POLITECNICO
MILANO 1863

Privacy-Preserving Convolutional Neural Networks through Homomorphic Encryption

Politecnico di Milano, DEIB

Carmen Barletta

Matr. 877129

Advisor: Prof. M. Roveri

Co-Advisor: S. Disabato

December 20, 2018

Introduction

- Machine Learning techniques have become popular thanks to their ability to solve complex problems.
- *Software as a Service* is a commonly used machine learning paradigm
  - Pros: fast, scalable, can be used by a large number of users☺
  - Cons: it involves an "honest but curious server"☹

A Threat To Data Owners' Privacy

Analyzed data, that might be sensible data, can be leaked on the outsourced server ☹☹

- The EU's General Data Protection Regulation requires that personal data is processed ensuring adequate protection.

Goals

Methodological solution to design a privacy-preserving Convolutional Neural Network through the usage of the Homomorphic Encryption:

1. Proposing a **methodology** to convert a plain CNN in a privacy-preserving CNN end able to execute calculi on encrypted data (i.e., images).
2. Defining a **heuristic** to estimate optimal encryption parameters.
3. Providing a **library** that allows to accomplish the aforementioned transformation.

HE: Brakerski/Fan-Vercauteren Scheme

Definition

An encryption scheme that allows computations to be done directly on encrypted data is said to be a **homomorphic encryption scheme**.

- Add a "small" noise component during encryption.
- SK is noise extracted from a discrete gaussian error distribution.
- PK is built using SK (can't distinguish this from a Uniform instance)

The main parameters to set are n,t and q. Indeed:

- Plaintext space is $R_t = \mathbb{Z}_t[x]/(x^n + 1)$
- Ciphertext space is $R_q = \mathbb{Z}_q[x]/(x^n + 1)$
- A message is encrypted by passing from R_t to R_q with $q > t$

A ciphertext can be a result of Ciphertext-Ciphertext and/or Ciphertext-Plaintext operations.

BFV-Noise

By making operations, the initial noise in the ciphertext increases.

- The **noise budget** $NB \cong log_2 \frac{q}{t}$ is the maximum noise is possible to add to the ciphertext before decryption fails:
  - each operation adds a different quantity of noise
  - it indicates the maximum number of operations it is possible to evaluate on a ciphertext (*Circuit depth*)

Infinity norm $|| \cdot ||_\infty$

The maximum value of a coefficient in a polynomial. It grows by applying operations on the polynomial.

parameter	**name**	**what it is**
n	poly_mod	max poly degree
q	coeff_mod	UB of max infinity norm of ciphertext poly
t	plain_mod	UB of max infinity norm of plaintext poly

Next Subsection

BFV - Problems(1)

Encryption Parameters Problem

We want to find the right mix of n, q and t for a given CNN.

CNNs require a big circuit depth $\Rightarrow$ NB sufficiently large!

- Setting a big n increases security, but there's a slowdown
  - Doubling $n \rightarrow$Doubling execution time
- Setting an unnecessary big q introduces security issues, but increases the NB
- Setting an unnecessary big t reduces NB...
  - ... but a too small t can introduce incorrectness!

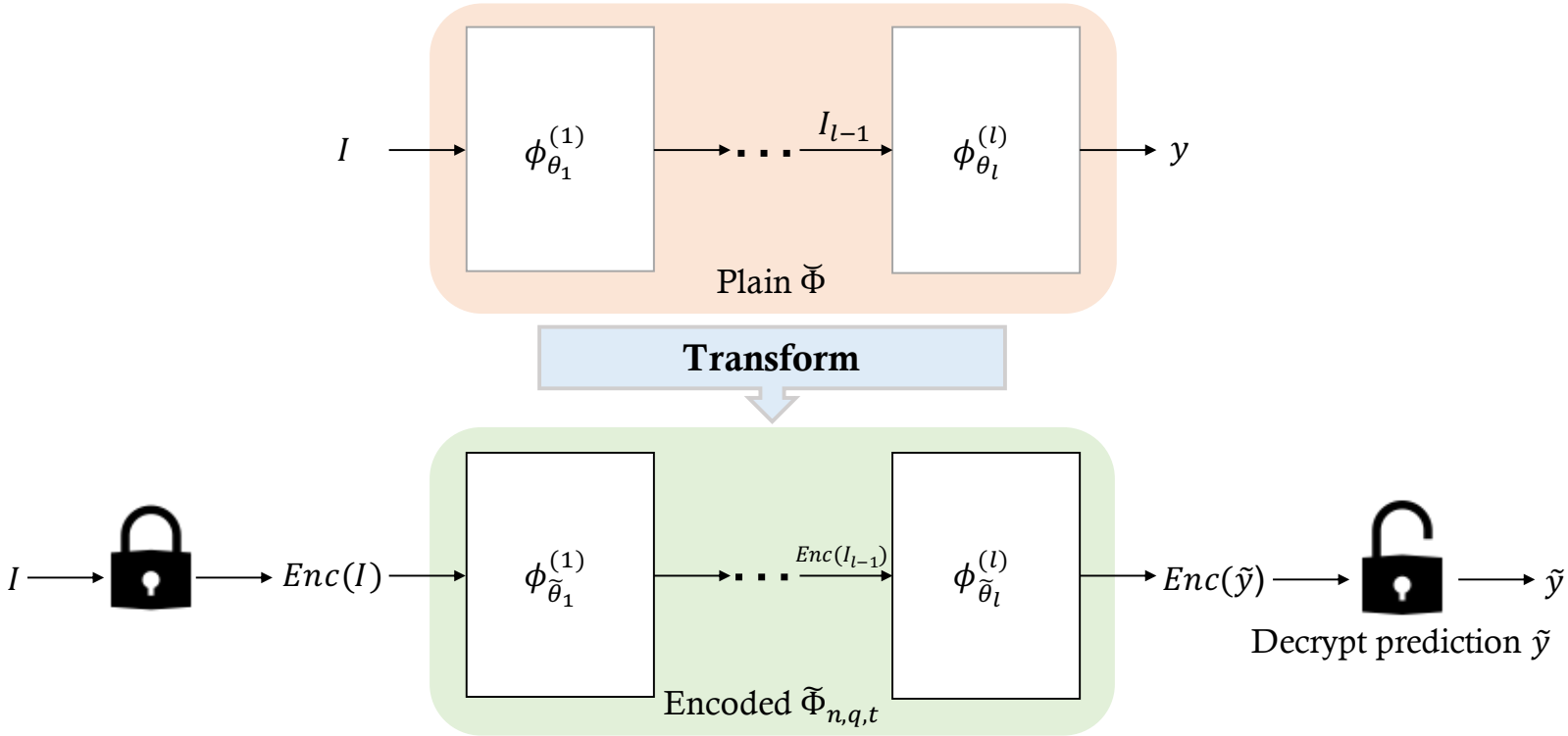
BFV - Problems(2)

- Operations allowed: **addition** and **multiplication**
  - ... can compute only polynomial functions
- Ciphertext-Ciphertext multiplications are the most expensive operations
- Some common CNN's operations:
  - Activation functions: ReLU, Sigmoid, Tanh
  - Pooling Functions: MaxPooling
- ... cannot be directly implemented

Function Problem

We want to find low degree polynomials that approximate these functions.

Solution Overview

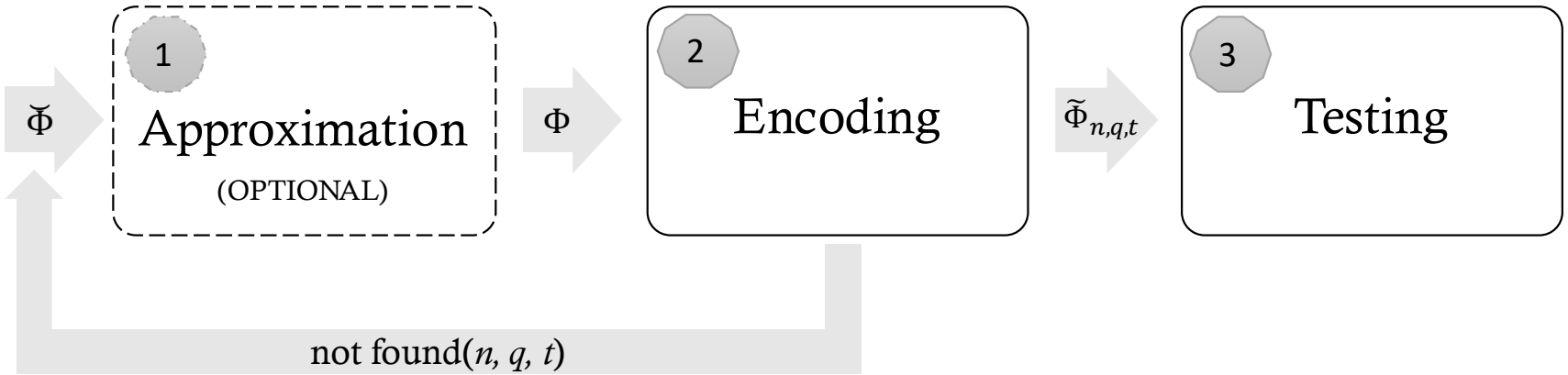


CNN Encoding Implications

CNN is not encrypted, but only encoded:

- Ciphertext-Plaintext operations are used
- Possible to use the same network with different key pairs

Solution - From a plain CNN $\breve{\Phi}$ to a privacy-preserving CNN $\tilde{\Phi}_{n,q,t}$



- STEP 1: solves the function approximation problem
- STEP 2: solves the encryption parameters choice problem
- STEP 3: tests the accuracy of the transformed CNN

Next Subsection

Step 1 Approximation (optional)

- $\breve{\Phi}$ is a pre-trained input CNN
- If $\breve{\Phi}$ contains non-polynomial functions, it is *approximated* with only polynomial functions:

ReLU $\longrightarrow$ Square

- ... otherwise GOTO STEP 2
- The produced approximated Φ is eventually retrained

Next Subsection

Step 2 Encoding

- After selecting encryption parameters, the approximated Φ is *encoded:*
  - each weight in Φ becomes a polynomial with coefficients modulo t and of maximum degree n

Fractional Encoding Example

Encode $r = 5.8125$ in base $B = 2,\ t \geq 2$:

$5.8125 = 2^2 + 2^0 + 2^{-1} + 2^{-2} + 2^{-4}$

`IntEncode(5,B=2)`$= x^2 + 1$

`FractionEncode(0.875,B=2)`$= -x^{n-1} - x^{n-2} - x^{n-4}$

`FractionEncode(5.875,B=2)`$= -x^{n-1} - x^{n-2} - x^{n-4} + x^2 + 1$

`FractionEncode(5.875,B=2)`$_{mod\ t}=$

$(t-1)x^{n-1} + (t-1)x^{n-2} + (t-1)x^{n-4} + x^2 + 1$

- The encoded $\tilde{\Phi}$ is produced
- ... but how do we select the optimal n, q and t?

Optimization Problem for Parameters

$$\min_{n,q,t} \ \alpha \sum_{i}^{D} (y_i - \tilde{y}_i)^2 + \gamma t + \delta n + \beta q$$

subject to

$t < q$

$q \geq 2$

$t > \max ||p||_\infty$ with $p \in R_t$ and generic intermediate result of $\tilde{\Phi}$

$$\log_2 (\frac{q}{t}) > \sum_{i=1}^{I} NB_{n,q,t}(\phi_{\tilde{\theta}_i}^{(i)})$$

$log_2 q \leq UB(\lambda, n)$

$n = 2^d \ s.t. \ d \in \mathbb{N}$

$t, q, n \in \mathbb{N}$

$\alpha, \beta, \gamma, \delta \geq 0$

Heuristic Binary Search of Plaintext Modulus t

Input

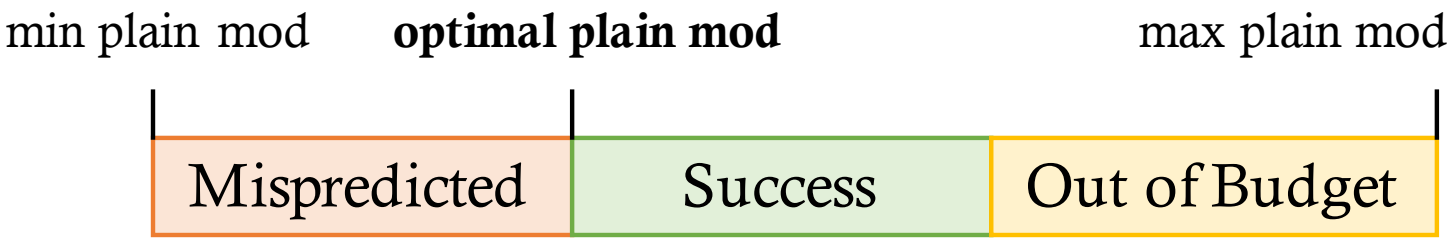
- CNN Φ
- couple (n,q)
- t range
- K images to test
- maximum number of accepted re-encryption steps

Output

- plaintext modulus parameter t
- best point/s in the CNN for the re-encryption/s

Goals

Find the minimal t s.t.:



Next Subsection

Step 3 Testing

- The encoded $\tilde{\Phi}_{n,q,t}$ is a **privacy-preserving CNN**
- The CNN is ready to be tested on encrypted images
  - provided that they are encoded and then encrypted using the same parameters
- The possible accuracy loss introduced by STEP 2 is computed

Next Subsection

Experimental Setup

- Machine 40-cores Intel Xeon CPU E5-2640 @ 2.40GHz with 128GB of RAM on Ubuntu 16.04
- Starting plain CNNs trained using PyTorch
- The encoding and testing of each CNN and the encryption of the data is achieved through the C++ CrCNN library
- Experimental setting on MNIST:
  - **9-Layers CNN**
  - 6-Layers CNN

Next Subsection

9 - Layers CNN (Approximation)

Starting CNN $\breve{\Phi}$	**Approximated CNN Φ**
Convolution $(5 \times 5 \times 20)$	Convolution $(5 \times 5 \times 20)$
Average Pooling $(2 \times 2 \times 1)$	Average Pooling $(2 \times 2 \times 1)$
Batch Normalization	Batch Normalization
Convolution $(3 \times 3 \times 50)$	Convolution $(3 \times 3 \times 50)$
ReLU	Square
Average Pooling $(2 \times 2 \times 1)$	Average Pooling $(2 \times 2 \times 1)$
Batch Normalization	Batch Normalization
Fully Connected $(800, 500)$	Fully Connected $(800, 500)$
Fully Connected $(500, 10)$	Fully Connected $(500, 10)$

9 - Layers CNN (Encoding+Testing)

Relative testing error

$$\epsilon_{\Delta} = \frac{1}{|D|} \cdot \sum_{i \in |D|} \mathbb{I}(y_i \neq \tilde{y}_i) \geq 0$$

Can be added to the encoded model $\tilde{\Phi}_{n,q,t}$ by non optimal encryption parameters

Binary Search $(n = 4096, q_{bits} = 109, max\_reenc = 1, K = 40)$ **Plaintext Mod Found** $t = 2^{29}$	
ϵ_{Δ}	$7/10000$
$Time_{FW}$ [s]	69.07
NB [bit]	69

9 - Layers CNN (Timings+Accuracy Tracking)

Encoded CNN $\tilde{\Phi}_{n=4096,q,t=2^{29}}$	**Threads' #**	**Time[s]**	**NB**
Convolution $(5 \times 5 \times 20)$	20	30.73	69
Average Pooling $(2 \times 2 \times 1)$	1	2.45	64
Batch Normalization	1	2.03	61
Convolution $(3 \times 3 \times 50)$	50	7.89	59
Square	50	0.65	53
Average Pooling $(2 \times 2 \times 1)$	1	0.76	13
Decryption+Encryption	1	3.20	7
Batch Normalization	1	0.68	69
Fully Connected $(800, 500)$	40	18.23	67
Fully Connected $(500, 10)$	50	2.45	59

9 - Layers CNN (Timings+Accuracy Tracking)

Encoded CNN $\tilde{\Phi}_{n=4096,q,t=2^{29}}$	**Threads' #**	**Time[s]**	**NB**
Convolution $(5 \times 5 \times 20)$	20	30.73	69
Average Pooling $(2 \times 2 \times 1)$	1	2.45	64
Batch Normalization	1	2.03	61
Convolution $(3 \times 3 \times 50)$	50	7.89	59
Square	50	0.65	53
Average Pooling $(2 \times 2 \times 1)$	1	0.76	13
Decryption+Encryption	1	3.20	7
Batch Normalization	1	0.68	69
Fully Connected $(800, 500)$	40	18.23	67
Fully Connected $(500, 10)$	50	2.45	59

Accuracy 9-layers CNN

Starting $\check{\Phi}$	**Approximated** Φ	**Encoded** $\tilde{\Phi}_{n=4096,q,t=2^{29}}$
98.25%	97.05%	96.98%

Conclusions and Future Works

This work:

- Proves that **HE+CNNs is a solution** to the privacy issues of cloud based ML
  - ...at the cost of a small accuracy loss and slower predictions.
- Provides a **library** to design privacy-preserving CNNs
  - freely downloadable at `https://github.com/barlettacarmen/CrCNN`

Further works can focus on:

- Performances improvement of the encoded CNN:
  - SIMD techniques
  - hardware accelerators, GPUs, FPGAs
- Training techniques of CNNs directly on encrypted data

Conclusions

Thank you for your attention

Questions?

Homomorphic Encryption

Definition

An encryption scheme that allows computations to be done directly on encrypted data is said to be a **homomorphic encryption scheme**.

- RSA (1978) had homomorphic properties, but Gentry (2009) proposed the 1^{st} FHE scheme
  - Can perform an arbitrary number of **additions** and **multiplications**
  - Practical implementation is unfeasible
- Semantic Security property: encrypting the same information twice can produce different ciphertexts (*Ciphertext indistinguishability* under Chosen Plaintext Attack).

Security of Most Homomorphic Encryption Schemes

$$\text{RLWE problem} \overset{\text{as hard as}}{\rightarrow} \text{SVP} \overset{\text{as hard as}}{\rightarrow} \text{NP-Hard}$$

BFV scheme

- SecretKeyGen(λ) sample $\mathbf{s} \leftarrow \chi$ and output $sk = \boldsymbol{s}$
- PublicKeyGen(sk) set $\boldsymbol{s} = sk$, sample $\boldsymbol{a} \leftarrow R_q$, small error $\boldsymbol{e} \leftarrow \chi$ and output

$$pk = (\boldsymbol{p}_0, \boldsymbol{p}_1) \coloneqq ([-(\boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e})]_q, \boldsymbol{a})$$

- Encrypt(pk, $\boldsymbol{m}$) to encrypt a message $\boldsymbol{m} \in R_t$, sample $\boldsymbol{u}, \boldsymbol{e}_1, \boldsymbol{e}_2 \leftarrow \chi$ and output

$$ct = (\boldsymbol{c}_0, \boldsymbol{c}_1) \coloneqq ([\boldsymbol{p}_0 \cdot \boldsymbol{u} + \boldsymbol{e}_1 + \Delta \cdot \boldsymbol{m}]_q, [\boldsymbol{p}_1 \cdot \boldsymbol{u} + \boldsymbol{e}_2]_q)$$

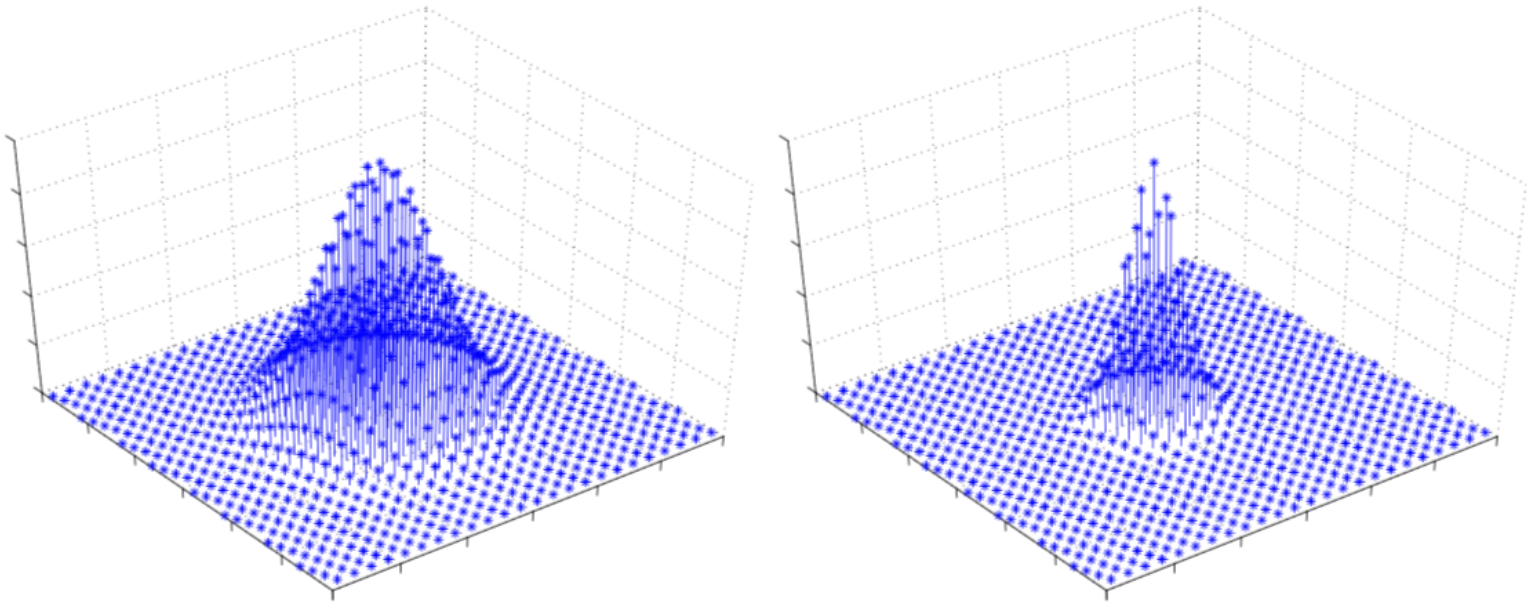
- Decrypt(sk, ct) set $\boldsymbol{s} = sk$ and $ct = (\boldsymbol{c}_0, \boldsymbol{c}_1)$ and compute

$$\left[\left\lfloor \frac{t \cdot [\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s}]_q}{q} \right\rceil \right]_t$$

Discrete Gaussian Distribution

Definition

Discrete Gaussian Distribution $D_{L,r}$ is the distribution whose support is L (which is typically a lattice), and in which the probability of each $\mathbf{x} \in L$ is proportional to $exp(-\pi|\mathbf{x}/r|^2)$



$D_{L,2}$ (left) and $D_{L,1}$ (right) for a two-dimensional lattice L. The z-axis represents probability.

BFV Plain Multiplication

Let $\text{ct} = (x_0, x_1)$ be a ciphertext encrypting m_1 with noise v, and let m_2 be a plaintext polynomial. Let N_{m_2} be an upper bound on the number of non-zero terms in the polynomial m_2. Let ct_{pmult} denote the ciphertext obtained by plain multiplication of ct with m_2. Then the noise in the plain product ct_{pmult} is $v_{pmult} = m_2 v$, and can be bounded as $||v_{pmult}||_\infty \leq N_{m_2} ||m_2||_\infty \, ||v||_\infty$.

Proof. By definition $ct_{pmult} = (m_2 x_0, m_2 x_1)$. Hence for some polynomials a, a' with integer coefficients,

$$\begin{aligned}
\frac{t}{q}\text{ct}_{\text{pmult}}(s) &= \frac{t}{q}(m_2 x_0 + m_2 x_1 s) \\
&= m_2 \frac{t}{q}(x_0 + x_1 s) \\
&= m_2 \frac{t}{q} ct(s) \\
&= m_2 (m_1 + v + at) \\
&= m_1 m_2 + m_2 v + m_2 at \\
&= [m_1 m_2]_t + m_2 v + (m_2 a - a')t,
\end{aligned}$$

where in the last line has been used $[m_1 m_2]_t = m_1 m_2 + a' t$. Hence the noise is $v_{pmult} = m_2 v$ and can be bounded as $||v_{pmult}||_\infty \leq N_{m_2} ||m_2||_\infty \, ||v||_\infty$.

BFV Plain Addition

Let $\text{ct} = (x_0, x_1)$ be a ciphertext encrypting m_1 with noise v, and let m_2 be a plaintext polynomial. Let ct_{padd} denote the ciphertext obtained by plain addition of ct with m_2. Then the noise in ct_{padd} is $v_{padd} = v - \frac{r_t(q)}{q} m_2$, and the bound is $||v_{padd}||_\infty \leq ||v||_\infty + \frac{r_t(q)}{q}||m_2||_\infty$.

Proof. By definition of plain addition $\text{ct}_{\text{padd}} = (x_0 + \Delta m_2, x_1)$. Hence for some polynomials a, a' with integer coefficients,

$$\begin{aligned}
\frac{t}{q}\text{ct}_{\text{padd}}(s) &= \frac{t}{q}(x_0 + \Delta m_2 + x_1 s) \\
&= \frac{\Delta t}{q} m_2 + \frac{t}{q}(x_0 + x_1 s) \\
&= \frac{\Delta t}{q} m_2 + \frac{t}{q} ct(s) \\
&= m_1 + v + \frac{q - r_t(q)}{q} m_2 + at \text{ (because q=}\Delta\cdot\text{t+}r_t(\text{q})\text{)} \\
&= m_1 + m_2 + v - \frac{r_t(q)}{q} m_2 + at \\
&= [m_1 + m_2]_t + v - \frac{r_t(q)}{q} m_2 + (a - a')t,
\end{aligned}$$

where in the last line has been used $[m_1 + m_2]_t = m_1 + m_2 + a't$. Hence the noise is $v_{padd} = v - \frac{r_t(q)}{q} m_2$ and can be bounded as $||v_{padd}||_\infty \leq ||v||_\infty + \frac{r_t(q)}{q}||m_2||_\infty$.

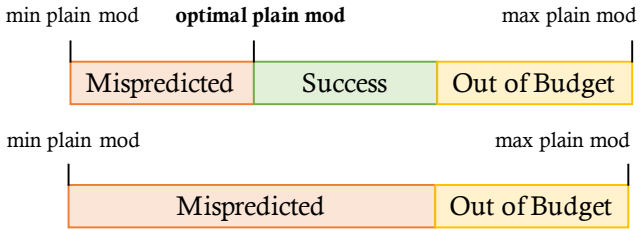
Default pairs (n, q) for 128-bit, 192-bit, and 256-bit λ-security levels.

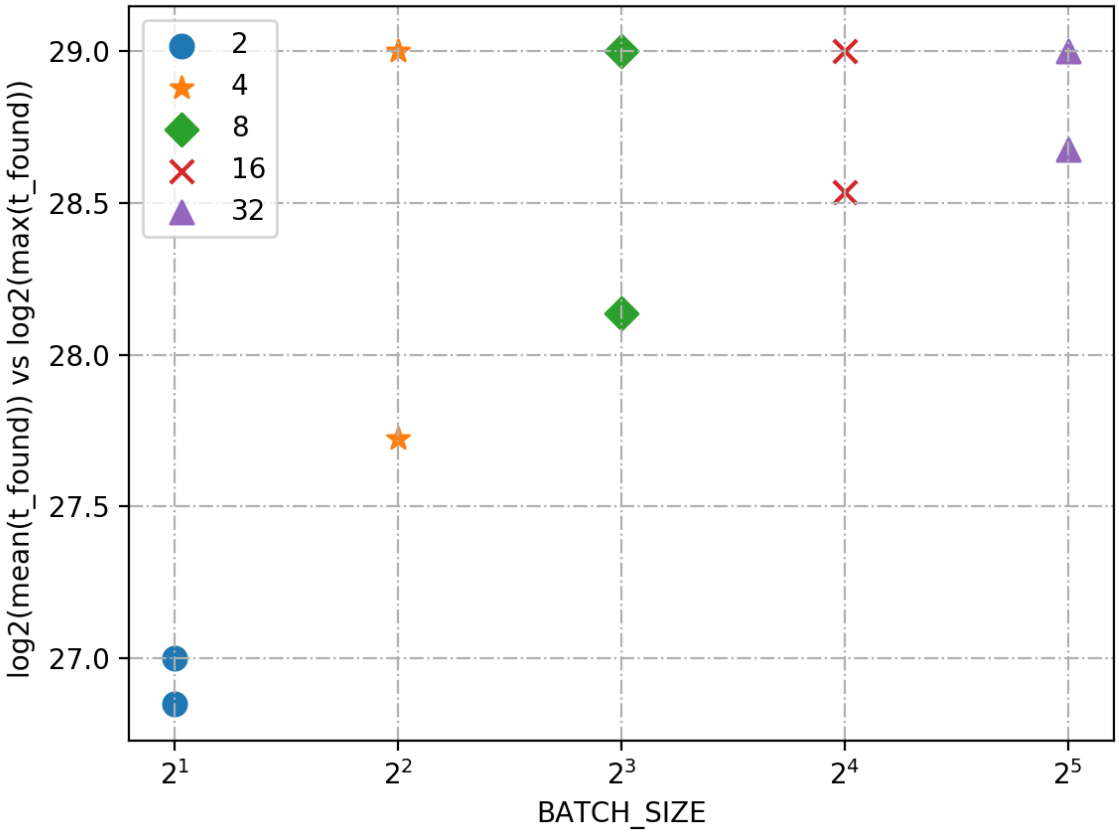
	Bit-length of q		
n	**128-bit security**	**192-bit security**	**256-bit security**
1024	27	19	14
2048	54	37	29
4096	109	75	58
8192	218	152	118
16384	438	300	237
32768	881	600	476

At present the following algorithms are covered:

- meet-in-the-middle exhaustive search
- Coded-BKW
- dual-lattice attack and small/sparse secret variant
- lattice-reduction + enumeration
- primal attack via uSVP
- Arora-Ge algorithm using Gröbner bases

Heuristic Binary Search (Follow-up 1)





Input: Model Φ, sorted list t_{list}, t_{min} and t_{max}, secuity level λ,q_0, number of K images to test
Output: Plain modulus $\tilde{t}$ or NOT_FOUND_T

```
1:  function SEARCH_T(Φ, t_list, t_min, t_max, λ, q_0, K)
2:      if t_max − t_min = 1 then                                   ▷ Base Case
3:          test ← TEST_PLAIN_MOD(Φ, t_list[t_min], λ, q_0, K)
4:          if test=SUCCESS then
5:              return t_list[t_min]
6:          end if
7:          if test=OUT_OF_BUDGET then
8:              return NOT_FOUND_T
9:          end if
10:         if TEST_PLAIN_MOD(Φ, t_list[t_max], λ, q_0, K)=SUCCESS then
11:             return t_list[t_max]
12:         end if
13:         return NOT_FOUND_T
14:     end if
15:     t_index ← t_min + (t_max − t_min)/2
16:     test ← TEST_PLAIN_MOD(Φ, t_list[t_index], λ, q_0, K)
17:     if test=SUCCESS or test=OUT_OF_BUDGET then                  ▷ Go Left
18:         t_smaller ← SEARCH_T(Φ, t_list, t_min, t_max − 1, λ, q_0, K)
19:         if t_smaller > 0 then                                   ▷ smaller p_mod found
20:             return t_smaller
21:         end if
22:         if test=SUCCESS then                                    ▷ smaller p_mod not found, prev ok
23:             return t_list[t_index]
24:         end if
25:         return NOT_FOUND_T                                      ▷ smaller p_mod not found, prev not ok
26:     end if
27:     if t_index ≥ t_max then                                     ▷ p_mod needed ∉ T_range
28:         return NOT_FOUND_T
29:     end if
30:     return SEARCH_T(Φ, t_list, t_min + 1, t_max, λ, q_0, K)     ▷ Go Right
```

Test Plain Modulus

global variables
  Y ▷ predicions given by model Φ
  D ▷ dataset
end global variables
Input: Model Φ, four integers plain modulus to test t_{test}, secuity level λ, upper bound for the coefficient modulus q_0, K images to test
Output: SUCCESS, OUT_OF_BUDGET or MISPREDICTED
function TEST_PLAIN_MOD(Φ, t_{test}, λ, q_0, K)
  $t \leftarrow t_{test}$ ▷ set encryption parameters
  $q \leftarrow q_0$
  $n \leftarrow \lambda(q_0)$
  $sk \leftarrow$ GEN_SEC_KEY(n, q, t)
  $pk \leftarrow$ GEN_PUB_KEY(sk)
  $\tilde{\Phi} \leftarrow$ ENCODE(Φ, n, t) ▷ Transform each θ_i in $\tilde{\theta}_i$
  for k **in** K **do**
    $Enc(k) \leftarrow$ ENCRYPT(ENCODE(k, n, t), pk)
    try
      $Enc(y_k) \leftarrow \tilde{\Phi}(Enc(k))$ ▷ Forward
    catch Out_Of_Budget_Exception
      return OUT_OF_BUDGET
    end try
    $\tilde{y_k} \leftarrow$ DECRYPT($Enc(y_k)$, sk)
    if $\tilde{y_k} \neq y_k$ **then**
      return MISPREDICTED
    end if
  end for
  return SUCCESS
end function

Case Study 2: 6-Layers CNN

Encoded CNN	Threads' #	Time(s)	NB
Convolution $(5 \times 5 \times 32)$	32	3.35	28
Average Pooling $(2 \times 2 \times 1)$	1	1.22	23
Convolution $(5 \times 5 \times 64)$	64	23.88	20
Average Pooling $(2 \times 2 \times 1)$	1	0.39	12
Decryption+Encryption	1	1.77	7
Fully Connected $(1024, 512)$	42	4.34	28
Fully Connected $(512, 10)$	42	0.62	20

Binary Search		
	Partial $t = 2^{16}$	**Full** $t = 2^{18}$
ϵ_Δ	15/10000	0
$Time_{FW}$	35.58	35.55
NB	28	26

Accuracy 6-layers CNN	
Starting $\check{\Phi}$	**Encoded** $\tilde{\Phi}_{n=2048,q,t=2^{16}}$
90%	89.85%