SHREC MEM SURG
SUMMER UNDERGRADUATE RESEARCH GROUP

SafetyNets: Privacy-Preserving Machine Learning as a Service

Carlos Vazquez, Sunwoong Kim, Donald Kline, Alan George, Alex Jones

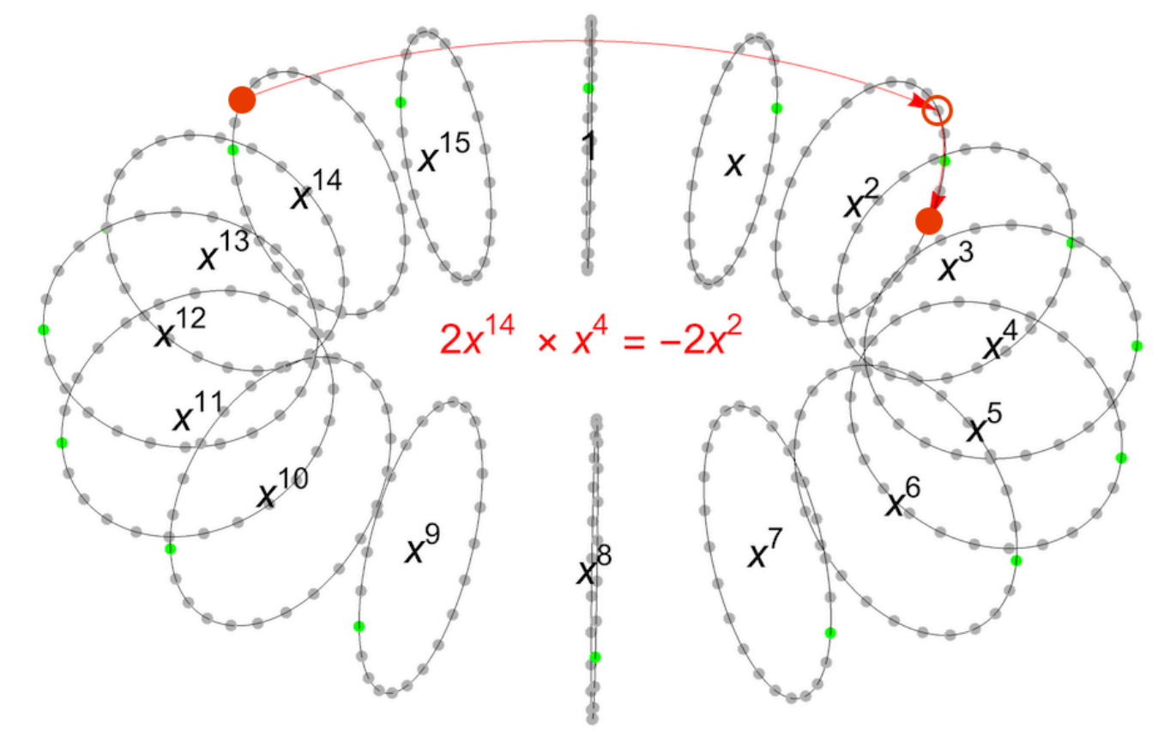
Mission-Critical Computing
NSF CENTER FOR SPACE, HIGH-PERFORMANCE, AND RESILIENT COMPUTING (SHREC)

PITT SWANSON ENGINEERING
ELECTRICAL & COMPUTER

Motivation & Background

The advent of cloud computing has created a demand for schemes to protect a client's data and a company's algorithms. For instance, suppose that a startup company were developing a learning algorithm that determines a person's risk of developing a certain disease given a number of medical and genetic factors. Before the 2009 discovery of additions and multiplication operations on encrypted data, this scenario seemed impossible due to confidentiality laws. This poster explores the option of homomorphically evaluating neural networks on encrypted data, granting the startup "access" to data it couldn't legally obtain before.

The Space of Ciphertexts

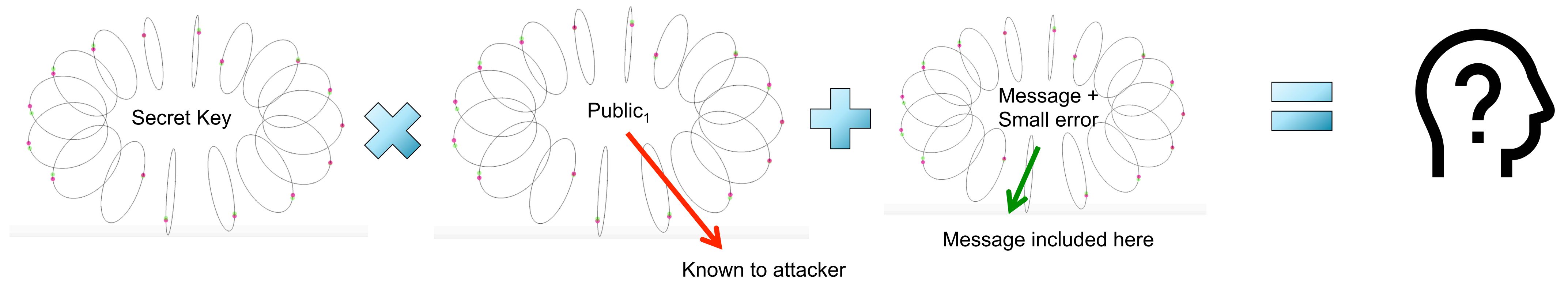


Ciphertexts are polynomials with the following properties:

1. Terms which exceed a maximum degree (usually ~2^{12}) "wrap around" back to degree 0, and are scaled by -1
2. Coefficients which exceed a maximum K wrap around back to 0

The "Ring Learning With Errors" Assumption

This assumption states that given a fixed secret key, and a Gaussian distribution of small error polynomials, even an unlimited number of the following expressions cannot reveal to an attacker the secret key.



Packing Many Integers into a Single Ciphertext

The degree wrap-around feature of ciphertext polynomials not only obscures the secret key via a "Rubik's Torus" action, but it also makes it possible to pack a vector into a ciphertext. In particular, the wrap-around is equivalent to always reducing a ciphertext modulo $x^n + 1$ (where n is the wrap-around degree) after operations. And since

$$x^n + 1 = (x - \zeta)(x - \zeta^3)\dots(x - \zeta^{2n-1}) \pmod{t},$$

where t is the wrap-around coefficient, and ζ is the number which, when raised to the 2nth power, is congruent to 1 modulo t. Then, the Chinese Remained Theorem (CRT) states that for any vector (of length n) of integers less than t, there exists a polynomial p(x) such that

$$p(x) \text{ (modulo } x-\zeta^{1+2i}) = \text{vector}[i]$$

The amazing thing about CRT batching is that given two vector-ciphertexts $p_1(x)$ and $p_2(x)$, their sum and products are the encryptions of the element-wise sum and element-wise products of their vectors. This fact is the driving force behind our rapid classification of handwritten digits.

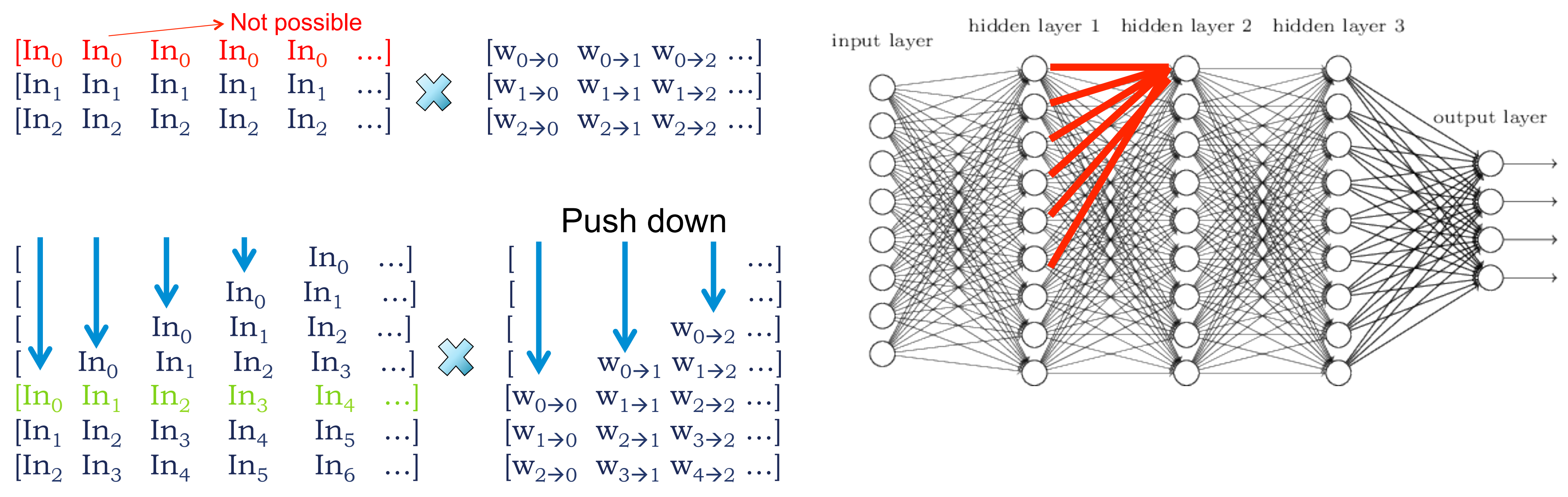
Results & Conclusions

These results compare an un-encrypted, un-binarized implementation, an encrypted implementation or a network with 784, 200, & 10 input, hidden, and output fully connected layers, and a un-encrypted CNN implementation. Although computation time is comparatively much higher for our encrypted network, it is a tremendous improvement over the 120 second encrypted evaluation of the un-batched 1024 → 784 neuron convolutional layer of our CNN.

Timing Analysis	**(Homomorphic Encryption)**	**(Without encryption)**	**(CNN)**
Time for encryption:	2.51 seconds		
Time for computation	1.59 seconds	0.17 ms	~15 ms
Time for decryption	0.5 ms		
Memory Usage			
Input image:	38.5 MB	6.3 KB	6.3 KB
output:	49 KB	1 B	1 B
Accuracy	89.1%	91.4%	98.2%

Approach

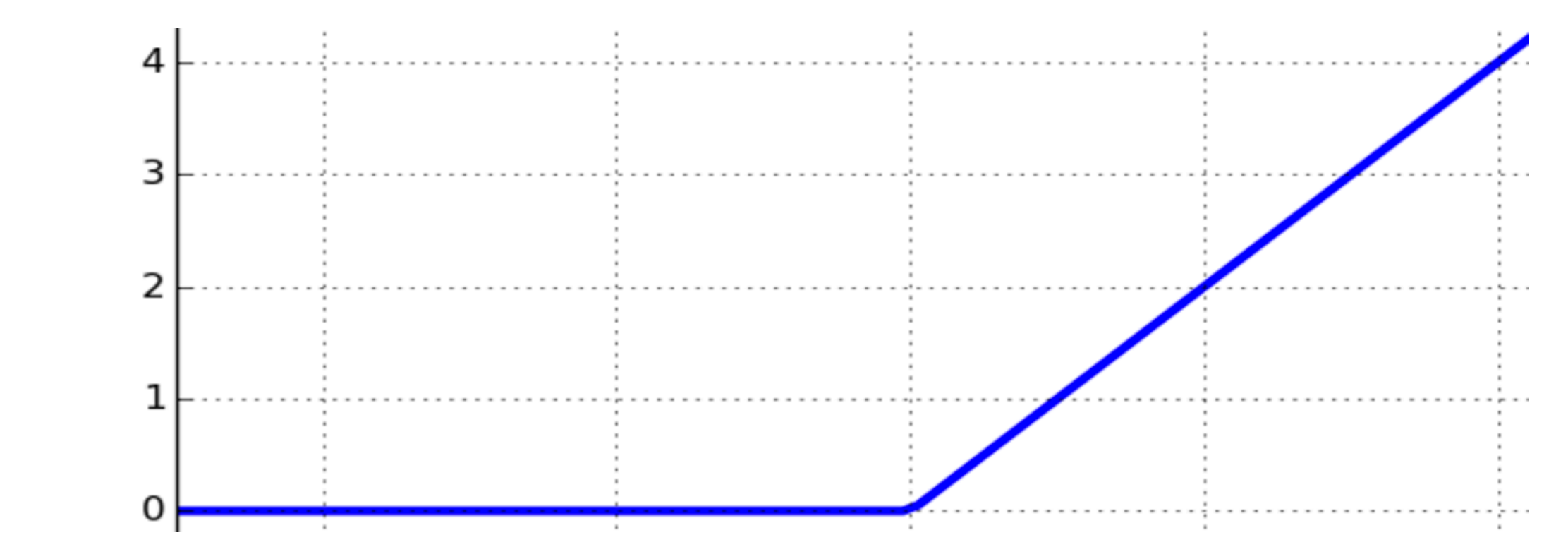
Challenge 1: Designing a convolution scheme for batched data



In our batching scheme, output activation batches are rotated using the Galois Automorphism feature of CRT batches (Microsoft SEAL library provides automorphism functions). Thus, after every layer of the neural network, the output activation is copied NEXT_LAYER_SIZE + PREV_LAYER_SIZE -1 times, and each ciphertext is then rotated as shown above.

Challenge 2: Selecting an activation function

A significant amount of time was spent attempting to find a scheme that computes the encrypted sign of a ciphertext's ith batched element. This would have enabled the use of the more accurate ReLU activation function. We finally accepted defeat on this endeavor, and settled on the square activation function, as it is very easily computable homomorphically. However, accuracy was sacrificed. Further work will focus on finding such a scheme.



*The ReLU activation is not computable using only additions and multiplications

Challenge 3: Dealing with explosive integer growth

Early on, we dealt with the scheme's integer-only restriction by multiplying the weights by a large factor (like 1000). The result was unprecedented growth in neuron activations even after a few layers due to the square activation function and the compounding factor of 1000. To solve this, we first binned the input image's pixels to a value between 0 and 10 (instead of 0 and 255). More significantly, we got rid of the factor 1000, and instead discretized the weights of out network to either +1 or -1 using the following algorithm:

```
Algorithm for the Binarization of Weights

1. Store the average of the weights in   _average
2. For every weight w:   w_binary = sign(w)*_average
3. Feed a training image into the network with binarized weights
4. Calculate gradients of the cost function with respect to each binarized weight
5. Update the floating point weights:   w ← w – learn_rate*gradient
6. Repeat for all training images
```

The result was a neural network with a much more manageable growth of activations. This makes it possible to design deep neural networks using our techniques.