cryptography

MDPI

Article

An Efficient Homomorphic Argmax Approximation for Privacy-Preserving Neural Networks

Peng Zhang *, Ao Duan and Hengrui Lu

College of Electronics and Information Engineering, Shenzhen University, Shenzhen 518060, China; 2110436077@email.szu.edu.cn (A.D.); 2300432009@email.szu.edu.cn (H.L.)
* Correspondence: zhangp@szu.edu.cn

Abstract: Privacy-preserving neural networks offer a promising solution to train and predict without user privacy leakage, and fully homomorphic encryption (FHE) stands out as one of the key technologies, as it enables homomorphic operations over encrypted data. However, only addition and multiplication homomorphisms are supported by FHE, and thus, it faces huge challenges when implementing non-linear functions with ciphertext inputs. Among the non-linear functions in neural networks, one may refer to the activation function, the argmax function, and maximum pooling. Inspired by using a composition of low-degree minimax polynomials to approximate sign and argmax functions, this study focused on optimizing the homomorphic argmax approximation, where argmax is a mathematical operation that identifies the index of the maximum value within a given set of values. For the method that uses compositions of low-degree minimax polynomials to approximate argmax, in order to further reduce approximation errors and improve computational efficiency, we propose an improved homomorphic argmax approximation algorithm that includes rotation accumulation, tree-structured comparison, normalization, and finalization phases. And then, the proposed homomorphic argmax algorithm was integrated into a neural network structure. Comparative experiments indicate that the network with our proposed argmax algorithm achieved a slight increase in accuracy while significantly reducing the inference latency by 58%, as the homomorphic sign and rotation operations were rapidly reduced.

Keywords: privacy-preserving neural networks; fully homomorphic encryption; non-linear functions; argmax

Citation: Zhang, P.; Duan, A.; Lu, H. An Efficient Homomorphic Argmax Approximation for Privacy-Preserving Neural Networks. *Cryptography* **2024**, *8*, 18. https://doi.org/10.3390/cryptography8020018

Received: 16 February 2024
Revised: 16 April 2024
Accepted: 29 April 2024
Published: 1 May 2024

1. Introduction

With the rapid advancement of artificial intelligence (AI), AI-driven products, like ChatGPT, have become integral in providing substantial assistance in people's daily lives. However, users who intend to utilize these AI-driven products must provide their own data to the AI models. This necessity raises critical questions about data security and user privacy, underscoring the need for technologies that safeguard user data against unauthorized access and misuse. When it comes to data privacy, the utilization of AI faces significant challenges. Privacy-preserving neural networks (PPNNs) have emerged as a promising solution to address the privacy concerns associated with neural network models. PPNN techniques, such as homomorphic encryption and secure multi-party computation, allow for the encryption of user data in such a way that AI models can be trained and inferenced on encrypted data without ever exposing the raw data. A PPNN enables model providers to process data without accessing users' sensitive information, thereby ensuring data privacy while utilizing the models.

Fully homomorphic encryption (FHE) stands out as a crucial technology in realizing a PPNN, as it enables model computations over encrypted data. An FHE-enabled PPNN makes it possible to provide a model prediction service without data privacy leakage. In 2016, Cryptonets [1] marked a pioneering achievement by integrating FHE into a PPNN,

showcasing its application in handwritten digit recognition. Subsequently, extensive studies have been conducted in this field. These research efforts have aimed at enhancing the efficiency and applicability of FHE in PPNNs, focusing on reducing the computational complexity and expanding the range of operations that can be performed on encrypted data. However, as FHE only supports the addition and multiplication homomorphisms, how to implement non-linear operations over the encrypted data remains one of the significant gaps between an FHE-enabled PPNN and traditional neural networks. To bridge this gap, various approximation techniques have been explored to approximate non-linear functions using the linear operations supported by FHE, thus enabling a broader spectrum of neural network functionalities within the encrypted domain.

In neural networks, non-linear operations, such as activation functions, maximum pooling, and argmax, play a vital role. At first, low-degree polynomials are substituted to approximate activation functions [1], but this approximation cannot support complex classification tasks due to large approximation errors. Then, Chabanne et al. [2] and Hesamifard et al. [3] employed high-order polynomials to approximate activation functions, but this approximation requires significant computing resources. Despite these advancements, the trade-off between approximation accuracy and computational overhead remained a critical challenge for researchers. Next, a study conducted by Wu et al. [4] demonstrated the use of trainable polynomials to approximate activation functions, but training a model is also a challenging task. This method introduced the concept of learning the approximation function directly from data, which represents a significant step toward more adaptive and efficient models. Recently, Lee et al. [5] proposed using a composition of low-degree minimax polynomials to approximate activation functions, which can efficiently achieve precise approximation.

Different from activation functions, the maximum pooling and argmax functions necessitate multiple comparisons between ciphertexts to derive their results. The argmax function is a mathematical operation that identifies the index of the maximum value in a given set of values. Unlike activation functions, which introduce non-linearity into a neural network and are applied element-wise to individual values, argmax is used in scenarios where the relative comparison of multiple values is necessary to determine a single output. The comparison process requires the use of multiple approximated versions of the sign function. Based on the homomorphic sign approximation algorithm, Lee et al. [5] successfully employed the sign function to approximate maximum pooling. Similarly, Phoenix [6] utilized polynomial approximations of the sign function to implement the argmax function.

In terms of model security, Lee et al. [7] used polynomial approximation for the softmax function, enabling an FHE-based PPNN to directly evaluate the softmax function, mitigating threats posed by model extraction attacks. This approach not only preserves the confidentiality of the model's parameters but also ensures that the model's output can be securely computed without revealing sensitive information. However, Truong et al. [8] argued that softmax is no longer sufficient for effective protection, and to further defend against model extraction attacks, argmax should replace softmax. This transition is pivotal for enhancing the security measures within neural networks, as argmax provides a more straightforward mechanism for decision-making that is less susceptible to adversarial manipulations. Next, Phoenix [6] homomorphically implemented argmax. Using the sign function repeatedly requires the introduction of bootstrapping operations or the allocation of larger initial parameters, thereby incurring greater computational costs.

When utilizing the sign function to implement the argmax function, the number of comparisons is the key factor. Note that each element is individually compared with all other elements in Phoenix. In order to improve the efficiency of argmax, we propose an efficient homomorphic argmax approximation. Our contributions are summarized as follows:

- We propose an efficient homomorphic argmax approximate algorithm **ArgmaxHE**$^+$ by employing a tree-structured comparison to determine the position of the maximum

value. The algorithm consists of four phases: rotation accumulation, tree-structured comparison, normalization, and finalization. Since the computation primarily involves homomorphic sign and rotation operations, we compared the theoretical counts of SgnHE and rotation operations to validate the efficiency improvement. By conducting this comparative analysis, our **ArgmaxHE**$^+$ offered a slight increase in accuracy while significantly reducing the inference latency by 58%.

- We integrated the proposed homomorphic argmax algorithm **ArgmaxHE**$^+$ into a PPNN and evaluated our approach on two datasets, namely, MNIST and CIFAR-10. Compared with a PPNN with **ArgmaxHE**, our PPNN with **ArgmaxHE**$^+$ offered a slight increase in accuracy while significantly reducing the inference latency for MNIST and CIFAR-10. This not only improves the user experience by delivering faster results but also enlarges the scope of applications for a PPNN by making it feasible to deploy in time-sensitive scenarios.

2. Related Works

In 2016, Cryptonets [1] made a groundbreaking connection between neural networks and FHE, achieving the first ciphertext inference of handwritten digits through a PPNN. Their model achieved an impressive 99% accuracy on the MNIST dataset but this came at the cost of a 250 s inference time. Subsequently, Lee et al. [7] extended the realm of PPNNs into deep learning by integrating FHE with ResNet, ultimately achieving a 92.43% accuracy on the CIFAR-10 dataset. This performance incurred only a slight 0.5% accuracy loss compared with plaintext inference, but the inference latency dramatically increased to 2.9 h. In pursuit of a reduced computational overhead, Dathathri et al. [9] turned to lightweight neural networks, such as SqueezeNet, to implement a PPNN. Lou et al. [10] further optimized the approach, ultimately achieving a remarkable result. They managed to attain an accuracy of 81.9% on the CIFAR-10 dataset in just 29.6 s—a significant reduction in inference latency.

One of the key points in the construction of an FHE-enabled PPNN is the implementation of non-linear functions. Some work used polynomials to approximate the non-linear functions. For instance, Cryptonets [1] substituted the square activation function for ReLU, Chabanne et al. [2] opted for a Taylor series approximation for ReLU, and Hesamifard et al. [3] chose Chebyshev polynomials to approximate the derivative of the ReLU. Furthermore, a study conducted by Wu et al. [4] demonstrated the use of trainable polynomials to approximate activation functions. While this method yielded promising results for smaller neural networks, it necessitated higher-precision approximation strategies when applied to deeper networks. Furthermore, Lee et al. [7] also approximated the softmax function. Within the softmax function, there are two non-linear operations: the exponential function and its inverse. Lee et al. utilized the least squares method to approximate the exponential function portion and employed the Goldschmidt division method to approximate the inverse function part.

Recently, Lee et al. [5] proposed a novel approach for approximating non-linear operations using high-order polynomials. This technique involves utilizing the minimax composite polynomial to approximate the sign function, followed by employing the sign function to implement various non-linear operations. The use of composite polynomials to approximate non-linear operations has been a recent focus. CKK19 [11], for instance, employed composite polynomials for iterative computations to approximate the comparison function. Subsequently, Cheon et al. [12] introduced further enhancements to the CKK19 approach, with the aim to optimize the computational complexity and reduce the execution time. Similarly, Phoenix [6] utilized polynomial approximations of the sign function to implement the argmax function.

Regarding safeguarding model security, Lee et al. [7] highlighted a vulnerability in traditional PPNN models where users are required to execute softmax operations. This setup allows users to perform multiple input–output pairs to extract model parameters. To counteract this potential model extraction threat, they introduced a polynomial approximation

of the softmax function. This adjustment enables PPNN models to undergo homomorphic evaluation of the softmax function directly on the server, enhancing the security. Following this, Boemer et al. [13] argued that relying solely on the softmax function is insufficient for providing effective model protection. They advocated for the replacement of the softmax function with the argmax function. Jovanovic et al. [6] further extended this idea in their work, implementing a composite polynomial approximation for the argmax function. By placing the argmax function on the server, they enabled homomorphic evaluation of argmax instead of softmax, adding an extra layer of security to PPNN models.

3. Preliminaries

3.1. RNS-CKKS

Cheon et al. [14] introduced a novel homomorphic encryption algorithm, namely, CKKS, which supports floating point arithmetic and is based on the ring learning with errors (RLWE) problem. RLWE is a mathematical framework that underpins the security of many lattice-based cryptographic schemes. Core operations in this algorithm include encoding, rescaling, and relinearization. Subsequently, Cheon et al. [15] optimized CKKS using the residue number system (RNS) and number theoretic transform (NTT) techniques, proposing the RNS-CKKS homomorphic encryption scheme. This scheme introduces a new ciphertext modular structure that allows for the use of the Chinese remainder theorem (CRT) decomposition of cyclotomic polynomials and NTT transformations on each RNS component. The CRT enables efficient modular arithmetic by breaking computations down into smaller components. Additionally, novel approximate modulus-switching schemes were introduced.

RNS-CKKS is an efficient variant of the somewhat homomorphic encryption scheme CKKS using the residue number system (RNS). In RNS-CKKS, plaintext and ciphertext consist of elements in a polynomial ring modulo a large integer $Q = \prod_{l=0}^{L} Q_l$, where the Q_l are distinct primes and L is the number of levels. As a leveled FHE scheme, RNS-CKKS can evaluate circuits with a depth up to a parameter L. The RNS representation enables replacing multi-precision operations on Q with computations modulo the base primes Q_l. This provides significant performance gains compared with the original CKKS scheme.

A message M is represented as a vector of real or complex numbers called slots. To encrypt, the message slots are scaled by an initial factor Δ and encoded into a plaintext polynomial. Using the public key pk, this encodes to a ciphertext polynomial $Enc_{pk}(x)$. The supported homomorphic operations are as follows:

- **Addition**: Slot-wise addition $\oplus$ of message vectors.
- **Subtraction**: Slot-wise subtraction $\ominus$ of message vectors.
- **Multiplication**: Slot-wise multiplication $\odot$ of message vectors.
- **Left rotation**: Cyclic left rotation *RotL* shifting slot positions.
- **Right rotation**: Cyclic right rotation *RotR* shifting slot positions.

The choice of parameters M and L involves tradeoffs between the efficiency and supported computation complexity. In this work, we introduce techniques to optimize the performance for argmax workloads over RNS-CKKS-encrypted data.

3.2. Composite Polynomial for Argmax in Phoenix

Phoenix [6] utilized the approximated sign function called SgnHE to implement argmax, and proposed a homomorphic argmax approximation algorithm called **ArgmaxHE**, which is described in Algorithm 1.

In this paper, the notation a^b signifies the repetition of a for b terms within a polynomial expression. The parameters d_q and d_p correspond to the degrees of odd and even polynomials, respectively, that are employed when approximating the sign function using composite polynomials. For detailed information on the specific methodologies concerning the use of polynomial approximation for the sign function, please refer to [12].

In each comparison, the SgnHE and rotation operations are used once. Ultimately, the position that wins in all the comparisons was assigned the value 1, while the others were

assigned 0. When $c = 8$, this method required seven comparisons, specifically including seven SgnHE and seven rotation operations, which were the main computation loads for **ArgmaxHE**.

Algorithm 1 ArgmaxHE in Phoenix

1: **Inputs:** $z = [z_1, \ldots, z_c, 0^{(M-c)}], d_q^{(1)}, d_p^{(1)}, d_q^{(2)}, d_p^{(2)}$
2: **Output:** $y = [y_1, \ldots, y_c, \#^{(M-c)}]$
3: $z \leftarrow z \oplus \mathrm{RotR}(z, c)$
4: $scores \leftarrow [0^M]$
5: $z_{\mathrm{rot}} \leftarrow z$
6: **for** j $\leftarrow 1$ to $c-1$ **do**
7: $\quad z_{\mathrm{rot}} \leftarrow \mathrm{RotL}(z_{\mathrm{rot}}, 1)$
8: $\quad diff \leftarrow z \ominus z_{\mathrm{rot}}$
9: $\quad signs \leftarrow \mathrm{SgnHE}(d_q^{(1)}, d_p^{(1)}, 4, diff)$
10: $\quad scores \leftarrow scores \oplus signs$
11: **end for**
12: $scores \leftarrow (scores \times [1/(2c-2)]) - [(c-2)/(2c-2)]$
13: $scores \leftarrow scores \times [1^c, 0^{M-c}]$
14: $signs \leftarrow \mathrm{SgnHE}(d_q^{(2)}, d_p^{(2)}, 4, scores)$
15: $y \leftarrow (y \times 0.5) + 0.5$
16: **return** y

4. An Efficient Homomorphic Argmax Approximation

The method described in Algorithm 1 to find the position of the maximum value involves comparisons between any two numbers, and one comparison consumes a large number of SgnHE and rotation operations, which are time consuming and level consuming. Thus, in this section, based on the knowledge that a composite polynomial can approximate non-linear functions with high accuracy, in order to improve the efficiency, we propose an efficient homomorphic argmax approximate algorithm **ArgmaxHE**$^+$ and give an example to explain it. And then, we compare the number of SgnHE and rotation operations in **ArgmaxHE** and **ArgmaxHE**$^+$ in theory to verify the efficiency improvement.

4.1. The Proposed Algorithm

Define $\mathbf{z} = [z_1, \ldots, z_c]$ to be the ciphertext of $\boldsymbol{\mu} = [\mu_1, \ldots, \mu_c]$, where $c = 2^k$. The proposed homomorphic argmax function is described in Algorithm 2. Mainly, it includes the following four phases:

- **Phase I: rotation accumulation (line 7 to line 9).**
  In this phase, all numbers in the ciphertext are accumulated though rotations.
- **Phase II: tree-structured comparison (line 10 to line 12).**
  Initially, all numbers are divided into several small groups by $\mathbf{m_k} = [m_{k0}, \ldots, m_{kk}]$, and each group contains 2^{j-1} numbers, where $j = 1, \ldots, k+1$. The purpose of this step is to make only the first number in each group remain unchanged, while the rest are set to 0.
  Then, pairs of small groups are combined to form a larger group, and comparisons are made to determine which of the two small groups is larger within each large group. If the first small group is larger, the first number in the large group is set to 1, with the remaining numbers being 0. If the second small group is larger, the first number in the large group is set to -1, with the rest of the numbers being 0.
- **Phase III: normalization (line 13 to line 16).**
  Normalization is involved to obtain standardized comparison results for each group. In the large group, at the position with the maximum, the corresponding number in **b** is set to 2, and the others are set to 0.
- **Phase IV: finalization (line 20 to line 21).**

To finalize the process, we subtract $2k-1$ from each element in $\mathbf{u}$. Subsequently, the SgnHE function is applied to transform the results into a logit composed solely of 1 and -1, which are transformed by multiplying each element by 0.5 and adding 0.5, resulting in a one-hot logit, where the only 1 corresponds to the maximum in $\boldsymbol{\mu}$.

Algorithm 2 Improved Homomorphic Argmax **ArgmaxHE**$^+$ when $c = 2^k$

1: **Inputs:** $\mathbf{z} = [z_1, \ldots, z_c]$, $d_q^{(1)}, d_p^{(1)}, d_q^{(2)}, d_p^{(2)}$
2: **Output:** One-hot logit $\mathbf{y} = [y_1, \ldots, y_c]$, where the only 1 corresponds to the maximum in $\boldsymbol{\mu}$, and the rest are 0
3: $\mathbf{u} \leftarrow [0^M]$
4: $\mathbf{a} \leftarrow \mathbf{z}$
5: $\mathbf{d} \leftarrow \mathbf{z}$
6: **for** $j \leftarrow 1$ to k **do**
7: **for** $l \leftarrow 1$ to $2^{j-1} - 1$ **do**
8: $\mathbf{a} \leftarrow \mathbf{d} \oplus \mathrm{RotL}(\mathbf{d}, l)$
9: **end for**
10: $\mathbf{a} \leftarrow \mathbf{a} \odot m_{k(j-1)}$
11: $\mathbf{a} \leftarrow \mathbf{a} \ominus \mathrm{RotL}(\mathbf{a}, 2^{j-1})$
12: $\mathbf{b} \leftarrow \mathrm{SgnHE}(d_q^{(1)}, d_p^{(1)}, 4, \mathbf{a}) \odot m_{k(j)}$
13: **for** $l \leftarrow 1$ to $j - 1$ **do**
14: $\mathbf{b} \leftarrow (\mathbf{b} \oplus \mathrm{RotR}(\mathbf{b}, l))$
15: **end for**
16: $\mathbf{b} \leftarrow (\mathbf{b} \ominus \mathrm{RotR}(\mathbf{b}, 2^{j-1})) \oplus [1^c]$
17: $\mathbf{u} \leftarrow \mathbf{u} \oplus \mathbf{b}$
18: $\mathbf{d} \leftarrow \mathbf{b} \odot \mathbf{d}$
19: **end for**
20: $\mathbf{y} \leftarrow \mathbf{u} \ominus [(2k-1)^c]$
21: $\mathbf{y} \leftarrow \mathrm{SgnHE}(d_q^{(2)}, d_p^{(2)}, 4, \mathbf{y}) \odot [0.5^c] \oplus [0.5^c]$
22: **return** $\mathbf{y}$

In Algorithm 2, $\mathbf{m_k} = [m_{k0}, \ldots, m_{kk}]$ is used to divide the numbers into several small groups with $2^{j-1} (j = 1, \ldots, k+1)$ numbers, and their roles are to make sure that only the first number in each group remains unchanged, while the rest are set to 0. For example, when $k = 3$, $\mathbf{m_3} = [m_{30}, \ldots, m_{33}] = [11111111, 10101010, 10001000, 10000000]$; when $k = 4$, $\mathbf{m_4} = [m_{40}, \ldots, m_{44}] = [1111111111111111, 1010101010101010, 1000100010001000, 1000000010000000, 1000000000000000]$.

Let us give an example to explain Algorithm 2. As shown in Figure 1, when $k = 3$, given any $z = (0.4, 0.3, 0.2, 0.6, 0.8, 0.7, 0.1, 0.5)$, there are a total of three rounds. As described in the algorithm, the actual objects being manipulated are ciphertexts, but the plaintexts are displayed in the figure for the purpose of providing a clearer and more intuitive illustration of this algorithm.

For the first round, namely, $j = 1$, phase I is skipped; in phase II, the small group with $2^{j-1} = 1$ numbers is shown in line 10, and the larger group with $2^j = 2$ numbers is shown in line 12; in phase III, the number at the position with the maximum in each large group is set to 2, and the others are set to 0. Next, $\mathbf{u}$ is used to record the comparison results, and $\mathbf{d}$ is prepared to be the input of the next round.

For the second round, namely, $j = 2$, in phase I, $\mathbf{a}$ is updated by accumulating $\mathbf{d}$ before and after one left rotation; in phase II, the small group with $2^{j-1} = 2$ numbers is shown in line 10, and the larger group with $2^j = 4$ numbers is shown in line 12; in phase III, the number at the position with the maximum in each large group is set to 2, and the others are set to 0. Next, $\mathbf{u}$ is used to record the comparison results, and $\mathbf{d}$ is prepared to be the input of the next round.

For the third round, namely, $j = 3$, in phase I, $\mathbf{a}$ is updated; in phase II, the small group has $2^{j-1} = 4$ numbers and the larger group has $2^j = 8$ numbers; in phase III, the number at

the position with the maximum in each large group is set to 2, and the others are set to 0. These steps are very similar to the previous two rounds such that **u** is obtained.

Lastly, in phase IV, a one-hot logit **y** with the only 1 at the position of the maximum is obtained.

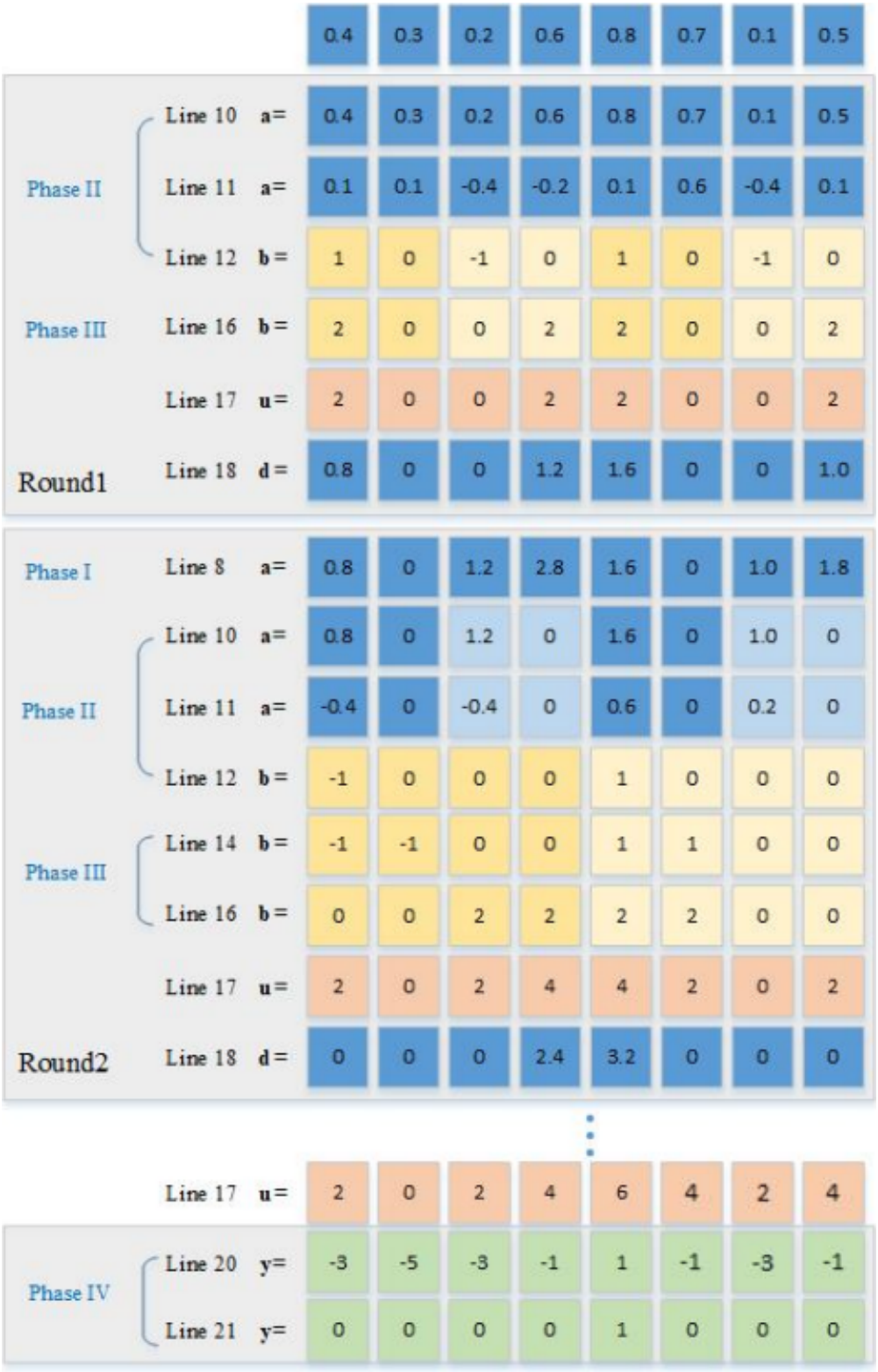

Figure 1. An example for our homomorphic argmax approximation algorithm.

4.2. Theoretical Analysis

As described in Algorithms 1 and 2, when $c = 2^k$, **ArgmaxHE** uses a total of 2^k SgnHEs, while **ArgmaxHE**$^+$ requires $k+1$ SgnHEs; **ArgmaxHE** needs to use $2^k - 1$ rotations, while **ArgmaxHE**$^+$ requires $k(k+1)$ rotations. The comparison of the number of SgnHE and rotation operations used in both algorithms is presented in Table 1.

Table 1. The numbers of SgnHE and rotation operations.

Algorithms	The Number of SgnHEs	The Number of Rotations
ArgmaxHE [6]	2^k	$2^k - 1$
ArgmaxHE$^+$	$k+1$	$k(k+1)$

For the rotations, when k is relatively small, **ArgmaxHE**$^+$ uses more operations than **ArgmaxHE**. But when k increases, the number of operations used by **ArgmaxHE**$^+$ becomes significantly less than **ArgmaxHE**. Notably, when $k = 5$, the number of operations in **ArgmaxHE**$^+$ and **ArgmaxHE** are 30 and 32, respectively. The comparison of the number of rotation operations is illustrated in Figure 2.

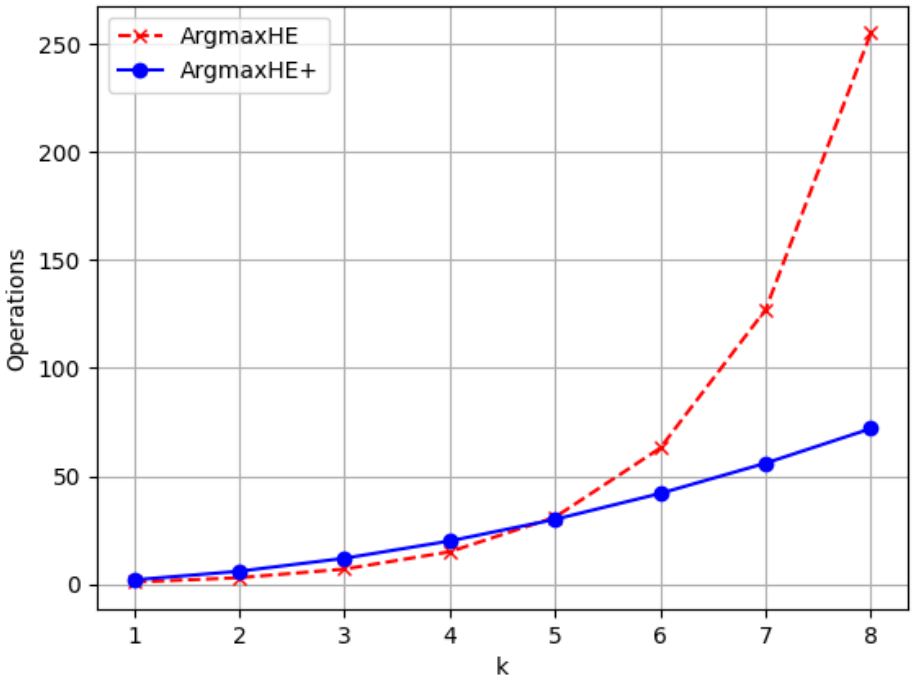

Figure 2. The numbers of rotation operations at different k values.

5. Experimental Results

In order to compare our proposed **ArgmaxHE**$^+$ algorithm with the **ArgmaxHE** algorithm [6], the platform, design, and analysis of experiments are introduced in this section.

5.1. Experimental Platform

In this work, one server was involved for all experimental tests. Specifically, the CPU of the server was an Intel (R) Core (TM) i9-10900X with 10 cores and a 256 GB RAM memory. The following experiments were implemented in this environment. Based on the Microsoft SEAL library [16], the data were encrypted by the FHE algorithm RNS-CKKS [15]. In order to ensure that all ciphertext could reach the 128-bit security level in the PPNN, the ring polynomial degree N was 131,072. We set the initial scale $\Delta = 2^{50}$ and the parameters $d_q^{(1)} = 6$, $d_p^{(1)} = 1$, $d_q^{(2)} = 2$, and $d_p^{(2)} = 2$ to meet the precision requirement of $\alpha = 0.001$. The above parameters were kept the same for the **ArgmaxHE** algorithm [6].

In our work, two distinct datasets were employed: the MNIST dataset and the CIFAR-10 dataset. The MNIST dataset, which is well-known for its extensive collection of handwritten digit images, is divided into 10 categories, each corresponding to a digit from 0 to 9. These images are single-channel grayscale with a resolution of 28 by 28 pixels. We employed $PPNN^1$ for the MNIST dataset. The employed neural network comprised four layers, with a hidden layer size of 32, consisting of two fully connected layers, an activation layer, and an argmax layer, which was kept the same for the **ArgmaxHE** algorithm [6]. The first fully connected layer was followed by a trainable activation function. The expression for this activation function was given by $ax^2 + bx$, where the parameters a and b were generated during the training phase. Subsequently, the second fully connected layer was followed by an argmax layer.

Meanwhile, the CIFAR-10 dataset was introduced; this dataset provides a more complex set of color images that encompassed 10 different classes featuring a variety of objects, such as vehicles and animals. The CIFAR-10 images, measuring 32 by 32 pixels, are multi-channel RGB images, providing a more challenging and diverse visual task for the network. We employed $PPNN^2$ for the CIFAR-10 dataset, which comprised three fully connected layers and two activation layers, with a hidden layer size of 128. The first two fully connected layers were each followed by an activation layer, while the third fully connected layer was followed by an argmax layer.

5.2. Experimental Design

First, we exclusively compared the accuracy and computation time of two argmax approximation schemes, namely, **ArgmaxHE** and **ArgmaxHE**$^+$. We assumed the input consisted of a set of 16 data logits denoting $c = 16$ and $k = 4$. The input here was $\boldsymbol{\mu}$, which consisted of a set of 16 random numbers drawn from the interval between -1 and 1, which was generated 1000 times for the experimental trials. We encrypted the input data to obtain $\mathbf{z}$ and fed it separately into **ArgmaxHE** and **ArgmaxHE**$^+$. Subsequently, we obtained the corresponding outputs and compared them with the theoretically expected one-hot results, as well as calculated the errors. This process was repeated 1000 times to derive the average error and average computation time.

Our neural network underwent rigorous training on the MNIST dataset, comprising 50,000 training images and 10,000 testing images. Key training parameters included a learning rate of 0.01 and weight decay set to 0.001. The training process extended over 100 epochs, resulting in optimized parameters for two fully connected layers. Additionally, the experiment yielded insights into the coefficients a and b of the trainable polynomial in the activation layer. These parameters collectively defined the network's structure and behavior.

Furthermore, to enhance the network's capabilities, we incorporated the CIFAR-10 dataset into our training regimen. This dataset comprised 50,000 training images and 10,000 testing images, with each one representing 32×32 pixel color images of natural scenes. Key training parameters included a learning rate of 0.01 and weight decay set to 0.001. The training process extended over 100 epochs. By leveraging this additional dataset, our neural network gained a richer understanding of the visual patterns and textures, enabling it to make more accurate predictions.

During the prediction phase, each experiment involved processing one image at a time, generating corresponding predictions. The predictions were organized into groups of 128, with a total of five groups, which constituted the set of experiments. For each group of 128 predictions, the success probability was statistically analyzed. The overall accuracy was determined by averaging the success probabilities across the five groups. The experiments encompassed three variants.

$PPNN^{1,2}$ + **Argmax**: Utilizing the described network structure, this variant introduced decryption operations before the argmax layer. The decrypted results were then fed into the argmax layer, yielding the final prediction.

$PPNN^{1,2}$ + **ArgmaxHE**: This variant also employed the same network structure, except for omitting the decryption operation before the argmax layer. Instead, it employed a polynomial approximation to homomorphically evaluate the argmax function, as detailed in Algorithm 1.

$PPNN^{1,2}$ + **ArgmaxHE**$^+$: Sharing the same network structure as the previous two, this variant used our proposed algorithm **ArgmaxHE**$^+$ in the argmax layer, as detailed in Algorithm 2.

5.3. Experimental Analysis

When we conducted the homomorphic argmax approximation experiments, we set specific parameters for a fair comparison between Algorithms 1 and 2. The data size was fixed at $c = 16$, and the number of slots was configured at $n = 65{,}536$ for both algorithms. During these experiments, we measured the time from feeding the encrypted data to producing the output, excluding the initialization, encryption, and decryption times. This allowed us to focus solely on the server's prediction time. These results are summarized in Table 2. Algorithm 1 achieved a final accuracy of 0.999426, with a total time consumption of 285,169 ms, while Algorithm 2 reached a higher final accuracy of 0.999459 in 119,639 ms, representing a 58% improvement. It is important to note that Algorithm 2 had a lower computation time due to the reduced number of SgnHEs, which were more costly than the rotation and scalar multiplication operations. Although Algorithm 2 included extra rotation and scalar multiplication operations, the overall execution time was significantly reduced. The accuracy improvement, or in other words, the reduction in the model's approximation error, was attributed to the decreased usage of SgnHEs.

Table 2. Comparisons of execution accuracy and time.

Algorithm	Accuracy	Time (ms)
ArgmaxHE [6]	0.999426	285,169
ArgmaxHE$^+$	0.999459	119,639

The training process concluded with a total training time of 700,964 ms. The achieved accuracy on the training set was 97.54%, while the accuracy on the test set reached 97.45%. It is noteworthy that these results were obtained by training directly on the plaintext data, without any encryption or obfuscation techniques applied. Additionally, the coefficients for the trainable polynomial activation function obtained upon completion on the MNIST dataset were determined to be a = 0.57742 and b = 0.59659. On the CIFAR-10 dataset, the coefficients for the first activation layer were a = -0.04536 and b = -1.00021, while for the second activation layer, they were a = -0.13628 and b = -1.82785.

Table 3 illustrates the application of argmax and homomorphic argmax to the neural network, showing the total computational time of the algorithms, excluding the communication time, such as the user data upload and server response times. It can be observed that **ArgmaxHE**$^+$ exhibited improvements in both the time and accuracy compared with **ArgmaxHE**. While there was a slight enhancement in the accuracy, a significant improvement was noted in terms of time, which was attributed to the reduced usage of SgnHE by five times. When c equalled a power of 2, the reduction in SgnHE usage was maximized, resulting in optimal performance enhancement for **ArgmaxHE**$^+$.

It is noteworthy that the CIFAR-10 dataset required a significantly longer processing time, approximately 7 to 9 times that of MNIST, which was mainly due to the increased complexity of the neural network architecture. Despite the improvement in execution time, particularly in the argmax operation, the overall enhancement was marginal. This was because the majority of the time was consumed during the inference phase, making the impact of argmax optimization less pronounced on the total processing duration. However, the notable improvements achieved with Algorithm 2 in both time efficiency and accuracy on the CIFAR-10 dataset served as a compelling indication of its potential applicability and effectiveness in other complex datasets and neural network configurations. This suggests

that the algorithm can be successfully adapted and yield beneficial results in a wide range of scenarios. Based on the experimental results, it is evident that our proposed solution has its limitations. While the accuracy of the experiments was acceptable on simpler datasets, such as MNIST, it fell short on more complex datasets, like CIFAR-10. This inadequacy can be attributed to two primary reasons: first, the simplicity of the neural network model, and second, the significant consumption of computational resources and memory due to the incorporation of encryption and decryption processes.

Table 3. Comparisons of execution accuracy and time in PPNN.

Schemes	Dataset	Accuracy	Time (ms)
$PPNN^1$ + **Argmax**	MNIST	0.9531	248,636
$PPNN^1$ + **ArgmaxHE**	MNIST	0.9393	427,621
$PPNN^1$ + **ArgmaxHE**$^+$	MNIST	0.9412	358,733
$PPNN^2$ + **Argmax**	CIFAR-10	0.4538	2,314,975
$PPNN^2$ + **ArgmaxHE**	CIFAR-10	0.4446	2,493,693
$PPNN^2$ + **ArgmaxHE**$^+$	CIFAR-10	0.4467	2,425,042

6. Conclusions

Inspired by employing polynomial-approximated signs to implement the argmax function, in this paper, we propose an efficient homomorphic argmax approximate algorithm, namely, **ArgmaxHE**$^+$, which employs a tree-structured comparison approach to determine the position of the maximum value. Instead of comparing one with the other elements, a pairwise tree-structured comparison was designed to improve the efficiency. Through the execution of comparative study, our **ArgmaxHE**$^+$ achieved a modest enhancement in precision, concurrently with a substantial decrease in inference latency, marking a reduction of 58%. Notably, we seamlessly integrated our proposed homomorphic argmax algorithm, **ArgmaxHE**$^+$, into a privacy-preserving neural network (PPNN) architecture. Compared with the original PPNN employing **ArgmaxHE**, our PPNN with **ArgmaxHE**$^+$ not only achieved a slight increase in accuracy but also, more significantly, delivered a significant reduction in inference latency. This approach can be applied to an FHE-based PPNN, as demonstrated in the experimental ciphertext predictions for MNIST and CIFAR-10. Looking ahead, the integration of techniques to introduce self-bootstrapping control noise in order to adapt to more complex models, along with hardware acceleration techniques, with an FHE-based PPNN holds great promise for further improving the efficiency and practicality.

Author Contributions: P.Z.: Conceptualization, Methodology, validation, manuscript review, editing, Supervision and project administration; A.D.:Conceptualization, Methodology, softwear, formal analysis, data curation, and original draft; H.L.: manuscript review and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in the study are included in the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Gilad-Bachrach, R.; Dowlin, N.; Laine, K.; Lauter, K.; Naehrig, M.; Wernsing, J. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 201–210.
2. Chabanne, H.; De Wargny, A.; Milgram, J.; Morel, C.; Prouff, E. Privacy-Preserving Classification on Deep Neural Network. *Cryptol. ePrint Arch.* **2017** . Available online: https://eprint.iacr.org/2017/035.pdf (accessed on 10 February 2024).
3. Hesamifard, E.; Takabi, H.; Ghasemi, M. Cryptodl: Deep neural networks over encrypted data. *arXiv* **2017**, arXiv:1711.05189.
4. Wu, W.; Liu, J.; Wang, H.; Tang, F.; Xian, M. Ppolynets: Achieving high prediction accuracy and efficiency with parametric polynomial activations. *IEEE Access* **2018**, *6*, 72814–72823. [CrossRef]

5. Lee, J.; Lee, E.; Lee, J.W.; Kim, Y.; Kim, Y.S.; No, J.S. Precise approximation of convolutional neural networks for homomorphically encrypted data. *IEEE Access* **2023**, *11*, 62062–62076. [CrossRef]
6. Jovanovic, N.; Fischer, M.; Steffen, S.; Vechev, M. Private and reliable neural network inference. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, Los Angeles, CA, USA, 7–11 November 2022; pp. 1663–1677.
7. Lee, J.W.; Kang, H.; Lee, Y.; Choi, W.; Eom, J.; Deryabin, M.; Lee, E.; Lee, J.; Yoo, D.; Kim, Y.S.; et al. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *IEEE Access* **2022**, *10*, 30039–30054. [CrossRef]
8. Truong, J.B.; Maini, P.; Walls, R.J.; Papernot, N. Data-free model extraction. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, 20–25 June 2021; pp. 4771–4780.
9. Dathathri, R.; Kostova, B.; Saarikivi, O.; Dai, W.; Laine, K.; Musuvathi, M. EVA: An encrypted vector arithmetic language and compiler for efficient homomorphic computation. In Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation, London, UK, 15–20 June 2020; pp. 546–561.
10. Lou, Q.; Jiang, L. HEMET: A homomorphic-encryption-friendly privacy-preserving mobile neural network architecture. In Proceedings of the International Conference on Machine Learning, Virtual, 18–24 July 2021; pp. 7102–7110.
11. Cheon, J.H.; Kim, D.; Kim, D.; Lee, H.H.; Lee, K. Numerical method for comparison on homomorphically encrypted numbers. In Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, 8–12 December 2019; pp. 415–445.
12. Cheon, J.H.; Kim, D.; Kim, D. Efficient homomorphic comparison methods with optimal complexity. In Proceedings of the Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, Republic of Korea, 7–11 December 2020; pp. 221–256.
13. Boemer, F.; Cammarota, R.; Demmler, D.; Schneider, T.; Yalame, H. MP2ML: A mixed-protocol machine learning framework for private inference. In Proceedings of the 15th International Conference on Availability, Reliability and Security, Virtual, 25–28 August 2020; pp. 1–10.
14. Cheon, J.H.; Kim, A.; Kim, M.; Song, Y. Homomorphic encryption for arithmetic of approximate numbers. In Proceedings of the Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, 3–7 December 2017; pp. 409–437.
15. Cheon, J.H.; Han, K.; Kim, A.; Kim, M.; Song, Y. A full RNS variant of approximate homomorphic encryption. In Proceedings of the Selected Areas in Cryptography–SAC 2018: 25th International Conference, Calgary, AB, Canada, 15–17 August 2018; pp. 347–368.
16. *Microsoft SEAL*, (Release 3.5); Microsoft Research: Redmond, WA, USA, 2020. Available online: https://github.com/Microsoft/SEAL (accessed on 10 February 2024).