Machine Learning with Partially Homomorphic Encrypted Data

IOP Conf. Series: Journal of Physics: Conf. Series **1108** (2018) 012112 doi:10.1088/1742-6596/1108/1/012112

Machine Learning with Partially Homomorphic Encrypted Data

K Muhammad[1], K A Sugeng[1,*] and H Murfi[1]

[1]Department of Mathematics, Universitas Indonesia, Kampus UI Depok, Depok 16424, INDONESIA
E-mail: [1]khalid.muhammad@sci.ui.ac.id, [1,*]kiki@sci.ui.ac.id, [1]hendri@ui.ac.id

Abstract. Machine learning had been widely used to analyze various kinds of data, including sensitive data such as medical and financial data. A trained machine learning model can be wrapped in a web application so that people can access it easily via internet. However, if the data to be analyzed is private or confidential, this will cause a problem; the application administrator may read the input. As shown by Dowlin et al. in their remarkable paper, this kind of problem can be solved with homomorphic encryption scheme. Paillier encryption scheme is one kind of encryption scheme that has homomorphic property. In this research, we will show that one type of machine learning model can take an input encrypted by Paillier encryption scheme and produce an encrypted output that shares the same key. A machine learning model will be trained with the MNIST database of hand-written digits. This model will be tested with the test data encrypted with Paillier encryption scheme. The experiment shows that the model achieved 92.92% accuracy on the test set.

1. Introduction

Machine learning had been widely used in various fields, from computer vision, natural language processing, to medical and finance. Goodfellow et al. defined machine learning algorithm as an algorithm that is able to learn from data; machine learning enables us to tackle tasks that are too difficult to solve with fixed programs written and designed by human beings [1]. A well-trained machine learning model can be wrapped in a web application so that people can access it easily via internet, Google Translate is such an example of this. However, people won't use it to analyze a private data because the application administrator may read their input. As shown by Dowlin et al., homomorphic encryption schemes have the ability to overcome this kind of problem. It preserves the structure of the message space such that we can perform operations such as addition and multiplication over the ciphertext space [2]. With a homomorphic encryption scheme, one can encrypt his/her data in advance before he/she inputs it to the machine learning model. The machine learning model will produce an encrypted output that shares the same key. The person with the key can then decrypt it. This decrypted output will have the same value to the output that are produced by taking the unencrypted data as an input. This way, the person can still uses the machine learning model with his/her private data without worrying that the application administrator will read his/her data. Furthermore, the application administrator also can't read the output produced by the application because it is still in an encrypted form.

This research is inspired by the remarkable work of Downlin et al. [3], they trained a deep convolutional neural network model with the MNIST database [4] of handwritten digits. The trained model is then modified to what they call CryptoNets, neural networks that can take an input encrypted

by a homomorphic encryption. They use YASHE' encryption scheme, a fully homomorphic encryption scheme, to encrypt the test set. The CryptoNet model achieved 98.9% accuracy when tested with the encrypted test set.

One of the encryption schemes that has a homomorphic property is Paillier encryption scheme, firstly introduced in 1999 by Pascal Paillier on his paper "Public-key cryptosystems based on composite degree residuosity classes" [5]. This encryption scheme is proven to be semantically secure against chosen-plaintext attack, moreover it is more efficient than Goldwasser-Micali cryptosystem, as well as the provably-secure RSA and Rabin schemes [6]. Paillier encryption scheme supports addition of ciphertexts and multiplication of a ciphertext with a plaintext (an unencrypted scalar). With this property in hand, we can do matrix multiplications and vector additions. Similar to what Dowlin et al. has shown, we will see that a trained shallow neural network model can be simplified to a matrix multiplication and a vector addition, hence this model can take an input encrypted by Paillier encryption scheme. We will also demonstrate this idea by training a model with the MNIST database [4].

2. Paillier Encryption Scheme

Paillier encryption scheme is an asymmetric encryption scheme. It uses (n, g) as the public-key, where n is a product of two big prime numbers p and q, and g is an element of $Z^*_{n^2}$ such that its order is a multiple of n. The two prime numbers p and q are used as a secret-key.

One version of this encryption scheme sets g equals to $n + 1$ and requires (p, q) to be a prime of equal length (in their binary representation). The requirement assures that $\phi(n) = (p-1)(q-1)$ has an inverse modulo n. The inverse of $\phi(n)$ is denoted by μ and used as the secret-key [5]. The scheme is summarized in Figure 1.

Key Generation:

- $p, q \in P$ with equal length
- $n = pq$
- $g = 1 + n$
- $\phi(n) = (p-1) \cdot (q-1)$
- $\mu = \phi(n)^{-1} \, mod \, n$ (μ is used as *private-key*)

Encryption:

- Plaintext $m < n$ $(m \in Z_n)$
- Choose $r < n$ dengan $\gcd(r, n) = 1$ randomly $(r \in Z^*_n)$
- Ciphertext $c = g^m . r^n \, mod \, n^2$

Decryption:

- Ciphertext $c < n^2$
- Plaintext $m = L\left(c^\lambda \, mod \, n^2\right) \cdot \mu \, mod \, n$

Figure 1. Paillier encryption scheme

Suppose that two plaintexts m_1 and m_2 are encrypted to $e(m_1) = g^{m_1}.r_1^n \ mod\ n^2$ and $e(m_2) = g^{m_2}.r_2^n \ mod\ n^2$ where $r_1, r_2 \in Z_n^*$,. We can see that

$$e(m_1) \cdot e(m_2)\ mod\ n^2 = (g^{m_1}.r_1^n \ mod\ n^2) \cdot (g^{m_2}.r_2^n \ mod\ n^2)\ mod\ n^2$$

$$e(m_1) \cdot e(m_2)\ mod\ n^2 = g^{m_1}.r_1^n.g^{m_2}.r_2^n \ mod\ n^2$$

$$e(m_1) \cdot e(m_2)\ mod\ n^2 = g^{m_1+m_2}.(r_1.r_2)^n \ mod\ n^2.$$

Since g has order n in $\mathrm{Z}_{\mathrm{n}^2}^*$ and $r_1, r_2 \in \mathrm{Z}_{\mathrm{n}}^*$ then

$$e(m_1) \cdot e(m_2)\ mod\ n^2 = g^{m_1+m_2\ mod\ n}.(r_1.r_2\ mod\ n)^n \ mod\ n^2, \text{ where } r_1.r_2\ mod\ n \in \mathrm{Z}_{\mathrm{n}}^*$$

$$e(m_1) \cdot e(m_2)\ mod\ n^2 = e(m_1 + m_2\ mod\ n) \quad (1)$$

Furthermore, if $m_1 + m_2 < n$ then

$$e(m_1) \cdot e(m_2)\ mod\ n^2 = e(m_1 + m_2).$$

Because of this property, Paillier encryption scheme is called a partially homomorphic encryption scheme. This property can be generalize to addition of n plaintexts

$$e(m_1) \cdot e(m_2) \cdot \ldots \cdot e(m_n)\ mod\ n^2 = e(m_1 + m_2 + \cdots + m_n\ mod\ n).$$

Now suppose that k is a positive integer then

$$e(m_1)^k\ mod\ n^2 = (g^{m_1} \cdot r_1^n\ mod\ n^2)^k\ mod\ n^2$$

$$e(m_1)^k\ mod\ n^2 = (g^{m_1} \cdot r_1^n)^k\ mod\ n^2$$

$$e(m_1)^k\ mod\ n^2 = (g^{m_1})^k \cdot (r_1^n)^k\ mod\ n^2$$

$$e(m_1)^k\ mod\ n^2 = g^{k.m_1} \cdot \left(r_1^k\right)^n\ mod\ n^2$$

Since g has order n in $\mathrm{Z}_{\mathrm{n}^2}^*$ and $r_1 \in \mathrm{Z}_{\mathrm{n}}^*$,

$$e(m_1)^k\ mod\ n^2 = g^{k \cdot m_1\ mod\ n} \cdot \left(r_1^k\ mod\ n\right)^n\ mod\ n^2, \text{ where } r_1^k\ mod\ n \in \mathrm{Z}_{\mathrm{n}}^*$$

$$e(m_1)^k\ mod\ n^2 = e(k \cdot m_1\ mod\ n) \quad (2)$$

Furthermore, if $km_1 < n$ then

$$e(m_1)^k\ mod\ n^2 = e(k \cdot m_1)$$

We will make use of these two properties to build a system where a machine learning model can take an input encrypted by Paillier encryption scheme and produce an output encrypted with the same key.

3. The Machine Learning Model

First, we will train a neural network model with description given as below:

1. Input layer with 784 neurons corresponding to flattened array of size 28x28 of the MNIST image data.
2. First hidden layer: fully connected (dense) layer with 256 neurons and identity function as the activation function
3. Second hidden layer: fully connected layer with 128 neurons, also having identity function as the activation function
4. Output layer with 10 neurons corresponding to 10 classes of the MNIST database (image of handwritten digits of the integers from 0 to 9).
5. There are $(784 \times 256) + (256 \times 128) + (128 \times 10) = 234.752$ weights parameter and $256 + 128 + 10 = 394$ biases parameter that can be optimized in the training process.

This model can be written in a form of matrix multiplications as below:

$$a^{(2)} = W^{(2)}a^{(1)} + b^{(2)} \quad (3)$$

$$a^{(3)} = W^{(3)}a^{(2)} + b^{(3)}$$
$$\hat{y} = a^{(4)} = softmax\left(W^{(4)}a^{(3)} + b^{(4)}\right)$$

where:

- $a^{(1)}$, $a^{(2)}$, $a^{(3)}$, and $a^{(4)}$ respectively is a column vector of length 784, 256, 128, and 10.
- $b^{(2)}$, $b^{(3)}$, and $b^{(4)}$ respectively is a column vector of length 256, 128, 10.
- $W^{(2)}$, $W^{(3)}$, and $W^{(4)}$ respectively is a matrix of size 256x784, 128x256, 10x128.
- $\hat{y}$ or $a^{(4)}$ is the output of the model in the form of a column vector of length 10.

After the training phase, the weights and biases will have a fixed value, so the model can then be simplified as

$$\hat{y} = softmax\left(W^{(4)}(W^{(3)}(W^{(2)}a^{(1)} + b^{(2)}) + b^{(3)}) + b^{(4)}\right)$$
$$\hat{y} = softmax\left(W^{(4)}W^{(3)}W^{(2)}a^{(1)} + W^{(4)}W^{(3)}b^{(2)} + W^{(4)}b^{(3)} + b^{(4)}\right)$$
$$\hat{y} = softmax\left(W'^{(2)}a^{(1)} + b'^{(2)}\right),$$

where

$$W'^{(2)} = W^{(4)}W^{(3)}W^{(2)} \text{ and } b'^{(2)} = W^{(4)}W^{(3)}b^{(2)} + W^{(4)}b^{(3)} + b^{(4)},$$

is a matrix of size 10x784 and a column vector of length 10 respectively [7].

Because of the softmax activation function in the output layer, this model cannot take an input encrypted with Paillier encryption scheme yet, softmax function is not a function that only consists of addition and scalar multiplication. However, we will see that the model is able to predict the input correctly without computing the softmax function in the output layer.

Neural networks model determine the inputs' class by looking at the index of the maximum value of $\hat{y}$. On the other hand, softmax function dosen't change the order of its input, for example if the input vector is $x = (8,7,10,6,9)$, $softmax(x) = (0.086, 0.031, 0.636, 0.011, 0.234)$, we see that the elements of x and $softmax(x)$ have the same order. In general, suppose that $x = (x_1, x_2, \dots, x_n)$ and WLOG we can assume that $x_1 < x_2 < \cdots < x_n$, since e is greater than 1 we can see that $e^{x_1} < e^{x_2} < \cdots < e^{x_n}$, finally since $\sum e^{x_i} = e^{x_1} + e^{x_2} + \cdots + e^{x_n}$ is positive then $\frac{e^{x_1}}{\sum e^{x_i}} < \frac{e^{x_2}}{\sum e^{x_i}} < \cdots < \frac{e^{x_n}}{\sum e^{x_i}}$, thus we have proven that in general case the elements of x and $softmax(x)$ have the same order. We can now safely drop the softmax function in the output layer of the model. The model is now simplified to

$$\hat{y} = W'^{(2)}a^{(1)} + b'^{(2)}. \quad (4)$$

Suppose that Alice have successfully trained a model that has a structure like (3) which she then simplify to (4). She then builds a web application so that people can access it easily via internet. Bob wants to use the model with his private data $m = [m_1\ m_2 \dots m_{784}]^T$. The output that Bob looking for is $\hat{y} = Wm + b$. Because m is private, Bob will encrypt it first to $e(m) = [e(m_1)\ e(m_2) \dots e(m_3)]^T$ which he then inputs it to Alices' app. In order to get $e(\hat{y})$, the app have to compute

$$\begin{bmatrix} \left((e(m_1)^{w_{1,1}} \cdot e(m_2)^{w_{2,1}} \cdot \dots \cdot e(m_{784})^{w_{784,1}}) \cdot b_1\right) mod\ n^2 \\ \left((e(m_1)^{w_{1,2}} \cdot e(m_2)^{w_{2,2}} \cdot \dots \cdot e(m_{784})^{w_{784,2}}) \cdot b_2\right) mod\ n^2 \\ \vdots \\ \left((e(m_1)^{w_{1,10}} \cdot e(m_2)^{w_{2,10}} \cdot \dots \cdot e(m_{784})^{w_{784,10}}) \cdot b_{10}\right) mod\ n^2 \end{bmatrix} \quad (5)$$

which equals to

$$\begin{bmatrix} \left(e\left(m_1 \cdot w_{1,1} + m_2 \cdot w_{2,1} + m_{784} \cdot w_{784,1} + b_1\right)\right) mod\ n^2 \\ \left(e\left(m_1 \cdot w_{1,2} + m_2 \cdot w_{2,2} + m_{784} \cdot w_{784,2} + b_2\right)\right) mod\ n^2 \\ \vdots \\ \left(e\left(m_1 \cdot w_{1,10} + m_2 \cdot w_{2,10} + m_{784} \cdot w_{784,10} + b_{10}\right)\right) mod\ n^2 \end{bmatrix}$$

$$= [e(\hat{y}_1)\ e(\hat{y}_2) \dots\ e(\hat{y}_{10})]^T = e(\hat{y})$$

Therefore, using equation (5) Alices' app can outputs $e(\hat{y})$ when given $e(m)$ as input. Bob can decrypt this this output to $\hat{y}$, i.e. the value that he was looking for from the trained model. This way, Bob can still obtain the value that he was looking for from Alices' model without having to give his private data to Alice. Furthermore, Alice, as the application admin, won't be able to read the output—which may be confidential as well—since it is in an encrypted form. Figure 2 gives an illustration on how this concept works.

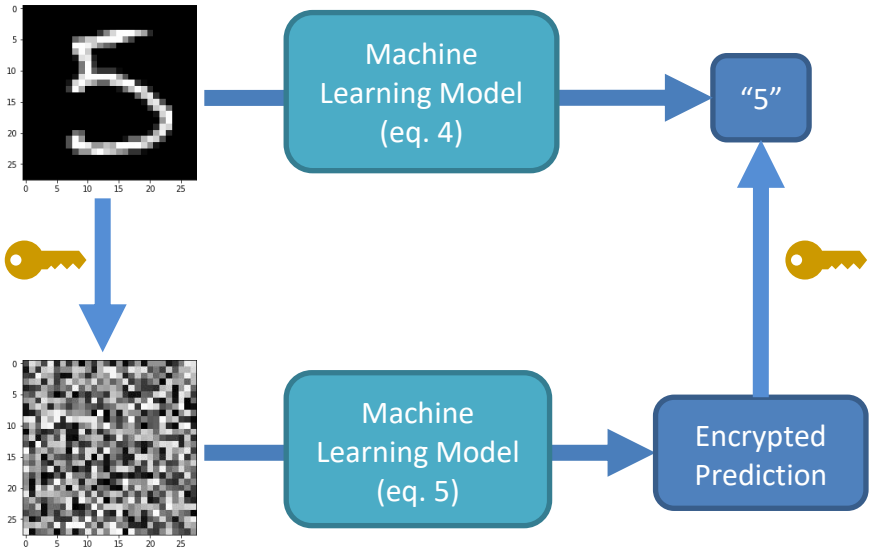

Figure 2. Summary

4. Empirical Result

We will implement the whole concept with Python programming language [8]. Three main library that we will use are Keras [9] for training the neural network model, NumPy [10] for matrix operations, and python-paillier [11] for using Paillier homomorphic encryption scheme. We use matplotlib [12] to plot the training progress.

We train the model in 150 epochs with Adam [13] as the optimizer. The best model is obtained in the 122th epoch with an accuracy of 92.92% on the test set. Figure 3 illustrates the training progress. After the training phase, we can extract the weights and biases of the trained model to NumPy arrays. We can then simplify the model to (4) using NumPy array operations. This simplified model—the model that doesn't have softmax on the output layer—gave the same output as the initial model. This confirms what we have discussed in the last section.

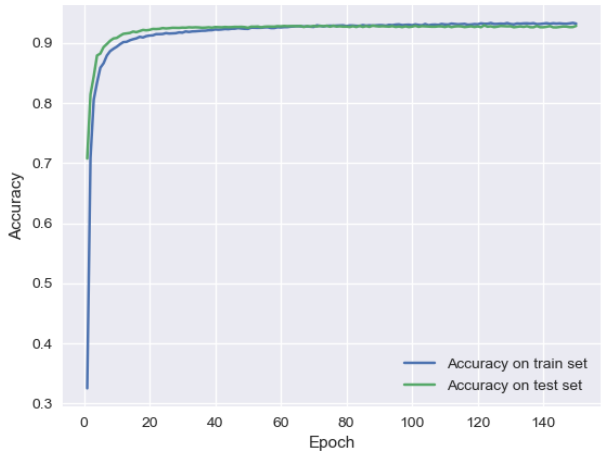

Figure 3. Training progress

The test set is encrypted pixel-by-pixel using phe, the encoding from floating point numbers to integers is handled by phe. Using equation (5), we get an encrypted prediction of the test set. After we decrypt the output of equation (5), we see that it results the same class prediction with the simplified model, it also gives a 92.92% accuracy.

5. Conclusion

Paillier encryption scheme is homomorphic over addition and scalar multiplication operation of ciphertexts. We have trained a neural network model and simplify the model to one matrix multiplication and one vector addition. By using the weights and biases of this simplified model, we can generate an encrypted prediction for private data.

Fully homomorphic encryption scheme such as FV [14] and YAHSE' [15] can be used for further research. This type of encryption allows addition and multiplication of ciphertexts, using these two operations we can compute polynomials, and by polynomials there are quite many thing we can do such as approximating functions using Taylors' theorem. Hesamifard, Takebi, & Ghasemi used approximation of ReLU function with polynomials to be used as an activation function in a neural network hidden layer [2]. Applying this concept for financial or medical data will also be interesting.

6. References

[1] Goodfellow I, Bengio Y, and Courville A 2016 Deep learning (Cambridge: MIT press)
[2] Hesamifard E, Takabi H, and Ghasemi M 2017 CryptoDL: Deep Neural Networks over Encrypted Data *Preprint arXiv:1711.05189*
[3] Dowlin N, Gilad-Bachrach R, Laine K, Lauter K, Naehrig M, and Wernsing J 2016 Cryptonets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy *International Conference on Machine Learning* pp 201-10
[4] LeCun Y 1998 The MNIST Database of Handwritten Digits http://yann.lecun.com/exdb/mnist/
[5] Paillier P 1999 Public-key Cryptosystems Based on Composite Degree Residuosity Classes *International Conference on the Theory and Applications of Cryptographic Techniques* pp 223-38 (Springer, Berlin, Heidelberg)
[6] Katz J and Lindell Y 2014 *Introduction to Modern Cryptography* (CRC press)
[7] Nielsen M A 2015 Neural Networks and Deep Learning (Determination Press) http://neuralnetworksanddeeplearning.com
[8] Van Rossum G 2007 Python Programming Language *USENIX Annual Technical Conference* **41** p 36
[9] Chollet F 2015 Keras *GitHub repository* https://github.com/keras-team/keras
[10] Oliphant T E 2006 A Guide to NumPy (USA: Trelgol Publishing)
[11] N1 Analytics 2014 python-paillier *GitHub repository* https://github.com/n1analytics/python-paillier
[12] Hunter J D 2007 Matplotlib: A 2D Graphics Environment *Computing in Science & Engineering*
[13] Kingma D P and Ba J 2014 Adam: A Method for Stochastic Optimization *Preprint arXiv*:1412.6980
[14] Fan J and Vercauteren F 2012 Somewhat Practical Fully Homomorphic Encryption *IACR Cryptology ePrint Archive*
[15] Bos J W, Lauter K, Loftus J, and Naehrig M 2013 Improved Security for a Ring-based Fully Homomorphic Encryption Scheme *IMA International Conference on Cryptography and Coding* pp 45-64 (Springer, Berlin, Heidelberg)

Acknowledgments

This research is funded by PITTA-UI 2018 Research Grant [Project Number = 2277/UN2.R3.1/HKP.05.00/2018].