Privacy-Preserving Machine Learning With Fully Homomorphic Encryption For Deep Neural Network

J. -W. Lee et al., "Privacy-Preserving Machine Learning With Fully Homomorphic Encryption for Deep Neural Network," in IEEE Access, vol. 10, pp. 30039-30054, 2022.

Introduction

Introduction

❖ PPML (Privacy-Preserving Machine Learning)

- **완전 동형 암호화(FHE, Fully homomorphic encryption)**는 암호화 상태에서도 연산이 가능하여 **개인정보 보호를 위한 머신러닝(PPML)**에 적합
- 기존 PPML 모델은 단순한 모델과 데이터셋에 한정되어 실제 데이터셋에는 비효율적

❖ 기존 한계점

- 비선형 활성화 함수(ReLU, Softmax 등)를 **단순한 다항식 함수로 대체**하여 정확도 제한
- **Bootstrapping**을 적용하지 않아 심층 신경망이나 다층 모델 구현이 어려움

Introduction

❖ 연구 기여

- **ResNet-20 모델을 RNS-CKKS 스킴 기반으로 구현**
  - Minimax 근사 방식으로 ReLU 및 Softmax 함수의 높은 정밀도 연산
  - Bootstrapping 기술을 사용하여 암호화된 데이터에서도 심층 신경망 평가 가능

- **결과**
  - CIFAR-10 데이터셋을 활용하여 모델 검증
    - 암호화된 CIFAR-10 데이터셋에서 **92.43%±2.65%** 정확도 달성
      - 원본 ResNet-20 CNN 모델(91.89%)과 매우 유사한 성능

⇒ FHE를 심층 PPML 모델에 성공적으로 적용하여 가능성 입증

Related Works

Limitations of Related Works s

❖ 1) Limitation of HE-Friendly Network

- 기존 머신러닝 모델을 HE(동형 암호) 스킴과 호환되도록 다시 설계하는 방식을 시도
  - CIFAR-10 데이터셋에서 91.5%의 정확도

➔ 단순한 데이터셋에만 효과적이며, 고급 데이터셋에서는 높은 정확도를 달성하기 어려움.

Limitations of Related Works s

❖ 1) Limitation of HE-Friendly Network

- 기존 머신러닝 모델을 HE(동형 암호) 스킴과 호환되도록 다시 설계하는 방식을 시도
  - CIFAR-10 데이터셋에서 91.5%의 정확도

➔ 단순한 데이터셋에만 효과적이며, 고급 데이터셋에서는 높은 정확도를 달성하기 어려움.

❖ 2) Limitation of Hybrid Model : FHE + MPC

- 다자간 연산(MPC)을 통해 비선형적 활성화 함수를 계산하는 접근 방식은 프라이버시를 보장하면서도 정확도를 확보할 수 있음.
  - 하지만 클라이언트가 활성화 함수 정보를 알아야 하며, 이는 모델 정보 유출 위험을 증가시킴.
- 클라이언트가 연산에 직접 관여해야 하므로 실용성 측면에서 한계가 있음.

Limitations of Related Works s

❖ 1) Limitation of HE-Friendly Network

- 기존 머신러닝 모델을 HE(동형 암호) 스킴과 호환되도록 다시 설계하는 방식을 시도
  - CIFAR-10 데이터셋에서 91.5%의 정확도

  ➔ 단순한 데이터셋에만 효과적이며, 고급 데이터셋에서는 높은 정확도를 달성하기 어려움.

❖ 2) Limitation of Hybrid Model : FHE + MPC

- 다자간 연산(MPC)을 통해 비선형적 활성화 함수를 계산하는 접근 방식은 프라이버시를 보장하면서도 정확도를 확보할 수 있음.
  - 하지만 클라이언트가 활성화 함수 정보를 알아야 하며, 이는 모델 정보 유출 위험을 증가시킴.
- 클라이언트가 연산에 직접 관여해야 하므로 실용성 측면에서 한계가 있음.

❖ 3) Limitation of Leveled Homomorphic Encryption

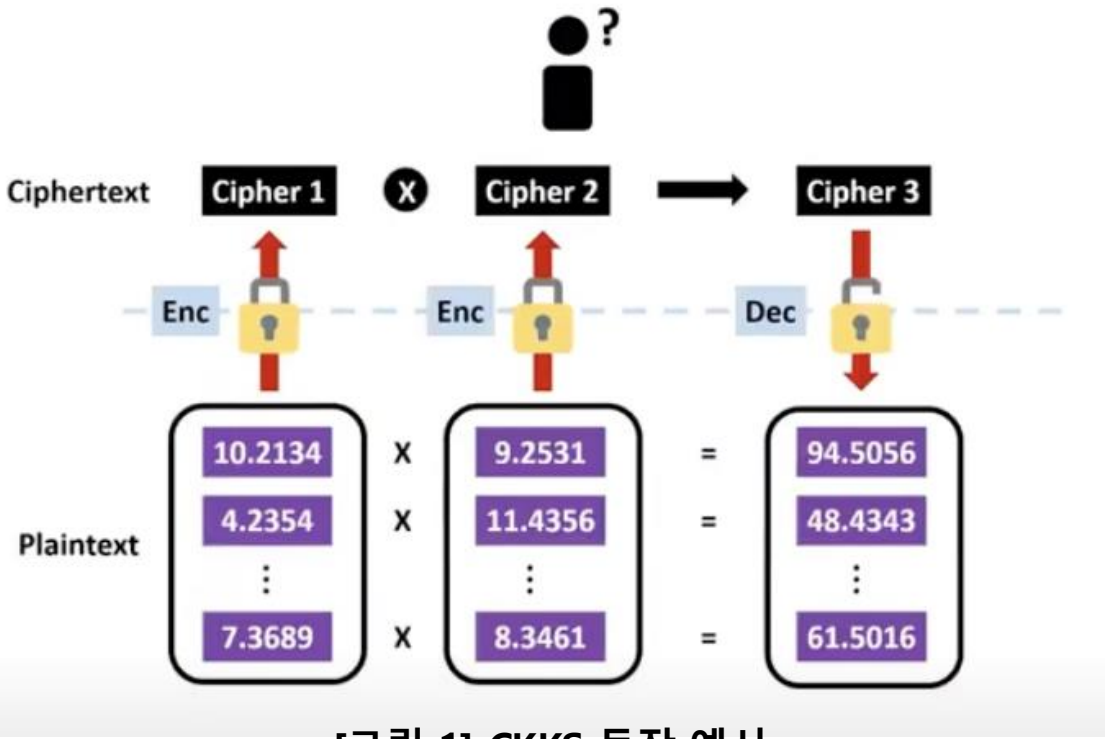
- 심층 신경망을 처리하기 위해 Leveled 동형암호를 사용하면, 매우 큰 매개변수를 설정해야 하므로 계산이 복잡해짐.
- 레벨형 암호화는 **회로 깊이**에 따라 실행 시간이 증가하여 대규모 심층 신경망에 비효율적.
  - 반면, **FHE** 스킴은 **Bootstrapping**을 사용하여 무제한 연산이 가능하므로, 복잡한 모델에서도 안정적인 연산이 가능

CKKS

CKKS

❖ CKKS Scheme

- **암호화된 데이터에서 실수 및 복소수에 대한 산술 연산(+.×)을 지원하는 FHE 스킴**
  - 암호화된 데이터 연산 시 scaling factor를 사용하여 데이터 크기를 조정하고, 연산 후 Rescaling을 통해 원래 크기로 복원

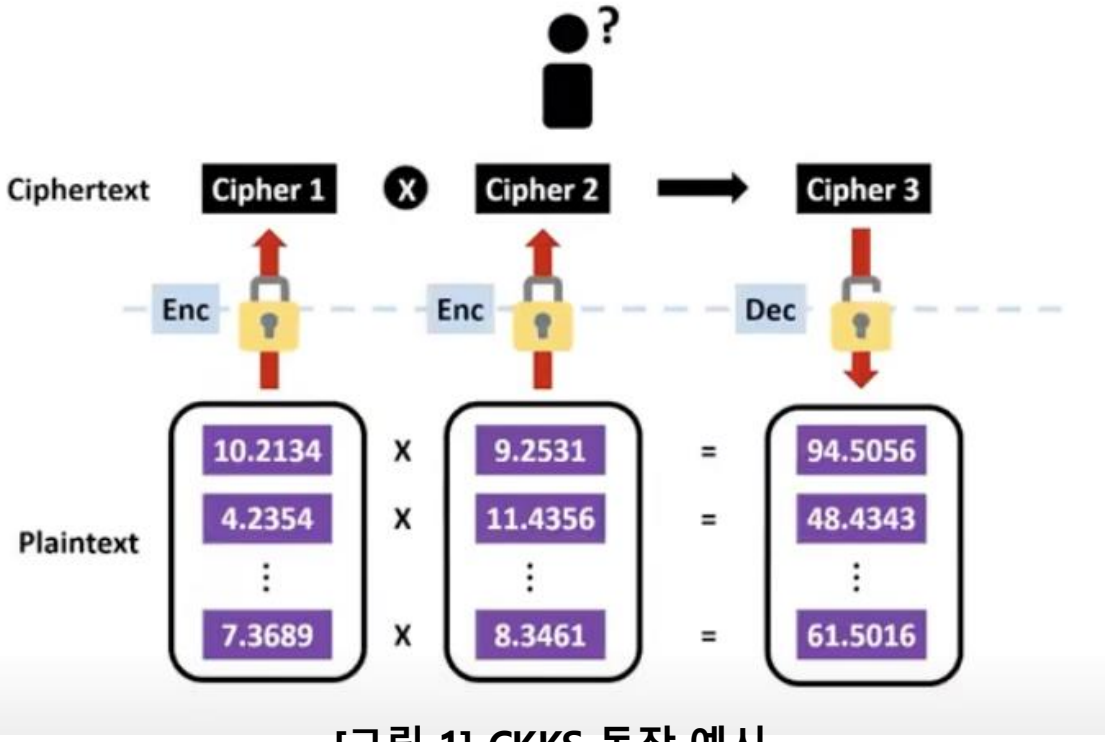


[그림 1] CKKS 동작 예시

CKKS

❖ CKKS Scheme

- **암호화된 데이터에서 실수 및 복소수에 대한 산술 연산(+.×)을 지원하는 FHE 스킴**
  - 암호화된 데이터 연산 시 scaling factor를 사용하여 데이터 크기를 조정하고, 연산 후 Rescaling을 통해 원래 크기로 복원



[그림 1] CKKS 동작 예시

$$Scaling\ Factor\ \Delta = 10^4$$

$$E(c') = E(a' \times b') = E(31416 \times 27183) = E(854074728)\ a' = 3.1416 \times 10^4, \quad b' = 2.7183 \times 10^4$$

RNS-CKKS

❖ RNS-CKKS Scheme

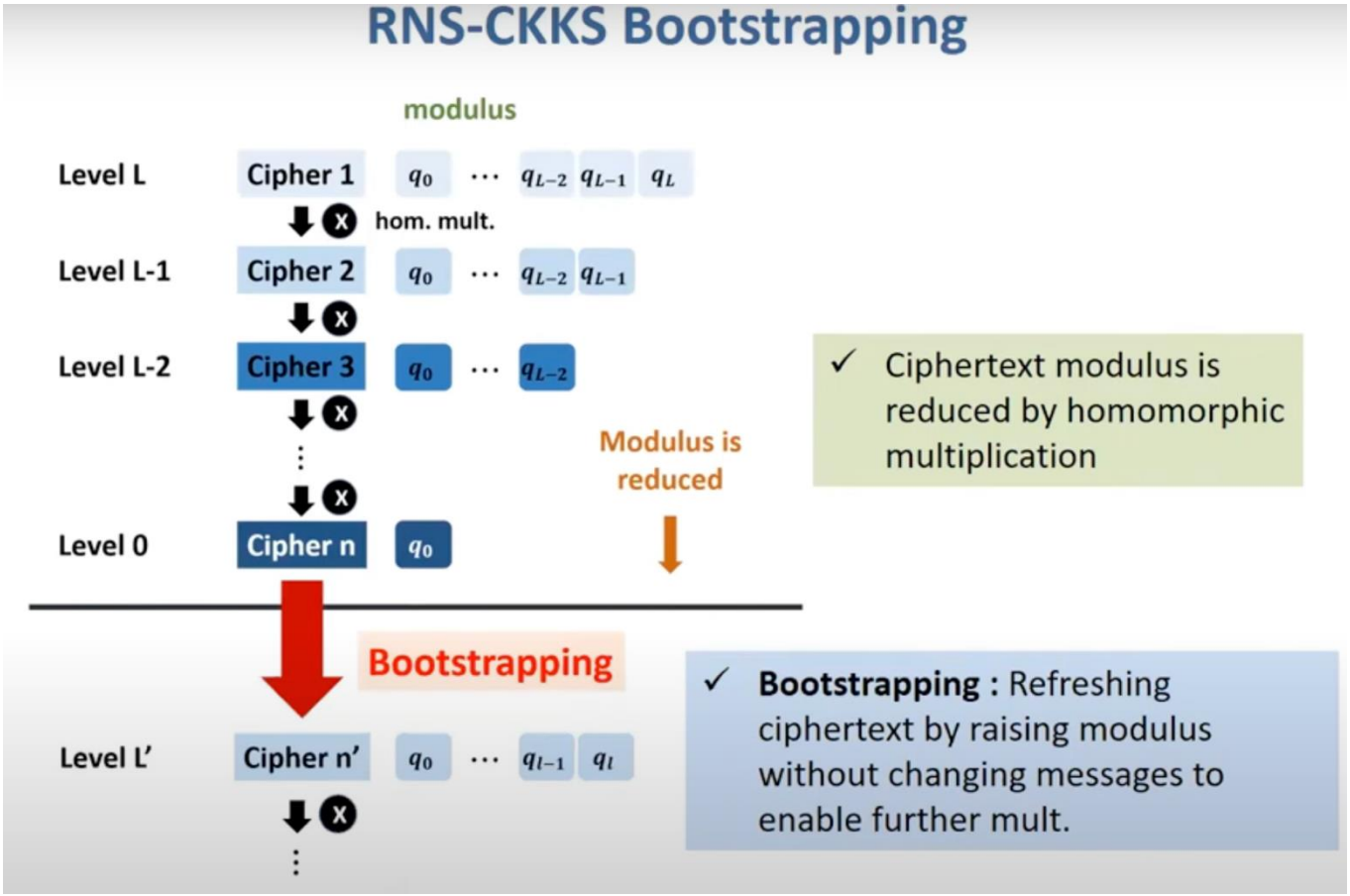
- CKKS 스킴의 높은 계산 복잡성을 해결하기 위해 **RNS(Residue Number System)** 도입
- RNS는 큰 정수를 **작은 정수로 분할**해 연산을 수행하여 연산 속도를 향상시킴

$$31416 \; mod \; 257 \; = \; 41 \qquad 27183 \; mod \; 257 \; = \; 66$$
$$31416 \; mod \; 263 \; = \; 64 \qquad 27183 \; mod \; 263 \; = \; 72$$

$$(64 \; \times \; 72) \; mod \; 263 \; = 187$$

Bootstrapping

❖ Bootstrapping의 필요성



[그림 2] RNS-CKKS Bootstrapping

Polynomial Optimization

Baby-Step Giant-Step Polynomial Evaluation

❖ Baby-Step Giant-Step

- **고차 다항식 평가를 효율적으로 수행하기 위한 기법**
- **필요성**
  - 동형암호를 활용하기 위해 ResNet의 비선형 연산 및 Bootstrapping을 고차 다항식으로 근사
  - RNS-CKKS 스킴에서 암호화된 입력에 대해 다항식을 평가할 때, 비스칼라 곱셈 수와 연산 깊이를 줄여야 함

Baby-Step Giant-Step Polynomial Evaluation

❖ Baby-Step Giant-Step

- 고차 다항식 평가를 효율적으로 수행하기 위한 기법
- 필요성
  - 동형암호를 활용하기 위해 ResNet의 비선형 연산 및 Bootstrapping을 고차 다항식으로 근사
  - RNS-CKKS 스킴에서 암호화된 입력에 대해 다항식을 평가할 때, 비스칼라 곱셈 수와 연산 깊이를 줄여야 함

❖ 이진 트리 기반 다항식 평가

- **DividePoly 알고리즘을 사용해 다항식을 재귀적으로 분할하고, 결과를 이진 트리 형태로 변환**
- **필요성**
  - Baby-Step Giant-Step 알고리즘을 이진 트리 구조로 수정하여, 더 직관적이고 체계적인 구현을 가능하게 함

Baby-Step Giant-Step Polynomial Evaluation

Algorithm 1: `DividePoly(p; k)`

Input : A degree-d polynomial p, a giant step parameter k
Output: A binary tree P with leaf having polynomials

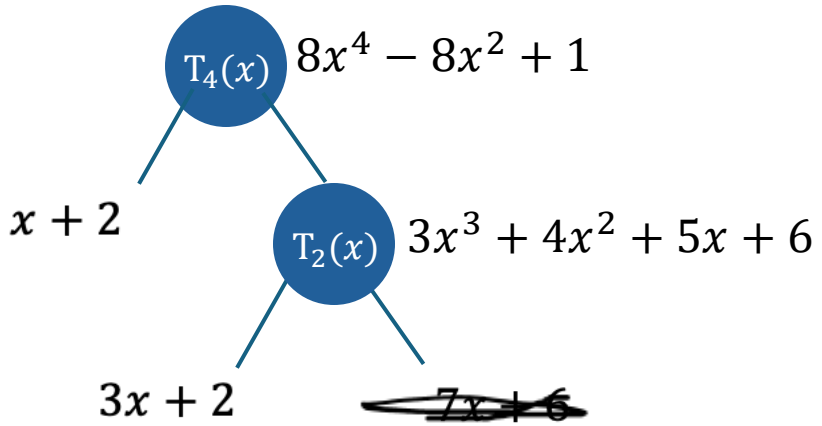
1 **if** $d < k$ **then**
2 | **return** a binary tree P with a single root node having p
3 **else**
4 | Find m such that $k \cdot 2^{m-1} < d \leq k \cdot 2^m$.
5 | Generate a binary tree P with a single root node having $T_{k \cdot 2^{m-1}}$.
6 | Divide p by $T_{k \cdot 2^{m-1}}$ to obtain the quotient q and the remainder r.
7 | Generate a binary tree Q using `DividePoly`$(q; k)$.
8 | Generate a binary tree R using `DividePoly`$(r; k)$.
9 | Append Q, R to the left child and the right child of the root in P, respectively.
10 | **return** P
11 **end**

❖ DivedePoly(p(x), 2)

- 다항식 $p(x) = x^5 + 2x^4 + 3x^3 + 4x^2 + 5x + 6$를 $k = 2$ 로 분할
- $p(x)$를 $T_4(x)$ 로 나눔

(몫) $q_1(x) = x + 2$ $q_2(x) = 3x + 2$

(나머지) $r_1(x) = 3x^3 + 4x^2 + 5x + 6$ $r_2(x) = 7x + 6$

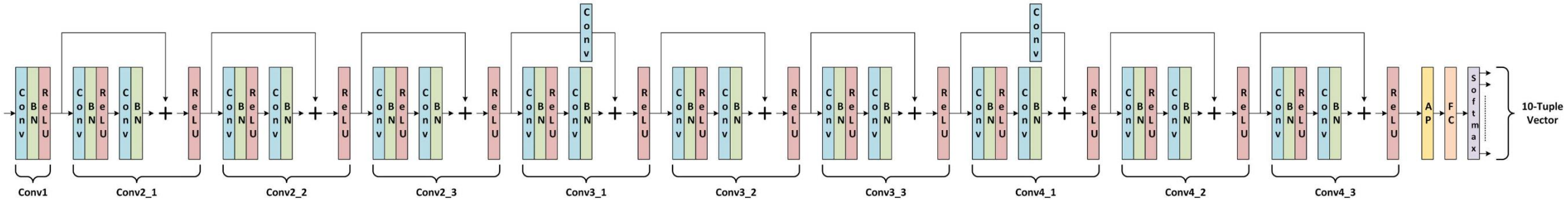


ResNet-20

ResNet-20

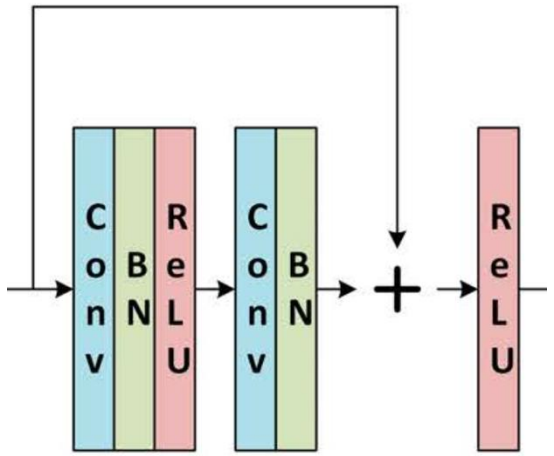
❖ ResNet-20 (Residual Network)

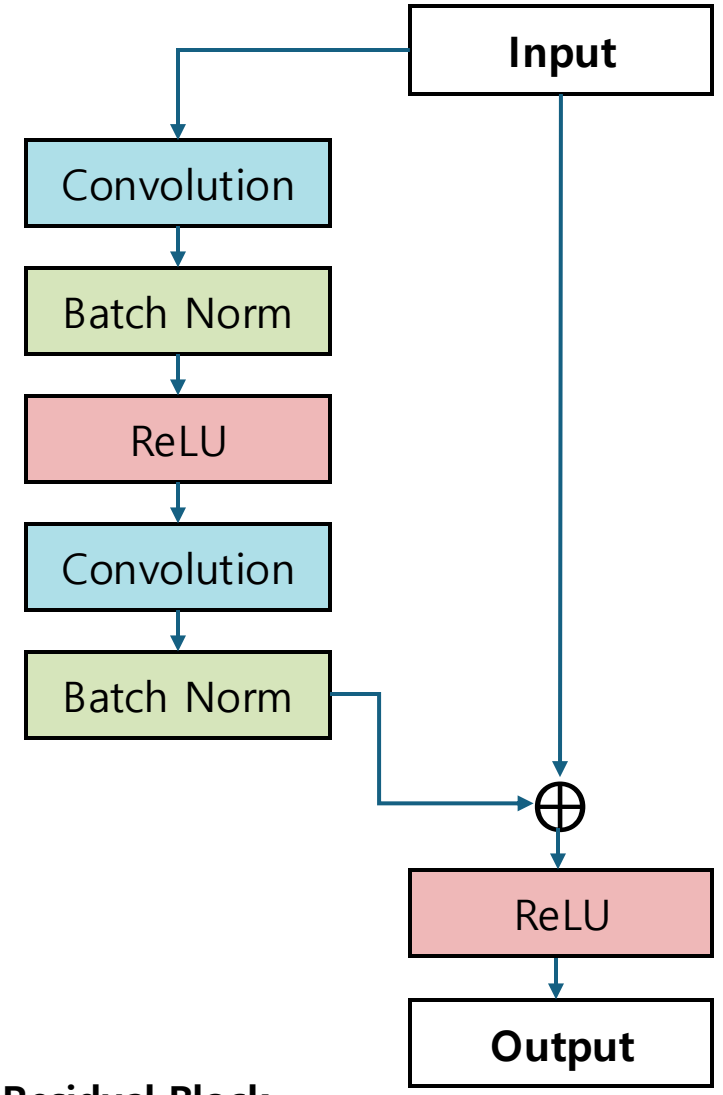
- 깊은 신경망에서 학습이 어려운 문제(**Gradient Vanishing**)를 해결하기 위해 설계된 구조
  - 기울기 소실(Gradient Vanishing) 문제: 신경망이 깊어질수록 역전파(Backpropagation) 과정에서 기울기(Gradient)가 점점 작아져 가중치가 거의 업데이트되지 않아 학습이 제대로 이루어지지 않는 현상
- 입력 값을 출력에 더하는 **Residual Connection** 사용
- 본 연구는 암호화된 데이터에서 동형 계산을 수행하기 위해 **RNS-CKKS 스킴 기반으로 ResNet-20를 재설계**



[그림 4] ResNet-20 모델 구조

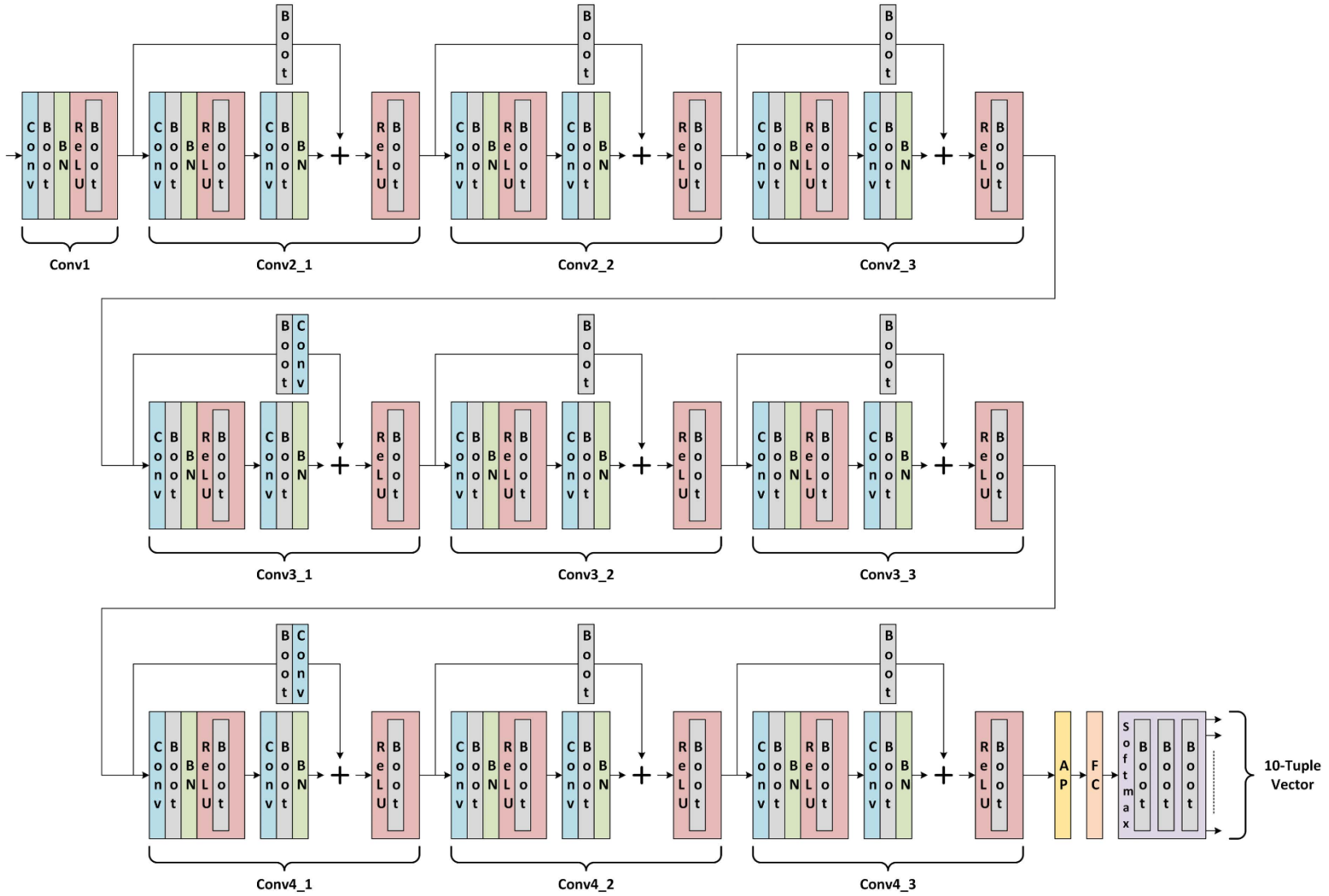
ResNet-20





[그림 5] ResNet-20의 Residual Block

Implementation ResNet-20 on RNS-CKKS



[그림 6] RNS-CKKS 기반 ResNet-20 모델 구조

Implementation ResNet-20 on RNS-CKKS

❖ Data Packing

- **필요성**: CIFAR-10 이미지 데이터를 RNS-CKKS를 적용한 ResNet-20 모델에서 처리하기 위해 데이터를 효율적으로 **암호화**해야 함
- **Sparse Packing**
  - CIFAR-10 이미지의 하나의 채널을 암호문에 $\mathbf{2^{10}}$**개의 슬롯**만을 사용하여 저장 (*cf. 설정 파라미터:* 2^{16})

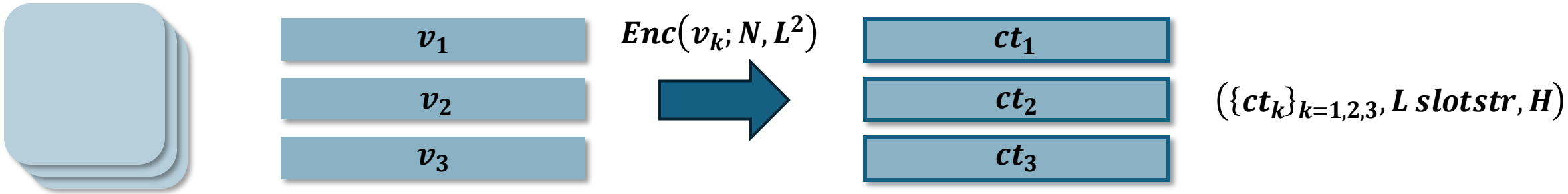
    ➜ Bootstrapping 시간 단축, Convolution 연산에서 rotation 비용 감소

Implementation ResNet-20 on RNS-CKKS

❖ Data Packing

- **필요성**: CIFAR-10 이미지 데이터를 RNS-CKKS를 적용한 ResNet-20 모델에서 처리하기 위해 데이터를 효율적으로 **암호화**해야 함
- **Sparse Packing**
  - CIFAR-10 이미지의 하나의 채널을 암호문에 $\mathbf{2^{10}}$**개의 슬롯**만을 사용하여 저장 (*cf. 설정 파라미터:* 2^{16})

    ➔ Bootstrapping 시간 단축, Convolution 연산에서 rotation 비용 감소
- **암호화된 텐서 구조**
  - 형식: $(\{ct_k\}_k, \ell, slotstr, h)$



[그림 7] EncTensor 알고리즘 시각화

Implementation ResNet-20 on RNS-CKKS

❖ Data Range and Precision

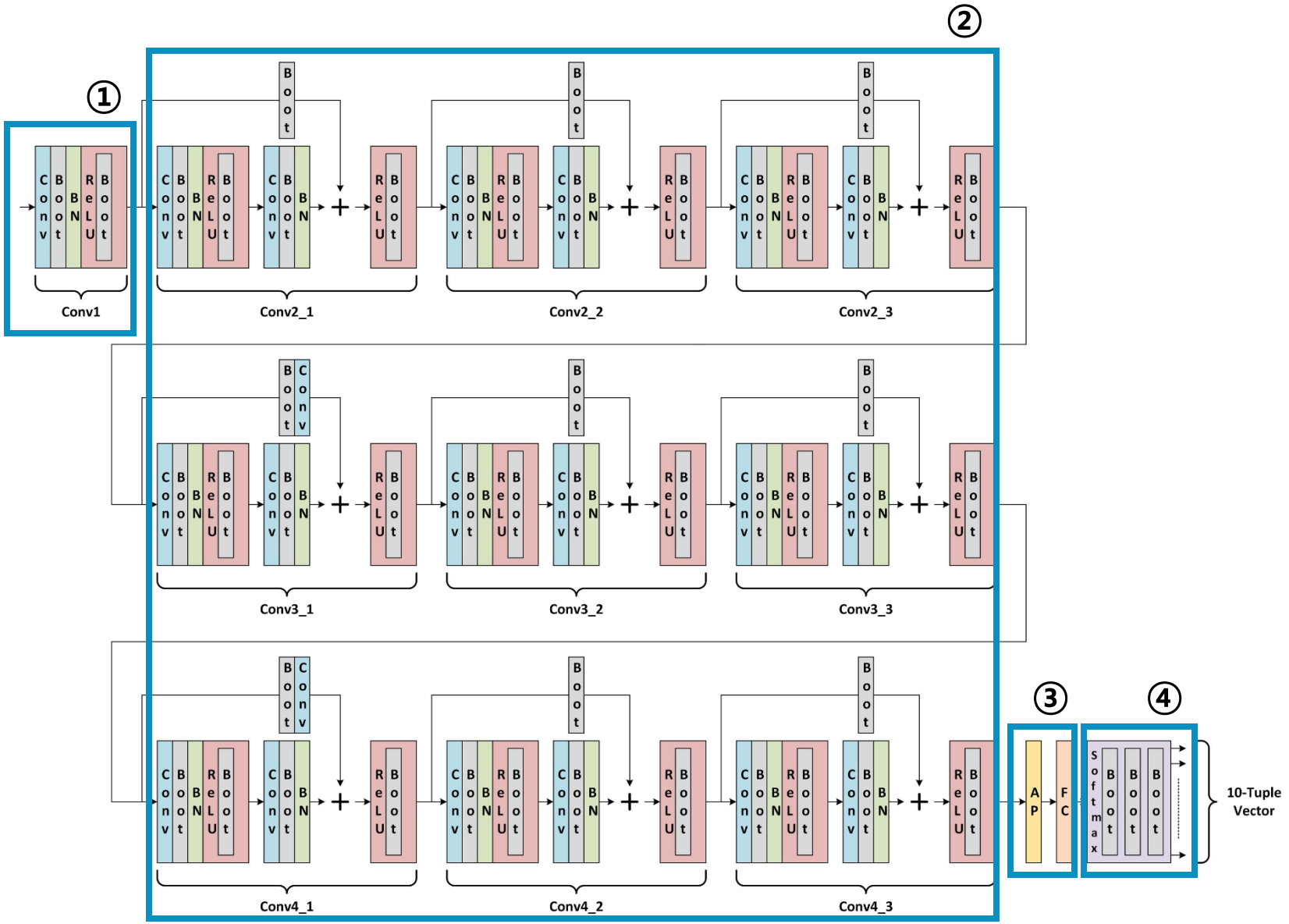
- **입력 범위 제한**
  - FHE는 활성화 함수를 처리할 때 비선형 연산을 직접 할 수 없고, 다항식으로 근사해야 함

    ➔ 이때 입력 데이터 값이 **특정 범위를 초과**하면 **계산 오류 및 학습 실패**로 이어질 수 있음
  - ReLU와 Softmax 함수의 최대 입력값이 37.1로 관찰됨

    ➔ 제한된 범위: $[-\mathbf{40}, \mathbf{40}]$

- **정밀도 설정**
  - **16비트 이하** 정밀도를 기반으로 다항식을 근사하여 발생하는 **오차를 최소화**
    - 다항식 근사 또는 함수 계산 시 소수점 이하 16비트의 정밀도가 안정성을 보장.

Implementation ResNet-20 on RNS-CKKS



[그림 6] RNS-CKKS 기반 ResNet-20 모델 구조

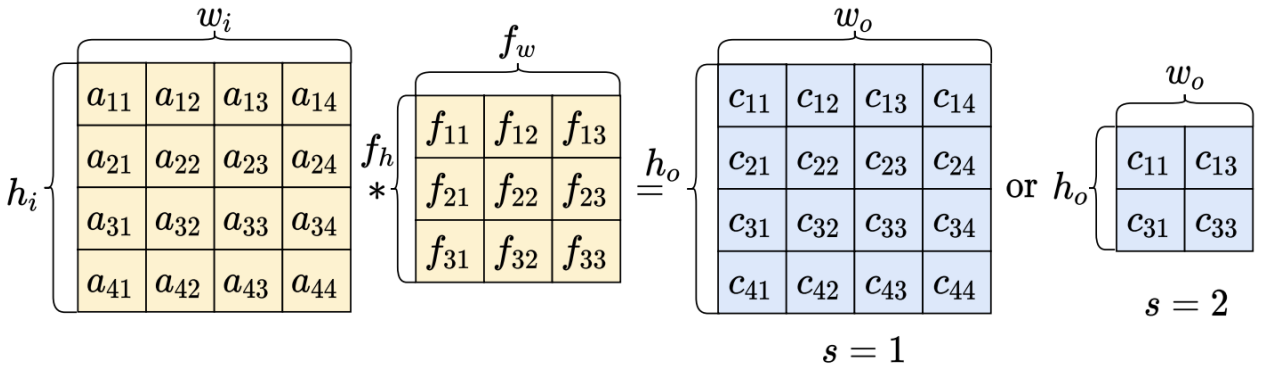
Convolution

❖ Convolution

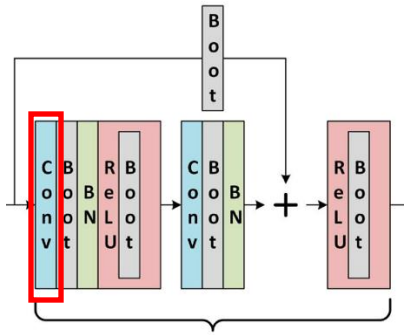
- **ResNet-20의 Convolution**
  - ResNet-20에서는 입력 크기를 유지하기 위해 **zero padding**을 사용한 **SISO (Single-Input Single-Output)** Convolution 수행

- **1) Nonstrided Convolution**
  - 스트라이드 값(str) = 1

- **2) Strided Convolution**
  - 스트라이드 값(str) = 2
  - Downsampling 수행



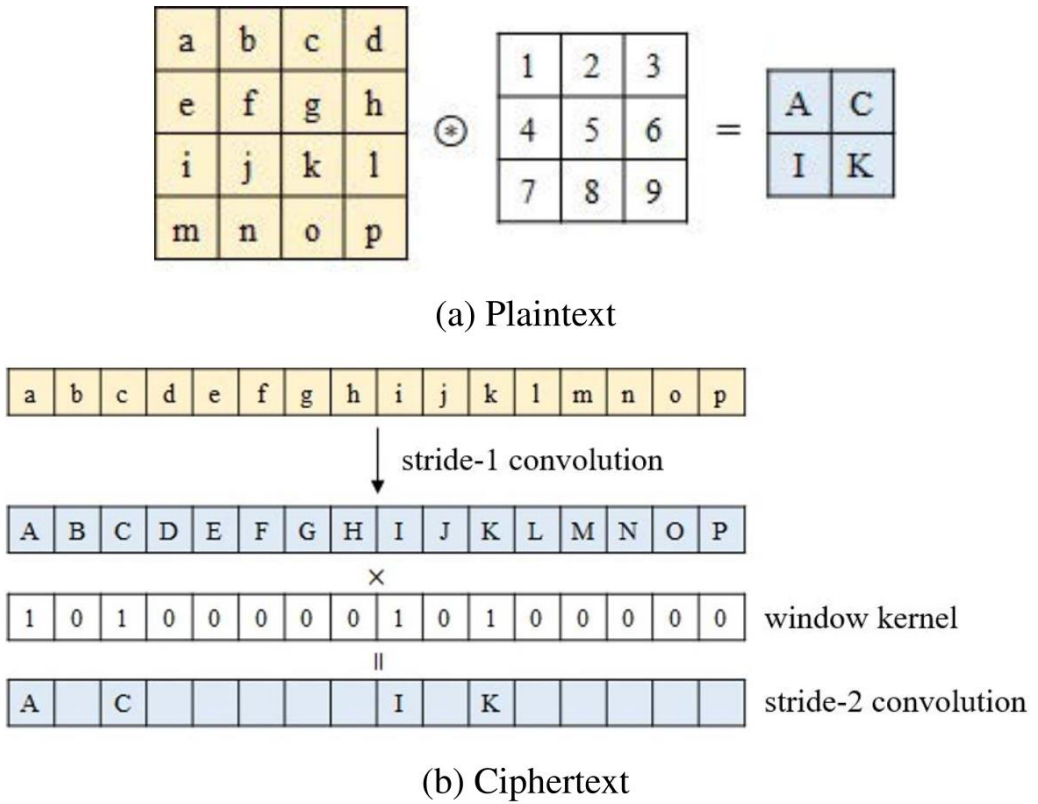
[그림 8] SISO Convolution for plaintext data



Convolution

❖ RNS-CKKS 기반 Convolution 최적화 방법

- **Strided Convolution을 Nonstrided Convolution 일부로 간주**
  - Nonstrided Convolution 수행 후 **윈도우 커널**을 사용해 **필요한 데이터만 남김**



[그림 9] Stride-2 convolution.

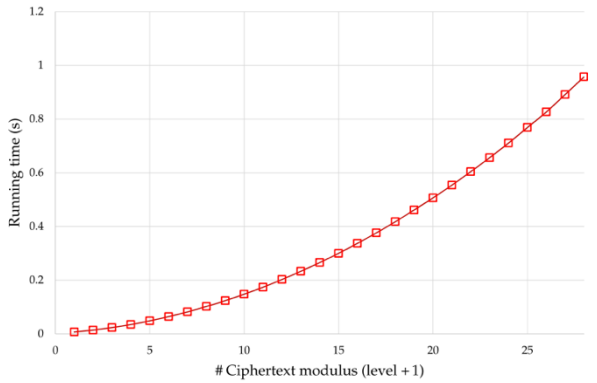
Bootstrapping Position

❖ 최적의 bootstrapping 위치

- **key-switching 연산량 비교 결과**
  - **Convolution** 연산에서 key-switching 연산이 훨씬 많음.

    ➔ 회전(rotation) 연산이 많기 때문.

- **제안된 최적 위치: Convolution 연산 직후에 bootstrapping 수행**



(a) T_{rot}-$(\ell + 1)$ graph

[그림 10] 암호문 레벨에 따른 실행 시간 증가 그래프

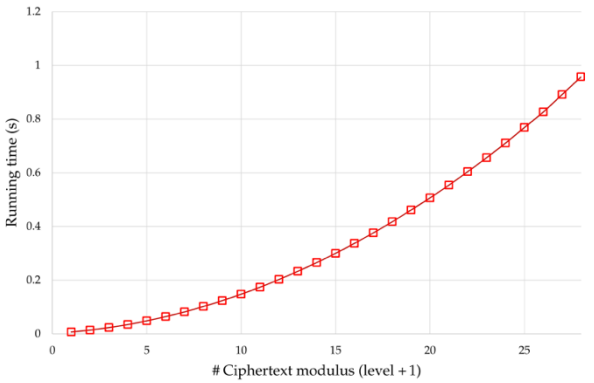
Bootstrapping Position

❖ 최적의 bootstrapping 위치

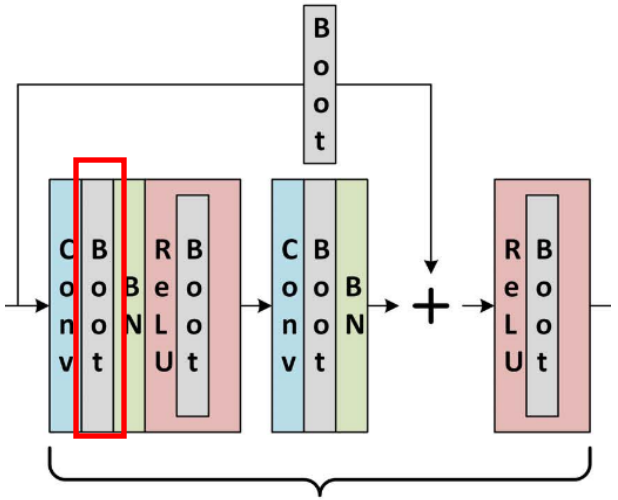
- **key-switching 연산량 비교 결과**
  - **Convolution** 연산에서 key-switching 연산이 훨씬 많음.

    ➔ 회전(rotation) 연산이 많기 때문.

- **제안된 최적 위치: Convolution 연산 직후에 bootstrapping 수행**



(a) T_{rot}-$(\ell + 1)$ graph

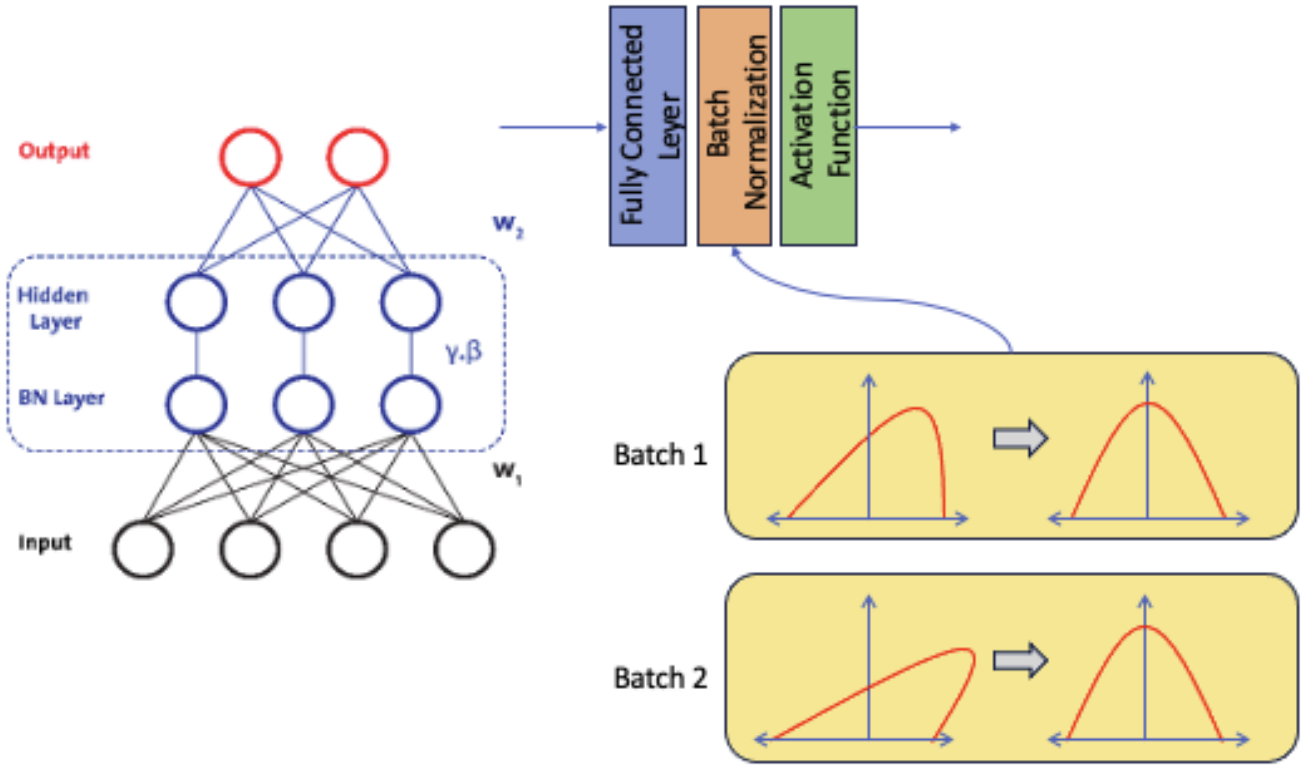


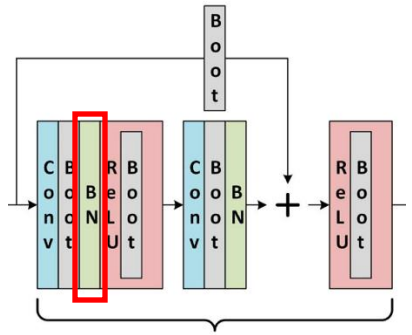
[그림 10] 암호문 레벨에 따른 실행 시간 증가 그래프

Batch Normalization

❖ Batch Normalization

- **배치 정규화 (Batch Normalization)**
  - 배치 정규화는 상수 계수를 포함하는 단순한 선형 연산으로, 동형 연산에서 쉽게 구현 가능





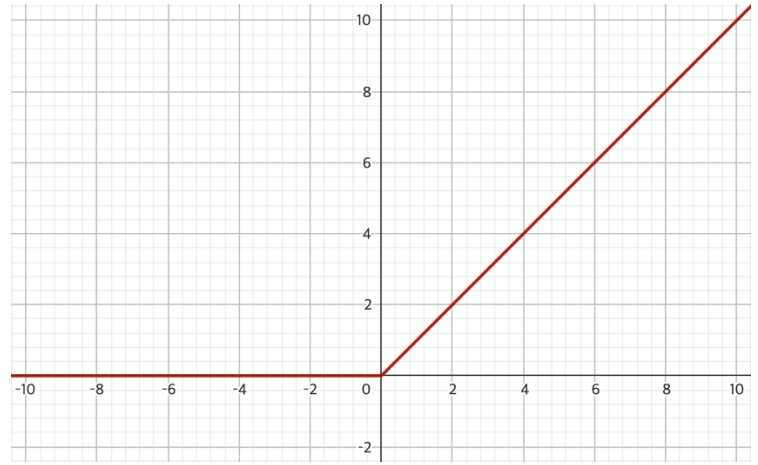
[그림 11] Batch Normalization 개념

ReLU

❖ ReLU

- **ReLU 함수 소개**
  - Rectified Linear Unit(ReLU): $\boldsymbol{ReLU(x) = max(0, x)}$

    ➔ 딥러닝 모델에서 비선형성을 추가하여 학습 성능을 개선하는 활성화 함수

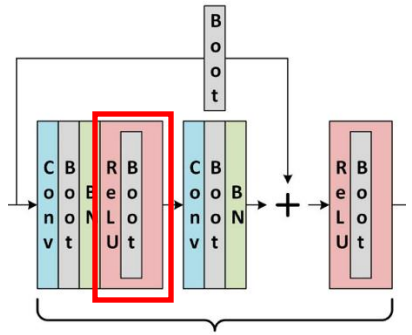


[그림 12] Minimax 합성으로 근사한 ReLU(x)

- **문제점**
  - 동형 암호 환경에서는 비산술(non-arithmetic) 연산을 직접 수행할 수 없음

    ➔ ReLU를 동형 암호 환경에서 구현하기 위해 다항식 근사 필요

- **해결책**
  - $ReLU(x) = \frac{1}{2}x\big(1 + sign(x)\big)$
    - $sign(x)$를 소규모 다항식의 Minimax 합성으로 근사



ReLU

❖ ReLU 구현 알고리즘: $ReLU(Tensorct, \{p_i\}_i)$

- **입력 데이터**
  - $Tensorct$: 암호화된 텐서 Tensort=$(ct_k, \ell, slotstr, h)$
  - $\{p_i\}_i$: $sign(x)$를 여러 구간으로 나눠, 각 구간에 대해 **근사된 다항식**
- **ReLU 적용**
  - 각 ct_k 에 대해 $sign(x)$ 를 근사하는 여러 **Minimax 다항식** p_i 를 순차적으로 적용
  - $(0.5 \odot ct_k) \otimes (1 + ct'_k)$: 근사된 ReLU 계산
- **결과**
  - 모든 채널에 ReLU를 적용한 결과가 업데이트된 텐서로 반환

Algorithm 6: ReLU(Tensorct, $\{p_i\}_i$)

Input : An encrypted tensor Tensorct $= (\{\mathsf{ct}_k\}_{k=0,\cdots,t-1}, \ell, \mathsf{slotstr}, t)$, sequence of composite polynomials for sign function $\{p_i\}_{i=0,\cdots,s-1}$

Output: An activated encrypted tensor with ReLU Tensorct′

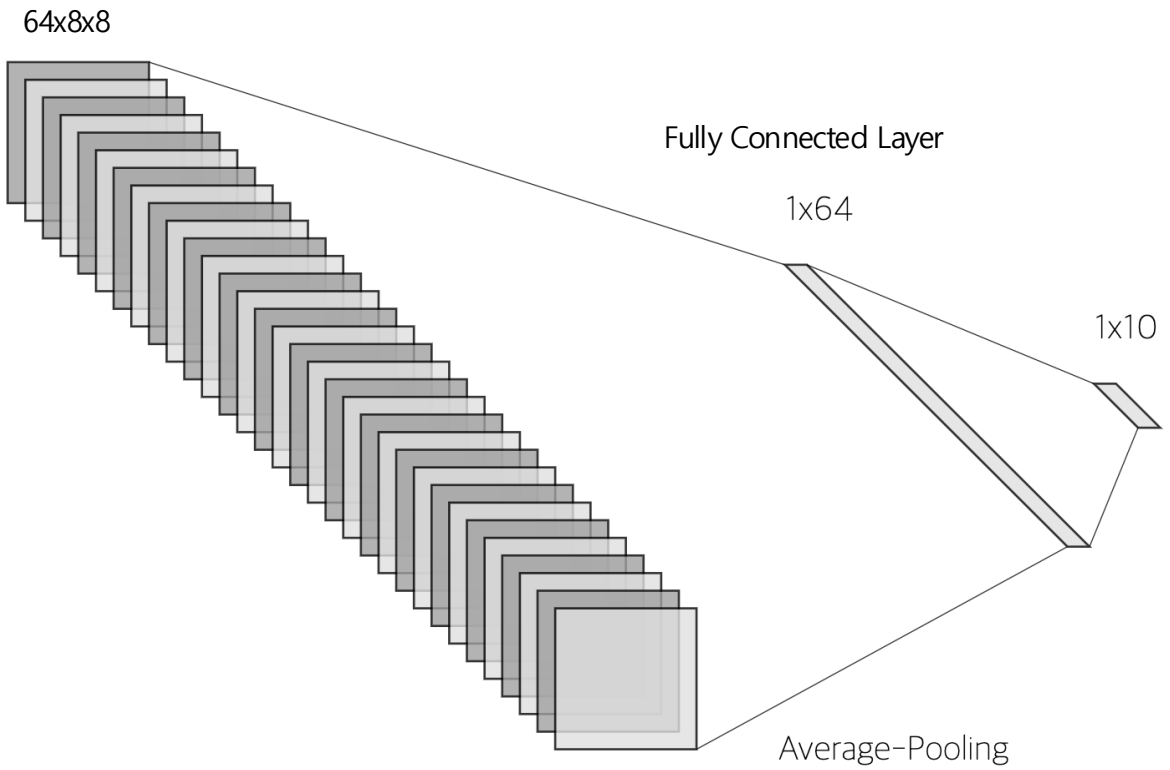
for $k = 0$ **to** $t - 1$ **do**
  $\mathsf{ct}'_k \leftarrow \mathsf{ct}_k$
  for $i = 0$ **to** $s - 1$ **do**
    $\mathsf{ct}'_k \leftarrow \mathtt{OddPolyEval}(\mathsf{ct}'_k, p_i)$
  end
  $\mathsf{ct}'_k \leftarrow (0.5 \odot \mathsf{ct}_k) \otimes (1 + \mathsf{ct}'_k)$
end

Average Pooling & Fully Connected Layer

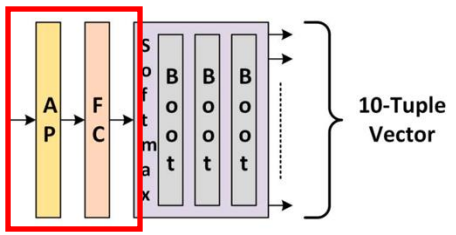
❖ Average Pooling and Fully Connected Layer

- **Average Pooling**
  - 각 채널의 평균을 계산하여 길이 64의 암호문 벡터 생성
    - 평균 계산 시 rotation을 수행하여 슬롯 값 합산

- **Fully Connected Layer**
  - 평균 풀링 결과와 가중치 행렬 W를 곱하여 최종 클래스 분류를 위한 벡터 생성
  - 결과: 길이가 10인 암호화된 벡터



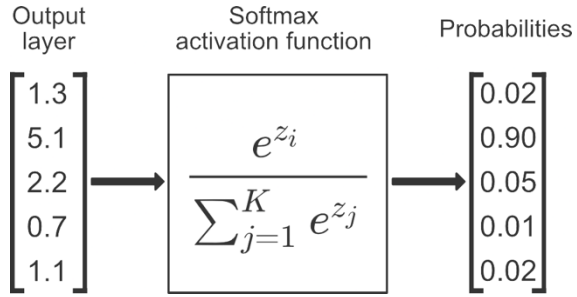
[그림 13] Average Pooling and Fully Connected Layer



Softmax

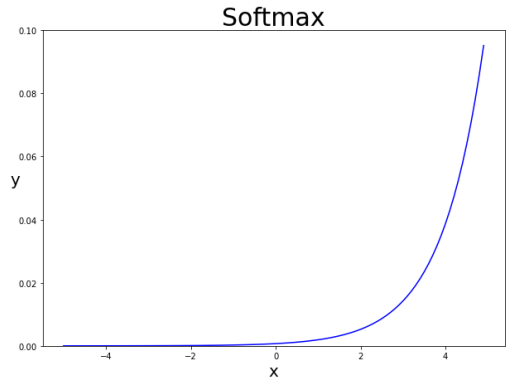
❖ Softmax

- **Softmax 함수 소개**
  - 입력 값을 클래스에 대한 확률 분포로 매핑하는 활성화 함수

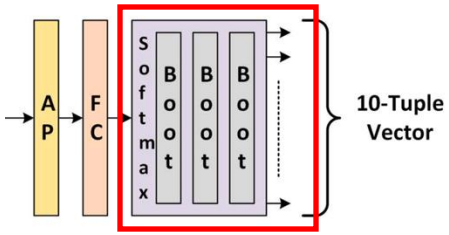


[그림 14] Softmax 함수 동작 예시

- **Softmax 구현의 어려움**
  - 지수 함수 e^{z_i}
    - 지수 함수는 입력값이 클수록 급격히 증가하기 때문에, 값이 너무 커져 암호화된 연산 범위를 초과하거나 노이즈 증폭 문제가 발생할 수 있음
  - 역함수 $\frac{1}{e^{z_i}}$
    - 입력값이 매우 작은 경우 역함수 계산 결과가 비정상적으로 커질 수 있음
      - 계산 결과의 안정성을 떨어뜨림



[그림 15] Softmax 함수



Softmax

❖ Softmax 구현 방안

1) 지수 함수 근사

- e^x를 $\left(e^{x/B}\right)^B$ 형태로 변환
- 입력 범위를 [-1, 1]로 제한 후 차수 12의 Minimax 다항식으로 근사화
  - B = 64

2) 역함수 근사

- Goldschmidt 분할 알고리즘 사용: $\frac{1}{y} \approx \prod_{i=0}^{n-1}\left(1+(1-y)^{2^i}\right),\ y = 1-x$

3) Gumbel Softmax

- λ 파라미터 추가 : $\frac{e^{x_i/\lambda}}{\sum_{j=0}^{T-1} e^{x_j/\lambda}}$
- 지수 함수 범위를 줄이고, 역함수 입력값 안정화
  - λ=4

Settings

❖ 모델 파라미터 및 학습 방법

- **테스트 데이터 사전 처리**
  - CIFAR-10 데이터셋, 1000개의 32×32 크기 RGB 이미지
- **학습 설정**
  - 학습률(Learning Rate)
    - 초기값 0.001
    - 80 epoch 이후 10배 감소, 120 epoch 이후 100 epoch까지 진행.
- **결과**
  - 분류 정확도: 학습된 모델 파라미터로 **91.89%**

Result

❖ RNS-CKKS parameter settings

- **Modulus Q : 111.6-bit 보안을 만족하는 길이로 설정**
  - Cheon et al.의 hybrid dual attack을 기준으로 보안 측정

[표 1] RNS-CKKS parameter settings

λ	Hamming Weight	Degree	Modulus Q	q_0	Special Prime	Scaling Factor	Evaluation Level	Bootstrapping Level
111.6	64	2^{16}	1450 bits	60 bits	60 bits	50 bits	11	13

Result

❖ Performance

- RNS-CKKS 스킴과 적절한 부트스트래핑 작업을 통해 ResNet-20이 성공적으로 수행될 수 있음 확인

[표 2] Classification accuracy of the ResNet-20 for plaintext and ciphertext and agreement ratio

Model	ResNet-20[1]	ResNet-20[2]	PPML ResNet-20	Agreement
Accuracy	91.89%	92.95% ± 2.56%	92.43% ± 2.65%	98.43%± 1.25%

[1] Classification accuracy verified with 10,000 images.
[2] Classification accuracy verified with 383 images which are used to test ResNet-20 on encrypted images.

Result

❖ Running time

- ResNet-20 전체와 모델 내 각 구성 요소에 대한 실행 시간 확인
- 한 이미지를 추론하는 데 약 3시간 소요

TABLE 5. The running time of the ResNet-20 and the percentage of time spent in each component relative to total time.

Layer	Conv	BN	ReLU	Boot	AP + FC	Softmax	Total time (s)
Time ratio	17.44%	13.55%	34.61%	31.55%	0.04%	2.81%	10,602

Result

❖ Running time

- 부트스트래핑이 컨볼루션 연산 이후에 수행된 경우, ReLU 이후에 수행된 경우에 비해 실행 시간이 27.8% 감소

TABLE 6. Comparison of the running time of ResNet-20 for two positions of the bootstrapping.

Bootstrapping position	After conv	After ReLU
Total Time (s)	10,602	14,694

Disscution

❖ 실행 시간 최적화 필요

- 단일 이미지를 추론하는 데 약 3시간이 소요
- 향후 연구 방향
  - GPU, FPGA, ASIC과 같은 가속기를 활용해 ResNet-20을 최적화하는 구현 방안 필요
  - RNS-CKKS의 데이터 패킹 방법을 적절히 활용하여 다수 이미지 처리 속도 개선 가능

❖ 보안 수준

- 제안된 모델의 보안 수준은 111.6비트로, 이는 보안성이 있지만 128비트 표준에는 미치지 못함
  - RNS-CKKS 파라미터 변경을 통해 보안 수준을 높일 수 있으나 실행 시간 증가 발생 가능

❖ 분류 정확도

- 암호화된 ResNet-20의 분류 정확도는 원본 ResNet-20과 거의 동일한 수준
  - ResNet-20을 암호화된 데이터에 대해 FHE 기반으로 성공적으로 구현

Conclusion

❖ 결론

- **기여**
  - RNS-CKKS 기반 FHE를 ResNet-20에 최초로 적용
  - ReLU, 부트스트래핑, Softmax 근사 기법 사용
- **주요 성과**
  - 원본 ResNet-20과 동일한 정확도 달성
  - 기존 PPML 모델 대비 최고 정확도 기록
- **의의**
  - FHE 기술의 재학습 없이 첨단 모델 적용 가능성 제시(수정하기)

End

End