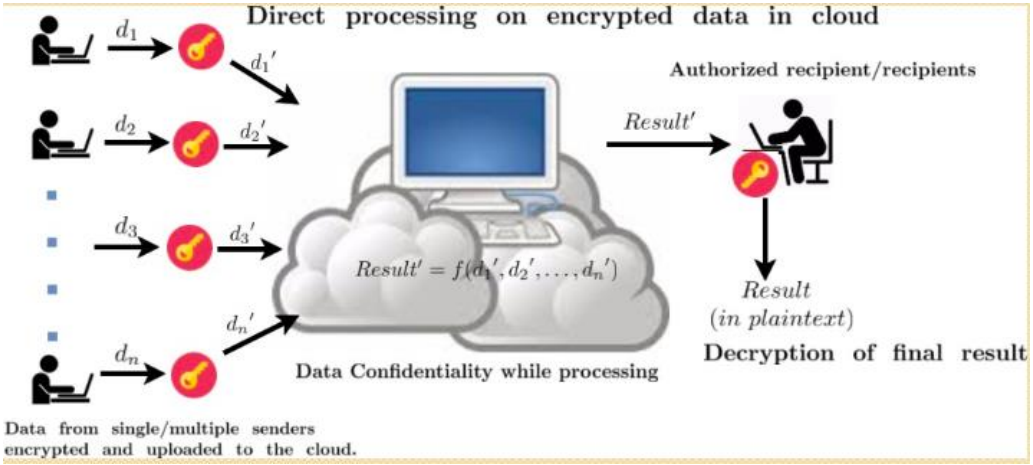
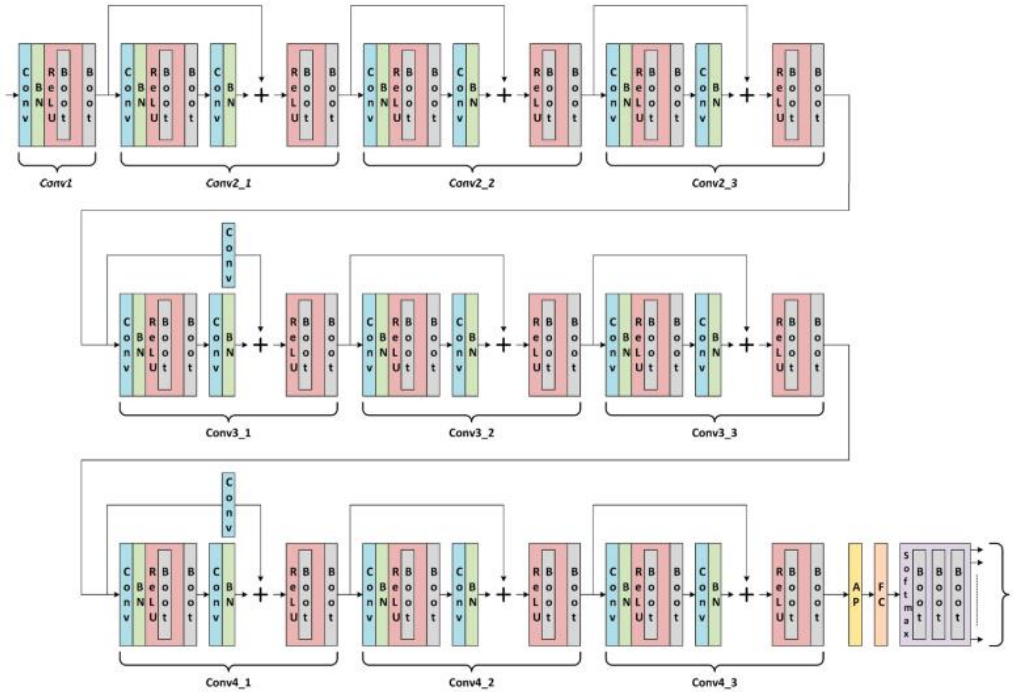
Privacy-Preserving Machine Learning with Fully Homomorphic Encryption

The privacy-preserving issue is one of the most practical problems for machine learning recently.

Fully homomorphic encryption (FHE) is one appropriate tool for privacy-preserving machine learning (PPML) to ensure strong security in the cryptographic sense.



Resnet 20: How to compute in ciphertext domain?

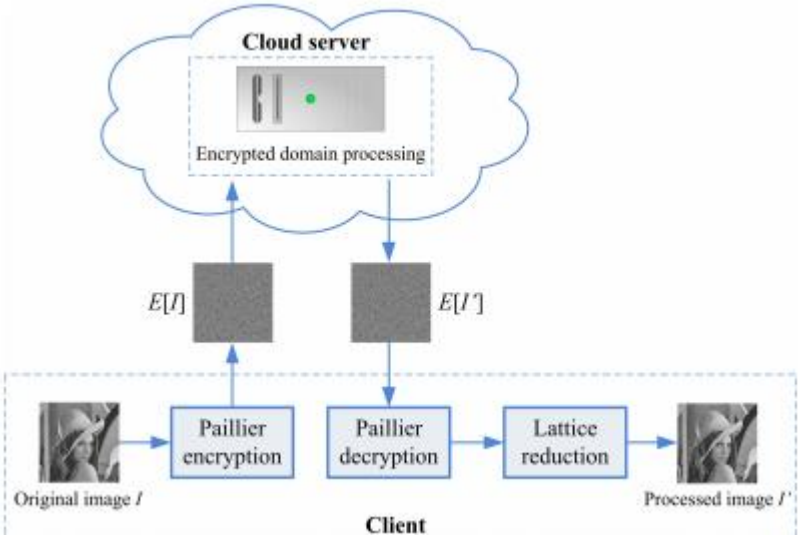


What is homomorohism?

- A group is an algebraic object consisting of a set together with a single binary operation, satisfying certain axioms.
- If G and H are groups, a homomorphism from G to H is a function $f : G \to H$ such that $f(g_1 * g_2) = f(g_1) * f(g_2)$ for any elements $g_1,\ g_2 \mathrm{in} G$, where $*$ denotes the operation in G and $*$ denotes the operation in H.

Homomorphism and Encryption

- Example in **Unpadded RSA (1977)** :
  If the RSA public key is modulus m and exponent e, then the encryption of a message x is given by $\mathcal{E}(x) = x^e \bmod m$. The homomorphic property is then $\mathcal{E}(x_1) \cdot \mathcal{E}(x_2) = x_1^e x_2^e \bmod m = (x_1 x_2)^e \bmod m = \mathcal{E}(x_1 \cdot x_2)$.
- **Hence, concept of homomorphism in encryption is not new**.



Paillier Encryption

Key Generation: KeyGen(p, q)

Input: $p, q \in \mathbb{P}$

Compute $n = pq$

Choose $g \in \mathbb{Z}^*_{n^2}$ such that

$$\gcd(L(g^\lambda \bmod n^2), n) = 1 \text{ with } L(u) = \frac{u-1}{n}$$

Output: (pk, sk)

public key: $pk = (n, g)$

secret key: $sk = (p, q)$

Encryption: Enc(m, pk)

Input: $m \in \mathbb{Z}_n$

Choose $r \in \mathbb{Z}^*_n$

Compute $c = g^m \cdot r^n \bmod n^2$

Output: $c \in \mathbb{Z}_{n^2}$

Decryption: Dec(c, sk)

Input: $c \in \mathbb{Z}_{n^2}$

Compute

$$m = \frac{L(c^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)} \bmod n$$

- **Homomorphic addition of plaintexts**

  The product of two ciphertexts will decrypt to the sum of their corresponding plaintexts,

  $$D(E(m_1, r_1) \cdot E(m_2, r_2) \bmod n^2) = m_1 + m_2 \bmod n.$$

  The product of a ciphertext with a plaintext raising g will decrypt to the sum of the corresponding plaintexts,

  $$D(E(m_1, r_1) \cdot g^{m_2} \bmod n^2) = m_1 + m_2 \bmod n.$$

Application in Federated Learning*

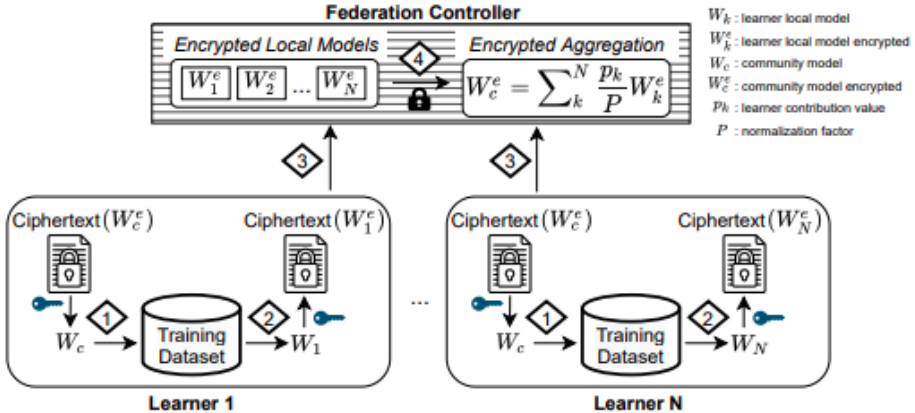


Figure Federated System Architecture with Encryption

***Secure Neuroimaging Analysis using Federated Learning with Homomorphic Encryption, SIPAIM 2021**

Concept of Fully Homomorphic Encryption

- Shared secret key: odd number p
- To encrypt a bit m:
  - Choose at random large q, small r
  - Output $c = pq + 2r + m$
    (Ciphertext is close to a multiple of p
    m = LSB of distance to nearest multiple of p)
- To decrypt : $m = (c \bmod p) \bmod 2$

- $c_1 = q_1 p + 2r_1 + m_1,\ c_2 = q_2 p + 2r_2 + m_2$
- $c_1 + c_2 = (q_1 + q_2)p + \underbrace{2(r_1 + r_2) + (m_1 + m_2)}_{\text{Distance to nearest multiple of p}}$

  $\rightarrow \quad c_1 + c_2 \bmod p = \underbrace{2(r_1 + r_2)}_{\text{error-term}} + (m_1 + m_2)$
- $c_1 * c_2 =$
  $(c_1 q_2 + q_1 c_2 - q_1 q_2)p + 2(2r_1 r_2 + r_1 m_2 + m_1 r_2) + m_1 * m_2$

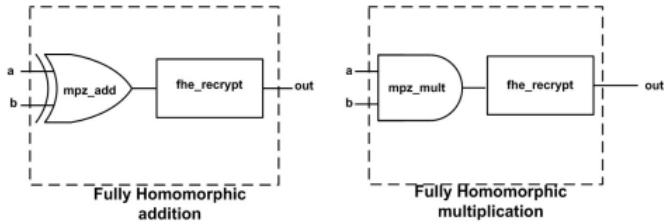
  $\rightarrow \quad c_1 * c_2 \bmod p = \underbrace{2(2r_1 r_2 + \ldots)}_{\text{error-term}} + m_1 * m_2$
- Homomorphic property retains till the error-term is within a certain limit.

$\rightarrow$ **Noise doubles on addition, squares on multiplication.**

Concept of Fully Homomorphic Encryption

- Restrict Noise
- **Bootstrap after each operation [keep noise within a certain limit]**
- Support homomorphic addition and multiplication
- That will support arbitrary operations

Performance and Feasibility??



- $c_1 = q_1 p + 2r_1 + m_1,\ c_2 = q_2 p + 2r_2 + m_2$
- $c_1 + c_2 = (q_1 + q_2)p + \underbrace{2(r_1 + r_2) + (m_1 + m_2)}_{\text{Distance to nearest multiple of p}}$

  $\rightarrow \quad c_1 + c_2 \bmod p = \underbrace{2(r_1 + r_2)}_{\text{error-term}} + (m_1 + m_2)$
- $c_1 * c_2 =$

  $(c_1 q_2 + q_1 c_2 - q_1 q_2)p + 2(2r_1 r_2 + r_1 m_2 + m_1 r_2) + m_1 * m_2$

  $\rightarrow \quad c_1 * c_2 \bmod p = \underbrace{2(2r_1 r_2 + \ldots)}_{\text{error-term}} + m_1 * m_2$
- Homomorphic property retains till the error-term is within a certain limit.

$\rightarrow$ **Noise doubles on addition, squares on multiplication.**

Summary

FHE is an encryption scheme whose ciphertexts can be processed with any deep Boolean circuits or arithmetic circuits without access to the data.

The security of FHE has been usually defined with indistinguishability under chosen-plaintext attack (IND-CPA) security, which is a standard cryptographic security definition.

If the client sends the public keys and the encrypted data with an FHE scheme to the PPML server, the server can perform all computation needed in the desired service before sending the encrypted output to the client.

Challenge/limitation: Performance and feasibility

Applications

Support from FHE Libraries

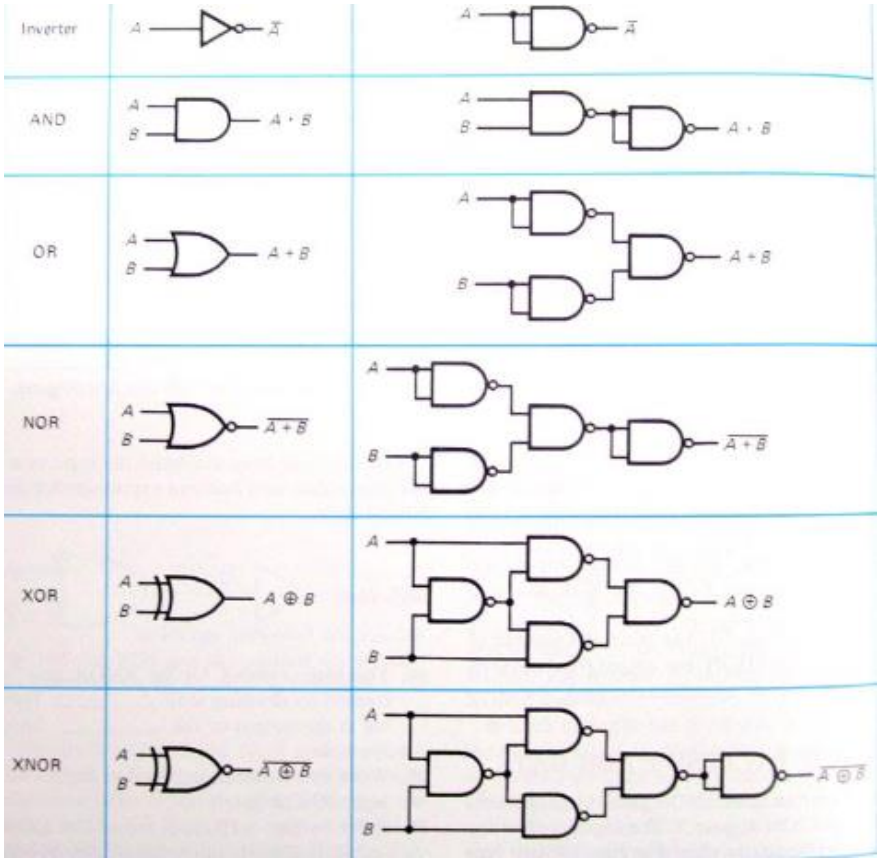
> "it must be emphasized that homomorphy is a theoretical achievement that merely lets us arithmetically add and multiply plaintexts encapsulated inside a ciphertext. In theory, this allows the execution of any algorithm complex manipulations like text replacements or similar, but putting this to practice requires the design (compilation) of a specific circuit representation for the algorithm at hand. This may be a nontrivial task." [2]

- Fully Homomorphic primitive circuits are used to realize the FHS circuit:
  - fhe_add: Add ciphertexts (XOR).
  - fhe_mul: Multiply ciphertexts (AND)

Arbitrary Operations

- A universal gate is **a gate which can implement any Boolean function without need to use any other gate type**.

- The NAND and NOR gates are universal gates.



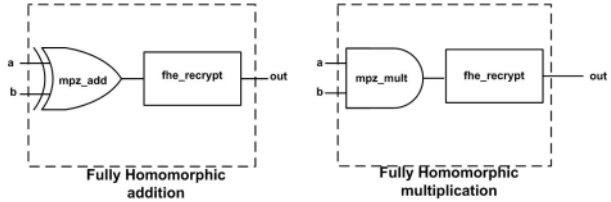


Figure: Fully Homomorphic addition and multiplication

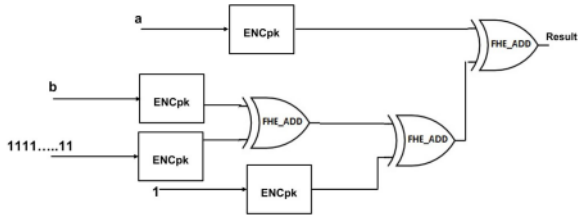


Figure: Fully Homomorphic Subtraction

Binary representation of 3 is: 0 0 1 1

1's Complement of 3 is: 1 1 0 0

2's Complement of 3 is: (1's Complement + 1) i.e.

1 1 0 0 (1's Compliment)

+ 1

1 1 0 1 (2's Complement i.e. -3)

Now 2 + (-3) = 0 0 1 0 (2 in binary)

+ 1 1 0 1 (-3)

1 1 1 1 (-1)

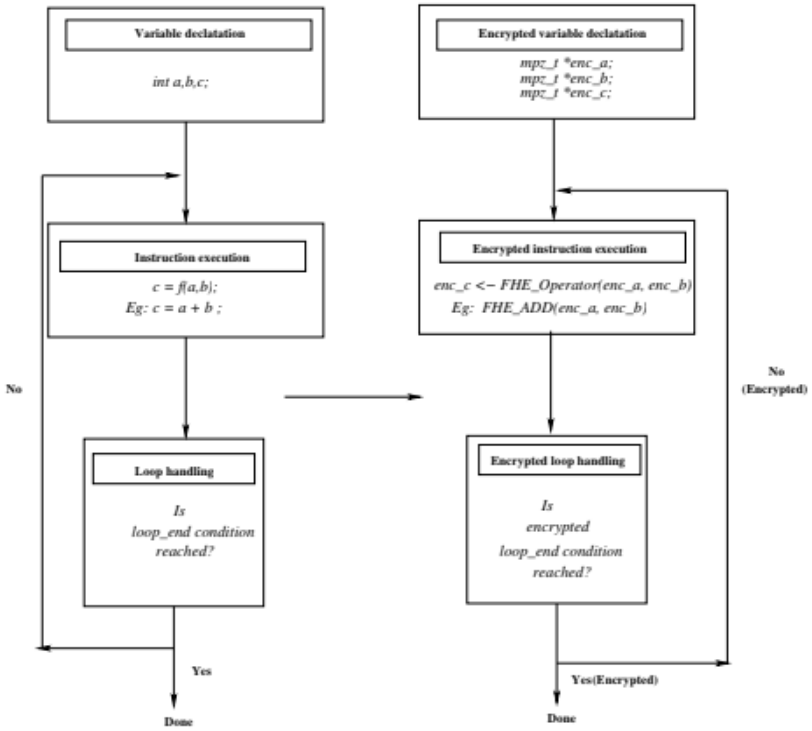
Now, to check whether it is -1 or not simply. takes 2's Complement of -1 and kept -ve sign as it is.

-1 = 1 1 1 1

2's Complement = -(0 0 0 0)

+ 1

-(0 0 0 1) i.e. -1


Variable declatation
int a,b,c;
Instruction execution
c = f(a,b);
Eg: c = a + b ;
Loop handling
Is
loop_end condition
reached?
No
Yes
Done
Encrypted variable declatation
mpz_t *enc_a;
mpz_t *enc_b;
mpz_t *enc_c;
Encrypted instruction execution
enc_c <– FHE_Operator(enc_a, enc_b)
Eg: FHE_ADD(enc_a, enc_b)
Encrypted loop handling
Is
encrypted
loop_end condition
reached?
No
(Encrypted)
Yes(Encrypted)
Done

1. Handing of arithmetic operations
2. Handling of Conditional Branches and detecting Termination Conditions
3. Supporting recursive programs

Major Hurdle

Intermediate encrypted conditions are incomprehensible by the underlying unencrypted processor.

- **Arithmetic Operators** perform addition, subtraction, multiplication and division of two operands.
- **Relational Operators** checks if the values of two operands are equal. These operators also decide the greater - lesser relations between two operands.
- **Logical Operators** include logical AND, OR and NOT operations among the operands.
- **Bitwise Operators** perform bitwise operations among the operands (like bitwise AND, OR operations).
- **Assignment Operators** assign values from right side operands to left side operand.

Operators are implemented using FHE library [3] premitives:

- FHE_add: Add ciphertexts (XOR) (bit-wise).
- FHE_mul: Multiply ciphertexts (AND) (bit-wise).
- FHE_fulladd: Add with carry in and carry out.

Few Challenges

Shift and Add algorithm in existing computers

- Addition and shifting both are performed if the data-bit equals to 1
- While working with encrypted data, no scope to identify whether the bit is $Enc(0)$ or $Enc(1)$.

Proposed Solution:

- Addition and shifting both the operations are performed for every bit.
- Final result computation is by encrypted decision making.
- Incurs significant overhead in terms of performance.

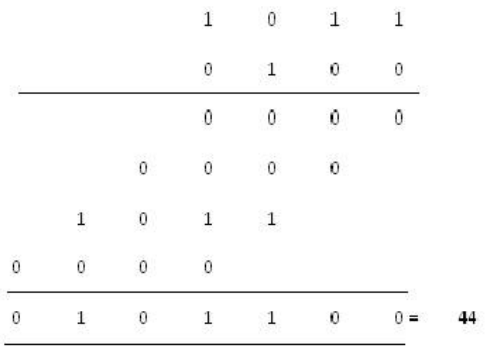
Encrypted DNN*

https://ieeexplore.ieee.org/document/9920696

Revisit Convolution: Multiplication

Eg. If we are multiply 11 * 4

Then 11 = 1011 4 = 0100



Shift and Add algorithm in existing computers

- Addition and shifting both are performed if the data-bit equals to 1
- While working with encrypted data, no scope to identify whether the bit is $Enc(0)$ or $Enc(1)$.

Proposed Solution:

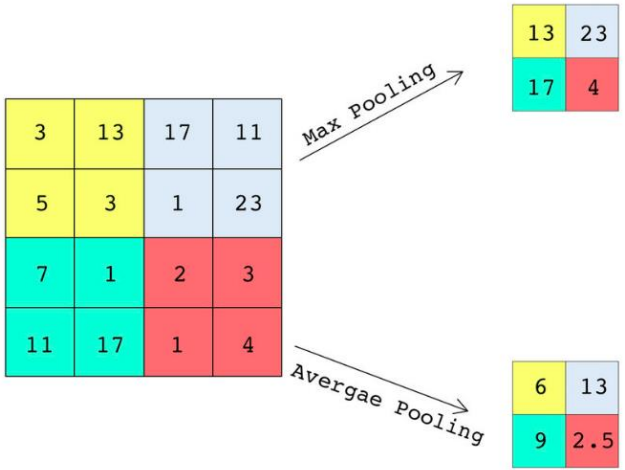
- Addition and shifting both the operations are performed for every bit.
- Final result computation is by encrypted decision making.
- Incurs significant overhead in terms of performance.

MaxPooling/ Average Pooling

Plaintext domain:

Average pooling method smooths out the image and hence the sharp features may not be identified when this pooling method is used.

Max pooling selects the brighter pixels from the image. It is useful when the background of the image is dark and we are interested in only the lighter pixels of the image.



Encrypted domain:

Max pooling is costlier now than average pooling (was mentioned not feasible as mentioned in this 2016 paper)

ReLU

- The sigmoid and the rectified linear activation functions are non-polynomial functions.
- One way is to approximate these functions with low-degree polynomials.
- chose to use the lowest-degree non-linear polynomial function, which is the square function: sqr(z) := z^2 .
- This limitation increases overfitting

$$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases} = \max\{0, x\} = x\mathbf{1}_{x>0}$$

Result

Layer	Description	Time to compute
Convolution layer	Weighted sums layer with windows of size 5×5, stride size of 2. From each window, 5 different maps are computed and a padding is added to the upper side and left side of each image.	30 seconds
1^{st} square layer	Squares each of the 835 outputs of the convolution layer	81 seconds
Pool layer	Weighted sum layer that generates 100 outputs from the 835 outputs of the 1^{st} square layer	127 seconds
2^{nd} square layer	Squares each of the 100 outputs of the pool layer	10 seconds
Output layer	Weighted sum that generates 10 outputs (corresponding to the 10 digits) from the 100 outputs of the 2^{nd} square layer	1.6 seconds

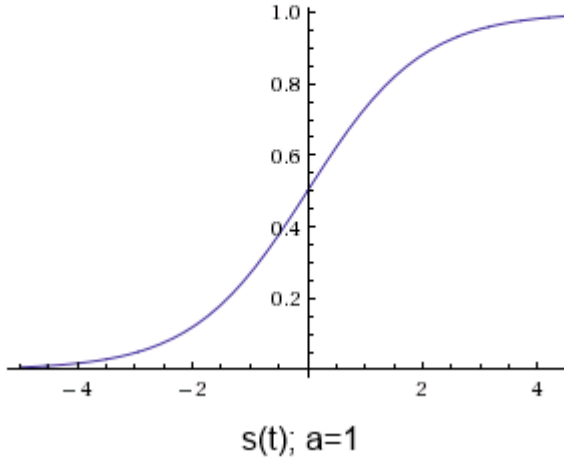
Compare this timing requirement with plaintext timing

Open Challenge: 1. Sigmoid Layer

- The sigmoid activation function is necessary for the training stage in order to get reasonable error terms when running the gradient descent algorithm.
- However, there is no good way of dealing with the sigmoid in the encrypted realm.
- Observation : This layer is not required in prediction stage

$$s(t) = \frac{1}{(1 + e^{-at})}$$

Plot:



s(t); a=1

How to do encrypted decision making?*

*https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7274687

Few Challenges

- Encrypted loop implies the execution of single or sequence of encrypted instructions.
- Initialization (or termination) condition of such loop execution is again the result of some encrypted operations: difficult with present unencrypted processors.
- Traditional loop control statements like go to control statement, which transfers control to a label.
- These labels should also be encrypted, which is impossible in existing unencrypted processor.

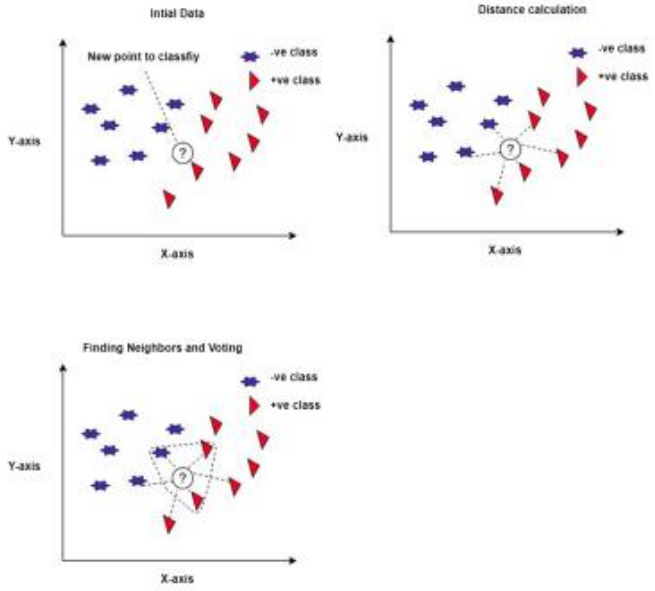
Existing Solution:

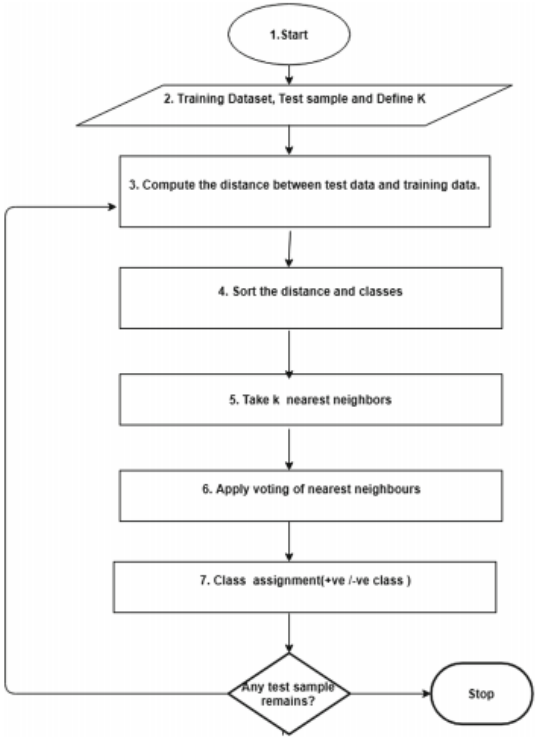
- Defining maximum loop-length with unencrypted value [3].
- Use of encrypted decision making to obtain the effective result.

Encrypted KNN Computation:

Ref: https://link.springer.com/content/pdf/10.1007/978-3-030-35869-3_13.pdf

KNN Algorithm





Distance Computation Step

Euclidean distance:

$$d(p,q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}.$$

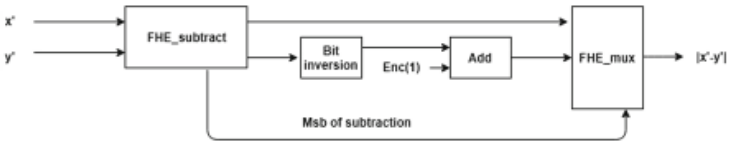


Fig. 4. Encrypted absolute value computation

Manhattan distance:

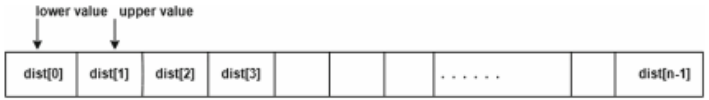
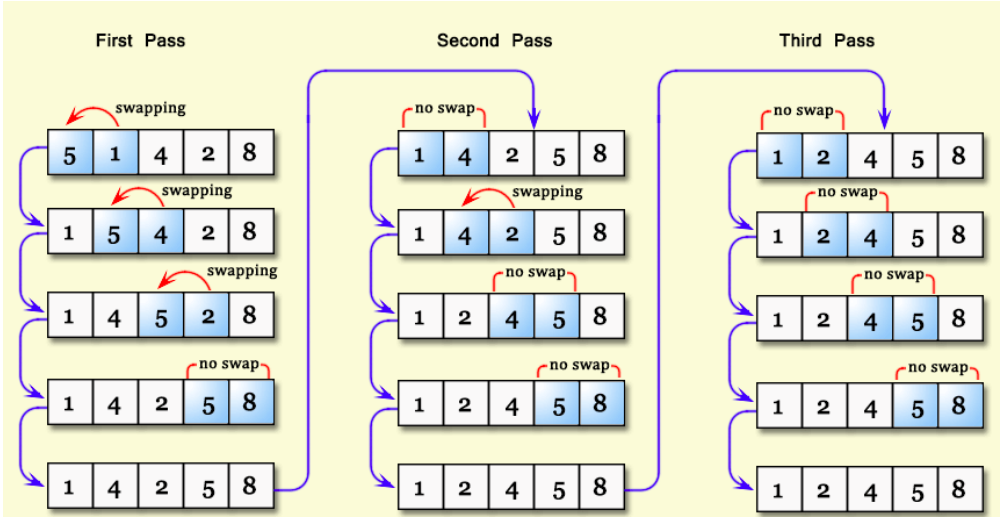


Fig. 5. Distance array

If there are two vectors, $\mathbf{p} = (p_1, p_2 \ldots, p_n)$ and $\mathbf{q} = (q_1, p_2 \ldots, q_n)$,

$$\sum_{i=1}^{n} |p_i - q_i|$$

Sorting Step




```c
void bubble_sort (int *x, int n) {
    int i, t, j = n, s = 1;
    while (s) {
        s = 0;
        for (i = 1; i < j; i++) {
            if (x[i] < x[i - 1]) {
                t = x[i];
                x[i] = x[i - 1];
                x[i - 1] = t;
                s = 1;
            }
        }
        j--;
    }
}
```

Sorting Step

```
void bubble_sort (int *x, int n) {
    int i, t, j = n, s = 1;
    while (s) {
        s = 0;
        for (i = 1; i < j; i++) {
            if (x[i] < x[i - 1]) {
                t = x[i];
                x[i] = x[i - 1];
                x[i - 1] = t;
                s = 1;
            }
        }
        j--;
    }
}
```

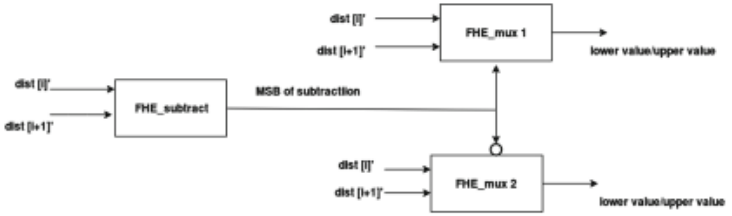


Fig. 6. Fully homomorphic swap

KNN Voting and Class Label Assignment

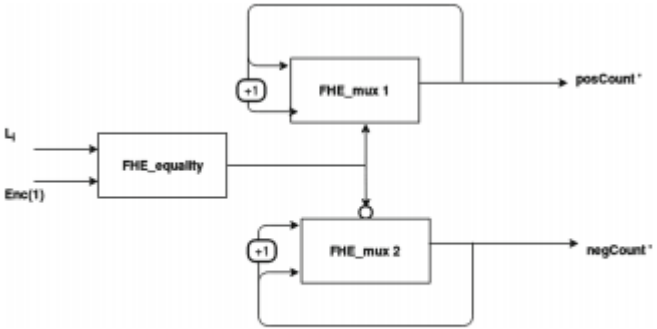


Fig. 7. Fully homomorphic prediction

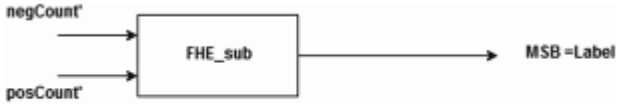


Fig. 8. Fully homomorphic label assignment

Encrypted NN Computation

Points to Ponder

- How to compute the convolution?
- How to compute the pooling layer?
- Feasibility of encrypted activation functions.

Key Observation

- Encrypted decision making (encrypted multiplexer) improves feasibility, but deteriorates timing performance.
- [In electronics, a multiplexer (or mux; spelled sometimes as multiplexor), also known as a data selector, is **a device that selects between several analog or digital input signals and forwards the selected input to a single output line**.
- Mux operator design demands encrypted multiplication]
- Try to reduce encrypted multiplication as far as possible by approximation to improve timing performance.

Multiplexer decides the output from the inputs based on the selection line. Considering the encrypted multiplexer takes two encrypted inputs A' and B', and a selector input S' (which is encrypted form of logic 1 or logic 0), the output Z' of FHE_Mux can be realized as:

$$Z' = A'.\overline{S'} + B'.S'. \quad (3)$$

Hence, considering S' as the encrypted condition, input A' will be assigned to Z', if S' is true ($Enc(1)$) else B' will be assigned to Z'. Now, consider a conditional statement, where decision making is performed based on a condition, whose encrypted value is S'. Thus, if the condition is true, S' is $Enc(1)$, else $Enc(0)$. Hence, the decision making can be realized by the foregoing multiplexer circuit. Note that, such an encrypted decision making can only be realized by a multiplexer circuit (as multiplexer is a universal circuit component), which requires the underlying homomorphism to support both multiplication and addition. This again emphasizes the requirement of FHE as underlying encryption scheme.

CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy, 2016, https://proceedings.mlr.press/v48/gilad-bachrach16.pdf

Common functions in NN

1. *Weighted-Sum* (convolution layer): Multiply the vector of values at the layer beneath it by a vector of weights and sum the results. The weights are fixed during the inference processes. This function is essentially a dot product of the weight vector and the vector of values of the feeding layer.
2. *Max Pooling*: Compute the maximal value of some of the components of the feeding layer.
3. *Mean Pooling*: Compute the average value of some of the components of the feeding layer.
4. *Sigmoid*: Take the value of one of the nodes in the feeding layer and evaluate the function $z \mapsto {}^{1}/_{(1+\exp(-z))}$.
5. *Rectified Linear*: Take the value of one of the nodes in the feeding layer and compute the function $z \mapsto \max(0, z)$.

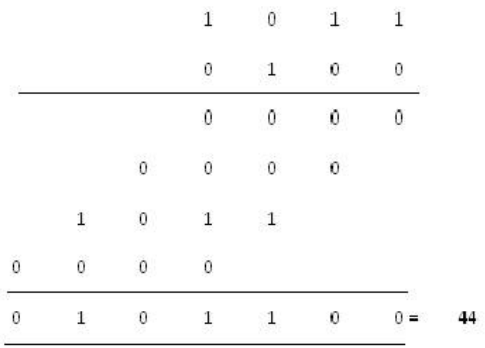
Tentative Size of Example NN

1. *Convolution Layer*: The input image is 28×28. The convolution has windows, or kernels, of size 5×5, a stride of $(2, 2)$, and a mapcount of 5. The output of this layer is therefore $5 \times 13 \times 13$.
2. *Square Activation Layer*: This layer squares the value at each input node.
3. *Scaled Mean Pool Layer*: This layer has $1 \times 3 \times 3$ windows, and again outputs a multi-array of dimension $5 \times 13 \times 13$.
4. *Convolution Layer*: This convolution has a kernel size of $1 \times 5 \times 5$, a stride of $(1, 2, 2)$, and a mapcount of 10. The output layer is therefore $50 \times 5 \times 5$.
5. *Scaled Mean Pool Layer*: As with the first mean pool, the kernel size is $1 \times 3 \times 3$, and the output is $50 \times 5 \times 5$.
6. *Fully Connected Layer*: This layer fully connects the incoming $50 \cdot 5 \cdot 5 = 1250$ nodes to the outgoing 100 nodes, or equivalently, is multiplication by a 100×1250 matrix.
7. *Square Activation Layer*: This layer squares the value at each input node.
8. *Fully Connected Layer*: This layer fully connects the incoming 100 nodes to the outgoing 10 nodes, or equivalently, is multiplication by a 10×100 matrix.
9. *Sigmoid Activation Function*: This layer applies the sigmoid function to each of the 10 incoming values.

Revisit Convolution: Multiplication

Eg. If we are multiply 11 * 4

Then 11 = 1011 4 = 0100



Shift and Add algorithm in existing computers

- Addition and shifting both are performed if the data-bit equals to 1
- While working with encrypted data, no scope to identify whether the bit is $Enc(0)$ or $Enc(1)$.

Proposed Solution:

- Addition and shifting both the operations are performed for every bit.
- Final result computation is by encrypted decision making.
- Incurs significant overhead in terms of performance.

Convolution Restriction

- the multiplications here are between precomputed weights and the values of the feeding layer.
- weights are not encrypted, it is possible to use the more efficient plain multiplication operation.

Leveled homomorphic encryption

- Bootstrapping/ recrypt is costly - performance bottleneck in practical adaptation

- allows adding and multiplying encrypted messages but requires that one knows in advance the complexity of the arithmetic circuit that is to be applied to the data.
- In other words, this cryptosystem allows to compute polynomial functions of a fixed maximal degree on the encrypted data.
- High degree polynomial computation requires the use of large parameters in the scheme, which results in larger encrypted messages and slower computation times.
- Hence, a primary task in making practical use of this system is to present the desired computation as a low degree polynomial.

Open Challenge: 2. Private Model

Encrypted Private Model Private Data Approach still impractical in real time scenario.

Additional Ref

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7274687