Homomorphic Encryption Based Secure Solutions to Logistic Regression and CNN

SPIA 2019 Group 4

August 12, 2019

Objective

- Classifiers —— given sets of independent variables (predictors Xi) to predict the dependent variable Y.
- Classifiers based on Homomorphic Encryption —— publicly operate on *ciphertexts* without decryption to ensure end-to-end semantically secure.
- Machine Learning Models concerning encrypted data

Homomorphic Encryption

What is Homomorphic Encryption?

A cryptosystem is *homomorphic* iff it decryption is a morphism

$$\texttt{Enc}(\mathbf{m}_1) \circledast \texttt{Enc}(\mathbf{m}_2) = \texttt{Enc}(\mathbf{m}_1 * \mathbf{m}_2)$$

More generally we want

$$\texttt{Eval}(f(x_1, \ldots, x_s)), \texttt{Enc}(\mathbf{m}_1), \ldots, \texttt{Enc}(\mathbf{m}_s)) = \texttt{Enc}(f(\mathbf{m}_1, \ldots, \mathbf{m}_s)$$

Normally in practice we consider operations $(+, \times)$

Homomorphic Encryption

Let $\mathbf{m}_1$ and $\mathbf{m}_2$ - messages from some message space $\mathcal{M}$ (numbers, polynomials, ring elements etc.)

$$\mathbf{m}_1 \xrightarrow{\texttt{Enc}} \mathsf{ct}_1$$

$$\xrightarrow{\circledast} \mathsf{ct}_* \xrightarrow{\texttt{Dec}} \mathbf{m}_1 * \mathbf{m}_2$$

$$\mathbf{m}_2 \xrightarrow{\texttt{Enc}} \mathsf{ct}_2$$

HEAAN Library

What is HEAAN?

HEAAN (Homomorphic Encryption for Arithmetics of Approximate Numbers) is one of the libraries that implements homomorphic encryption scheme which supports four approximate operations —— addition, multiplication, rotation and conjunction, as well as a rescaling procedure called bootstrapping to manage the accuracy magnitude of plaintexts.

HEAAN Library

The main idea is to treat an encryption noise as part of error occurring during approximate computation.

$$\mathrm{m} \xrightarrow{\texttt{Enc}} \mathrm{ct} \xrightarrow{\texttt{Dec}} \langle ct, sk \rangle = m + e \pmod{q}$$

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. e is made small enough so that the significant figures of m remain intact as $\tilde{m} = m + e$. This can further be guaranteed by scaling the message before encryption.

LWE Problem

- $\vec{s} \xleftarrow{\$} \mathbb{Z}_q^n$
- LWE instance: $(\vec{a}, \langle \vec{a}, \vec{s} \rangle + e)$, where $\vec{a} \xleftarrow{\$} \mathbb{Z}_q^n$ and $e \xleftarrow{\phi} \mathbb{Z}_q$
- LWE problem: distinguish LWE instance $(\vec{a}, \langle \vec{a}, \vec{s} \rangle + e)$ from pure random $(\vec{a}, b) \xleftarrow{\$} \mathbb{Z}_q^{n+1}$ without $\vec{s}$ knowledge

How to distinguish with $\vec{s}$ knowledge?

- We have $(\vec{a}, b)$ Consider $\langle (\vec{a}, b), (-\vec{s}, 1) \rangle = b - \langle \vec{a}, \vec{s} \rangle$.
- If $(\vec{a}, b) \in$ LWE then $b - \langle \vec{a}, \vec{s} \rangle = e$ is small
- If $(\vec{a}, b) \xleftarrow{\$} \mathbb{Z}_q^{n+1}$ then $b - \langle \vec{a}, \vec{s} \rangle$ is uniform over $\mathbb{Z}_q$

We normally define a secret key sk as sk $= (-\vec{s}, 1)$

RLWE Problem

- $\mathcal{R} = \mathbb{Z}[x]/\Phi_M(x)$, $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$
- $\mathbf{s} \xleftarrow{\$} \mathcal{R}_q$
- RLWE instance: $(\mathbf{a}, \mathbf{a}\cdot\mathbf{s} + \mathbf{e})$, where $\mathbf{a} \xleftarrow{\$} \mathcal{R}_q$ and $\mathbf{e} \xleftarrow{\phi} \mathcal{R}_q$
- RLWE problem: distinguish RLWE instance $(\mathbf{a}, \mathbf{a}\cdot\mathbf{s} + \mathbf{e})$ from pure random $(\mathbf{a}, \mathbf{b}) \xleftarrow{\$} \mathcal{R}_q^2$ without $\mathbf{s}$ knowledge

How to distinguish with $\mathbf{s}$ knowledge?

- We have $(\mathbf{a}, \mathbf{b})$. Consider $\langle(\mathbf{a}, \mathbf{b}), (-\mathbf{s}, 1)\rangle = \mathbf{b} - \mathbf{a}\cdot\mathbf{s}$.
- If $(\mathbf{a}, \mathbf{b}) \in$ RLWE then $\mathbf{b} - \mathbf{a}\cdot\mathbf{s} = \mathbf{e}$ is small
- If $(\mathbf{a}, \mathbf{b}) \xleftarrow{\$} \mathcal{R}_q^2$ then $\mathbf{b} - \mathbf{a}\cdot\mathbf{s}$ is uniform over $\mathcal{R}_q$

We normally define a secret key sk as sk $= (-\mathbf{s}, 1)$

RLWE Problem

More precisely, the plaintext space of the HEAAN scheme is $\mathbb{C}^{n/2}$ for some power-of-two integer n. To deal with the complex plaintext vector efficiently, Cheon et al. proposed plaintext encoding/decoding methods which exploits a ring isomorphism $\Phi = \mathbb{R}[X]/(X^n + 1) \longrightarrow \mathbb{C}^{n/2}$.

Packing Method

We consider the data set as a matrix of $\mathbb{R}^{n \times f}$ 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.

$$Z = \left[\begin{array}{cccc|ccc} z_{0,0} & z_{0,1} & \cdots & z_{0,f-1} & 0 & \cdots & 0 \\ z_{1,0} & z_{1,1} & \cdots & z_{1,f-1} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ z_{n-1,0} & z_{n-1,1} & \cdots & z_{n-1,f-1} & 0 & \cdots & 0 \\ \hline 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \end{array}\right]$$

HEAAN Style

- $\mathbf{s} \xleftarrow{\$} \mathcal{R}_q$, $\mathsf{sk} = (-\mathbf{s}, 1)$, $\mathbf{m} \in \mathcal{R}$, $\mathsf{ct} \in \mathcal{R}_q^2$
- $\mathtt{Enc}(\mathbf{m}) = \mathsf{ct}$ such that $\langle \mathsf{ct}, \mathsf{sk} \rangle = \mathbf{m} + \mathbf{e}$
- Sample $(\mathbf{a}, \mathbf{b}) \longleftarrow$ RLWE, then $\mathsf{ct} = (\mathbf{a}, \mathbf{b} + \mathbf{m})$

If $\mathbf{b} = \mathbf{a} \cdot \mathbf{s} + \mathbf{e}$,

$$\mathtt{Dec}(\mathsf{ct}) = \lfloor \langle \mathsf{ct}, \mathsf{sk} \rangle \rceil_q = \lfloor \mathbf{a} \cdot \mathbf{s} + \mathbf{e} + \mathbf{m} - \mathbf{a} \cdot \mathbf{s} \rceil_q =$$
$$= \lfloor \mathbf{e} + \mathbf{m} \rceil_q = \mathbf{m} + \mathbf{e}$$

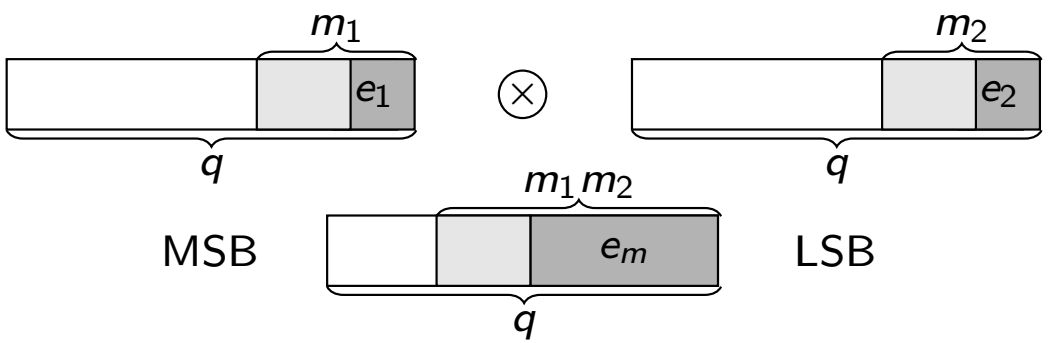


Figure 1: Homomorphic multiplication of ciphertexts for approximate arithmetic

Logistic Regression

What is Logistic Regression

Logistic Regression is a widely used machine learning classifier used to assign observations to a number of discrete classes.

In our project, the learning data consists of pairs (x_i, y_i) where $x_i = (x_{i1}, \ldots, x_{if}) \in \mathcal{R}^f$ and a dependent variable $y_i \in \pm 1$. Logistic regression aims to find an optimal $\beta \in \mathcal{R}^{f+1}$ which minimize the loss function

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

Approximate Sigmoid Function

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.

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

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.

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

Homomorphic Evaluation of the Gradient Descent

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

$$\mathrm{ct}_\beta^{(0)} = \mathtt{Enc}\begin{bmatrix} \beta_0^{(0)} & \beta_1^{(0)} & \cdots & \beta_f^{(0)} \\ \beta_0^{(0)} & \beta_1^{(0)} & \cdots & \beta_f^{(0)} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_0^{(0)} & \beta_1^{(0)} & \cdots & \beta_f^{(0)} \end{bmatrix}.$$

The goal of each iteration is to update the modeling vector $\boldsymbol{\beta}^{(t)}$ using the gradient of loss function:

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

Homomorphic Evaluation of the Gradient Descent

Step 1: For given two ciphertexts ct_z and $\mathrm{ct}_\beta^{(t)}$, compute their multiplication and rescale it by p bits:

$$\mathrm{ct}_1 \leftarrow (\mathtt{Mult}(\mathrm{ct}_\beta^{(t)}, \mathrm{ct}_z); p).$$

The output ciphertext contains the values $z_{ij} \cdot \beta_j^{(t)}$ in its plaintext slots,

$$\mathrm{ct}_1 = \mathtt{Enc} \begin{bmatrix} z_{10} \cdot \beta_0^{(t)} & z_{11} \cdot \beta_1^{(t)} & \cdots & z_{1f} \cdot \beta_f^{(t)} \\ z_{20} \cdot \beta_0^{(t)} & z_{21} \cdot \beta_1^{(t)} & \cdots & z_{1f} \cdot \beta_f^{(t)} \\ \vdots & \vdots & \ddots & \vdots \\ z_{n0} \cdot \beta_0^{(t)} & z_{n1} \cdot \beta_1^{(t)} & \cdots & z_{nf} \cdot \beta_f^{(t)} \end{bmatrix}.$$

Homomorphic Evaluation of the Gradient Descent

Step 2: To obtain the inner product $\mathbf{z}_i^T \boldsymbol{\beta}^{(t)}$, the public cloud aggregates the values of $z_{ij}\beta_j^{(t)}$ in the same row. This step can be done by adding ct_1 to its rotations recursively:

$$\mathrm{ct}_1 \leftarrow \mathtt{Add}(\mathrm{ct}_1, (\mathrm{ct}_1; 2^j)),$$

for $j = 0, 1, \ldots, \log(f+1) - 1$. Then the output ciphertext ct_2 encrypts the inner product values $\mathbf{z}_i^T \boldsymbol{\beta}^{(t)}$ in the first column and some "garbage" values in the other columns, denoted by $\star$,

$$\mathrm{ct}_2 = \mathtt{Enc}\begin{bmatrix} \mathbf{z}_1^T\boldsymbol{\beta}^{(t)} & \star & \cdots & \star \\ \mathbf{z}_2^T\boldsymbol{\beta}^{(t)} & \star & \cdots & \star \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{z}_n^T\boldsymbol{\beta}^{(t)} & \star & \cdots & \star \end{bmatrix}.$$

Homomorphic Evaluation of the Gradient Descent

Step 3: This step performs a constant multiplication in order to annihilate the garbage values. It can be obtained by computing the encoding polynomial $c \leftarrow (C; p_c)$ of the matrix

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

using the scaling factor of 2^{p_c} for some integer p_c. The parameter p_c is chosen as the bit precision of plaintexts so it can be smaller than the parameter p.

Homomorphic Evaluation of the Gradient Descent

Step 3: Finally we multiply the polynomial c to the ciphertext ct_2 and rescale it by p_c bits:

$$\mathsf{ct}_3 \leftarrow ((\mathsf{ct}_2; c); p_c).$$

The garbage values are multiplied with zero while one can maintain the inner products in the plaintext slots. Hence the output ciphertext ct_3 encrypts the inner product values in the first column and zeros in the others:

$$\mathsf{ct}_3 = \mathrm{Enc}\begin{bmatrix} \mathbf{z}_1^T\beta^{(t)} & 0 & \cdots & 0 \\ \mathbf{z}_2^T\beta^{(t)} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{z}_n^T\beta^{(t)} & 0 & \cdots & 0 \end{bmatrix}.$$

Homomorphic Evaluation of the Gradient Descent

Step 4: The goal of this step is to replicate the inner product values to other columns. Similar to Step 2, it can be done by adding the input ciphertext to its column shifting recursively, but in the opposite direction

$$\mathsf{ct}_3 \leftarrow \mathtt{Add}(\mathsf{ct}_3, (\mathsf{ct}_3; -2^j))$$

for $j = 0, 1, \ldots, \log(f+1) - 1$. The output ciphertext ct_4 has the same inner product value in each row:

$$\mathsf{ct}_4 = \mathtt{Enc}\begin{bmatrix} \mathbf{z}_1^T\beta^{(t)} & \mathbf{z}_1^T\beta^{(t)} & \cdots & \mathbf{z}_1^T\beta^{(t)} \\ \mathbf{z}_2^T\beta^{(t)} & \mathbf{z}_2^T\beta^{(t)} & \cdots & \mathbf{z}_2^T\beta^{(t)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{z}_n^T\beta^{(t)} & \mathbf{z}_n^T\beta^{(t)} & \cdots & \mathbf{z}_n^T\beta^{(t)} \end{bmatrix}.$$

Homomorphic Evaluation of the Gradient Descent

Step 5: This step simply evaluates an approximating polynomial of the sigmoid function, $\mathsf{ct}_5 \leftarrow g(\mathsf{ct}_4)$ for some $g \in \{g_3, g_5, g_7\}$. The output ciphertext encrypts the values of $g(\mathbf{z}_i^T \boldsymbol{\beta}^{(t)})$ in its plaintext slots:

$$\mathsf{ct}_5 = \mathtt{Enc} \begin{bmatrix} g(\mathbf{z}_1^T \boldsymbol{\beta}^{(t)}) & \cdots & g(\mathbf{z}_1^T \boldsymbol{\beta}^{(t)}) \\ g(\mathbf{z}_2^T \boldsymbol{\beta}^{(t)}) & \cdots & g(\mathbf{z}_2^T \boldsymbol{\beta}^{(t)}) \\ \vdots & \ddots & \vdots \\ g(\mathbf{z}_n^T \boldsymbol{\beta}^{(t)}) & \cdots & g(\mathbf{z}_n^T \boldsymbol{\beta}^{(t)}) \end{bmatrix}.$$

Homomorphic Evaluation of the Gradient Descent

Step 6: The public cloud multiplies the ciphertext ct_5 with the encrypted dataset ct_z and rescales the resulting ciphertext by p bits:

$$\mathsf{ct}_6 \leftarrow (\mathtt{Mult}(\mathsf{ct}_5, \mathsf{ct}_z); p).$$

The output ciphertext encrypts the n vectors $g(\mathbf{z}_i^T \boldsymbol{\beta}^{(t)}) \cdot \mathbf{z}_i$ in each row:

$$\mathsf{ct}_6 = \mathtt{Enc} \begin{bmatrix} g(\mathbf{z}_1^T \boldsymbol{\beta}^{(t)}) \cdot z_{10} & \cdots & g(\mathbf{z}_1^T \boldsymbol{\beta}^{(t)}) \cdot z_{1f} \\ g(\mathbf{z}_2^T \boldsymbol{\beta}^{(t)}) \cdot z_{20} & \cdots & g(\mathbf{z}_2^T \boldsymbol{\beta}^{(t)}) \cdot z_{2f} \\ \vdots & \ddots & \vdots \\ g(\mathbf{z}_n^T \boldsymbol{\beta}^{(t)}) \cdot z_{n0} & \cdots & g(\mathbf{z}_n^T \boldsymbol{\beta}^{(t)}) \cdot z_{nf} \end{bmatrix}.$$

Homomorphic Evaluation of the Gradient Descent

Step 7: This step aggregates the vectors $g(\mathbf{z}_i^T \boldsymbol{\beta}^{(t)})$ to compute the gradient of the loss function. It is obtained by recursively adding ct_6 to its row shifting:

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

for $j = \log(f+1), \ldots, \log(f+1) + \log n - 1$. The output ciphertext is

$$\mathrm{ct}_7 = \mathtt{Enc} \begin{bmatrix} \sum_i g(\mathbf{z}_i^T \boldsymbol{\beta}^{(t)}) \cdot z_{i0} & \cdots & \sum_i g(\mathbf{z}_i^T \boldsymbol{\beta}^{(t)}) \cdot z_{if} \\ \sum_i g(\mathbf{z}_i^T \boldsymbol{\beta}^{(t)}) \cdot z_{i0} & \cdots & \sum_i g(\mathbf{z}_i^T \boldsymbol{\beta}^{(t)}) \cdot z_{if} \\ \vdots & \ddots & \vdots \\ \sum_i g(\mathbf{z}_i^T \boldsymbol{\beta}^{(t)}) \cdot z_{i0} & \cdots & \sum_i g(\mathbf{z}_i^T \boldsymbol{\beta}^{(t)}) \cdot z_{if} \end{bmatrix},$$

as desired.

Homomorphic Evaluation of the Gradient Descent

Step 8: For the learning rate α_t, it uses the parameter p_c to compute the scaled learning rate $\Delta^{(t)} = \lfloor 2^{p_c} \cdot \alpha_t \rceil$. The public cloud updates $\boldsymbol{\beta}^{(t)}$ using the ciphertext ct_7 and the constant $\Delta^{(t)}$:

$$\mathsf{ct}_8 \leftarrow (\Delta^{(t)} \cdot \mathsf{ct}_7; p_c),$$
$$\mathsf{ct}_\beta^{(t+1)} \leftarrow \mathtt{Add}(\mathsf{ct}_\beta^{(t)}, \mathsf{ct}_8).$$

In this way, it becomes

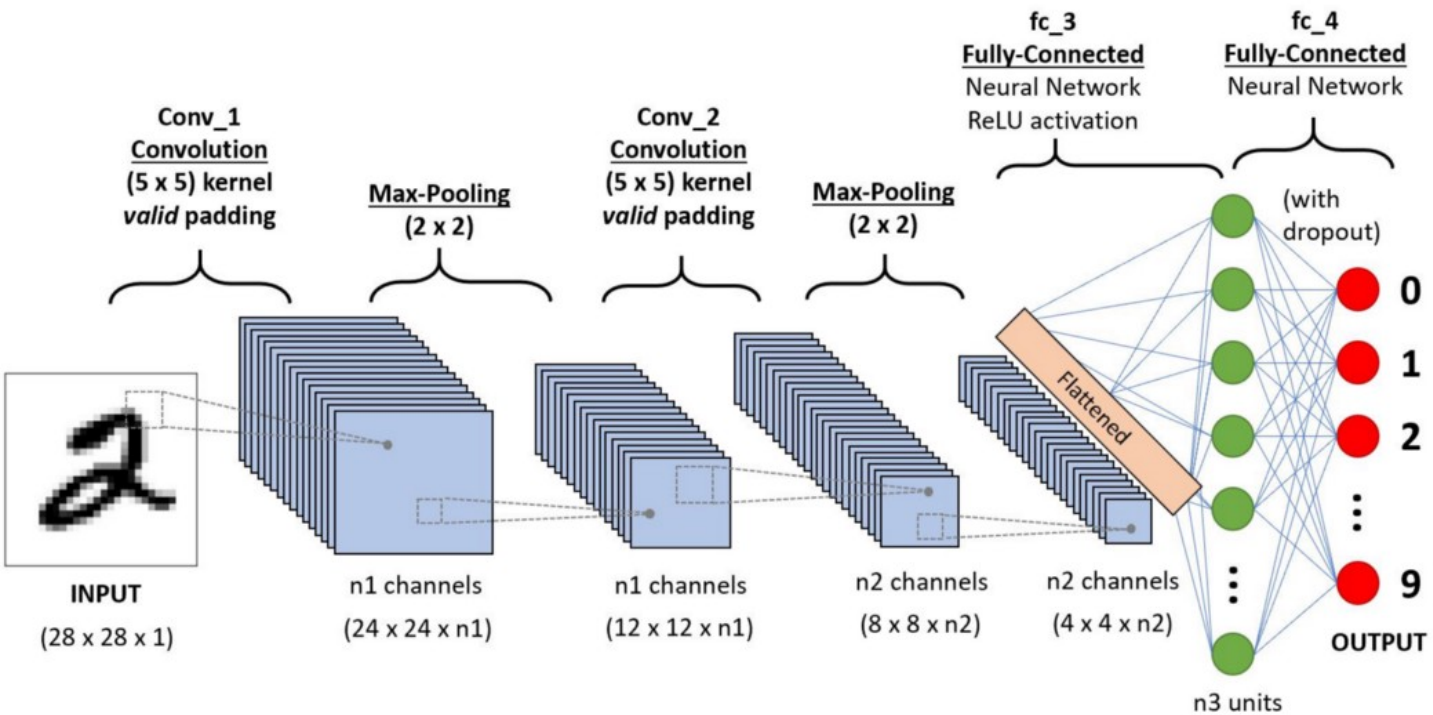
$$\mathsf{ct}_\beta^{(t+1)} = \mathtt{Enc}\begin{bmatrix} \beta_0^{(t+1)} & \beta_1^{(t+1)} & \cdots & \beta_f^{(t+1)} \\ \beta_0^{(t+1)} & \beta_1^{(t+1)} & \cdots & \beta_f^{(t+1)} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_0^{(t+1)} & \beta_1^{(t+1)} & \cdots & \beta_f^{(t+1)} \end{bmatrix}.$$

where $\beta_j^{(t+1)} = \beta_j^{(t)} + \frac{\alpha_t}{n} \sum_i g(\mathbf{z}_i^T \boldsymbol{\beta}^{(t)}) \cdot z_{ij}$.

Convolutional Neural Network

What is CNN

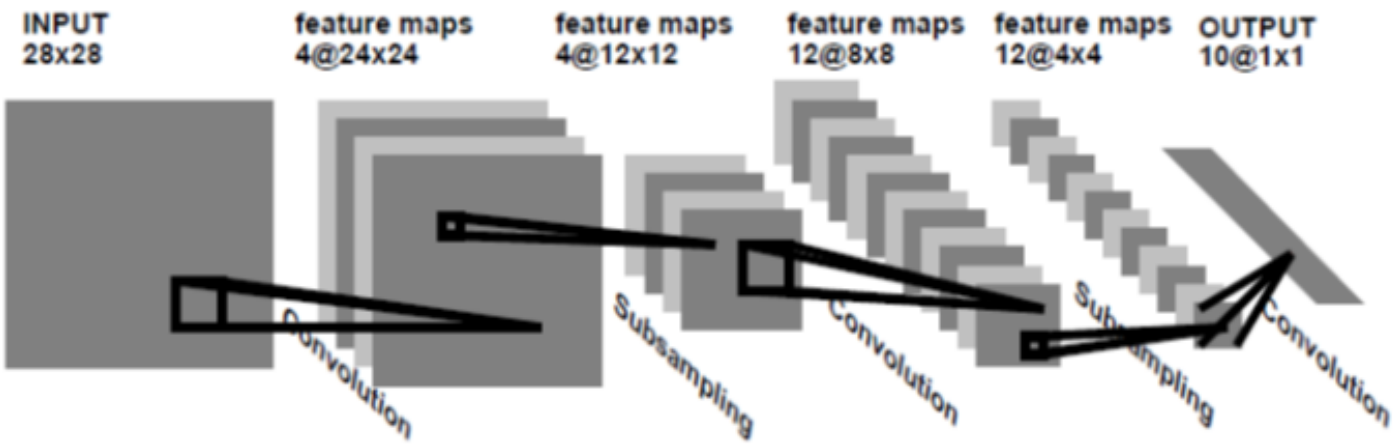
Convolutional neural network (CNN) which is a widely applied multi-layer neural network to analyze visual imagery has been developing rapidly in recent years.



LeNet-1

What is LeNet-1

LeNet-1 is one of the early models of CNN which is designed to learn pixel images of handwritten characters. Taking pixel images of size 28× 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.



LeNet-1 1st Covolutional 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 $w_{n,m} = \sum_{0 \leqslant i,j < 5} c_{i,j} \cdot z_{n+i,m+j}$. $(0 \leqslant n, m < 24)$

LeNet-1 1st Convolutional Layer

Also, this can be represented as

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

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

LeNet-1 1st Convolutional Layer

So in ciphertext we evaluate it as

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

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

LeNet-1 1st Convolutional Layer

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

LeNet-1 2nd Convolutional Layer

In the second convolution, we have to evaluate the matrix represented as below:

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

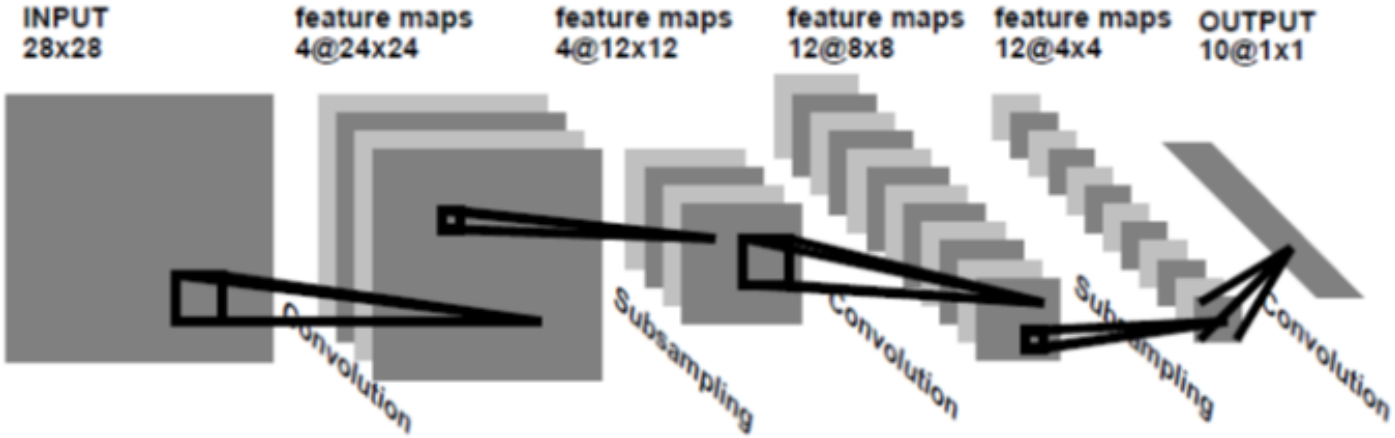
LeNet-1 2nd Convolutional Layer

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.

LeNet-1 1st Average 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.
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.

LeNet-1 1st Average Pooling Layer

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

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

LeNet-1 2nd Average Pooling Layer

The second average pooling layer takes input of size 8x8x12 and output of size 4x4x12. 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. We get

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

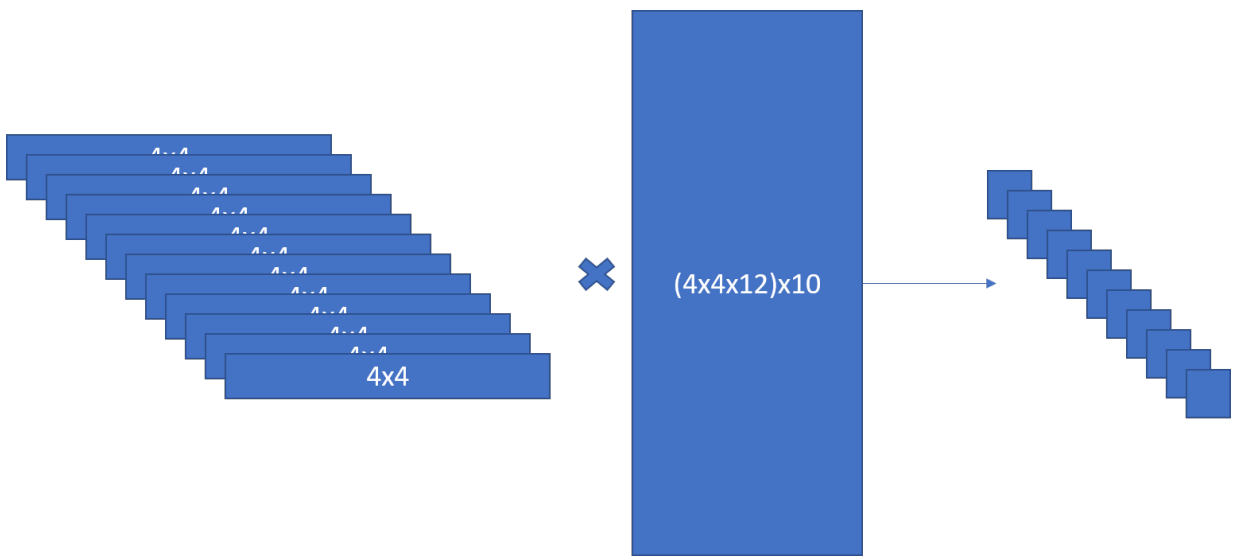
LeNet-1 2nd Average Pooling Layer

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

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

LeNet-1 Fully Connected Layer

The fully connected layer is the last layer of Lenet-1 before the softmax function, which consists 1920 $(12 \times 4 \times 4 \times 10)$ trainable parameters. It takes 12 channels of units of size 4×4 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. In this way, ten probabilities correspondent to each value are calculated.



LeNet-1 Fully Connected Layer

The output from the second pooling layer is 12 ciphertexts. Each encrypts a vector of 32×32 including the garbage values. The 16 values needed is scattered as follows:

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

The weights should be placed carefully according to the position of the above meaningful values.

LeNet-1 Softmax Function

In order to turn the ten values into the probability of occurrence, they need to be passed through an activation function, which is usually a softmax function:

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

where n is the number of labels, z_i is the value of output corresponding to label i.
However, this function contains a lot of exponent arithmetic, which is difficult to calculate in HEAAN, unlike the example of logistic regression. The value of exponential function vary drastically and as a result, we have to approximate it in a wide range. This implies calculation in a much higher degree with lower efficiency.

LeNet-1 Softmax Function

Therefore, we do not evaluate this function and simply return the input value of softmax layer. If the client receives these values, he or she can decrypt it and use normal softmax function instead to calculate the probability before picking the largest value as the final result, which should be the same is the ideal output of the encrypted softmax function.

$$\mathsf{ct} \xrightarrow{softmax} \mathsf{ct}_P \xrightarrow{\mathtt{Dec}} P$$

$$\mathsf{ct} \xrightarrow{\mathtt{Dec}} \mathbf{m} \xrightarrow{softmax} P'$$

$$P \approx P'$$

Summary

- 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.

Limitations and Future Directions

- Currently, HEAAN library can only perform addition, multiplication, conjugation and rotation.
- Other more complicated operations are expensive to calculate or approximate, especially for non-linear functions.
- Therefore, it is costly to calculate some important functions and operations such as trigonometric functions, sigmoid functions and differentiation.
- This makes many basic machine learning algorithms difficult to be implemented, including back propagation and activation functions.
- Improvements are in great need!

Limitations and Future Directions

- Most variations of CNN are composed of convolutional layers, pooling layers and fully connected layers as in the case of LeNet-1.
- On the basis of LeNet-1, we can implement other types of CNN with the help of bootstrapping.
- If proper approximation is applied and the cost is minimized, it is possible to develop secured solutions to other classifiers and machine learning models.
- The future is still bright!

Q & A

Thank You!

Bibliography

[1] J. H. Cheon and Y. S. Song, *Homomorphic Encryption Method of a Plurality of Messages Supporting Approximate Arithmetic of Complex Numbers*, Feb. 2018.

[2] 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.

[3] A. Smola and M. Li, *Dive into Deep Learning*, 2019.

[4] 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.

[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].

[6] S. Sasha, "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way", *Towards Data Science*, Dec. 2018. [Online], Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Accessed Aug.11, 2019]