarized as follows:

- We develop a general framework SecureNets, which can efficiently and securely outsource neural network inference to the cloud.
- In SecureNets, the user performs computations with only $\mathcal{O}(n^2)$ complexity, where n is the dimension of the data, while the cloud conducts computations with $\mathcal{O}(n^3)$ complexity. Moreover, the cloud only conducts traditional linear algebra operations, which have much lower computational burden compared to the computations based on homomorphic encryptions.
- We show that the transformed data and weights are secure. In particular, the data transformation schemes in SecureNets are CPA-secure in both value and structure.
- Experiment results demonstrate that SecureNets can significantly accelerate the inference process of deep models. Specifically, compared to a previous scheme CryptoNets, SecureNets increases the throughput by 104.45% while reducing the data transmission size by 69.78% per instance.

2. Preliminaries

In this section, we introduce the threat model and the secure definitions considered in this paper.

2.1. Threat Model

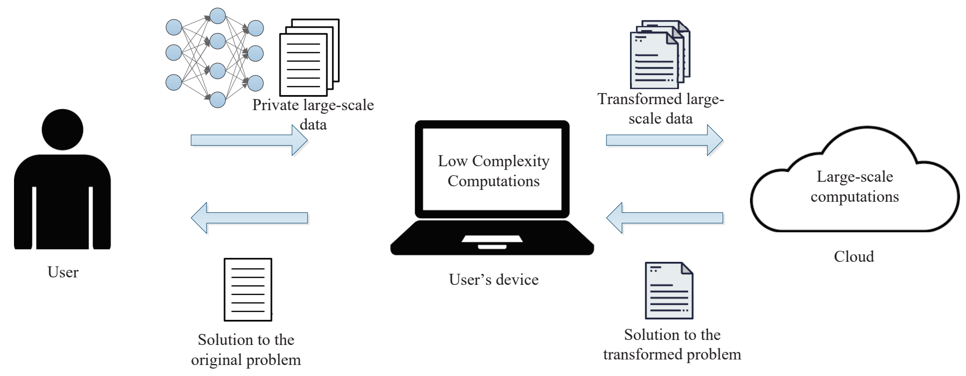

Private large-scale data
Transformed large-scale data
Low Complexity Computations
Large-scale computations
User
User's device
Cloud
Solution to the original problem
Solution to the transformed problem

Figure 1: The architecture for securely outsourcing inference of deep neural networks.

As shown in Fig. 1, we consider that a user has a well-trained deep neural network, which is a valuable intellectual property of the user, and also has sensitive data that needs to be processed by this model. Besides, we consider a malicious cloud. In particular, the cloud attempts to learn sensitive information about both the user's data and his/her model from the outsourced data, operations and the results of its own computations. In addition, the cloud may return inaccurate computation results to the user for financial motivations, or even deviate the proposed protocols and return erroneous results.

2.2. Security Definitions

To securely outsource the deep neural network inference, we adopt the concept of indistinguishability under a chosen-plaintext attack (CPA) (Katz and Lindell (2007)). We formulate the input of a deep neural network as a matrix (for examples, if the input data is a vector, we aggregate a batch of samples to build a matrix; if the input data is a high-dimensional tensor, we concatenate the multiple dimensions to construct a matrix). Besides, we convert the weights in each neural layer into a weight matrix (for example, the weights in each layer of an artificial neural network are obviously a weight matrix; in a convolutional neural network, the convolutional kernels can be converted to Toeplitz matrices (Vasudevan et al. (2017))). Therefore, we focus on the secure outsourcing of matrices and matrix operations. Note that in a matrix, both the elements' values and the matrix structure carry important private information. In the following, we formally define CPA security in value and in structure, respectively.

We first introduce the definition of a pseudorandom function (Lindell and Katz (2014)).

Definition 1 *Let F be a function and f a truly random function. F is a pseudorandom function if for all probabilistic polynomial-time distinguishers D, there exists a negligible function μ such that*

$$\left| Pr\left[D^F(1^n)=1)\right] - Pr\left[D^f(1^n)=1)\right]\right| \leqslant \mu, \tag{1}$$

Distinguishers D^F and D^f have oracle access to function F and f, respectively.

The following Definition 2 given by (Salinas et al. (2016b)) defines CPA-security in value.

Definition 2 *CPA-security in Value (Salinas et al. (2016b)): A matrix transformation scheme has indistinguishable transformations in value under a chosen-plaintext attack (or is CPA-secure) if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function μ, such that the probability of distinguishing two matrix transformations in value in a CPA indistinguishability experiment is less than $1/2 + \mu$.*

To protect a matrix's structure, the positions of the non-zero values in a matrix need to be protected.

Definition 3 *CPA-security in Structure (Salinas et al. (2016b)): A permutation scheme has indistinguishable permutations under a chosen-plaintext attack (or is CPA-secure) if for all a probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function μ, such that the probability of distinguishing two permutations in a CPA indistinguishability experiment is less than $1/2 + \mu$.*

3. SecureNets: Secure Inference of Deep Neural Networks

In this section, we develop a secure and efficient outsourcing framework for deep neural network inference. First, we formulate the mathematical model of a deep neural network. Then we introduce the secure matrix transformation schemes for protecting matrix values and structure, respectively. Building upon secure matrix transformations, we finally elaborate the details of SecureNets.

3.1. Mathematical Model of A General Deep Neural Network

In general, an L layer feed-forward neural network can be modeled as follows. We denote the input to the network as $\mathbf{x} \in \mathbb{F}_p^{n_0}$, where $\mathbb{F}_p$ represents the finite field of order p, with p being a large prime, and n_0 is the dimension of the input sample. Layer $l \in [1, L]$ has n_l neurons, and is defined by a weight matrix $\mathbf{W}_l \in \mathbb{F}_p^{n_l \times n_{l-1}}$ and a bias vector $\mathbf{b}_l \in \mathbb{F}_p^{n_l \times 1}$. Its output $\mathbf{y}_{l+1} \in \mathbb{F}_p^{n_l \times 1}$ is defined as[1]

$$\mathbf{y}_{l+1} = \sigma\left(\mathbf{W}_l \mathbf{y}_l + \mathbf{b}_l\right) \ \forall l \in [1, L], \tag{2}$$

where $\sigma(\cdot)$ is a non-linear activation function (particularly, when $l = L$, $\sigma(\cdot)$ is the activation function of the output layer, which typically is the softmax function).

Without loss of generality, we consider a batch of m samples as the input matrix. We redefine $\mathbf{W}_l = [\mathbf{W}_l \quad \mathbf{b}_l] \in \mathbb{F}_p^{n_l \times (n_{l-1}+1)}$, $\mathbf{y}_l^T = [\mathbf{y}_l^T \quad 1]$, and $\mathbf{Y}_l = [\mathbf{y}_l^1 ... \mathbf{y}_l^m] \in \mathbb{F}_p^{(n_{l-1}+1) \times m}$, where $\mathbf{y}_l^k$ is the output of the $(l-1)$th layer of sample k for $1 \leqslant k \leqslant m$. Thus, Eq. (2) can be rewrite as

$$\mathbf{Z}_l = \mathbf{W}_l \mathbf{Y}_l, \tag{3}$$

$$\mathbf{Y}_{l+1} = \sigma(\mathbf{Z}_l), \tag{4}$$

where $l \in [1, L]$ and $\mathbf{Z}_l \in \mathbb{F}_p^{n_l \times m}$. Note that Eq. (3) is a matrix multiplication and Eq. (4) is a column-wise non-linear transformation. This model can capture both fully connected and convolutional layers, and in the latter case the weight matrix is sparse.

3.2. Secure Matrix Transformation

To securely utilize an untrusted cloud server, the user must protect his/her private data and model weights, which can both be denoted as matrices as demonstrated above. Here, we introduce the secure matrix transformation methods developed in our previous works (Salinas et al. (2016b)). Particularly, we consider a private matrix $\mathbf{X} \in \mathbb{F}_p^{m \times n}$, with $(i, j) \in \mathcal{S}_\mathbf{X}$, where $\mathcal{S}_\mathbf{X}$ is the structure of $\mathbf{X}$ and defined as

$$\mathcal{S}_\mathbf{X} = \{(i, j) | x_{i,j} \neq 0, \forall i \in [1, n] \ \forall j \in [1, m]\}. \tag{5}$$

3.2.1. Secure Matrix Multiplication

To hide the values of non-zero values in $\mathbf{X}$, the user performs the following multiplications

$$\tilde{\mathbf{X}} = \mathbf{D}\mathbf{X}\mathbf{F}. \tag{6}$$

1. The input is considered as layer 0, and hence we have $\mathbf{y}_1 = \mathbf{x}$.

In Eq. (6), $\mathbf{D} \in \mathbb{F}_p^{m\times m}$ is a diagonal matrix that is defined as

$$d_{i,j} = \begin{cases} v_i & i = j \text{ for } i,j \in [1,m], \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

The value v_i (for $i \in [1,m]$) is the ith element of a vector $\mathbf{v_X} \in \mathbb{F}_p^{m\times 1}$ and determined by using a function $F_c : \{0,1\}^w \times \{0,1\}^w \to \{0,1\}^w$, i.e.,

$$v_i = F_c(r_i, g), \forall i \in [1,m], \tag{8}$$

where r_i is a random string and g is a constant one.

$\mathbf{F} \in \mathbb{F}_p^{m\times m}$ in Eq. (6) is also a diagonal matrix, i.e.

$$f_{i,j} = \begin{cases} t_i & i = j \text{ for } i,j \in [1,n], \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

where t_i is the ith element of vector $\mathbf{t} \in \mathbb{F}_p^{n\times 1}$ and set to an arbitrary positive constant.

Therefore, although the structure of $\tilde{\mathbf{X}}$ in Eq. (6) remains the same with $\mathbf{X}$, the non-zero elements are given by

$$\tilde{h}_{i,j} = d_{i,i} f_{j,j} h_{i,j} = v_i t_j h_{i,j} \quad \forall i,j \in \mathcal{S}_\mathbf{H} \tag{10}$$

where $(i,j) \in \mathcal{S}_\mathbf{X}$.

(Salinas et al. (2016b)) prove that if $F_c(\cdot,\cdot)$ is a pseudorandom function, the matrix transformation in Eq. (6) is computationally indistinguishable in value under a chosen-plaintext attack (CPA), i.e., CPA-secure in value.

3.2.2. Secure Matrix Permutations

Eq. (6) protects a matrix's non-zero values but it still reveals the structure information of the matrix. Then we perform matrix permutation to hide $\mathbf{X}$'s structure $\mathcal{S}_\mathbf{X}$ by randomly permutating the rows and columns of $\tilde{\mathbf{X}}$.

To randomly permute $\tilde{X}$'s row index vector $\mathbf{e} \in \mathbb{R}^{m\times 1}$, the user computes the following

$$\mathbf{e}' = \mathcal{M}(\mathbf{e}), \hat{\mathbf{e}}' = F_c'(\mathbf{r}, \mathbf{e}'), \hat{\mathbf{e}} = \mathcal{M}^{-1}(\hat{\mathbf{e}}'), \tag{11}$$

where $\mathcal{M} : \mathbb{R}^m \to \{0,1\}^k$ (for $k = \lceil \log_2 m! \rceil$) is a function that maps index vectors to bit strings, $F_c' : \{0,1\}^w \times \{0,1\}^w \to \{0,1\}^w$ is a permutation function, $\mathbf{r} \in \{0,1\}^k$ is a random bit string; and $\mathcal{M}^{-1} : \{0,1\}^k \to \mathbb{R}^m$ is the inverse of $\mathcal{M}$.

We denote these operations as

$$Perm^{F_c'}(\mathbf{r}, \mathbf{e}) = \hat{\mathbf{e}}. \tag{12}$$

We also use $Perm^{F_c'}(\mathbf{r}', \mathbf{u})$ to denote the random permutation of $\tilde{X}$'s column index vector $\mathbf{u} \in \mathbb{R}^n$, where $\mathbf{r}' \in \{0,1\}^k$ is a random bit string.

The user can thus perform both row index vector permutation and column index vector permutation by

$$\hat{\mathbf{X}} = \mathbf{E}\tilde{\mathbf{X}}\mathbf{U} \tag{13}$$

where $\mathbf{E} \in \mathbb{R}^{m \times m}$ and $\mathbf{U} \in \mathbb{R}^{n \times n}$ in Eq. (13) are permutation matrices, and their elements are

$$e_{i,j} = \delta_{\pi(i),j} \quad \forall i \in [1, m], j \in [1, m],$$
$$u_{i,j} = \delta_{\pi(i),j} \quad \forall i \in [1, n], j \in [1, n],$$

where the function $\pi(\cdot)$ maps an original index i to its permuted index, i.e., $\pi(i) = \hat{e}_i$ (for $i \in [1, m]$) and $\pi(i) = \hat{u}_i$ (for $i \in [1, n]$). Besides, the Kronecker delta function is given by

$$\delta_{i,j} = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases} \tag{14}$$

(Salinas et al. (2016b)) also prove that if $F'_c(\cdot, \cdot)$ is a pseudorandom function, the matrix transformation in Eq. (13) is computationally indistinguishable in structure under a CPA, i.e., CPA-secure in structure.

3.3. SecureNets

Algorithm 1 The SecureNets Framework

```
Input: Deep Model $\mathbb{M}$ with $L$ layers and data $\mathbf{X}$
Initialization: Generate weights matrices $\mathbb{M} := \{\mathbf{W}_1, \mathbf{W}_2, \cdots, \mathbf{W}_L\}$
for $l$=1 to $L$ do
    User(Secure Matrix Transformation):
        Transform the data $\mathbf{Y}_l$ to $\hat{\mathbf{Y}}_l$
        Transform the weight matrix $\mathbf{W}_l$ to $\hat{\mathbf{W}}_l$
        Transfer $\hat{\mathbf{Y}}_l$ and $\hat{\mathbf{W}}_l$ to the cloud
    Cloud(Masked Matrix Multiplication):
        Calculate $\hat{\mathbf{Z}}_l = \hat{\mathbf{W}}_l \hat{\mathbf{Y}}_l$
        Return $\hat{\mathbf{Z}}_l$
    User(Correctness Verification):
        if ($\hat{\mathbf{Z}}_l$ is correct) then:
            User(True Result Recovery):
                Recover $\mathbf{Z}_l$ from $\hat{\mathbf{Z}}_l$
            Conduct column-wise non-linear transformation $\mathbf{Y}_{l+1} = \sigma(\mathbf{Z}_l)$
        else:
            Send a recalculation request
        end if
end for
Output: Inference result $\mathbf{y}_{L+1}$
```

We first describe the general idea of the proposed SecureNets. As shown in Algorithm 1, in the initialization phase, the user requires to conduct inference by using his/her deep neural network model. As described in Sec. 3.1, the user generates weight matrices $\mathbf{W}_1, \mathbf{W}_2, \cdots, \mathbf{W}_L$ for the deep model $\mathbb{M}$. Then for layer $l \in [1, L]$, the user locally conducts secure matrix transformations on the data $\mathbf{X}$ and the weights matrix $\mathbf{W}_l$, which are sent to the cloud to perform matrix multiplication. When the cloud sends back its computation result, the client first checks the correctness of the cloud's result. If the result is correct, the user recovers the original results and repeats the process for the next layer. If not, the user requires the cloud to conduct the computation again. The iterations continue until the user obtains the final inference results.

We divide the SecureNets framework into four main phases, *secure matrix transformation*, *matrix multiplication outsourcing*, *correctness verification*, and *true result recovery*, which are detailed below. Note that the cloud only performs the masked matrix multiplication while other phases are performed locally. The details are as follows.

3.3.1. Secure Matrix Transformation

Recall that the input to layer l is $\mathbf{Y}_l$ and the weight matrix is $\mathbf{W}_l$. Both of them contain sensitive information that need to be protected. To this end, the user applies secure matrix transformation to $\mathbf{W}_l$ and $\mathbf{Y}_l$ as follows:

$$\hat{\mathbf{W}}_l = \mathbf{V_W}\mathbf{W}_l\mathbf{T_W}, \tag{15}$$

where $\mathbf{V_W}$ is formed by a pseudorandom permutation matrix and a pseudorandom diagonal matrix, i.e., $\mathbf{V_W} = \mathbf{E_W}\mathbf{D_W}$, and $\mathbf{T_W}$ is formed by a diagonal matrix of arbitrary positive constants and a pseudorandom permutation matrix, i.e., $\mathbf{T_W} = \mathbf{F_W}\mathbf{U_W}$. Similarly, the user conceals $\mathbf{Y}_l$ as follows:

$$\hat{\mathbf{Y}}_l = \mathbf{V_Y}\mathbf{Y}_l\mathbf{T_Y}, \tag{16}$$

where $\mathbf{V_Y} = \mathbf{U_W}^{-1}\mathbf{F_W}^{-1}$, and $\mathbf{T_Y}$ is formed by a diagonal matrix of arbitrary positive constants and a pseudorandom permutation matrix, i.e., $\mathbf{T_Y} = \mathbf{F_Y}\mathbf{U_Y}$. As mentioned before, both Eq. (15) and Eq. (16) are CPA-secure in both value and structure, and hence can protect the user's model information and his/her data, respectively.

Computational Complexity Recall that $\mathbf{W}_l \in \mathbb{F}_p^{n_l \times (n_{l-1}+1)}$ and $\mathbf{Y}_l \in \mathbb{F}_p^{(n_{l-1}+1)\times m}$. For simplicity, we assume $n_l = m = n$ for all $l \in [1, L]$. For permutation matrices, they have the orthogonal property, i.e., $\mathbf{U_W}\mathbf{U_W}^{\intercal} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix, so we have $\mathbf{U_W}^{-1} = \mathbf{U_W}^{\intercal}$. Therefore, the complexity of computing $\mathbf{U_W}^{-1}$, $\mathbf{F_W}^{-1}$ is $\mathcal{O}(n)$. For the permutation matrices $\mathbf{E_W}, \mathbf{U_W}, \mathbf{U_Y}$ and the diagonal matrices $\mathbf{D_W}, \mathbf{F_W}, \mathbf{F_Y}$, they just have n non-zero elements in the matrices, one for each row and each column. Thus, for $\mathbf{Y}_l$ and $\mathbf{W}_l$, the computational complexity is $\mathcal{O}(n^2)$ when multiplied by a diagonal matrix or a permutation matrix. In summary, the computational complexity of secure matrix transformation is $\mathcal{O}(n^2)$.

3.3.2. Matrix Multiplication Outsourcing

After transforming the input $\mathbf{Y}_l$ and $\mathbf{W}_l$ to masked matrices $\hat{\mathbf{Y}}_l$ and $\hat{\mathbf{W}}_l$, the user transmits them to the cloud to perform matrix multiplication:

$$\hat{\mathbf{Z}}_l = \hat{\mathbf{W}}_l\hat{\mathbf{Y}}_l, \tag{17}$$

which can be derived as

$$\begin{aligned}\hat{\mathbf{Z}}_l &= \hat{\mathbf{W}}_l \hat{\mathbf{Y}}_l \\ &= (\mathbf{V_W W}_l \mathbf{T_W})(\mathbf{V_Y Y}_l \mathbf{T_Y}) \\ &= \mathbf{V_W W}_l (\mathbf{F_W U_W})(\mathbf{U_W^{-1} F_W^{-1}}) \mathbf{Y}_l \mathbf{T_Y} \\ &= \mathbf{V_W W}_l \mathbf{Y}_l \mathbf{T_Y}.\end{aligned} \tag{18}$$

$\mathbf{Z}_l$ is then sent back to the user for verification and result recovery.

Computational Complexity Note that the masked matrices $\hat{\mathbf{Y}}_l$ and $\hat{\mathbf{W}}_l$ remain the same sizes as $\mathbf{Y}_l$ and $\mathbf{W}_l$. Reall that we assume $n_l = m = n$ for all $l \in [1, L]$. Thus, the computational complexity of this phase is $\mathcal{O}(n^3)$ in the cloud.

3.3.3. Correctness Verification

We consider a malicious cloud that may derivate from the proposed protocols and return erroneous results. Thus, the user needs check the correctness of the result returned by the cloud. Specifically, the user randomly selects $\hat{z}^l_{i,j}$ from $\hat{\mathbf{Z}}_l$, where $(i, j) \in \mathcal{S}_{\hat{\mathbf{Z}}_l}$ and $\mathcal{S}_{\hat{\mathbf{Z}}_l}$ is the structure of $\hat{\mathbf{Z}}_l$. Since $\hat{\mathbf{Z}}_l = \hat{\mathbf{W}}_l \hat{\mathbf{Y}}_l$, the user just checks

$$\hat{z}^l_{i,j} \overset{?}{=} \hat{w}^l_{(i,:)} \cdot \hat{y}^l_{(:,j)}, \tag{19}$$

where $\hat{w}^l_{(i,:)}$ is the ith row in $\hat{\mathbf{W}}_l$ and $\hat{y}^l_{(:,j)}$ is the jth column in $\hat{\mathbf{Y}}_l$. If Eq. (19) holds, the user can ensure the correctness of the computation for this element $\hat{z}^l_{i,j}$. To be more confident in the correctness of the computing, the user can select c points in $\hat{\mathbf{Z}}_l$ to verify, where $1 < c \ll n$. This partial correctness verification has been studied in the Freivalds' algorithm (Freivalds (1977)), where it is proved that in $\mathcal{O}(cn^2)$ time the algorithm can verify a matrix product with probability of failure less than 2^{-c}.

Computational Complexity The verification of each element in the returned matrix $\hat{\mathbf{Z}}_l$ is an inner vector product, whose complexity is $\mathcal{O}(n)$. If the user verifies c points, the computational complexity of the correctness verification phase is $\mathcal{O}(cn)$.

3.3.4. True Result Recovery

Once the returned result passes the verification, the user can recover the true result, which is

$$\mathbf{Z}_l = \mathbf{W}_l \mathbf{Y}_l. \tag{20}$$

Compared to Eq. (18), the user can recover the true result by

$$\begin{aligned}\mathbf{Z}_l &= \mathbf{V_W^{-1}} \hat{\mathbf{Z}}_l \mathbf{T_Y^{-1}} \\ &= (\mathbf{E_W D_W})^{-1} \hat{\mathbf{Z}}_l (\mathbf{F_Y U_Y})^{-1} \\ &= \mathbf{D_W^{-1} E_W^{-1}} \hat{\mathbf{Z}}_l \mathbf{U_Y^{-1} F_Y^{-1}}.\end{aligned} \tag{21}$$

Once the user gets $\mathbf{Z}_l$, the user conducts column-wise transformation and obtains $\mathbf{Y}_{l+1}$. Then, the process continues to the next layer.

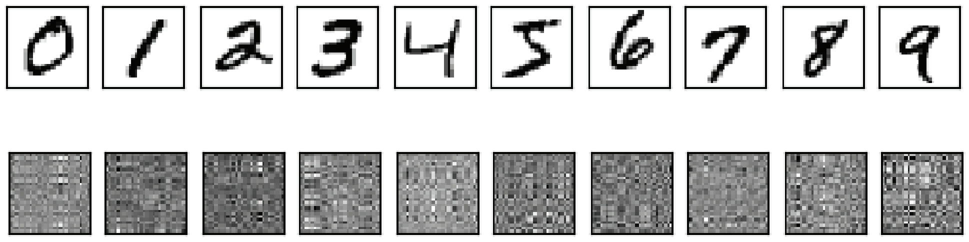
Computational Complexity As we discussed in Sec. 3.3.1, the inverses of matrices $\mathbf{D}_{\mathbf{W}}^{-1}$, $\mathbf{E}_{\mathbf{W}}^{-1}$, $\mathbf{U}_{\mathbf{Y}}^{-1}$, and $\mathbf{F}_{\mathbf{Y}}^{-1}$ each only has complexity of $\mathcal{O}(n)$, and the multiplication between each of them and $\hat{\mathbf{Z}}_l$ has complexity of $\mathcal{O}(n^2)$. The computational complexity of column-wise transformation is $\mathcal{O}(n^2)$. Thus, the total complexity is $\mathcal{O}(n^2)$ in this phase.

3.3.5. Computational Complexity Analysis

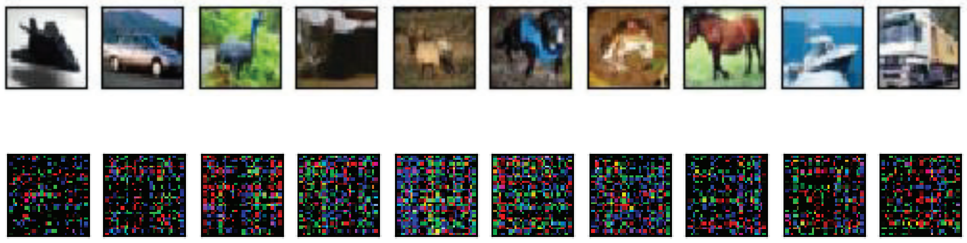
To summarize the computational complexity analysis of SecureNets for each layer, the computational complexity at the user is $\mathcal{O}(n^2)$, and $\mathcal{O}(n^3)$ at the cloud. In contrast, if the user performs one layer computing locally, the computational complexity would be $\mathcal{O}(n^3)$. Therefore, when there are L layers in a deep neural network model, SecureNets can reduce the computational complexity from $\mathcal{O}(n^3L)$ to $\mathcal{O}(n^2L)$ by securely outsourcing intensive computations to the cloud. Thus, SecureNets can significantly reduce the local computing time and accelerate the inference process.

4. Discussions on Privacy

The goal of SecureNets is to provide a secure and efficient outsourcing scheme for inference of general deep models. SecureNets aims to protect the user's data $\mathbf{X}$ and the model $\mathbb{M}$, which includes the weights $\mathbf{W}_1, \mathbf{W}_2, \cdots, \mathbf{W}_L$. In this section, we analyze the privacy of SecureNets.



(a) Images from MNIST and their corresponding masked images



(b) Images from CIFAR10 and their corresponding masked images

Figure 2: Images and their corresponding SecureNets transformed images

Human Perception As shown in Fig. 2, we select several images and display them with their corresponding images masked by the secure matrix transformation scheme introduced before. In Fig. 2(a), we select different images from the MNIST dataset, which is a digit recognition database where each image only has one channel. In Fig. 2(b), we select different images from CIFAR10, which is an image classification database where each image has three channels. From the figures, we can see that the images transformed by our methods are unrecognizable with human eyes. Moreover, after secure matrix transformation, it is impossible to distinguish the different types of images in the same dataset, which makes the masked images hard to convey any sensitive knowledge. Thus, from the human perception perspective, SecureNets well preserves the privacy of the data.

Privacy Analysis Inspecting the secure outsourcing scheme proposed in Sec. 3, we observe that the cloud only has access to the transformed input and weight matrices of each layer, and thus it is unable to learn private information from the user since the transformations are CPA-secure. Specifically, the user sends the matrices $\hat{\mathbf{Y}}_l$ and $\hat{\mathbf{W}}_l$ to the cloud. However, according to *secure matrix transformation*, the transformed matrices are CPA-secure in both value and structure. Therefore, the cloud is unable to find out any information about the true data $\mathbf{X}$ ($\mathbf{X} = \mathbf{Y}_1$) and layer weights $\mathbf{W}_1, \mathbf{W}_2, \cdots, \mathbf{W}_L$. Due to page limit, we do not include the proof for CPA-security in this paper. In summary, SecureNets can protect the user's data $\mathbf{X}$ as well as the model $\mathbb{M}$.

5. Performance Evaluation

In this section, we conduct experiments to evaluate the performance of the proposed SecureNets.

5.1. Experiment Setup

Datasets We evaluated SecureNets on two classification tasks, MNIST dataset for digit recognition and CIFAR10 for image classification. For both datasets, we train the deep model with $50,000$ images and test on $10,000$ images.

Neural Networks In the experiments, we apply three neural networks to the SecureNets framework, two for MNIST task and one for CIFAR-10 task. First, we use a multilayer perceptron (MLP) for MNIST with two hidden layers of 512 nodes. Both of the hidden layers are followed by rectifier activations. The output layer uses softmax activation.

Second, we use a convolutional neural network (CNN) for MNIST with two convolutional layers with 5×5 filters, a stride of 1 and a mapcount of 16 and 32 for the first and second layer, respectively. Each convolutional layer is followed by rectifier activations and 2×2 max-pooling with a stride of 2. The fully connected layer uses softmax activation.

For CIFAR-10, we use a four-layer network with two convolutional layers followed by two fully-connected layers. The convolutional layers use 5×5 convolutions with stride 1, followed by a rectifier activation and 2×2 max pooling, with 64 channels each. The first fully connected layer uses rectifier activation and the second uses softmax activation.

Experiment Environment We train the deep models with Keras, then convert the model to be suitable for the SecureNets framework. For user side, we run on a laptop with 2.7 GHz CPU, 8 GB RAM, and a 256 GB solid state drive, and we run the cloud side on an Amazon Elastic Compute Cloud (EC2) with 1 P2 instance. We implemented SecureNets by Python's scientific computing libraries Numpy, and we did not optimize the SecureNets application with multi-thread programming or parallel computing. Thus, SecureNets runs slowly than using Keras directly. To fairly analyze the performance of our proposed scheme, we use Numpy to implement the mentioned models and then run them locally.

5.2. Performance Analysis

In this section, we analyze the performance of SecurityNets in terms of model accuracy and runtime cost to demonstrate its efficiency.

Table 1: The runtime (s) and accuracy (%) of SecureNets

Methods	MNIST with MLP				
	local	cloud	trans.	total	Acc.
Local	25.74	-	-	25.74	98.32%
SecureNets (0)	18.91	25.15	5.82	49.88	98.32%
SecureNets (10)	19.06	26.23	6.01	51.30	98.32%
SecureNets (100)	19.35	24.72	5.90	49.97	98.32%
Methods	**MNIST with CNN**				
	local	cloud	trans.	total	Acc.
Local	3968.0	-	-	3968.00	99.21%
SecureNets (0)	628.3	851.8	12.56	1492.66	99.21%
SecureNets (10)	659.7	831.4	10.98	1502.08	99.21%
SecureNets (100)	714.6	842.7	11.65	1568.95	99.21%
Methods	**CIFAR10 with CNN**				
	local	cloud	trans.	total	Acc.
Local	6437.3	-	-	6437.30	74.68%
SecureNets (0)	1005.0	1303.7	45.12	2353.82	74.68%
SecureNets (10)	1055.2	1386.9	41.98	2484.08	74.68%
SecureNets (100)	1140.8	1285.7	49.51	2476.01	74.68%

Accuracy Analysis One benefit of SecureNets is that it has no restrictions on the structure of deep models. From Table 1, we can see that for all three models, the performance of networks remains the same, even with a different number c, $c = 0, 10, 100$, of verification points, which demonstrates that SecureNets accelerates the inference process without any accuracy compromise. For the MNIST dataset, SecureNets with MLP achieved the accuracy of 98.32%, and SecureNets with CNN achieved the state-of-art performance accuracy (99.21%). For CIFAR10, SecureNets with CNN achieved 75.68%.

Timing analysis In Table 1, we also compare the runtime cost for three different models. For each model, we process $50,000$ images locally, in SecureNets without verification, in SecureNets with $c = 10$ verification points, and in SecureNets with $c = 100$ verification points, respectively.

Table 2: Performance Comparison of SecureNets and CryptoNets

	CryptoNets	SecureNets
Accuracy	98.95%	99.15%
Throughput per hour	58,982	120,590
Message size (User to Cloud) per instance	91.875 KB	18.75 KB
Message size (Cloud to User) per instance	1.17 KB	9.37 KB
Total message size per instance	93.046 KB	28.12 KB

Compared with running predict models locally, SecureNets with CNN reduces by 62.38% for MNIST and 63.43% for CIFAR10 in terms of the total computing runtime. However, the SecureNets extends the total run-time for MLP model for MNIST. The reason is that the MLP model for MNIST is too simple and the power of cloud computing has not been fully used. Even though, SecureNets do reduce the local run-time for this "shallow" model. We also observed that with increasing of the models' complexity, the reduction percentages of the local run-time becomes more significant. Further, we also demonstrated the efficiency of the proposed verification methods. With more verification points, the users can ensure the correctness of the cloud service more confidently. Meanwhile, the local computational time almost remains the same as the ones without the verification scheme. Therefore, the proposed SecureNets framework is suitable for complex deep models to process large-scale data.

5.3. Performance Comparison

In this section, we compare the performance of SecureNets with one of the state-of-art deep model outsourcing schemes, the CryptoNets. In the CryptoNets, the user protects its data by Homomorphic encryption and outsources the inference process to the cloud.

For the CryptoNets, the local computer handles the encryption and decryption process. The computational complexity of Homomorphic encryption depends on the length of the public key, which usually is $\mathcal{O}(2^n)$. Meanwhile, the local computational complexity in the proposed SecureNets is only $\mathcal{O}(n^2)$. The cryptography-based data protection methods also inevitably introduce redundant data, which increases the transmission cost when outsourcing data processing to the cloud. In our framework, we utilize secure matrix transformation to protect the data. The matrix transformation only masks the sensitive information of the data while the data are remaining the original size. Moreover, the CryptoNets assumes the cloud is always honest and reliable, so it does not have additional verification scheme to ensure the correctness of the returned results. Therefore, the SecureNets scheme is more suitable for the cloud service renter to deploy deep models for large-scale data processing securely.

In Table 2, we compare the performance of SecureNets and the CryptoNets. In this experiment, the CryptoNets and SecureNets are using similar deep convolutional neural networks trained on MNIST. The structure of both CNN models are the same, and the difference lies in the activation functions. The CryptoNets can only apply to the neural networks which can be transformed into arithmetic circuits. Therefore, the activation function of CNN in CryptoNets is the polynomial function. Meanwhile, we use rectifier activation function for the CNN in SecureNets. From the table, we can see that both SecureNets and the CryptoNets can achieve the state-of-art performance on MNIST dataset. Nonetheless, compared to SecureNets, the restriction on activation functions slightly impacts the performance of the CryptoNets. To be clear, the accuracy error is CNNs testing error, and the conversion does not compromise the precision of the model. In the experiment, we define the system throughput as the number of images can be processed in one hour. We observe that the CryptoNets can make 58,982 predictions per hour while SecureNets scheme can process $120,590$ images per hour, which increase the throughput by 104.45%. In terms of data transmission, the CryptoNets is a non-interactive cloud computing paradigm and SecureNets is an interactive cloud computing paradigm. The CryptoNets only transmit and receive messages once while SecureNets executes several times, thus the CryptoNets should transmit smaller amounts of data. However, the encryption introduces redundancy to the CryptoNets' message size. We observe that although the CryptoNets scheme costs fewer transmission data from the cloud to the user, SecureNets reduces the total message size by 69.78% per instance.

6. Conclusions

In this paper, we have presented SecureNets, a new framework that allows a client to securely outsource deep neural networks based inference to an untrusted cloud. Particularly, building upon CPA-secure matrix transformations, we have designed a secure and efficient outsourcing protocol for general deep neural network inference, which can protect both a user's data and his/her the model and verify the correctness of the cloud computing results. We have shown that the proposed SecureNets can significantly reduce the local computational complexity and accelerate the inference process of deep models.

Moreover, the proposed SecureNets scheme is a general secure outsourcing framework which can be applied to different types of neural networks. Although we have only discussed feed-forward neural networks, it can also be applied to neural networks with cross-layer structures. Taking recurrent neural networks as an example, we can divide each node into several matrix multiplications, which can be outsourced to the cloud, and other operations which remain locally.

Acknowledgments

This work was partially supported by the US National Science Foundation under grants CNS-1602172 and CNS-1566479.

References

Mauro Barni, Claudio Orlandi, and Alessandro Piva. A privacy-preserving protocol for neural-network-based computation. In *Proceedings of the 8th workshop on Multimedia and security*, pages 146–151. ACM, 2006.

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

Rusins Freivalds. Probabilistic machines can use less running time. In *IFIP congress*, volume 839, page 842, 1977.

Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Annual Cryptology Conference*, pages 465–482. Springer, 2010.

Zahra Ghodsi, Tianyu Gu, and Siddharth Garg. Safetynets: Verifiable execution of deep neural networks on an untrusted cloud. In *Advances in Neural Information Processing Systems*, pages 4675–4684, 2017.

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.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Jinlong Ji, Xuhui Chen, Changqing Luo, and Pan Li. A deep multi-task learning approach for ecg data analysis. In *Biomedical & Health Informatics (BHI), 2018 IEEE EMBS International Conference on*, pages 124–127. IEEE, 2018a.

Jinlong Ji, Changqing Luo, Xuhui Chen, Lixing Yu, and Pan Li. Cross-domain sentiment classification via a bifurcated-lstm. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 681–693. Springer, 2018b.

Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007. ISBN 1584885513.

Junzuo Lai, Robert H Deng, HweeHwa Pang, and Jian Weng. Verifiable computation on outsourced encrypted data. In *European Symposium on Research in Computer Security*, pages 273–291. Springer, 2014.

Weixian Liao, Changqing Luo, Sergio Salinas, and Pan Li. Efficient secure outsourcing of large-scale convex separable programming for big data. *IEEE Transactions on Big Data*, 2017.

Yehuda Lindell and Jonathan Katz. *Introduction to modern cryptography*. Chapman and Hall/CRC, 2014.

Fang Liu, Wee Keong Ng, and Wei Zhang. Encrypted gradient descent protocol for outsourced data mining. In *Advanced Information Networking and Applications (AINA), 2015 IEEE 29th International Conference on*, pages 339–346. IEEE, 2015.

Changqing Luo, Kaijin Zhang, Sergio Salinas, and Pan Li. Secfact: Secure large-scale qr and lu factorizations. *IEEE Transactions on Big Data*, 2017.

Changqing Luo, Jinlong Ji, Xuhui Chen, Ming Li, Laurence T Yang, and Pan Li. Parallel secure outsourcing of large-scale nonlinearly constrained nonlinear programming problems. *IEEE Transactions on Big Data*, 2018.

Claudio Orlandi, Alessandro Piva, and Mauro Barni. Oblivious neural network computing via homomorphic encryption. *EURASIP Journal on Information Security*, 2007(1): 037343, 2007.

Nicolas Papernot, Patrick D. McDaniel, Arunesh Sinha, and Michael P. Wellman. Towards the science of security and privacy in machine learning. *CoRR*, abs/1611.03814, 2016. URL http://arxiv.org/abs/1611.03814.

Cheolsoo Park, Clive Cheong Took, and Joon-Kyung Seong. Machine learning in biomedical engineering. *Biomedical Engineering Letters*, 8(1):1–3, Feb 2018. ISSN 2093-985X. doi: 10.1007/s13534-018-0058-3. URL https://doi.org/10.1007/s13534-018-0058-3.

Alessandro Piva, Claudio Orlandi, M Caini, Tiziano Bianchi, and Mauro Barni. Enhancing privacy in remote data classification. In *IFIP International Information Security Conference*, pages 33–46. Springer, 2008.

Sergio Salinas, Xuhui Chen, Jinlong Ji, and Pan Li. A tutorial on secure outsourcing of large-scale computations for big data. *IEEE Access*, 4:1406–1416, 2016a.

Sergio Salinas, Changqing Luo, Weixian Liao, and Pan Li. Efficient secure outsourcing of large-scale quadratic programs. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pages 281–292. ACM, 2016b.

Sergio Salinas, Changqing Luo, Xuhui Chen, Weixian Liao, and Pan Li. Efficient secure outsourcing of large-scale sparse linear systems of equations. *IEEE Transactions on Big Data*, 4(1):26–39, 2018.

Amir Shpilka, Amir Yehudayoff, et al. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends® in Theoretical Computer Science*, 5(3–4): 207–388, 2010.

Richard Socher, Yoshua Bengio, and Christopher D Manning. Deep learning for nlp (without magic). In *Tutorial Abstracts of ACL 2012*, pages 5–5. Association for Computational Linguistics, 2012.

Karthik Mahendra Raje Urs. Harnessing the cloud for securely outsourcing large-scale systems of linear equations. *IEEE Transactions on Parallel and Distributed Systems*, 24: 1172–1181, 2013.

Aravind Vasudevan, Andrew Anderson, and David Gregg. Parallel multi channel convolution using general matrix multiplication. In *Application-specific Systems, Architectures and Processors (ASAP), 2017 IEEE 28th International Conference on*, pages 19–24. IEEE, 2017.