서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

香港科技大學
THE HONG KONG
UNIVERSITY OF SCIENCE
AND TECHNOLOGY

SNU & HKUST - 2019

Summer Research Program in Industrial and Applied Mathematics

Final Report:

Homomorphic Encryption Based Secure Solutions

Sponsor

CryptoLab

Industrial Mentor

Andrey Kim

Academic Mentor

Jaehyun Park

Student members:

Minyi Wang, Qinglan Cao
Wootae Kim, Siwoo Park

Contents

1 Introduction **3**

2 Homomorphic Encryption **4**
2.1 HEAAN Library 4

3 Logistic Regression Model Training Phase **5**
3.1 Plaintext Logistic Regression 5
3.2 Preparation for Encrypted Logistic Regression 5
3.2.1 Packing Method 6
3.2.2 Approximate Sigmoid function 6
3.3 Encrypted Logistic Regression 7

4 CNN Model Prediction Phase **8**
4.1 LeNet1 Model 8
4.2 HE-friendly CNN architecture 8
4.2.1 Max Pooling Layer 8
4.2.2 Softmax Layer 9
4.3 Packing Method 9
4.4 Encrypted Convolutional Layer 10
4.5 Encrypted Average Pooling Layer 12
4.6 Encrypted Fully Connected Layer 15

5 Conclusion **17**
5.1 Summary 17
5.2 Limitations and Possible Future Directions 17
5.3 Acknowledgements 17

6 Bibliography **18**

Abstract

This project aims to develop homomorphic encryption based secure solutions over encrypted data to make predictions about the dependent variable Y when given the data consisting of independent variables (predictors Xi). In order to acheive that, homomorphic encrypted machine learning models based on HEAAN (Homomorphic Encryption for Arithmetics of Approximate Numbers) library were devised. We first trained a logistic regression model concerning encrypted data. Due to the limited operations in the encrypted state, special packing methods, bootstrapping methods and approximate logistic function are designed to satisfy the requirements. The estimated parameters got relatively reasonable results when compared with the original parameters learned on plaintexts. Then we built a LeNet1 Convolutional Neural Network for prediction phase and also obtained results with acceptable quality during the comparison testing with plaintexts. In this paper we provide the detailed processes of our work and some of the possible improvements which can be attempted in the future.

1 Introduction

The goal of this project is to continue applying machine learning classifiers used in plaintexts to encrypted texts based on HEAAN library. In chapter 2 we briefly introduce the content of homomorphic encryption and HEAAN library, which is one of the encryption schemes. HEAAN library stores data in vector form and only supports polynomials. Since all the operations in the encrypted state depend on the library, specific functions and methods are required to implement such as approximate sigmoid function and packing methods, which are clarified in the steps needed in this report. Logistic regression model training phase and CNN model prediction phase are demonstrated in chapter 3 and chapter 4 respectively. The logistic regression model also includes Nesterov's Accelerated Gradient Descent. Both are explained with pseudocodes and conducted comparisons between plaintext and ciphertext. In the last chapter, we present current limitations and possible future development directions.
All the code we used has been made available to public at https://github.com/tldn0718/Homomorphic-Encryption-for-SPIA2019. Any kind of interest or contribution to our project would be welcomed.

2 Homomorphic Encryption

A cryptosystem is *homomorphic* for some operation * iff the encryption function Enc satisfies

$$\mathrm{Enc}(\mathbf{m}_1) * \mathrm{Enc}(\mathbf{m}_2) = \mathrm{Enc}(\mathbf{m}_1 * \mathbf{m}_2).$$

More generally we want

$$f(\mathrm{Enc}(\mathbf{m}_1), \ldots, \mathrm{Enc}(\mathbf{m}_s)) = \mathrm{Enc}(f(\mathbf{m}_1, \ldots, \mathbf{m}_s))$$

Normally in practice we consider operations $(+, \times)$.
With homomorphic encryption, ciphertexts can be publicly operated without decryption to ensure end-to-end semantically secure.

2.1 HEAAN Library

HEAAN is a library that implements homomorphic encryption scheme for arithmetics of approximate numbers. The main idea is to treat an encryption noise as part of error occurring during approximate computations. That is, an encryption ct of message m by the secret key sk will have a decryption structure of the form $\langle ct, sk \rangle = m + e \pmod q$ where e is a small error inserted to guarantee the security of hardness assumptions such as the learning with errors (LWE), the ring-LWE (RLWE) and the NTRU problems. If e is small enough compared to the message, this noise is not likely to destroy the significant figures of m and the whole value $\tilde{m} = m + e$ can replace the original message in approximate arithmetic. One may multiply a scale factor to the message before encryption to reduce the precision loss from encryption noise. For homomorphic operations, we always maintain our decryption structure small enough compared to the ciphertext modulus so that computation result is still smaller than q.

3 Logistic Regression Model Training Phase

3.1 Plaintext Logistic Regression

Logistic regression is one of the statistical techniques for inferring the values of discrete dependent variables from the input values of several independent variables.
Concretly, dataset for training logistic regression consists of n pairs $(\mathbf{x}_i, y_i)$ where $i = 1, \ldots, n$. Here, $\mathbf{x}_i \in \mathbb{R}^f$ is a vector of independent variables which have f features and y_i is a dependent variable according to $\mathbf{x}_i$. The purpose of logistic regression is to find out $\beta \in \mathbb{R}^{f+1}$ which expresses

$$\Pr(y_i|\mathbf{x}_i) = \frac{1}{1 + \exp(-y_i(1, \mathbf{x}_i)^T\beta)} = \sigma(-y_i(1, \mathbf{x}_i)^T\beta)$$

for best where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function. If the dependent variable is assumed to be in only two cases, like success and failure, the value can be expressed as ± 1. This problem can be rephrased by calculating $\beta \in \mathbb{R}^{f+1}$ which maximizes the likelihood

$$\prod_{i=1}^{n} \Pr(y_i|\mathbf{x}_i) = \prod_{i=1}^{n} \frac{1}{1 + \exp(-y_i(1, \mathbf{x}_i)^T\beta)},$$

or equivalently minimizes the loss function, defined as the negative log-likelihood:

$$J(\beta) = \frac{1}{n} \sum_{i=1}^{n} \log(1 + \exp(-\mathbf{z}_i^T\beta))$$

where $\mathbf{z}_i = y_i \cdot (1, \mathbf{x}_i)$ for $i = 1, \ldots, n$.
For gradient descent, we have to calculate the gradient of the cost function with respect to β which can be expressed by

$$\nabla J(\beta) = -\frac{1}{n} \sum_{i=1}^{n} \sigma(-\mathbf{z}_i^T\beta) \cdot \mathbf{z}_i$$

. We choose a random initial $\beta^{(0)}$ at first of the gradient descent. Then at each time step t we update the value of β using the equation

$$\beta^{(t+1)} \leftarrow \beta^{(t)} + \frac{\alpha_t}{n} \sum_{i=1}^{n} \sigma(-\mathbf{z}_i^T\beta^{(t)}) \cdot \mathbf{z}_i$$

where α_t is a learning rate at step t.

3.2 Preparation for Encrypted Logistic Regression

Since possible operations in encrypted states are very limited, training logistic regression model using homomorphic encryption is more difficult than training in plaintext state. We have to implement the gradient descent method using only addition and multiplication between ciphertexts, constants multiplication, and rotate sum. In this process, we have to consider how to pack data, which is how to encode it for encryption. In addition, since it is impossible to evaluate the transcendental function in the homomorphic encryption, the Sigmoid function required in the logistic regression training process should be approximated as a polynomial function. In this section, we explain how we pack the dataset and approximate Sigmoid function to polynomial.

3.2.1 Packing Method

The dataset we have is the set of $\mathbf{z}_i = y_i \cdot (1, \mathbf{x}_i) \in \mathbb{R}^{f+1}$ for $i = 1, \ldots, n$. Thus we consider the data set as a matrix of $\mathbb{R}^{n \times (f+1)}$ by encoding each one data as a row of matrix. It is not too hard to pad the matrix such that its numbers of row and column will be power of 2. Second, we convert this matrix to a long one-dimension array by unroll the matrix row-wise.

$$Z = \begin{bmatrix} z_{10} & z_{11} & \cdots & z_{1f} \\ z_{20} & z_{21} & \cdots & z_{1f} \\ \vdots & \vdots & \ddots & \vdots \\ z_{n0} & z_{n1} & \cdots & z_{nf} \end{bmatrix}.$$

3.2.2 Approximate Sigmoid function

The most common way to approximate the transcendence function as a polynomial function is the Taylor Approximation. However, since the Taylor Approximation is a local method that approximates a function only near a certain point, there is a very large error for approximating a function over an interval. Thus, we used the Least Square Method to approximate on a certain interval. Least Square Method is to find coefficients of $l_n(x)$ that minimizes the mean square error (MSE) to approximate the function $f(x)$ defined in interval I to polynomial $l_n(x)$ of degree n. The mean square error is defined as the integral value of square the difference between the two functions, i.e.

$$MSE = \int_I (f(x) - l_n(x))dx$$

This problem is fully solved and has a solution of closed form. If we define the inner product of two function as below:

$$< f, g >= \int_I (f(x)g(x))dx$$

then the approximated polynomial $l_n$is as below:

$$l_n(x) = \sum_{j=0}^{n} \frac{< f, P_j >}{< P_j, P_j >} P_j(x)$$

where P_j is the Legendre polynomial of degree j We approximate sigmoid function for degree 3 over $[-8, 8]$. The approxmited polynomial is

$$p(x) = 0.5 + 0.15012x - 0.001593x^3$$

where the largest error 0.11432 is at $x = -8$. We can evaluate this polynomial faster as below:

$$p(x) = (-0.001593x^2 + 0.15012)x + 0.5$$

3.3 Encrypted Logistic Regression

We implement the encrypted logistic regression based on the algorithm presented in the paper of Kim et al.[1]. The brief explanation of the algorithm is

Algorithm 1 Encrypted Logistic Regression

procedure LR($\mathtt{ct}_z, \mathtt{ct}_\beta$)

$\mathtt{ct} \leftarrow \mathtt{ReScale}(\mathtt{Mult}(\mathtt{ct}_\beta, \mathtt{ct}_z); p)$

for $j = 0$ to $\log(f+1) - 1$ **do**

$\mathtt{ct} \leftarrow \mathtt{Add}(\mathtt{ct}, \mathtt{Rotate}(\mathtt{ct}; 2^j))$

end for

$c \leftarrow \mathtt{Encode}(C; p_c)$

$\mathtt{ct} \leftarrow \mathtt{ReScale}(\mathtt{CMult}(\mathtt{ct}; c); p_c)$

for $j = 0$ to $\log(f+1) - 1$ **do**

$\mathtt{ct} \leftarrow \mathtt{Add}(\mathtt{ct}, \mathtt{Rotate}(\mathtt{ct}; -2^j))$

end for

$\mathtt{ct} \leftarrow p(ct)$

for $j = \log(f+1)$ to $\log(f+1) + \log n - 1$ **do**

$\mathtt{ct}_6 \leftarrow \mathtt{Add}(\mathtt{ct}, \mathtt{Rotate}(\mathtt{ct}; 2^j))$

end for

$\mathrm{ct} \leftarrow \mathtt{ReScale}(\Delta \cdot \mathtt{ct}; p_c) \quad \mathtt{ct}_\beta \leftarrow \mathtt{Add}(\mathtt{ct}_\beta, \mathtt{ct})$

Here, each matrix is of form as below:

$$C = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \cdots & 0 \end{bmatrix},$$

.

$$\mathtt{ct}_z = \mathtt{Enc} \begin{bmatrix} z_{10} & z_{11} & \cdots & z_{1f} \\ z_{20} & z_{21} & \cdots & z_{1f} \\ \vdots & \vdots & \ddots & \vdots \\ z_{n0} & z_{n1} & \cdots & z_{nf} \end{bmatrix},$$

$$\mathtt{ct}_\beta = \mathtt{Enc} \begin{bmatrix} \beta_0 & \beta_1 & \cdots & \beta_f \\ \beta_0 & \beta_1 & \cdots & \beta_f \\ \vdots & \vdots & \ddots & \vdots \\ \beta_0 & \beta_1 & \cdots & \beta_f \end{bmatrix}.$$

4 CNN Model Prediction Phase

4.1 LeNet1 Model

Convolutional neural network (CNN) which is a widely applied multi-layer neural network to analyze visual imagery has been developing rapidly in recent years. One of the early models, LeNet-1 [2], is designed to learn pixel images of handwritten characters. Taking pixel images of size 28x 28 as input, it consists of 5 layers with weights, including 2 convolutional layers, 2 subsampling layers and 1 fully connected layer at the end, as illustrated in the graph below.

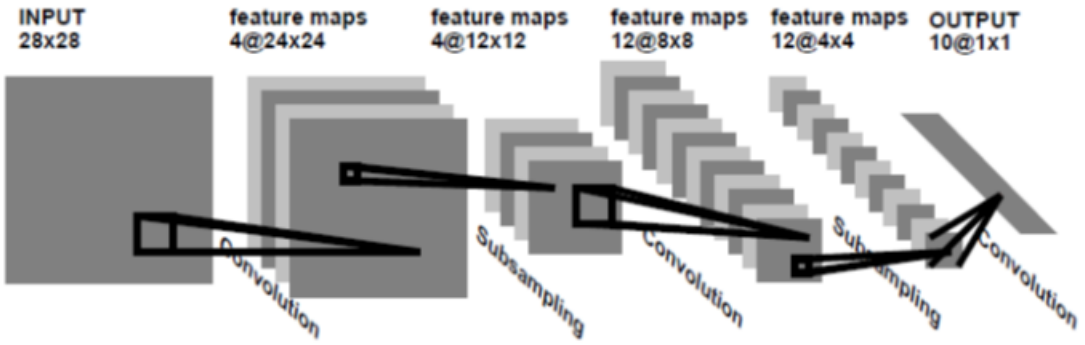

Figure 1: LeNet-1 [5]

4.2 HE-friendly CNN architecture

CNN has a wide variety of structures because it is one of the areas that is currently being studied deeply. Choosing more advantageous structure to homomorphic encryption among these is critical to optimizing the results of our experiments. We considered Max Pooling layer and Softmax layer as bottleneck and finds a nicer algorithm to replace it.

4.2.1 Max Pooling Layer

Pooling layer serves to progressively mitigate the spatial size of the representation and reduce the amount of parameters and computation in the network, so it is usually followed by the convolutional layers. The layer computes a single output for each location traversed by the fixed-shape window (sometimes known as the pooling window).[1]

Max pooling layer is currently the most popular pooling method among the various pooling methods, which can be said to be the only method used nowadays. Usually max pooling divides the input values into square areas that do not overlap each other, and then outputs only the largest value for each area. For example, a max pooling layer, which takes 4 by 4 matrix as input value and outputs a 2 by 2 matrix, divides the input matrix into 4 squares, and outputs only the largest of the 4 values in each area. Below is the concrete example.

$$\begin{bmatrix} 3 & 7 & -1 & 5 \\ 5 & 1 & 2 & -3 \\ 3 & 1 & 2 & 4 \\ 6 & -2 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & -3 \\ -2 & 0 \end{bmatrix}$$

Although max pooling may be the most commonly used pooling method at present, it is very inefficient in terms of homomorphic encryption. Max pooling requires a lot of comparision computations, while in HEEAAN, comparision computations are implemented by:

$$\max(x, y) = \lim_{n \to \infty} \frac{a^{n+1} + b^{n+1}}{a^n + b^n}$$

This requires a lot of multiplications and also needs division, which is inefficient in HEAAN. As a result, max pooling in HEAAN tend to be slower than other pooling methods. Therefore, we choose the average pooling layer, which outputs the average of values of each area instead of maximum value. This requires only a few additions and one multiplication by constant, which it is much more efficient than max pooling. As for the concrete example above, the output will be as below:

$$\begin{bmatrix} 3 & 7 & -1 & 5 \\ 5 & 1 & 2 & -3 \\ 3 & 1 & 2 & 4 \\ 6 & -2 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 4 & 0.75 \\ 2 & 1.75 \end{bmatrix}$$

4.2.2 Softmax Layer

Softmax layer is a layer which changes the output of DNN network to a probability value for the last output. Usually, it is at the end of the network and calculates the probability with which the input value corresponds to the specific label. By definition, the sum of probability values for all labels is to be 1, and the formula is as follows:

$$\Pr(y_i|X_i) = \frac{e^{z_i}}{\sum_{k=1}^{n} e^{z_k}}$$

where n is the number of labels, z_i is the value of output corresponding to label i.

Evaluating softmax layer requires a lot of evaluation on exponential function. Since non-linear functions are difficult to compute in HEAAN, we have to approximate it using polynomials as we do for sigmoid function. But unlike the example of logistic regression, the values vary drastically so we have to approximate exponential function in a wide range. This implies calculation in a much higher degree which is inefficient. Thus we do not evaluate this layer and simply return the input value of softmax layer. If the client receives these values, he or she can decrypt it and choose the largest value. Then that result would be same as the output of softmax layer.

4.3 Packing Method

Since the HEAAN library only supports power of 2 number of slots, we pack the input through enlarging the matrix from 28× 28 to 32×32 by adding zeros at the end:

$$\left[\begin{array}{cccc|ccc} z_{0,0} & z_{0,1} & \cdots & z_{0,27} & 0 & \cdots & 0 \\ z_{1,0} & z_{1,1} & \cdots & z_{1,27} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ z_{27,0} & z_{27,1} & \cdots & z_{27,27} & 0 & \cdots & 0 \\ \hline 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \end{array}\right]$$

and this matrix is stored in 1,024-slot vector form as

$$(z_{0,0}, z_{0,1}, \cdots, z_{0,27}, 0, 0, 0, 0, z_{1,0}, z_{1,1}, \cdots, z_{1,27}, 0, 0, 0, 0, \cdots, z_{27,0}, z_{27,1}, \cdots, z_{27,27}, 0, 0, \cdots, 0).$$

4.4 Encrypted Convolutional Layer

There is no padding for convolution in Lenet1 Model. So the convolution kernel operates to the matrix as

$$\begin{bmatrix} z_{0,0} & z_{0,1} & z_{0,2} & \cdots & z_{0,27} \\ z_{1,0} & z_{1,1} & z_{1,2} & \cdots & z_{1,27} \\ z_{2,0} & z_{2,1} & z_{2,2} & \cdots & z_{2,27} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ z_{27,0} & z_{27,1} & z_{27,2} & \cdots & z_{27,27} \end{bmatrix} \otimes \begin{bmatrix} c_{0,0} & c_{0,1} & \cdots & c_{0,4} \\ c_{1,0} & c_{1,1} & \cdots & c_{1,4} \\ \vdots & \vdots & \ddots & \vdots \\ c_{4,0} & c_{4,1} & \cdots & c_{4,4} \end{bmatrix}$$

$$= \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & \cdots & w_{0,23} \\ w_{1,0} & w_{1,1} & w_{1,2} & \cdots & w_{1,23} \\ w_{2,0} & w_{2,1} & w_{2,2} & \cdots & w_{2,23} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{23,0} & w_{23,1} & w_{23,2} & \cdots & w_{23,23} \end{bmatrix}$$

where $w_{n,m} = \sum_{0 \le i,j < 5} c_{i,j} \cdot z_{n+i,m+j}$. $(0 \le n, m < 24)$

Also, this can be represented as

$$\begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & \cdots & w_{0,23} \\ w_{1,0} & w_{1,1} & w_{1,2} & \cdots & w_{1,23} \\ w_{2,0} & w_{2,1} & w_{2,2} & \cdots & w_{2,23} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{23,0} & w_{23,1} & w_{23,2} & \cdots & w_{23,23} \end{bmatrix}$$

$$= \sum_{0 \le i,j < 5} c_{i,j} \cdot \begin{bmatrix} z_{i,j} & z_{i,j+1} & \cdots & z_{i,j+23} \\ z_{i+1,j} & z_{i+1,j+1} & \cdots & z_{i+1,j+23} \\ \vdots & \vdots & \ddots & \vdots \\ z_{i+23,j} & z_{i+23,j+1} & \cdots & z_{i+23,j+23} \end{bmatrix}$$

So in ciphertext we evaluate it as

$$\text{Enc}\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,23} & * & \cdots & * \\ w_{1,0} & w_{1,1} & \cdots & w_{1,23} & * & \cdots & * \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ w_{23,0} & w_{23,1} & \cdots & w_{23,23} & * & \cdots & * \\ * & * & \cdots & * & * & \cdots & * \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ * & * & \cdots & * & * & \cdots & * \end{bmatrix}$$

$$= \textstyle\sum_{0\le i,j<5} c_{i,j} \cdot \text{Enc}\begin{bmatrix} z_{i,j} & z_{i,j+1} & \cdots & z_{i,j+23} & * & \cdots & * \\ z_{i+1,j} & z_{i+1,j+1} & \cdots & z_{i+1,j+23} & * & \cdots & * \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ z_{i+23,j} & z_{i+23,j+1} & \cdots & z_{i+23,j+23} & * & \cdots & * \\ * & * & \cdots & * & * & \cdots & * \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ * & * & \cdots & * & * & \cdots & * \end{bmatrix}$$

$$= \sum_{0\le i,j<5} c_{i,j} \cdot \text{Rotate}_{(i\cdot 32+j)}\text{Enc}\begin{bmatrix} z_{0,0} & z_{0,1} & \cdots & z_{0,27} & 0 & \cdots & 0 \\ z_{1,0} & z_{1,1} & \cdots & z_{1,27} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ z_{27,0} & z_{27,1} & \cdots & z_{27,27} & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \end{bmatrix}$$

Here is the pseudocode for this method:

Algorithm 2 Convolution 1

procedure CONVOLUTION$_1$(CT_Z, M, r)

  $ct_R \leftarrow 0$

  for $i = 0$ TO r **do**

    for $j = 0$ TO r **do**

      $ct_{i,j} \leftarrow Rotate(ct_Z, i \cdot 32 + j)$

      $ct_{i,j} \leftarrow Mult(ct_{i,j}, M_{i,j})$

      $ct_R \leftarrow Add(ct_R, ct_{i,j})$

    end for

    return ct_R

where CT_Z is input ciphertext, M is $r \times r$ convolution kernel matrix.
In the second convolution, we have to evaluate the matrix represented as below:

$$\begin{bmatrix} z_{0,0} & 0 & z_{0,2} & 0 & z_{0,4} & \cdots \\ 0 & 0 & 0 & 0 & 0 & \cdots \\ z_{2,0} & 0 & z_{2,2} & 0 & z_{2,4} & \cdots \\ 0 & 0 & 0 & 0 & 0 & \cdots \\ z_{4,0} & 0 & z_{4,2} & 0 & z_{4,4} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

So we decided to rotate by two slots for convolution, rather than changing the matrix to be of the form as below:

$$\begin{bmatrix} z_{0,0} & z_{0,2} & z_{0,4} & \cdots \\ z_{2,0} & z_{2,2} & z_{2,4} & \cdots \\ z_{4,0} & z_{4,2} & z_{4,4} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

Other methods are similar to convolution 1. Here is the pseudocode for this:

Algorithm 3 Convolution 2

procedure CONVOLUTION$_2$(CT_Z, M, r)

$ct_R \leftarrow 0$

for $i = 0$ TO r **do**

for $j = 0$ TO r **do**

$ct_{i,j} \leftarrow Rotate(ct_Z, 2 \cdot (i \cdot 32 + j))$

$ct_{i,j} \leftarrow Mult(ct_{i,j}, M_{i,j})$

$ct_R \leftarrow Add(ct_R, ct_{i,j})$

end for

return ct_R

4.5 Encrypted Average Pooling Layer

The first average pooling layer takes input of size 24x24x4 and output of size 12x12x4. 4 is the number of channels and we do the pooling for each channel independently. Considering the length of the encrypted text is unalterable, for each 2x2 block we average the values and place it in the top left

lattice. (The lattices are marked red.) We get

$$\begin{bmatrix} z_{0,0} & * & z_{0,2} & * & z_{0,4} & \cdots \\ * & * & * & * & * & \cdots \\ z_{2,0} & * & z_{2,2} & * & z_{2,4} & \cdots \\ * & * & * & * & * & \cdots \\ z_{4,0} & * & z_{4,2} & * & z_{4,4} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

Then we change the rest garbage values to zero by multiplying an encrypted constant vector

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 & \cdots \\ 0 & 0 & 0 & 0 & 0 & \cdots \\ 1 & 0 & 1 & 0 & 1 & \cdots \\ 0 & 0 & 0 & 0 & 0 & \cdots \\ 1 & 0 & 1 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

Finally the output would be

$$\begin{bmatrix} z_{0,0} & 0 & z_{0,2} & 0 & z_{0,4} & \cdots \\ 0 & 0 & 0 & 0 & 0 & \cdots \\ z_{2,0} & 0 & z_{2,2} & 0 & z_{2,4} & \cdots \\ 0 & 0 & 0 & 0 & 0 & \cdots \\ z_{4,0} & 0 & z_{4,2} & 0 & z_{4,4} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

The pseudocode for this layer is as follows:

Algorithm 4 Average Pooling 1

procedure POOLING$_1$(CT$_Z$,c)
  for $j = 0$ TO c **do**
    $ct_R \leftarrow Add(ct_Z, Rotate(ct_Z, 1))$
    $ct_R \leftarrow Add(ct_R, Rotate(ct_R, 32))$
    $ct_R \leftarrow Mult(ct_R, 0.25)$
    $ct_{Const} \leftarrow Enc(1, 0, 1, 0, \ldots, 0, 0, 0, 0, \ldots)$
    $ct_R \leftarrow Mult(ct_R, ct_{Const})$
    return ct_R
  end for
end procedure

The second average pooling layer takes input of size 8x8x12 and output of size 4x4x12. Identically the pooling for each channel is done independently. For each 4x4 block we average the top left value

in each 2x2 block within and place it in the top left lattice. For example, $z_{0,0}$, $z_{0,2}$, $z_{2,0}$ and $z_{2,2}$ are averaged to $z_{0,0}$ in the first 4x4 block. (The lattices are marked red.) We get

$$\begin{bmatrix} z_{0,0} & * & * & * & z_{0,4} & * & * & * & z_{0,8} & \cdots \\ * & * & * & * & * & * & * & * & * & \cdots \\ * & * & * & * & * & * & * & * & * & \cdots \\ * & * & * & * & * & * & * & * & * & \cdots \\ z_{4,0} & * & * & * & z_{4,4} & * & * & * & z_{4,8} & \cdots \\ * & * & * & * & * & * & * & * & * & \cdots \\ * & * & * & * & * & * & * & * & * & \cdots \\ * & * & * & * & * & * & * & * & * & \cdots \\ z_{8,0} & * & * & * & z_{8,4} & * & * & * & z_{8,8} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

Same as the first pooling layer, we change the rest garbage values to zero by multiplying an encrypted constant vector

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

Finally the output would be

$$\begin{bmatrix} z_{0,0} & 0 & 0 & 0 & z_{0,4} & 0 & 0 & 0 & z_{0,8} & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ z_{4,0} & 0 & 0 & 0 & z_{4,4} & 0 & 0 & 0 & z_{4,8} & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ z_{8,0} & 0 & 0 & 0 & z_{8,4} & 0 & 0 & 0 & z_{8,8} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

The pseudocode for this layer is as follows:

Algorithm 5 Average Pooling 2

procedure POOLING$_2$(CT$_Z$,c)

  for $j = 0$ TO c **do**

    $ct_R \leftarrow Add(ct_Z, Rotate(ct_Z, 2 \cdot 1))$

    $ct_R \leftarrow Add(ct_R, Rotate(ct_R, 2 \cdot 32))$

    $ct_R \leftarrow Mult(ct_Z, 0.25)$

    $ct_{Const} \leftarrow Enc(1, 0, 0, 0, 1, 0, 0, 0 \ldots, 0, 0, 0, 0, 0, 0, 0, 0 \ldots)$

    $ct_R \leftarrow Mult(ct_R, ct_{Const})$

    return ct_R

  end for

end procedure

4.6 Encrypted Fully Connected Layer

The fully connected layer is the last layer of Lenet-1, which consists 1920 (12x4x4x10) trainable parameters. It takes 12 channels of units of size 4x4 from the second pooling layer as input. The weighted sums computed from the dot product between the input vector and their weight vector is added up with biases before being passed through an activation function. In this way, ten probabilities correspondent to each value are calculated. In our implementation of encrypted Lenet-1 model, the output from the second pooling layer is 12 ciphertexts which encrypts a vector of 32x32 including garbage values. The 16 values needed is scattered as the following.

$$\begin{bmatrix}
z_{0,0} & * & * & * & z_{0,4} & * & * & * & z_{0,8} & * & * & * & z_{0,12} & * & \cdots & * \\
* & * & * & * & * & * & * & * & * & * & * & * & * & * & \cdots & * \\
* & * & * & * & * & * & * & * & * & * & * & * & * & * & \cdots & * \\
* & * & * & * & * & * & * & * & * & * & * & * & * & * & \cdots & * \\
z_{4,0} & * & * & * & z_{4,4} & * & * & * & z_{4,8} & * & * & * & z_{4,12} & * & \cdots & * \\
* & * & * & * & * & * & * & * & * & * & * & * & * & * & \cdots & * \\
* & * & * & * & * & * & * & * & * & * & * & * & * & * & \cdots & * \\
* & * & * & * & * & * & * & * & * & * & * & * & * & * & \cdots & * \\
z_{8,0} & * & * & * & z_{8,4} & * & * & * & z_{8,8} & * & * & * & z_{8,12} & * & \cdots & * \\
* & * & * & * & * & * & * & * & * & * & * & * & * & * & \cdots & * \\
* & * & * & * & * & * & * & * & * & * & * & * & * & * & \cdots & * \\
* & * & * & * & * & * & * & * & * & * & * & * & * & * & \cdots & * \\
z_{12,0} & * & * & * & z_{12,4} & * & * & * & z_{12,8} & * & * & * & z_{12,12} & * & \cdots & * \\
* & * & * & * & * & * & * & * & * & * & * & * & * & * & \cdots & * \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \\
* & * & * & * & * & * & * & * & * & * & * & * & * & * & \cdots & *
\end{bmatrix}.$$

Due to the existence of these garbage values, the weights need to be placed with caution into locations corresponding to the 16 slots among the total 1024 slots. Then in each of the ten iterations, 12 weight

vectors are encrypted and multiplied with the 12 ciphertext inputs from the last layer, which is then summed up into one ciphertext. In each of the 10 ciphertexts calculated from each iterations, all the meaningful value is added up into one slot by minimum number of rotation and adding, which will be used for later decryption and activation.

5 Conclusion

5.1 Summary

In this report, we demonstrated in detail how we implemented the two important algorithms, namely, logistic regression and convolutional neural network, using the HEAAN library. We mainly described where the obstacles exist in designing the algorithm for ciphertext and the methods we used to address them. Although we only implemented two elementary classifiers and models, the results of our projects indicate the possibility of accomplishing more complicated tasks with the HEAAN library.

5.2 Limitations and Possible Future Directions

Currently, it is expensive for HEAAN library to perform more complicated operations other than addition, multiplication and rotation, making it difficult to approximate non-linear functions including activation functions and differentiation, which is the reason why the learning phase of neural network is hard to implement. In view of this, methods for reducing the cost while preserving the accuracy of encrypted calculations are in great need.
On the other hand, since most variations of CNN are composed of convolutional layers, pooling layers and fully connected layers as in the case of LeNet-1, with the assistance of bootstrapping, we can further implement other types of CNN. Moreover, if proper approximation is applied and the cost is minimized, it is possible to develop secured solutions to other classifiers and machine learning models.

5.3 Acknowledgements

Our team would like to acknowledge Professor Jung Hee Cheon and CryptoLab for providing this project, as well as our mentors Andrey Kim and Jaehyun Park who offerred valuable guidance and enormous help without which we couldn't have accomplished this task. We would also like to thank Professor Otto van Koert for his helpful discussion. Last but not least, we want to express our gratitude to the math departments of SNU and HKUST, especially Professor Junho Lee and Professor Albert Ku for organizing this program successfully and providing us with this incredible opportunity to research and explore in SNU.

6 Bibliography

[1] A. Kim, Y. Song, M. Kim, K. Lee, and J. H. Cheon, *Logistic Regression Model Training Based on the Approximate Homomorphic Encryption*, vol. 11, no. Suppl 4, p. 83, Oct. 2018.
[2] J. H. Cheon and Y. S. Song, *Homomorphic Encryption Method of a Plurality of Messages Supporting Approximate Arithmetic of Complex Numbers*, Feb. 2018.
[3] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based Learning Applied to Document Recognition*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[4] A. Smola and M. Li, *Dive into Deep Learning*, 2019.
[5] S.H. Tsang, "Review: LeNet-1, LeNet-4, LeNet-5, Boosted LeNet-4 (Image Classification)," *Medium*, August, 2018. [Online], Available: https://medium.com/@sh.tsang/paper-brief-review-of-lenet-1-lenet-4-lenet-5-boosted-lenet-4-image-classification-1f5f809dbf17. [Accessed Aug.11, 2019].