SEAL-DL

(Simple Encrypted Arithmetic Library – Deep Learning)

Contents

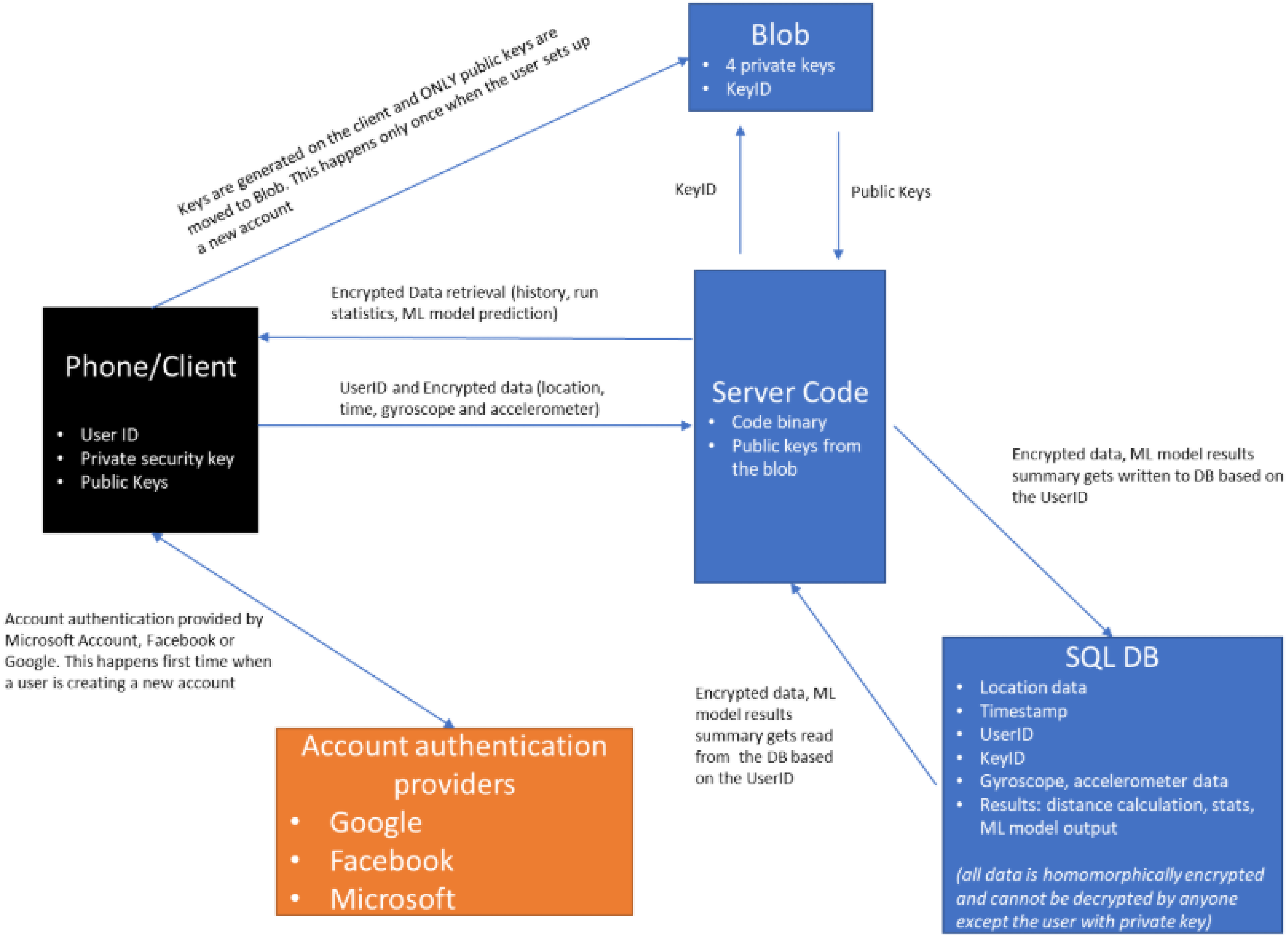
- SEAL DEMO
- HE
- DL
- Efficient Matrix Product
- Implement

SEAL DEMO(https://github.com/microsoft/SEAL-Demo)

- **AsureRun**
  - User의 '달리기'정보(gps위치정보, 가속도등)를 활용하여 운동강도(Intensity)를 분류(low, Medium, High)하는 Machine Learning기반의 추론 Android APP
  - SEAL을 이용하여 User의 정보를 Mobile APP(Client)에서 Encrypt하여 Server, Asuze DB, Account provider(Google, Facebook, Mircrosoft)와 통신하는 Architecture
  - 사내에서 Test하기에는 환경적인 문제 및 build 문제가 많아 선택하지 않은 예제
- **Cloud Functions Demo**
  - Seal기반의 Matrix 연산(덧샘, 뺄셈, 곱셈) 예제로 Client에서 encrypt한 Matrix data를 를 server에서 연산하여 client에서 결과를 확인 할 수 있는 예제
  - C#언어, Azure Functions(이벤트 기반 서버리스 컴퓨팅 플랫폼)을 이용
  - **추가로, Mnist Data(숫자 손글씨 이미지)를 가지고 Meachine Learning Model(Fully connected 1-layer)를 구현**

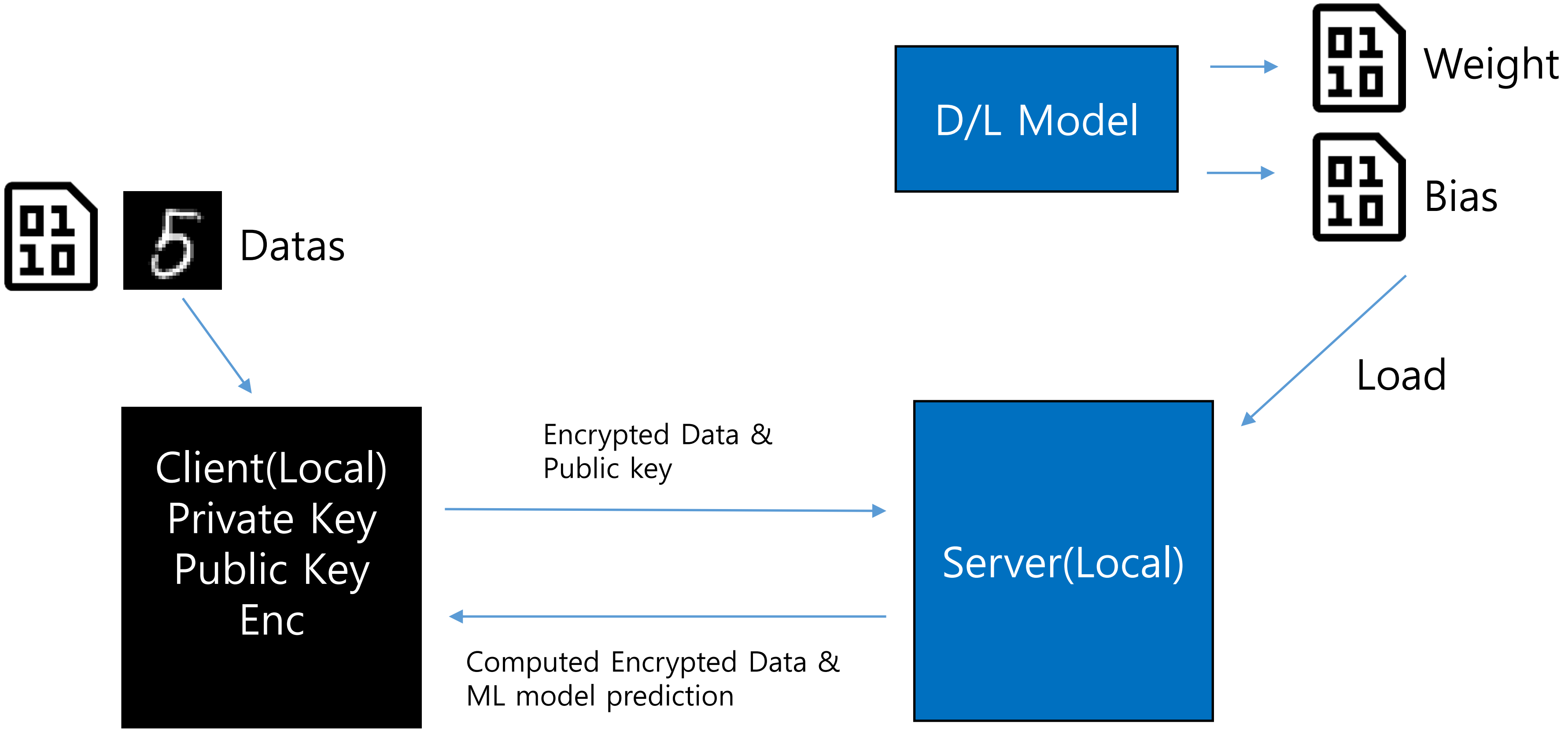
SEAL DEMO

- **AsureRun Architecture**



SEAL DEMO

- **Cloud Functions Demo Architecture**



Homomorphic Encryption(CKKS)

– Secret key: $\mathrm{sk} = (1, s)$

– Encryption key: (b, a) s.t. $b = -a \cdot s + e \ (mod\ q)$

– Enc(m): $(b, a) + (m, 0) = (\mathrm{b} + m, \mathrm{a}) = (\beta, \alpha) = (c_0, c_1) = \mathbf{ct}$

– Dec(ct, sk): $< \mathrm{ct}, \mathrm{sk} > \, = \, < (c_0, c_1), (1, s) > \, = c_0 + c_1 \cdot s$

$$= b + m + a \cdot s$$

$$= -\cancel{a \cdot s} + e + m + \cancel{a \cdot s}$$

$$= m + e$$

Ciphertext Vector (Polynomials)

Secret Key

Noisy Message

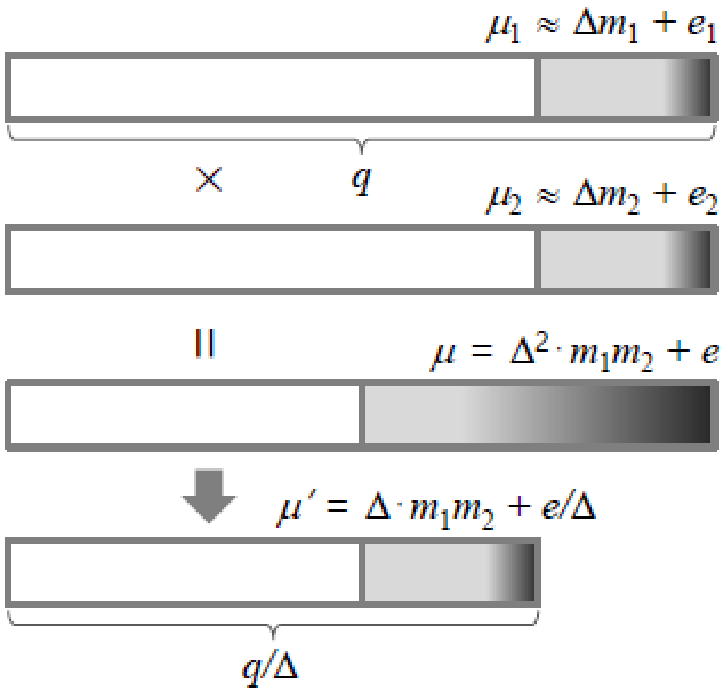
Homomorphic Encryption(CKKS)

- **Homomorphic Operations**
  - Input: $\mu_1 \approx \Delta m_1 + e_1$
    $\mu_2 \approx \Delta m_2 + e_2$
  - Addition: $\mu_1 + \mu_2 \approx \Delta \cdot (m_1 + m_2)$
  - Multiplication: $\mu = \mu_1 \mu_2 \approx \Delta^2 \cdot m_1 m_2$
  - Rescaling(Rounding):

$$ct \mapsto ct' = \lfloor ct/\Delta \rceil$$

$\Rightarrow$ if <ct, sk> $\approx m \ (mod\ q)$,

<ct', sk> $\approx m/\Delta \ (mod\ q/\Delta)$

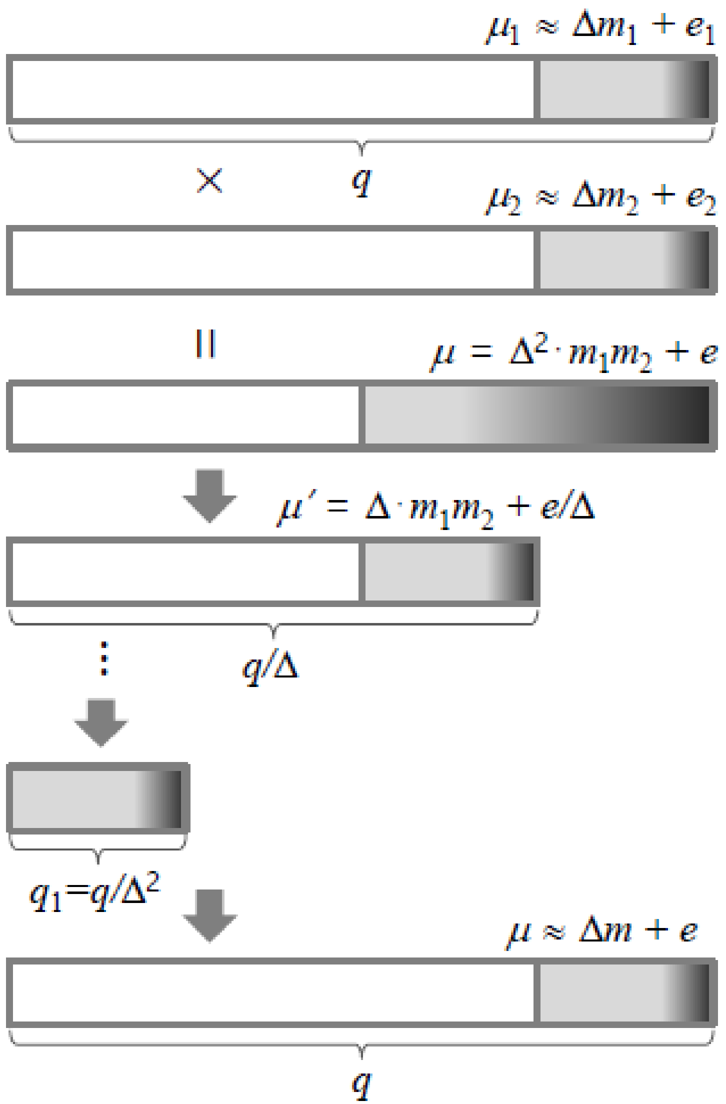


Homomorphic Encryption(CKKS)

- **Homomorphic Operations**
  - Input: $\mu_1 \approx \Delta m_1 + e_1$
    $\mu_2 \approx \Delta m_2 + e_2$
  - Addition: $\mu_1 + \mu_2 \approx \Delta \cdot (m_1 + m_2)$
  - Multiplication: $\mu = \mu_1\mu_2 \approx \Delta^2 \cdot m_1 m_2$
  - Rescaling: $\mu' = \Delta^{-1} \cdot \mu \approx \Delta \cdot m_1 m_2$
  - Leveled HE: $(q_L = \Delta^L) > \ldots > (q_1 = \Delta)$
  - Bootstapping:

    refresh a ciphertext for more computation

    $(q_1 = \Delta) \rightarrow (q_L = \Delta^L)$



Homomorphic Encryption(CKKS)

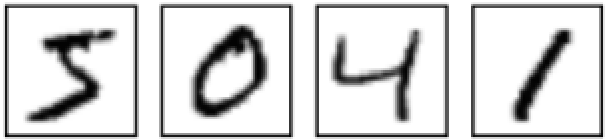
- **Multiplication**
  - A ciphertext contains two polynomials(dimension=2)
    - $ct_1 = (\beta_1, \alpha_1) \rightarrow ct_1(s) = \beta_1 + \alpha_1 s \approx m_1$
    - $ct_2 = (\beta_2, \alpha_2) \rightarrow ct_2(s) = \beta_2 + \alpha_2 s \approx m_2$
  - After multiplication, the dimension of ciphertext increases
    - $m_1 m_2 \approx ct_1(s) \cdot ct_2(s) = (\beta_1 + \alpha_1 s) \cdot (\beta_2 + \alpha_2 s)$
      $= \beta_1\beta_2 + (\alpha_1\beta_2 + \alpha_2\beta_1)s + \alpha_1\alpha_2 s^2$
      $= < (\beta_1\beta_2, \alpha_1\beta_2 + \alpha_2\beta_1, \alpha_1\alpha_2)(1, s, s^2) >$
    - **Mult**$(ct_1, ct_2) \rightarrow \boldsymbol{ct_3} = (\boldsymbol{c_0}, \boldsymbol{c_1}, \boldsymbol{c_2}) = (\beta_1\beta_2,\ \alpha_1\beta_2 + \alpha_2\beta_1, \alpha_1\alpha_2)$

Increased Dimension

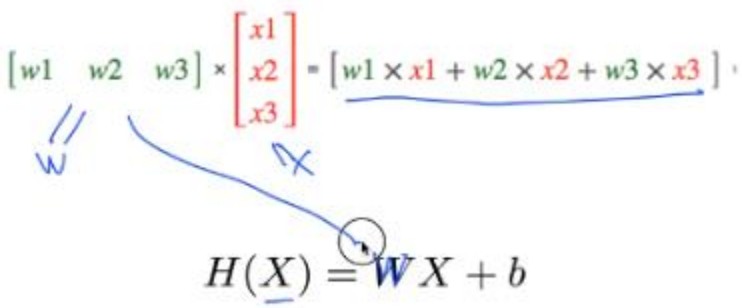
- **Relinearization**
  - Reduce the increased dimension of ciphertext
    - $\boldsymbol{ct_3} = (\boldsymbol{c_0}, \boldsymbol{c_1}, \boldsymbol{c_2}) \Longrightarrow \boldsymbol{ct_3'} = (\boldsymbol{c_0'}, \boldsymbol{c_1'}) = (\boldsymbol{c_0} + \boldsymbol{c_2 s^2}, \boldsymbol{c_1})$
    - $ct_3' = (\beta_3, \alpha_3) \rightarrow \beta_3 + \alpha_3 s = c_0 + c_2 s^2 + c_1 s \ \approx m_1 m_2$
  - $\boldsymbol{s^2}$ is provided with encrypted format, called evaluation key
    - **Enc**$(s^2, s) \rightarrow evk = (k_0, k_1)$
    - **Relin**$(ct_3, evk) \rightarrow ct_3' = (c_0', c_1')$

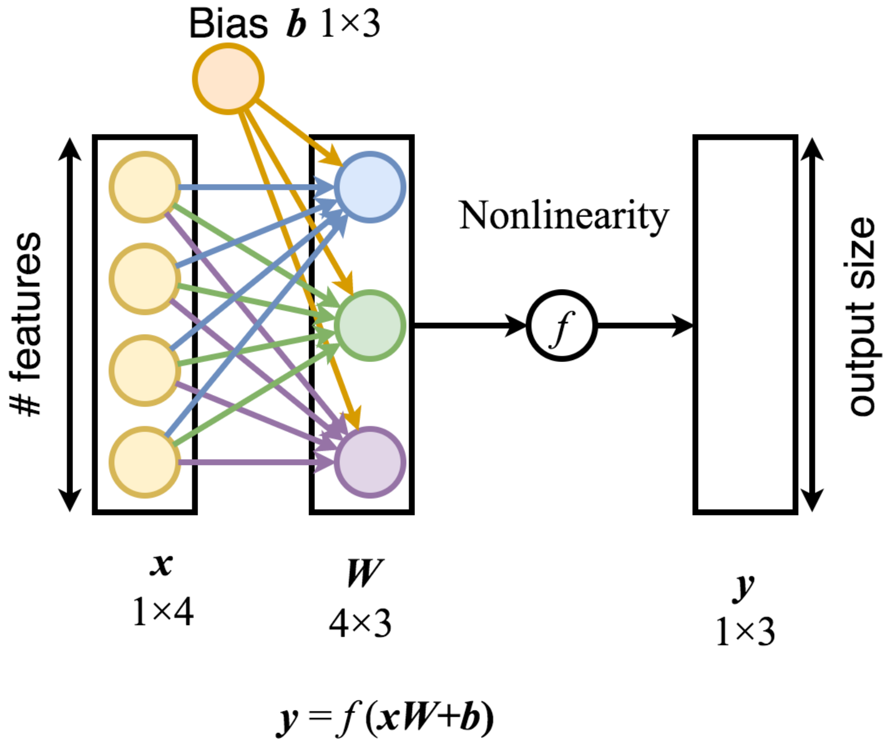
DL(MNIST-FC1)

- **MNIST : 손글씨 숫자 이미지로 간단한 컴퓨터 비전 해상도 28x28 데이터셋**



- **Fully-connected layers**





DL(MNIST-FC1)

- Model
  - Tensorflow 기반
  - Fully Connected 1-layer 구현
  - Wx + b = y (non activation func)

  - W,b Initial value : Zero
  - Optimizer : GradientDescent
  - Train factor : 0.5
  - Epoch : 10

  - Acc : 92.2%

```python
import tensorflow as tf

# 변수들을 설정한다.
x = tf.placeholder(dtype=tf.float32, shape=[None, 784])
W = tf.Variable(dtype=tf.float32, initial_value=tf.zeros([784, 10]))
b = tf.Variable(dtype=tf.float32, initial_value=tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)

# cross-entropy 모델을 설정한다.
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

# 경사하강법으로 모델을 학습한다.
init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)
for i in range(1000):
    print("i : {}".format(i))
    batch_xs, batch_ys = mnist.train.next_batch(550)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

# 학습된 모델이 얼마나 정확한지를 출력한다.
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))

# 모델 저장
saver = tf.train.Saver()
saver.save(sess, './save_model/mnist', global_step=0, write_meta_graph=False)
```

Efficient Matrix Product

(https://github.com/microsoft/SEAL-Demo/blob/master/CloudFunctionsDemo/ClientBasedFunctions/ClientBasedFunctions/MatrixProduct.md)

- 단순한 방식(Iterative)의 행렬곱 알고리즘 대부분은 Θ(n^3)의 시간 복잡도를 가진다.
- SEAL은 여러개의 숫자들을 한번에 연산(+, -, *)할 수 있는 Batch연산기능과 Rotation기능을 지원하기 때문에 특정한 방식으로 행렬을 전처리 하면 행렬곱을 Θ(n)의 시간으로 연산이 가능
- 단순방식

$$\begin{matrix} a & b & c \\ d & e & f \\ g & h & i \end{matrix} \bullet \begin{matrix} A \\ B \\ C \end{matrix} = aA + bB + cC \quad dA + eB + fC \quad gA + hB + iC$$

- Row Rotate & Switch

$$\begin{matrix} \overleftarrow{+0} \\ \overleftarrow{+1} \\ \overleftarrow{+2} \end{matrix} \begin{matrix} a & b & c \\ d & e & f \\ g & h & i \end{matrix} \xrightarrow{\text{Rot}} \begin{matrix} a & b & c \\ e & f & d \\ i & g & h \end{matrix} \xrightarrow{\text{S}} \begin{matrix} a & e & i \\ b & f & g \\ c & d & h \end{matrix}$$

```
int[,] cyclicDiagsMatrix = new int[dimension, dimension];
for (int r = 0; r < dimension; r++)
{
    for (int c = 0; c < dimension; c++)
    {
        cyclicDiagsMatrix[r, c] = paddedMatrixa[c, (c + r) % dimension];
    }
}
```

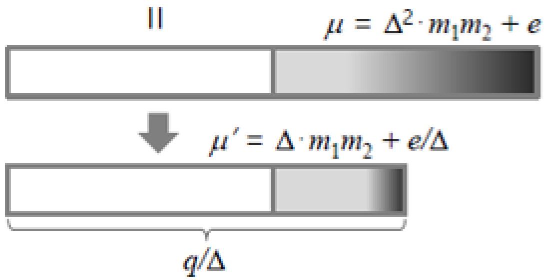
- Cyclic Mutiplication & Row Add

$$\begin{matrix} a & e & i \\ b & f & g \\ c & d & h \end{matrix} * \begin{matrix} A & B & C \\ B & C & A \\ C & A & B \end{matrix} = \begin{matrix} aA & eB & iC \\ bB & fC & gA \\ cC & dA & hB \end{matrix} \xrightarrow{\text{Row Add}} aA + bB + cC \quad dA + eB + fC \quad gA + hB + iC$$

Implement(Client)

- **Init Processing**
  - Generate SEALContext

```
EncryptionParameters parms = new EncryptionParameters(SchemeType.CKKS);
parms.PolyModulusDegree = 8192;
parms.CoeffModulus = CoeffModulus.Create(CKKS_PolyModulusDegree, new int[] { 60, 40, 40, 60 });
ckks_context_ = new SEALContext(parms);
```



PolyModulusDegree	max CoeffModulus bit-length
1024	27
2048	54
4096	109
8192	218
16384	438
32768	881

Rescaling factors
Rescaling 순서(2^60, 2^40, 2^40, 2^60)

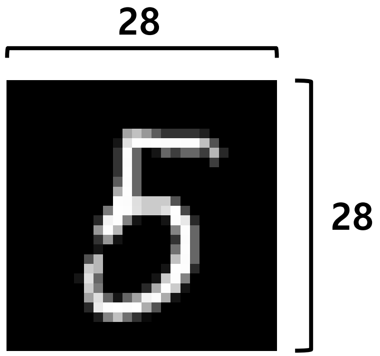
PolyModulusDegree 크기에 따라 할당 할 수 있는 CoffModulus의 최대 사이즈가 달라짐

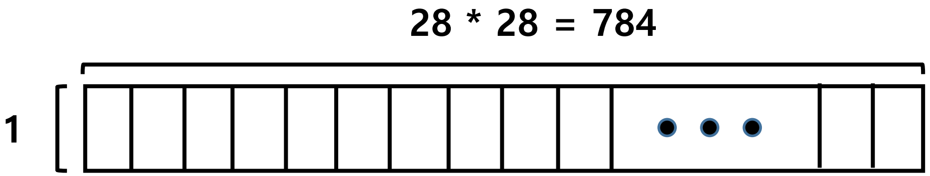
  - Generate variety Classes from SEALContext

```
KeyGenerator ckks_keygen = new KeyGenerator(GlobalProperties.CKKS_Context);
ckks_encryptor_ = new Encryptor(GlobalProperties.CKKS_Context, ckks_keygen.PublicKey);
ckks_decryptor_ = new Decryptor(GlobalProperties.CKKS_Context, ckks_keygen.SecretKey);
ckks_encoder_ = new CKKSEncoder(GlobalProperties.CKKS_Context);
```

Implement(Client)

- **Data Processing**






```
double[,] matrix = new double[1, 784];
```

```
double[] batchArray = new double[encoder.SlotCount];

for (int r = 0; r < rows; r++)
{
    for (int c = 0; c < cols; c++)
    {
        batchArray[r * cols + c] = (double)matrix[r, c];
    }
}

Plaintext plain = new Plaintext();
double scale = Math.Pow(2.0, 40);
encoder.Encode(batchArray, scale, plain);
```

SlotCount는 한꺼번에 연산할 수 있는 Batch Size로 CKKS에서는 PolyModulusDegree/2

Context에서 CoffModulus를 { 60, 40, 40, 60 }으로 설정하였기 때문에 유효숫자를 **20bit**로 맞추기 위해 초기 scale은 40으로 설정함
2^40 * 2^40 = 2^80
Rescaling시 2^80 / 2^60 = 2^20

Batch배열을 polynomial 형태의 Plaintext로 인코딩

Implement(Client)

- **Encrypt Processing & Data Encoding**
  - Encrypt Data & base64 encoding

```
using (MemoryStream ms = new MemoryStream()
{
    Ciphertext cipher = new Ciphertext();
    encryptor.Encrypt(plain, cipher);
    cipher.Save(ms);
    byte[] bytes = ms.ToArray();
    return Convert.ToBase64String(bytes);
}
```

  - Publickey base64 encoding

```
MemoryStream ms = new MemoryStream();
ckks_pk_.Save(ms);
string b64pk = Convert.ToBase64String(ms.ToArray());

string json = $"{{ ₩"sid₩": ₩"{sid}₩", ₩"matrixa₩": ₩"{b64matrixa}₩", ₩"pk₩": ₩"{b64pk}₩" }}";
```

Implement(Server)

- **Init & Data decoding**
  - Generate Evaluator

```
ckks_evaluator_ = new Evaluator(CKKS_Context);
```

  - base64 decoding

```
Ciphertext result = new Ciphertext();
byte[] bytes = Convert.FromBase64String(b64);
using (MemoryStream ms = new MemoryStream(bytes))
{
    result.Load(context, ms);
}

return result;
```

  - Publickey Load

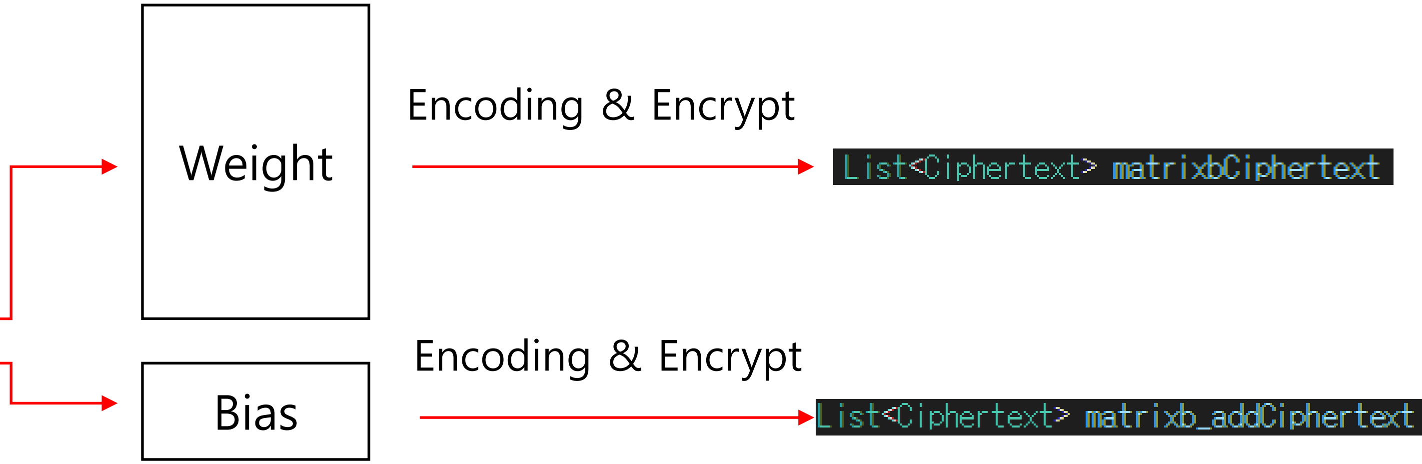
```
PublicKey ckks_pk = new PublicKey();
byte[] pk = Convert.FromBase64String(b64pk);
using (MemoryStream ms = new MemoryStream(pk))
{
    ckks_pk.Load(GlobalProperties.CKKS_Context, ms);
}
```

Implement(Server)

- **Compute ML Processing**
  - Load Model Weight, bias

```
const int w_row = 784;
const int w_col = 10;

double[,] matrixb = new double[w_row, w_col];
double[,] matrixb_add = new double[1, w_col]
double[,] matrixb_add_t = new double[1, w_col]
```



Implement(Server)

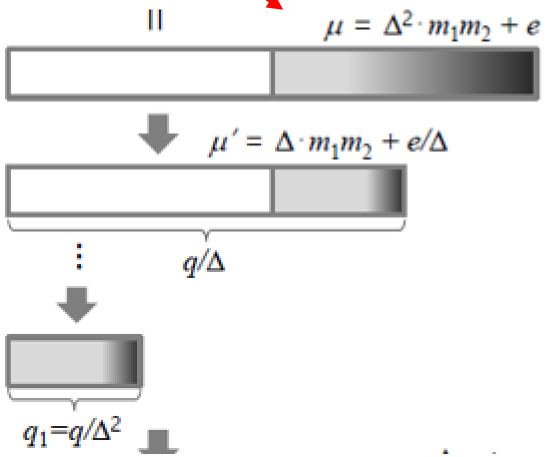
- **Compute ML Processing**

$$\mathbf{Mult}(ct_1, ct_2) \rightarrow \boldsymbol{ct_3} = (\boldsymbol{c_0}, \boldsymbol{c_1}, \boldsymbol{c_2}) = (\beta_1\beta_2,\ \alpha_1\beta_2 + \alpha_2\beta_1, \alpha_1\alpha_2)$$

```
for (int i = 0; i < matrixbCiphertext.Count; i++)
{
    Ciphertext currProduct = new Ciphertext();
    GlobalProperties.CKKS_Evaluator.Multiply(c_matrixa, matrixbCiphertext[i], currProduct);
    GlobalProperties.CKKS_Evaluator.MultiplyInplace(matrixb_addCiphertext[i], matrixb_add_tCiphertext[i]);
    GlobalProperties.CKKS_Evaluator.AddInplace(currProduct, matrixb_addCiphertext[i]);
    GlobalProperties.CKKS_Evaluator.RelinearizeInplace(currProduct, rlk);

    GlobalProperties.CKKS_Evaluator.RescaleToNextInplace(currProduct);
    GlobalProperties.CKKS_Evaluator.ModSwitchToNextInplace(currProduct);
    tempResult.Add(currProduct);
}
```

Multiply시 Dimension이 증가하여 덧셈연산이 불가 -> 덧셈할 데이터를 동일 Dimension으로 맞추기 위해 [1]*n 배열을 Multiply해줌



Implement(Client)

- **Decrypt Processing & Result**
  - Decrypt & Decoding

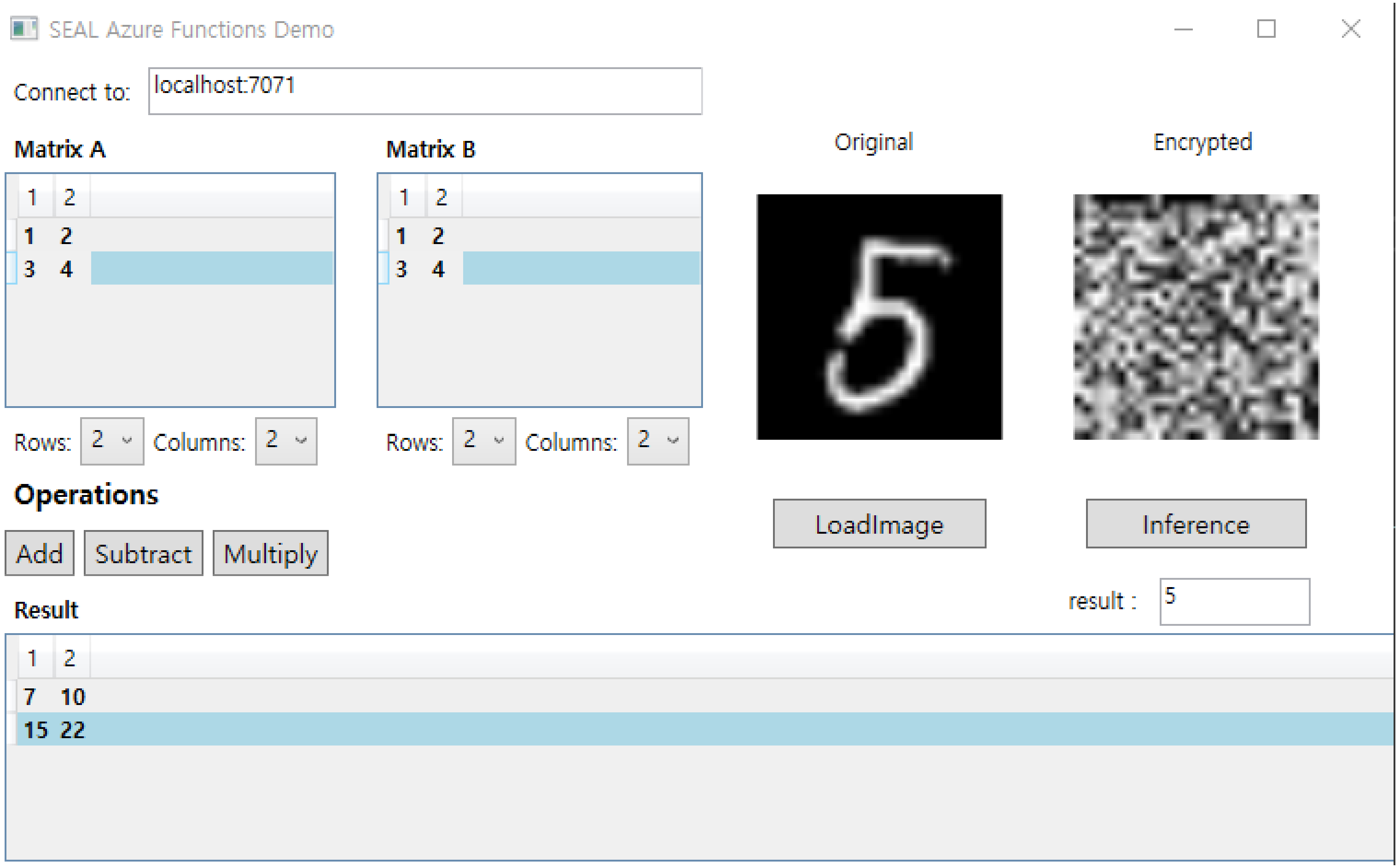
```
List<Ciphertext> resultCipher = Utilities.Base64ToCiphertextList(resultb64, GlobalProperties.CKKS_Context);
List<Plaintext> result = new List<Plaintext>();
for (int i = 0; i < resultCipher.Count; i++)
{
    Plaintext ptmp = new Plaintext();
    ckks_decryptor_.Decrypt(resultCipher[i], ptmp);
    result.Add(ptmp);
}
```

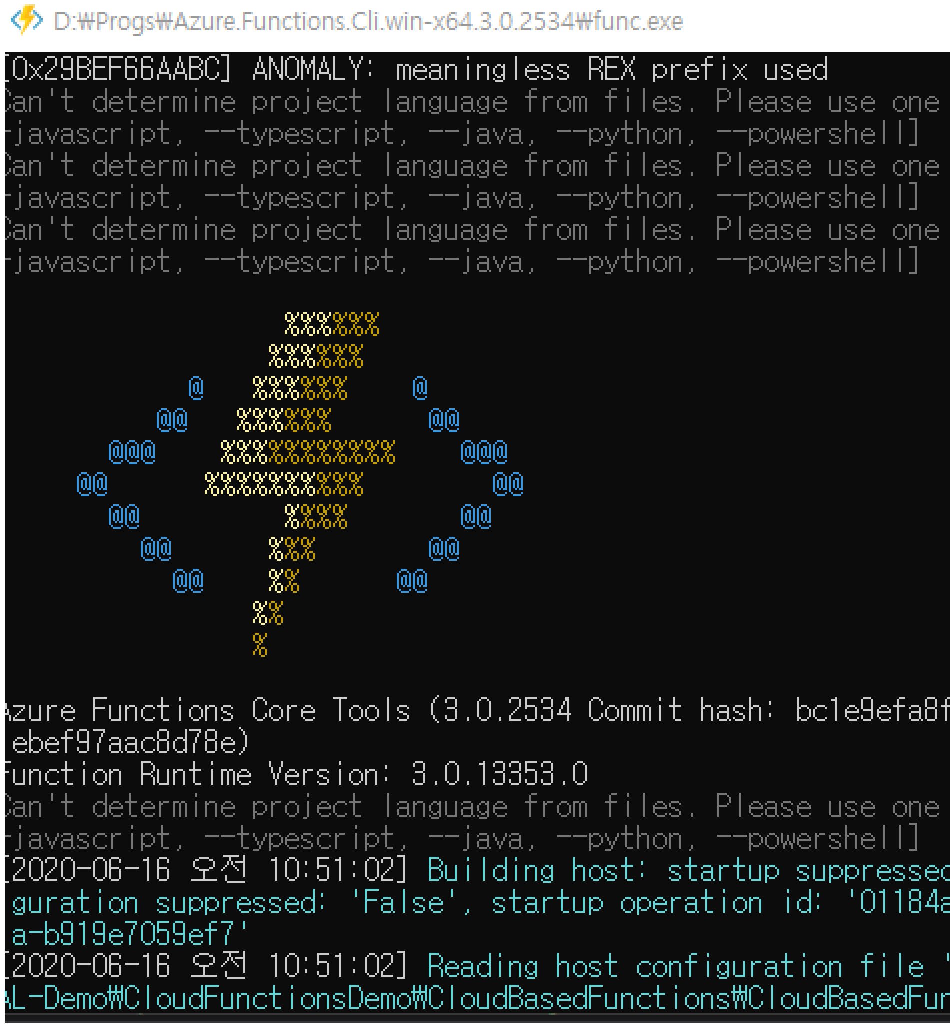
```
encoder.Decode(plains[i], batchArray);
```

  - Result

```
int classfy_count = result.GetLength(dimension: 1);
double maxv = 0.0;
int maxi = 0;
for (int i = 0; i < classfy_count; i++)
{
    if (maxv < result[0, i])
    {
        maxv = result[0, i];
        maxi = i;
    }
}
textBox.Text = Convert.ToString(maxi);
```

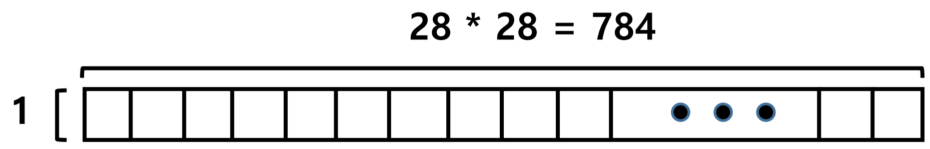
RUN

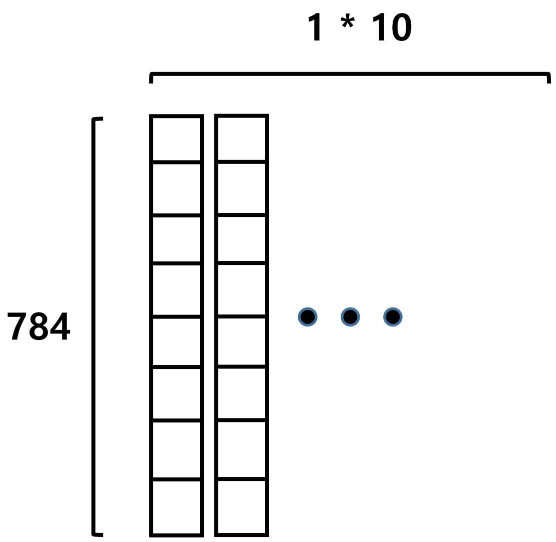

SEAL Azure Functions Demo
Connect to: localhost:7071
Matrix A
Matrix B
Original
Encrypted
Rows: 2 Columns: 2
Rows: 2 Columns: 2
Operations
Add Subtract Multiply
LoadImage
Inference
result : 5
Result
1 2
7 10
15 22


D:₩Progs₩Azure.Functions.Cli.win-x64.3.0.2534₩func.exe
[0x29BEF66AABC] ANOMALY: meaningless REX prefix used
Can't determine project language from files. Please use one
-javascript, --typescript, --java, --python, --powershell]
Azure Functions Core Tools (3.0.2534 Commit hash: bc1e9efa8f
ebef97aac8d78e)
Function Runtime Version: 3.0.13353.0
[2020-06-16 오전 10:51:02] Building host: startup suppressed
guration suppressed: 'False', startup operation id: '01184a
a-b919e7059ef7'
[2020-06-16 오전 10:51:02] Reading host configuration file '
L-Demo₩CloudFunctionsDemo₩CloudBasedFunctions₩CloudBasedFun

Multi Inference

- Mnist Data 전체(10000) 이미지에 대해 Test위하여 Inference 동작방식 변경
  i) Memory issue 해결위해 begin – update – end 방식으로 구현
  ii) Server에서 모든 Inference연산 수행 위해 Matrix Product 방식 변경

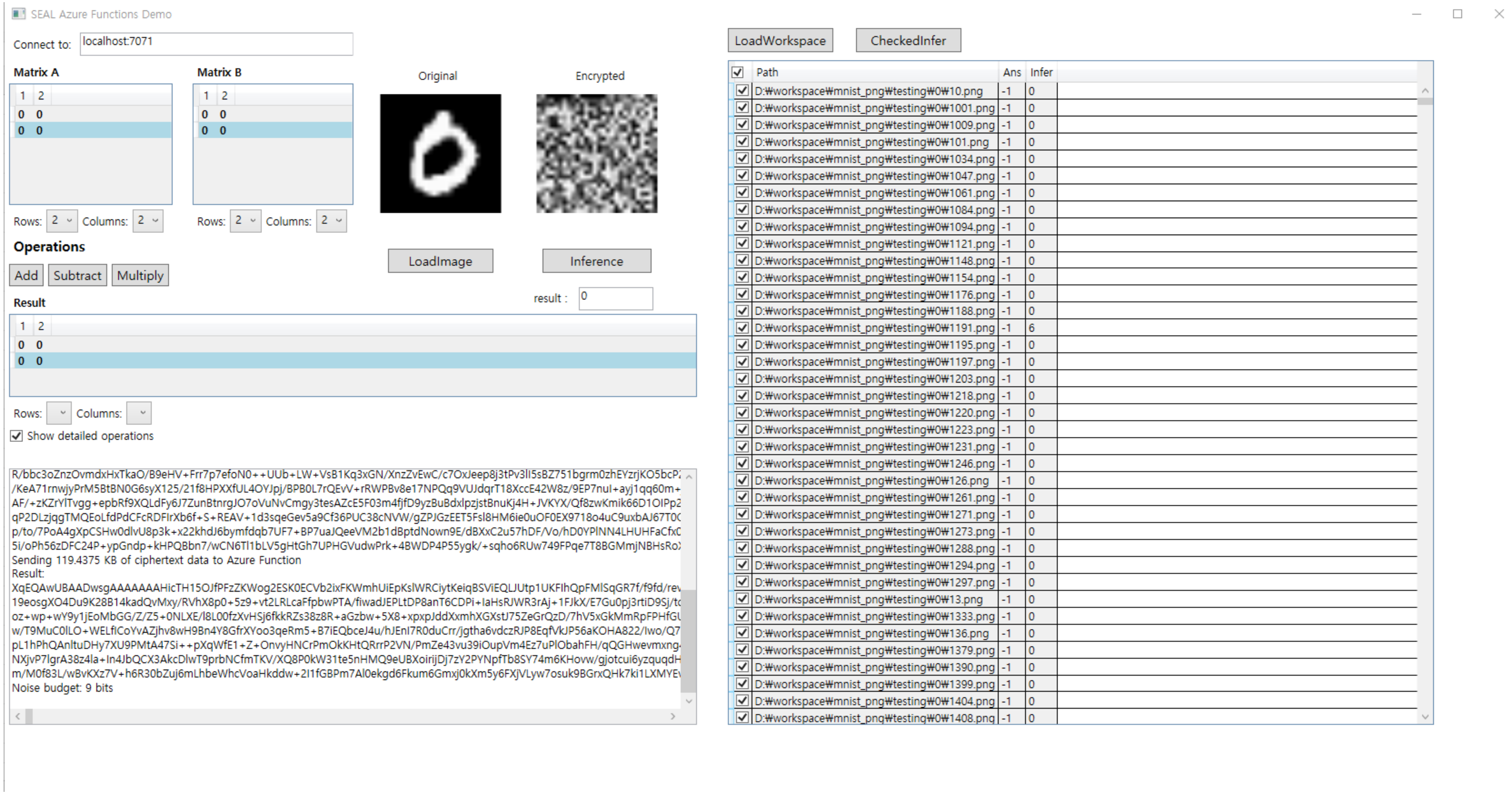




➡ **Ciphertext * 784 update**

PolyModulus를 8192로 하면 slot은 4096만큼 사용할 수 있으며 Mnist Data의 Class 갯수는 10개 이므로 <u>409개의 이미지를 한번에 Inference 가능</u>
(but, 1개할때랑 409개할때랑 동일 시간 소요)

Multi Inference



Multi Inference

- TestSet – Mnist

TrainSet	validate	test
45000	5000	10000

- Result
  - Acc : 92.13% -> 90.13%(▼2%)

	orginal	SEAL
acc	92.13%	90.13%
time	-	561ms

seal

Y/Yhat	0	1	2	3	4	5	6	7	8	9
0	965			1			6	1	7	
1		1087	2	2			4	1	38	1
2	13	6	884	15	9		14	9	72	10
3	4		11	919	1	4	4	7	51	9
4	2	1	2	1	888		12	1	21	54
5	16	2	2	57	12	591	19	8	176	9
6	12	3	3	2	9	3	915	1	10	
7	4	10	19	10	8			897	11	69
8	5	3	3	12	5	2	4	2	932	6
9	11	4	1	8	19			5	26	935

py

Y/Yhat	0	1	2	3	4	5	6	7	8	9
0	960		8	3	1	10	12	2	7	11
1		1107	7		1	3	3	7	6	6
2	2	2	909	20	2	3	4	21	6	2
3	2	2	17	918	1	32	2	9	19	10
4		1	13		921	9	11	6	9	38
5	4	2	2	28		775	14	1	28	6
6	9	4	15	2	11	17	907		10	
7	1	2	14	12	2	6	3	947	13	26
8	2	15	40	18	9	30	2	2	866	7
9			7	9	34	7		33	10	903

BootStrapinghttps://github.com/snucrypto/HEAAN

- HEAAN Lib에는 CKKS schem에 한해 BootStrap 존재
  Sample Code도 같이 제공

```
void TestScheme::testBootstrap(long logq, long logp, long logSlots, long logT) {
	cout << "!!! START TEST BOOTSTRAP !!!" << endl;

	srand(time(NULL));
	SetNumThreads(8);
	TimeUtils timeutils;
	Ring ring;
	SecretKey secretKey(ring);
	Scheme scheme(secretKey, ring);

	timeutils.start("Key generating");
	scheme.addBootKey(secretKey, logSlots, logq + 4);
	timeutils.stop("Key generated");

	long slots = (1 << logSlots);
	complex<double>* mvec = EvaluatorUtils::randomComplexArray(slots);

	Ciphertext cipher;
	scheme.encrypt(cipher, mvec, slots, logp, logq);

	cout << "cipher logq before: " << cipher.logq << endl;

	scheme.modDownToAndEqual(cipher, logq);
	scheme.normalizeAndEqual(cipher);
	cipher.logq = logQ;
	cipher.logp = logq + 4;

	Ciphertext rot;
```

BootStraping

- **Modulo Down**

Plaintext : $2^6 \equiv 2^2\ (MOD\ \ 2^4)$

$Down\ Modulo : 2^6 \equiv 2^3 (MOD\ \ 2^3)$

$Rescaling\ : 2^5 \equiv 2^2 (MOD\ \ 2^3)$

- **Modulo Up**

Plaintext : $2^6 \equiv 2^2\ (MOD\ \ 2^4)$

$Up\ Modulo : 2^6 \equiv 2^1 (MOD\ \ 2^5)$

$Bootstraping\ : B(2^6) \equiv 2^2 (MOD\ \ 2^5)$

$$m\ (MOD\ q)$$

$$\Big\downarrow \text{Modulo Up } (q \ll Q)$$

$$\boxed{qI} + m\ (MOD\ Q)$$

BootStraping

- **How to delete qI !? -** Approximation of the Modular Reduction Function F(t)

  HEAAN에서는 modulo를 증가시켰을때 나오는 qI항을 제거하기 위한 함수 F(t)에 대한 Approximation를 다음과 같이 제안함

$$F(t) \approx S(t) = \frac{q}{2\pi} \sin\left(\frac{2\pi t}{q}\right)$$

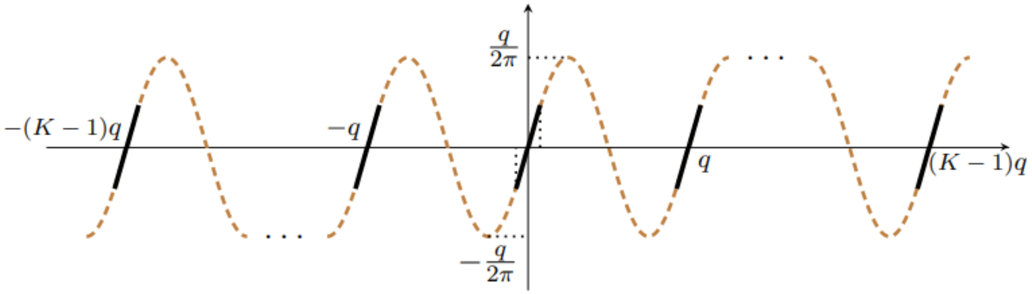


Fig. 1. Modular reduction and scaled sine functions

$$m\ (MOD\ q)$$

$$\Big\downarrow \text{Modulo Up } (q \ll Q)$$

$$t = qI + m\ (MOD\ Q)$$

$$\Big\downarrow \text{F(t)}$$

$$m\ (MOD\ Q)$$

BootStraping

- **Small Angle Approximation**

Sin함수는 각도가 충분히 작을때 다음과 같이 근사가능

$$\sin\theta \approx \theta \;\; (\text{small } \theta)$$
$$\sin(2\pi + \theta) = \sin\theta$$

$$\sin\left(\frac{2\pi t}{q}\right) = \sin\left(2\pi I + \frac{2\pi m}{q}\right) = \sin\left(\frac{2\pi m}{q}\right) \approx \frac{2\pi m}{q} \qquad (m \ll q)\ (t = Iq + m)$$

$$S(t) = \frac{q}{2\pi}\sin\left(\frac{2\pi t}{q}\right) \approx m$$

BootStraping

- **Complex Exponential Function**

  sin함수는 다음과 같이 complex exponential 함수를 이용해 표현 가능 하며,
  exp는 곱과 합만으로 연산 가능한 Tayler급수로 표현 가능하다.
  => 동형암호 연산으로 sin함수표현가능

$$\begin{cases} \exp(i\theta) = \cos\theta + i \cdot \sin\theta, \\ \exp(2i\theta) = (\exp(i\theta))^2, \end{cases}$$

$$\sin\left(\frac{2\pi m_j}{q}\right) = \frac{1}{2}\left(\exp\left(\frac{2\pi i m_j}{q}\right) - \exp\left(\frac{-2\pi i m_j}{q}\right)\right)$$

BootStraping

- **BootStraping process**

$$m\ (MOD\ q)$$

$$\Big\downarrow \text{ Modulo Up } (q \ll Q)$$

$$t = qI + m\ (MOD\ Q)$$

$$\Big\downarrow \text{ F(t)}$$

$$m\ (MOD\ Q)$$

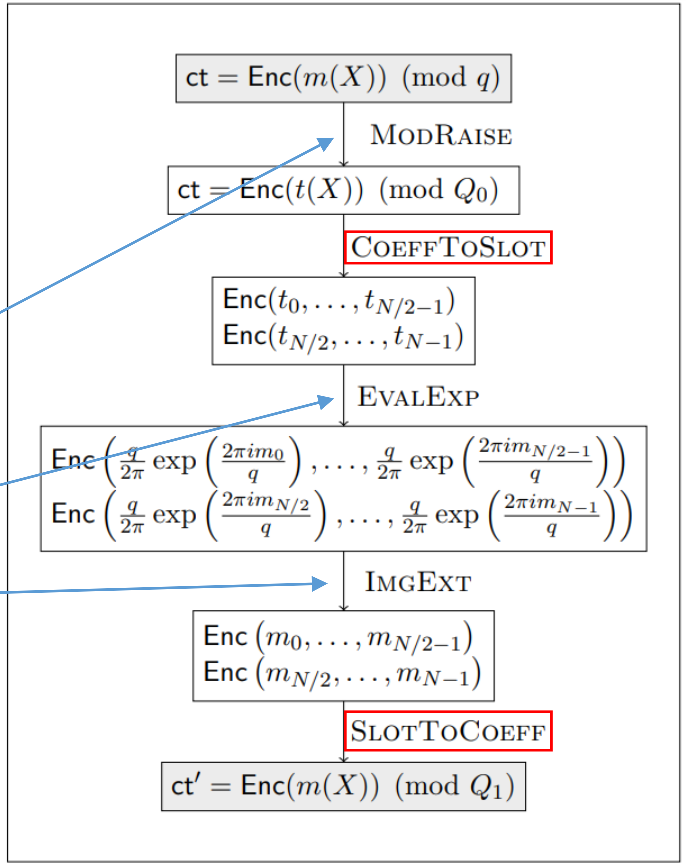


Fig. 2. Pipeline of our bootstrapping process

BootStraping

- **CoeffToSlot & SlotToCoeff**

HEAAN은 CRT Packing을 사용하기 때문에 F(t)를 적용하기
전에 Plaintext Slot상태로 UnPacking을 해줘야 한다. (※ Conjugate연산이 필요하기 때문)

– $\Phi_8(x) = x^4 + 1 \equiv (x - \zeta)(x - \zeta^3)(x - \zeta^5)(x - \zeta^7)$

$$\zeta = exp\left(\frac{2\pi i}{8}\right) = (1 + i)/\sqrt{2}$$

– Plaintext $z = (3 + 4i, 2 - i)$ → $m(x) = t_0 + t_1 x + t_2 x^2 + t_3 x^3$

$$m(x) = 3 + 4i \mod (x - \zeta) \Leftrightarrow m(\zeta) = 3 + 4i$$
$$m(x) = 2 - i \mod (x - \zeta^3) \Leftrightarrow m(\zeta^3) = 2 - i$$
$$m(x) = 2 + i \mod (x - \zeta^5) \Leftrightarrow m(\zeta^5) = 2 + i$$
$$m(x) = 3 - 4i \mod (x - \zeta^7) \Leftrightarrow m(\zeta^7) = 3 - 4i$$

$$U \cdot P = S$$

$$U^{-1} \cdot S = P$$

$$\begin{pmatrix} 1 & \zeta & \zeta^2 & \zeta^3 \\ 1 & \zeta^3 & \zeta^6 & \zeta^9 \\ 1 & \zeta^5 & \zeta^{10} & \zeta^{15} \\ 1 & \zeta^7 & \zeta^{14} & \zeta^{21} \end{pmatrix} \cdot \begin{pmatrix} t_0 \\ t_1 \\ t_2 \\ t_3 \end{pmatrix} = \begin{pmatrix} 3+4i \\ 2-i \\ 2+i \\ 3-4i \end{pmatrix} \Leftrightarrow \begin{pmatrix} t_0 \\ t_1 \\ t_2 \\ t_3 \end{pmatrix} = \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ \zeta^7 & \zeta^5 & \zeta^3 & \zeta \\ \zeta^6 & \zeta^2 & \zeta^6 & \zeta^2 \\ \zeta^5 & \zeta^7 & \zeta & \zeta^3 \end{pmatrix} \cdot \begin{pmatrix} 3+4i \\ 2-i \\ 2+i \\ 3-4i \end{pmatrix} \approx \frac{1}{4} \begin{pmatrix} 10 \\ 4\sqrt{2} \\ 10 \\ 2\sqrt{2} \end{pmatrix}$$

Vandermonde Matrix

U

$$\rightarrow m(x) = \frac{1}{4}(10 + 4\sqrt{2}x + 10x^2 + 2\sqrt{2}x^3)$$

BootStraping

- **Code <coeffToSlotAndEqual>**

```
NTL_EXEC_RANGE(k, first, last);
for (long j = first; j < last; ++j) {
    multByPolyNTT(tmpvec[j], rotvec[j], bootContext->rpvec[j], bootContext->bndvec[j], bootContext->logp);
}
NTL_EXEC_RANGE_END;

for (long j = 1; j < k; ++j) {
    addAndEqual(tmpvec[0], tmpvec[j]);
}

cipher.copy(tmpvec[0]);
for (long ki = k; ki < slots; ki += k) {
    NTL_EXEC_RANGE(k, first, last);
    for (long j = first; j < last; ++j) {
        multByPolyNTT(tmpvec[j], rotvec[j], bootContext->rpvec[j + ki], bootContext->bndvec[j + ki], bootContext->logp);
    }
    NTL_EXEC_RANGE_END;
    for (long j = 1; j < k; ++j) {
        addAndEqual(tmpvec[0], tmpvec[j]);
    }
    leftRotateFastAndEqual(tmpvec[0], ki);
    addAndEqual(cipher, tmpvec[0]);
}
reScaleByAndEqual(cipher, bootContext->logp);
```

BootStraping

- **Code <evalExpAndEqual>**

```
conjugate(tmp, cipher);
subAndEqual(cipher, tmp);
divByPo2AndEqual(cipher, logT + 1); // bitDown: logT + 1
exp2piAndEqual(cipher, bootContext->logp); // bitDown: logT + 1 + 3(logq + logI)
for (long i = 0; i < logI + logT; ++i) {
    squareAndEqual(cipher);
    reScaleByAndEqual(cipher, bootContext->logp);
}
conjugate(tmp, cipher);
subAndEqual(cipher, tmp);
multByPolyNTT(tmp, cipher, bootContext->rp1, bootContext->bnd1, bootContext->logp);
Ciphertext tmprot;
leftRotateFast(tmprot, tmp, slots);
addAndEqual(tmp, tmprot);
multByPolyNTTAndEqual(cipher, bootContext->rp2, bootContext->bnd2, bootContext->logp);
leftRotateFast(tmprot, cipher, slots);
addAndEqual(cipher, tmprot);
addAndEqual(cipher, tmp);
```

BootStraping

- **Code <slotToCoeffAndEqual>**

```
NTL_EXEC_RANGE(k, first, last);
for (long j = first; j < last; ++j) {
    multByPolyNTT(tmpvec[j], rotvec[j], bootContext->rpvecInv[j], bootContext->bndvecInv[j], bootContext->logp);
}
NTL_EXEC_RANGE_END;

for (long j = 1; j < k; ++j) {
    addAndEqual(tmpvec[0], tmpvec[j]);
}
cipher.copy(tmpvec[0]);

for (long ki = k; ki < slots; ki+=k) {
    NTL_EXEC_RANGE(k, first, last);
    for (long j = first; j < last; ++j) {
        multByPolyNTT(tmpvec[j], rotvec[j], bootContext->rpvecInv[j + ki], bootContext->bndvecInv[j + ki], bootContext->logp);
    }
    NTL_EXEC_RANGE_END;

    for (long j = 1; j < k; ++j) {
        addAndEqual(tmpvec[0], tmpvec[j]);
    }

    leftRotateFastAndEqual(tmpvec[0], ki);
    addAndEqual(cipher, tmpvec[0]);
}
reScaleByAndEqual(cipher, bootContext->logp);
```