Effective Activation Functions for Homomorphic Evaluation of Deep Neural Networks

SRINATH OBLA¹, XINGHAN GONG¹, ASMA ALOUFI¹, (Student Member, IEEE), PEIZHAO HU¹, (Member, IEEE), AND DANIEL TAKABI², (Member, IEEE)

¹Department of Computer Science, Rochester Institute of Technology, Rochester, NY 14623 USA
²Department of Computer Science, Georgia State University, Atlanta, GA 30302 USA

Corresponding author: Peizhao Hu (e-mail: Peizhao.Hu@rit.edu).

ABSTRACT CryptoNets and subsequent work have demonstrated the capability of homomorphic encryption (HE) in the applications of private artificial intelligence (AI). In convolutional neural networks (CNNs), many computations are linear functions such as the convolution layer which can be homomorphically evaluated. However, there are layers such as the activation layer which is comprised of non-linear functions that cannot be homomorphically evaluated. One of the most commonly used methods is approximating these non-linear functions using low-degree polynomials. However, using the approximated polynomials as activation functions introduces errors which could have a significant impact on accuracy in classification tasks. In this paper, we present a systematic method to construct HE-friendly activation functions for CNNs. We first determine what properties in a good activation function contribute to performance by analyzing commonly used functions such as Rectified Linear Units (ReLU) and Sigmoid. Then, we compare polynomial approximation methods and search for an optimal range of approximation for the polynomial activation. We also propose a novel weighted polynomial approximation method tailored to the output distribution of a batch normalization layer. Finally, we demonstrate the effectiveness of our method using several datasets such as MNIST, FMNIST, CIFAR-10.

INDEX TERMS Private AI, Homomorphic encryption, Activation function, Deep neural networks

I. INTRODUCTION

Deep neural networks have made significant contributions to solving complex tasks, especially in computer vision. In some applications, these networks require a large volume of private data; for example, lung cancer [1] and diabetic retinopathy detection [2]. However, regulations such as HIPAA [3] and GDPR [4] impose stringent restrictions on the use of private data by third-party service providers.

Homomorphic encryption (HE) supports arithmetic operations, such as addition and multiplication, on encrypted data without decrypting it first. Specifically, given two messages \( m \) and \( m' \) and homomorphic addition and multiplication operations \( \oplus \) and \( \otimes \), we have \( \text{Enc}(m) \oplus \text{Enc}(m') \) which decrypts to \( m + m' \) and \( \text{Enc}(m) \otimes \text{Enc}(m') \) which decrypts to \( m \times m' \). For simplicity, we will use normal arithmetic operators to represent homomorphic operations in the rest of the paper. Generally speaking, any algorithm that can be reduced to just these arithmetic operations can be homomorphically evaluated on encrypted data.

CNNs comprise of linear functions which can be homomorphically evaluated using homomorphic addition and multiplication. However, CNNs also consist of layers such as the activation layer that depend on non-linear functions such as ReLU, Sigmoid. Since we cannot represent these functions efficiently using addition and multiplication, they cannot be homomorphically evaluated.

A. CURRENT SOLUTIONS

To evaluate CNNs on homomorphically encrypted data, developing effective and HE-friendly support for non-linear functions has been an active topic of research in recent years [5]–[9]. There are three commonly used methods to support non-linear functions: use a polynomial substitution [5], precompute function for discrete values and store these values in look-up tables [6], and use low-degree polynomials to approximate the non-linear functions [7]–[9].
When CryptoNets [5] was first introduced, it used a single variable monomial with degree 2, that is a square function \( f(x) = x^2 \), to substitute those commonly used activation functions in the activation layer. Unfortunately, this substitution polynomial grows exponentially and causes instability during training in CNNs. As a consequence, CryptoNets cannot sustain more than one activation layer and has significant impact on accuracy (98.95% compared to 99.77% in state-of-the-art performance on the MNIST handwritten digit dataset [10]).

Another approach [6] is to precompute discrete samples of a non-linear activation function and store these discrete values in a lookup table. During homomorphic evaluation, the program performs a table lookup obliviously. However, low sampling rates would reduce floating-point precision and have a direct impact on the performance of neural networks. Conversely, increasing the rate of sampling could lead to large lookup tables.

A more commonly used approach is to approximate non-linear functions with low-degree polynomials which can be evaluated on encrypted data efficiently. As examples, Chabanne et al. [7] investigated the effectiveness of using Taylor-series polynomials to approximate ReLU activation function. Together with other techniques such as batch normalization (BN) [11] which sets a bound on inputs for each activation layer, these approximated polynomials help the networks to achieve 99.30% accuracy. CryptoDL [8] examined other polynomial approximation methods such as numerical analysis, standard/modified Chebyshev polynomials in addition to the Taylor series approach. Using these approximation methods, CryptoDL evaluated the effectiveness of different approximated polynomials for activation functions such as ReLU, Sigmoid, Tanh on MNIST and CIFAR-10 [12] datasets.

Observed from the existing works, the polynomial approximation approach is more robust than the table lookup approach and yields higher accuracy than simply using low-degree substitution functions. But, the existing works also show that identifying the best performing polynomial is a difficult task and is often customized for a dataset.

B. OUR CONTRIBUTIONS

In this paper, we aim to develop a systematic method to construct effective and HE-friendly activation functions for CNNs. While pursuing this goal, we made the following contributions.

- We first conduct extensive experiments using different datasets (MNIST, FMNIST, and CIFAR-10) to study the characteristics of effective activation functions in CNNs. We share the lessons learned for future research. For example, we discuss why having bounded derivative in activation functions is crucial for achieving high accuracy in CNNs. Also, we explore a region between \([-3,3]\) because almost 99% of inputs of an activation layer lie within this region.

- We leverage these lessons learned and propose a weighted polynomial approximation technique for finding effective and HE-friendly activation functions. Experimental results obtained from the three datasets show that our proposed method is able to identify HE-friendly activation functions which yield higher or the same accuracy as those customized functions proposed in the state-of-the-art works.

C. EXPERIMENT DATASETS

To validate our hypothesis, we run tests on multiple datasets with different neural network configurations as below.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>Easy</td>
<td>Medium</td>
<td>Hard</td>
</tr>
<tr>
<td>Image type</td>
<td>Grayscale</td>
<td>Grayscale</td>
<td>RGB</td>
</tr>
<tr>
<td>Image dimension</td>
<td>28 × 28</td>
<td>28 × 28</td>
<td>32 × 32</td>
</tr>
<tr>
<td>Num. of Classes</td>
<td>10,000</td>
<td>10,000</td>
<td>50,000</td>
</tr>
<tr>
<td>Test set size</td>
<td>10,000</td>
<td>10,000</td>
<td>10,000</td>
</tr>
</tbody>
</table>

D. ORGANIZATION

Section II briefly introduces the background of CNNs, HE, and methods for polynomial approximation. Section III reviews the related work. This is followed by a comprehensive analysis of the characteristics of effective activation functions in Section IV. We propose our method for finding effective and HE-friendly activation functions in Section V. In Section VI, we conduct experiments to study the performance of our weighted polynomial approximation in the classification tasks. We conclude this paper in Section VII.

II. BACKGROUND AND PRELIMINARIES

In this section, we provide brief descriptions of the background and preliminaries of convolutional neural networks, homomorphic encryption, and methods for polynomial approximation. Throughout this paper, we will use notations defined in Table 1 in our discussions.

A. CONVOLUTIONAL NEURAL NETWORKS

A CNN is a type of deep neural network that is used primarily for analyzing image data. It consists of a stack of layers, as shown in Fig. 1. The use of CNN for image analysis has two phases: training and inference. During inference, a CNN transforms image data from the input layer, through layers of computations, into the scores of each label in the output layer. Each layer consists of weights \( \omega \) that are adjusted during the network training phase using a technique called backpropagation [14]. In backpropagation, the weights corresponding to the transformations are adjusted to optimize a loss function based on the error between the prediction made by the network and the ground truth. More formally, we model a CNN as a sequence of transformations, such that at a layer \( \ell - 1 \) some functions \( f \) apply computations to the \( g \) input neurons \( x_1^{\ell-1}, x_2^{\ell-1}, ..., x_g^{\ell-1} \), yielding the value of \( h \) output neurons \( x_j^\ell = f(x_i^{\ell-1}); i \in \{1, ..., g\}; j \in \{1, ..., h\} \).
at the layer $\ell$. Depending on the type of a CNN layer, the function $f$ consists either linear or non-linear transformation, as we will discuss below.

1) Convolutional layer (CONV)

In image processing, a CONV layer reduces the complexity of input data by summarizing the neighboring pixels based on convolutional filtering, as illustrated in Fig. 2. In other words, the CONV uses trained filters to extract local information from a previous layer and significantly reduces the number of parameters to be optimized. Through the training phase, a sequence of $d$ filters, each of the size of $s \times s$, will be generated. In every filter, each weight $\omega_{ij}^{\ell}$ and bias $\beta_i^{\ell}$, where $i', j' \in s; k' \in d$ at layer $\ell$, is learned from the training dataset. For the value of each neuron at layer $\ell$, we compute $x_{i'j'k'}^{\ell} = \beta_{i'}^{\ell} + \sum_{i=1}^{s} \sum_{j=1}^{s} \sum_{k=1}^{d} \omega_{ijk}^{\ell} x_{i'j'k'}^{\ell-1}$ using $k'$ filter and some neurons of the previous layer at $\ell - 1$. Essentially, a convolution operation consists of many dot-products of matrices or weights $\omega_{ij}^{\ell}$ of a filter and neurons applied on the region of $\ell$ with sliding window $|s/2|, j' \leq j \leq |s/2|$. As illustrated in Fig. 2. When done, we slide the filter to the right to calculate the value for the next position in the feature map. This layer consists only linear transformations, hence it can be supported homomorphically.

2) Batch normalization layer (BAT)

This operation was introduced by Ioffe and Szegedy [11] to accelerate the training of deep neural networks. The input to the batch normalization layer is transformed to have zero mean and unit variance using the statistics observed in a batch. Normalizing the data in this manner makes the neural network optimization smoother [15] and results in stable and predictable training. During inference, the network uses the moving averages learned during training to normalize the data. Algorithm 1 illustrates the calculations taken place in the batch normalization. Note, $\gamma$ and $\beta$ are parameters obtained from training. In many CNNs, the BAT layer is optional. But, in our work we depend on this layer to bound the inputs that pass through the activation layer.

![Algorithm 1: Batch normalization over a mini batch X](image)

3) Activation layer (ACT)

The CONV layers enable the CNN to classify data with linearity, while non-linearity is captured by ACT layers using some non-linear functions $f(x)$. We compute the value of each neuron at the current layer as $x_{i}^{\ell} = f(x_{i}^{\ell-1}); i \in g; i' \in h$, where $g$ is the input neurons and $h$ is the output neurons.
There are many variants of activation functions depending on the characteristics of input data and application requirements. We focus on the following four classical activation functions as they often serve as the building blocks for other variants.

- **Rectified Linear Unit (ReLU):** \( f(x) = \max(0, x) \)
- **Logistic function (Sigmoid):** \( f(x) = \frac{1}{1 + e^{-x}} \)
- **Hyperbolic tangent (Tanh):** \( f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \)
- **Softplus:** \( f(x) = \log(1 + e^x) \)

Fig. 3(a) shows these activation functions for comparison. Among these activation functions, Sigmoid and Tanh become saturated after input ranges; hence they get relatively small gradients. To address this problem, there have been a range of newer activation functions which are similar to ReLU, as shown in Fig. 3(b).

Recent work also showed that we can design effective activation functions in a piecewise manner. For examples, Qin et al. [16] introduced the improved Sigmoid function — Isigmoid, which tunes the saturated region by a learnable parameters, and Wang et al. [17] proposed ReLTanh, which treats the negative and positive saturation region as two straight lines each with different slopes. These piecewise treatments are to address the vanishing gradient and bias shift problems. We will discuss these variants in more details in Sec. IV-F.

Classical activation functions like ReLU and Tanh have been further optimized for specific applications with the addition of tunable parameters. However, because the focus of our paper is to support activation functions for HE-CNNs in general, we choose to focus on the classical activations for our analysis and experiments. Fig. 3(d) shows two examples of these activation functions that are HE friendly. Among them, CryptoNets [5] used a square function but achieved low accuracy. Chabanne et al. [7] used a degree 4 polynomial that approximates ReLU and achieved better accuracy than CryptoNets.

4) **Pooling layer (POOL)**

The objective of this layer is to reduce the spatial size hence reducing the number of neurons. Neurons from the previous layer are partitioned into regions. For each of these region, a pooling function is applied to all neurons outputting a value for a neuron in the current layer. Applying pooling layer allows a larger number of filters to be used in deeper convolution layers which help in extracting complex features. A pooling layer is commonly used after an activation layer. Together with the CONV and ACT layers, they form a structure which is repeated through the network; i.e., CONV–ACT–POOL.

Two commonly used pooling functions are max-pool and average-pool. The average-pool function \( f(X) = \frac{\sum_{i \in n} x_i; x_i \in X} {n} \) can be computing easily, but the max-pool function \( f(X) = \max(x_1, x_2, ..., x_n); x_i \in X \) is a non-linear function which requires similar treatment as a non-linear activation function.

5) **Fully Connected layer (FC)**

In this layer, every neuron \( x_{i\ell}^{\ell}; i \in h \) at layer \( \ell \) is associated with every other neuron \( x_{j\ell-1}^{\ell-1}; i \in g \) in the previous layer \( \ell-1 \). There is a weight \( \omega_{ij\ell}^{\ell-1} \) on each of these connections, and a bias \( \beta_{i\ell}^{\ell} \) associated with \( x_{i\ell}^{\ell} \). We calculate the inner product yielding \( x_{i\ell}^{\ell} = \beta_{i\ell}^{\ell} + \sum_i \omega_{ij\ell}^{\ell-1} x_{i\ell-1}^{\ell-1} \). Note that we simplify the notations, especially subscripts, for clarity. Figure 4 depicts a common structure of neurons, weights and connections in a fully connected network.

6) **Softmax layer (SOFTMAX)**

The softmax operation is commonly used in deep neural networks to transform the final outputs into a set of probabilities. Usually used in classification problems with more than 2 classes, this linear transformation converts all values from the previous layer to the range \((0, 1)\) but they sum to 1. Each of these values can be interpreted as the probability score of the input belonging to the corresponding output class. Consequently, the class with the highest probability is the
class prediction made by the neural network. More formally, the softmax layer can be represented as
\[ \sigma(x_j) = \frac{e^{x_j}}{\sum_{t=1}^{k} e^{x_t}} \text{ where } j = \{1, ..., k\} \] (1)

B. HOMOMORPHIC ENCRYPTION

HE is a class of encryption schemes that support computations such as addition and multiplication on encrypted data. Existing HE schemes can be divided into three main types based on the homomorphic operations supported by the evaluation function. In Partial HE (PHE) schemes, the evaluation function supports either addition (i.e., additive homomorphism), such as Goldwasser-Micali [18] and Paillier cryptosystem [19], or multiplication (i.e., multiplicative homomorphism), such as ElGamal cryptosystem [20], but not both. In contrast, Fully HE (FHE) allows arbitrary number of additions and multiplications. Somewhat HE (SWHE) schemes support both addition and multiplication on the ciphertexts. Yet, the number of multiplications allowed is limited due to the inherited construction of the scheme where ciphertexts contain noise that exponentially scales with multiplications. In general, HE scheme is a tuple of PPT algorithms HE = (KeyGen, Enc, Eval, Dec). We define each algorithm as follow.

- HE.KeyGen(1^λ) \to (pk, sk): Given a security parameter \( \lambda \) determining the security level, the key generation algorithm outputs a public key pk, a private key sk.
- HE.Enc(pk, m) \to c: Given a public key pk and a message m, the encryption algorithm outputs a ciphertext c.
- HE.Eval(pk, f, c, c') \to c_{eval} \text{ : Given a public key pk, two ciphertexts } c, c', \text{ and the homomorphic function } f, \text{ the evaluation algorithm outputs the evaluated ciphertext } c_{eval} = f(c, c').
- HE.Dec(sk, c) \to m: Given a ciphertext c encrypted under pk and the corresponding secret key sk, the decryption algorithm outputs the message m.

Note, the HE.Eval algorithm homomorphically performs a defined function \( f \) on the ciphertexts. This function is constructed using HE.EvalAdd and HE.EvalMult which are homomorphic addition and multiplication respectively. Many HE schemes instantiate these common algorithms for integer-based computations, such as the BGV [21] and BFV [22], [23] schemes. Another HE scheme, the CKKS scheme [24], computes on fixed-point arithmetic, which is most appropriate for scientific research that often deals with floating-point data. Most well-known HE libraries (e.g., Palisade [25], Microsoft SEAL [26], and HElib [27]) have support for CKKS. Further reading on the construction of HE schemes can be found in the following survey [28]–[32].

Since we cannot homomorphically evaluate non-linear functions (i.e., \( \text{ACT, POOL} \)), we construct our CNNs differently, replacing non-linear layers with HE-friendly alternatives. For the HE-friendly pooling layers, we use sum-pooling or scaled-mean pooling proposed by CryptoNets [5] to avoid the division in a typical average-pooling layer. In this paper, we focus on finding effective polynomial activation functions to ensure high classification accuracy.

We assume a system model with two phases: training and inference, as illustrated in Fig. 5. In the training phase, a set of unencrypted labeled images are used to train the CNN model and obtain the weights \( \omega \) and the biases \( \beta \). The training starts with typical activation function. Then, the weights and biases are fine-tuned by replacing the activation function with its HE-friendly approximated polynomial. Detailed view of this process is shown in Fig. 12. In this paper, we do not focus on our discussion on the training since it is performed on data in the clear. We focus on the inference phase that is performed in the encrypted domain. Specifically, the output weight vector from the training phase is homomorphically encrypted and used in the secure evaluation of the CNN on homomorphically encrypted images with HE-friendly activation functions.

C. POLYNOMIAL APPROXIMATION

We apply methods in numerical approximation to construct HE-friendly activation functions for CNNs. In numerical approximation, polynomials are used as elementary elements to approximate other functions. In this section, we introduce three approximation methods used in our research, including Taylor expansion, best uniform approximation, best square approximation. These polynomial approximation methods are supported by the theoretical foundations, including Weierstrass approximation theorem [33], Stone-Weierstrass theorem [34], and Haar theorem [35].

1) Taylor expansion

Taylor’s Theorem [36] is the most elementary approximation method in numerical analysis. It expands a k times differential function into the general k-th order Taylor polynomial \( T_n(x) = \sum_{n=0}^{k} \frac{f^{(n)}(x_0)}{n!}(x-x_0)^n + R_k \) around a given point \( x_0 \) with \( k \in \mathbb{Z} \). Then we define \( \sum_{n=0}^{k} \frac{f^{(n)}(x_0)}{n!}(x-x_0)^n \) as Taylor Series and \( R_k = \frac{f^{(n+1)}(t)}{(n+1)!}(x-x_0)^n \text{ as the remainder of Taylor polynomial. Taylor expansion has been widely used in computational optimization because it can convert complex mathematical expressions into Taylor polynomials with only addition and multiplication operations preserved [37], [38].}

2) Best uniform approximation

Best uniform approximation is based on \( L^\infty \) norm, defined as \( \|f(x)\|_\infty = \max_{x \in [a, b]} |f(x)| \), where \([a, b]\) is a region of interest as shown in Fig. 6(a). It aims to minimize the maximum distance between \( f(x) \) and approximation polynomial \( p(x) \). In Fig 6(a), we denote the error function in best uniform approximation as \( \epsilon = \|f(x) - p(x)\|_\infty = \max \{|f(x) - g(x)|\} \). To generate the approximation polynomial, we have to solve the optimization problem: \( \min \max \{|f(x) - g(x)|\} \). But since the problem is hard to solve, a general and practical way is to directly expand \( f(x) \) into Chebyshev series [39] taken as the
approximation polynomial. Chebyshev series is a class of orthogonal polynomials defined as \( T_n(x) = \cos(n \cdot \arccos(x)) \) with \( x \in [-1, 1] \). Given \( T_0(x) = 1 \) and \( T_1(x) = x \), it is convenient to calculate Chebyshev polynomials: \( T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \). Therefore, we can first expand \( f(x) \) into Chebyshev series in \([-1, 1]\). Then, we calculate its Chebyshev coefficients \( c_k = \frac{1}{2} f^{(k)}(1) \) and adjust approximation domain from \([-1, 1]\) to \([a, b]\), for the \( k\)-th order Taylor polynomial. Algorithm 2 gives an elaborate description of the Chebyshev approximation.

**Input:** \( c_k, T_k(x) \)  
**Output:** \( f(x) \)  
1. Rewrite \( f(x) \) into Chebyshev Series form:  
   \[
   f(x) = \sum_{k=0}^{n} c_k T_k(x), \quad x \in [-1, 1]
   \]  
2. Compute Chebyshev coeff.: \( c_k = \frac{1}{2} f^{(k)}(1) \frac{T_k(x)}{\sqrt{1-x^2}} \)  
3. Convert approximation domain from \([-1, 1]\) to \([a, b]\):  
   \[
   x = \frac{2x' - a - b}{b - a}
   \]

**Algorithm 2:** Best Uniform Approximation

3) Best squares approximation  
Best squares approximation is another approximation method. It is based on \( L^2 \) norm, defined as \( \|f(x)\|_2 = \left( \int_a^b (f(x))^2 \, dx \right)^{1/2} \), where \([a, b]\) is the region of interest as shown in Fig. 6(b). The method aims to minimize the area between \( f(x) \) and approximation polynomial \( p(x) \). In Fig. 6(b), we denote \( A \) as the total area between \( f(x) \) and \( p(x) \) in \([a, b]\). We can also define the error function in best squares approximation as \( \epsilon = \|f(x) - p(x)\|_2 = \int_a^b (f(x) - p(x))^2 \, dx \). In our experiment, given discrete data points, we use least squares method [40] to approximate \( f(x) \). In Section V, the novel weighted method is an extension of least square approximation. Instead of evenly getting data points from \( f(x) \) in a given domain, we collect points in a weighted way.

**III. RELATED WORK**

After CryptoNets [5] demonstrated the ability to homomorphically evaluate neural network on encrypted data for secure image classification, several follow-up work [7]–[9], [41] attempted to improve the classification accuracy by extending the depth of the neural network.

However, a critical flaw in the CryptoNets design is the use of a non-linear monomial (i.e., square function \( x^2 \)) instead of those well-performing non-linear activation functions (Sec. II-A3). This substitution caused CryptoNets inability to train a deep network. As reported by the authors, CryptoNets cannot sustain a network with more than two \texttt{CONV-ACT-POOL} layers while maintaining high efficiency and accuracy. The reason is that the square function \( x^2 \) is not an effective activation function because it has unbounded derivative. Also, the use of sum-pooling (to avoid division operation) instead of average- or max-pooling added some negative impacts to the accuracy. As a result, CryptoNets achieved 98.95% on the MNIST handwritten digit dataset, which is not too far from the state-of-the-art result of 99.77% trained and tested on plaintext image data. Fig. 3 shows how the square function used in CryptoNets behaves in comparison to other well-performing activation functions.

With a goal to achieve higher accuracy, Chabanne et al. [7] attempted to approximate ReLU activation function using Taylor series polynomials (see Sec. II). Fig. 3 shows their best-performing approximated polynomial with degree 4; i.e., \texttt{Polyfit4}. This relatively low degree polynomial allows their neural network to be evaluated efficiently on en-
crypted data, because the required multiplicative depth is low and homomorphic evaluation is relatively efficient. However, this polynomial grows to infinity rather quick even for small values on the x-axis. This means the outputs of an activation layer can be very large causing a problem in training the network. To address this problem, Chabanne et al. proposed to use a batch normalization (BAT) layer before every activation layer, as illustrated in Fig. 1, to normalize the inputs of the activation layer into a specific range. Also, Chabanne et al. proposed a two-stage training process where the network is first trained with the original activation function to achieve stable training. In the second stage the activation function is replaced with the approximation of the activation and the weights are fine-tuned by continuing training the network at a low learning rate. Using these new methods, they improved the accuracy 99.30% on the MNIST dataset.

Hesamifard et al. proposed CryptoDL [8] for more complex dataset, CIFAR-10 [12], in addition to the MNIST dataset. Based on the conclusions from CryptoNets, CryptoDL attempted to find HE-friendly activation functions that have a bounded derivative. For this reason, their approach focused on approximating the derivative of a bounded function (i.e., Sigmoid) and using its integral as the HE-friendly activation function, as illustrated in Fig. 7. The integral of the approximation of Sigmoid behaves similar to a Softplus which is also a good activation function. Using this approach, CryptoDL achieved a classification accuracy of 91.55% on the CIFAR-10 dataset.

Modern deep learning architectures are significantly deeper than the networks discussed in the above approaches. Faster CryptoNets [9] presented a practical implementation of deep CNNs that can process on encrypted data using transfer learning. In transfer learning, a pre-trained network is modified by retraining the final layers on the data of the new task. Their proposed approach used a 50 layers residual network (ResNet) where only the final few layers perform inference on encrypted data; i.e., the feature maps are encrypted. The initial layers process on plaintext data. The authors tested this new method on a diabetic retinopathy dataset [42] and achieved a classification accuracy of 76.47% (Baseline ResNet scoring 80.61%).

Instead of using Taylor series approximation method, Chabanne et al. suggested to make the coefficients of the polynomial as trainable parameters as well. Wu et al. [41] explored this idea and trained a model with the polynomial coefficients in each activation layer as trainable parameters. While the model achieved a classification accuracy of 99.70%, it is at the expense of optimizing a large number of parameters. More importantly, the trained model may not be widely applicable in practice because this model is too specific to the training set.

In this paper, we investigate the research questions of whether we can systematically find effective and HE-friendly activation functions. To better explain our findings, let’s first review what contribute to an effective activation functions in CNNs.

IV. ANALYZING EFFECTIVE ACTIVATION FUNCTIONS

As discussed in Section II, the primarily role of the activation layer is to capture the non-linearity of complex features in CNNs. Naturally, many effective activation functions are non-linear.

A linear function \( f(x) \) is one which satisfy both of the following properties: \( f(x + y) = f(x) + f(y) \) and \( f(\alpha x) = \alpha f(x) \). Nonlinear functions are those who do not follow the above definition. Complex tasks such as classification of images or speech involves the separation of non-linear data and can be performed well only using non-linear models. Non-linear activation functions such as ReLU and Sigmoid enable neural networks with the capability to perform this task. This is because neural networks with effective non-linear activation functions are universal function approximators [43]. Every complex task can be abstracted as a function that maps an input to an output. Without the use of such activation functions, the networks would essentially be a linear model.

A. DIFFERENTIABLE

Backpropagation is the most popular and effective training method that have been used for neural networks. This training method makes use of the derivatives of the functions used in the network to calculate how much to change the weights during optimization. Due to this reason, it is necessary for the activation function to be differentiable.

A function \( f \) is said to be differentiable if the derivative \( f'(x) \) exists. This property ensures that the derivative is defined and exists at every value. The functions Softplus, Sigmoid and Tanh and their derivatives are differentiable over the entire domain. However, activation functions including ReLU and Leaky ReLU are only piecewise differentiable because they are piecewise functions and do not have derivatives at the origin point. In practical application of backpropagation, we just set the value of the derivative of ReLU at the origin point as zero.

B. CONTINUOUS

A function \( f(x) \) is said to be continuous at a given point \( x_0 \) if \( \lim_{x \to x_0} f(x) = f(x_0) \). For example, the function \( f(x) = \frac{1}{x} \) is continuous on the domain \( \mathbb{R} \setminus \{0\} \), but is not continuous...
over the domain $\mathcal{R}$ because it is undefined at $x = 0$. If an activation function $g(x)$ is differentiable with $x \in [a, b]$, then based on the necessity theorem, we can state that they are also continuous over the domain $[a, b]$. Therefore, if we can generate a differential activation function within $[a, b]$, then it must also continuous on $[a, b]$. In normal unencrypted CNNs, since we have showed the differentiable property of Softplus, Sigmoid and Tanh, we can also state that they are continuous over domain $[a, b]$. Although activation functions including ReLU and Leaky ReLU are non-differentiable, they are continuous over the entire domain. Polynomial is also a classic class of continuous functions over the domain $\mathcal{R}$ and linear combination of polynomials is also continuous. So it is a good way to use polynomials as main approximation component.

C. MONOTONIC

A function is $f$ monotonic when it is either increasing or decreasing, i.e., the function always grows in a single direction. During training, a neuron's weight might be changed to increase or decrease its influence on neurons in the next layer. A monotonic activation function makes this behaviour predictable and thus enables faster optimization. Using a non-monotonic activation function might have the opposite effect because of how non-monotonic functions change in direction of growth. Most of the effective activation functions such as ReLU, Sigmoid, Tanh, and Swish are monotonic in nature.

However, it is important to note that neural networks with non-monotonic activation functions can be optimized as well – The network might require a longer training time. Monotonic trigonometric functions such as the periodic sine wave have been used as activation functions to train neural networks successfully [44].

D. BOUNDED DERIVATIVE

It has been observed that activation functions with bounded derivatives contribute to effective performance in neural networks. This is because a bounded derivative restricts the training algorithm from making large updates to the weights of the network [45]. Not preventing this phenomenon could lead to large fluctuations in the network weights and create instability during the training process.

With this intuition, a common property seen among well-performing activations is a bounded derivative. The same property can be seen in well-performing activations discovered by Ramachandran et al. [46]. The research used a reinforcement learning technique to find effective activation functions from a search space of functions. The search space was composed of a combination of linear and non-linear functions such as $x^2$, $\sin(x)$, $x_1 + x_2$. Every well-performing activation discovered using this process has a bounded derivative. The most effective activation discovered was Swish. Defined as $x \cdot \text{sigmoid}(\beta x)$, Fig. 8(c) shows how the derivative of Swish stays bounded with different values of $\beta$, which the network can adjust as a trainable parameter. Figure 8(b) shows the bounded derivatives of the effective activation functions ReLU, Sigmoid and Tanh (Fig. 8(a)).

E. ZERO-CENTERED

It has been long known that neural networks can learn faster if activation functions in hidden layers are centered around zero [47], [48]. LeCun et al. [47] gave a strict proof to the zero-centered property of effective activation functions. In this section, we provided a more intuitive interpretation to explain this observation.

We first denote $\vec{x} = (x_1, x_2, \ldots, x_n)$ as a $n$-dimensional input vector and $\vec{w}_1 = (w_1, w_2, \ldots, w_n)$ as an initial weight vector. Then with given threshold $b$, we can define a neuron model as an input-output map $f(\vec{x}; \vec{w}, b)$, satisfying $f(\vec{z}) = f(\sum_{i=1}^n w_i x_i + b)$. The gradient descent process of each parameter $w_i \in \vec{w}$ based on the given cost function $L(\vec{x})$ and learning rate $\eta$, can be shown as $w_{i+1} = w_i - \eta \frac{\partial L}{\partial w_i} = w_i - \eta \frac{\partial L}{\partial x_i} = w_i - \eta \frac{\partial L}{\partial z} \frac{\partial z}{\partial x_i}$. Hence, the renewal equation above shows that the updated direction of parameter $w_i$ is entirely based on the value of $x_i$ because parameters including $\eta$, $\frac{\partial L}{\partial w_i}$ and $\frac{\partial L}{\partial x_i}$ can all be seen as constant term.

To explain the zero-centered property, we further assumed that the optimal weight vector $\vec{w}^* = (w_1^*, w_2^*, \ldots, w_n^*) \neq \vec{w}_1$ and the previous neuron unit takes Sigmoid $g(x) = \frac{1}{1+e^{-x}}$, a typical non-zero-centered function, as activation functions. Then, due to $g(x) \in (0, 1)$, we can know that $x_i > 0$ for each input value $x_i \in \vec{x}$, $i = 1, \ldots, n$, which will cause the renewal process of each $w_i \in \vec{w}_1$ to take a z-path from the initial weight vector $\vec{w}_1$ to the optimal weight vector $\vec{w}^*$ and the learning speed of neural networks will be much slower.

F. PIECEWISE

In literature, recent work showed that we can design effective activation functions in a piecewise manner [49]. Nonzero-centered activation functions including Sigmoid suffer from the vanishing gradient problem that can cause the learning process of neural networks to slow. But an exception within these functions is ReLU. ReLU can greatly speed up the learning process because of its rectification for inputs $x \leq 0$. However, it will lead to another so-called "Dead ReLU" problem, which prevents networks from learning efficient weights by not activating in the early stage for any input $x \leq 0$. A solution to this problem is to use an activation function with a non-zero slope in its derivative.

Qin et al. [16] introduced the improved Sigmoid function — Isigmoid, as shown in Figs. 9(a) and (b). The function adapts the Sigmoid activation with tunable parameters to have a non-zero slope in its derivative.

$\text{Isigmoid}(x) = \begin{cases} \alpha(x - \beta) + \text{Sigmoid}(\beta) & x \geq \beta \\ \text{Sigmoid}(x) & -\beta < x < \beta \\ \alpha(x + \beta) + \text{Sigmoid}(-\beta) & x \leq \beta \end{cases}$

The derivative of Isigmoid is calculated as

$\text{Isigmoid}'(x) = \begin{cases} \alpha & |x| \geq \beta \\ \text{Sigmoid}'(x) & x < \beta \end{cases}$
V. FINDING EFFECTIVE POLYNOMIAL ACTIVATION

Recent works have tried to create effective polynomial activations and have achieved reasonable results. Chabanne et al. \cite{7} demonstrated a classification accuracy of 99.30\% with a degree 2 polynomial on the MNIST dataset and CryptoDL achieved 99.52\% classification accuracy on the MNIST dataset and 91.55\% classification accuracy on the CIFAR-10 dataset.

Polynomial activation functions are differentiable, continuous and non-linear but are not monotonic and do not have a bounded derivative. In order to achieve these results, some additional steps have been implemented. Some of these include using a batch normalization layer to bound derivatives, mimicking the structure of effective activation functions and using a two-phase training approach for better optimization. It is also necessary for the approximations to be a low degree polynomial to maintain an acceptable multiplicative depth for homomorphic encryption. In the following paragraphs, we explain in detail how these techniques contribute to effective polynomial activations.

Polynomial functions with a degree greater than 2 do not have bounded derivatives. Monomials and Polynomials with coefficients set to 1 grow very fast and consequently have large outputs. Using these functions without any change to the coefficients lead to training instability and results in sub-optimal network performance. CryptoNets reported this behavior and cause when they used the monomial \(x^2\) as their activation function. This effect, however, can be partially controlled by modifying the structure of the polynomial and controlling the range of inputs to the function. The rapid growth of the polynomial can be controlled by approximating an activation function such as ReLU. The resulting polynomial function is more suitable as it closely replicates the steady growth of ReLU as shown in Fig. 10(a).

One thing in common among these two example activation functions is that authors tuned the saturated region of the input data with tunable parameters. Rather than flatten out after some ranges, these new activation functions incorporate a small slope, in both the negative and positive regions, to address the vanishing gradient and bias shift problem. Other functions, such as Leaky ReLU and Swish, also follow the same principle to tackle these problems.

where \(\text{Sigmoid}'(x) = e^{-x}/(1 + e^{-x})^2\).

Wang et al. \cite{17} proposed ReLTanh, which treats the negative and positive saturation region as two straight lines each with different slopes, as shown in Figs. 9 (c) and (d).

\[
\text{ReLTanh}(x) = \begin{cases} \frac{Tanh'(\alpha)(x - \alpha) + Tanh(\alpha)}{\alpha} & x \geq \alpha \\ \frac{Tanh(x)}{\beta} & \beta < x < \alpha \\ \frac{Tanh'(\beta)(x - \beta) + Tanh(\beta)}{\beta} & x \leq \beta \end{cases}
\]

where \(Tanh'(x) = 4/(e^x + e^{-x})^2\). The derivative of ReLTanh is calculated as

\[
\text{ReLTanh}'(x) = \begin{cases} -Tanh''(\alpha) & x \geq \alpha \\ Tanh'(x) & \beta < x < \alpha \\ Tanh''(\beta) & x \leq \beta \end{cases}
\]

where \(Tanh''(x) = 8(e^{-2x} - e^{2x})/(e^x + e^{-x})^4\). In this function, \(\alpha\) and \(\beta\) are thresholds that determine the start positions of the non-zero slope and the amount of slope. The network can also learn these thresholds during training to further reduce the output error.

One thing in common among these two example activation functions is that authors tuned the saturated region of the input data with tunable parameters. Rather than flatten out after some ranges, these new activation functions incorporate a small slope, in both the negative and positive regions, to address the vanishing gradient and bias shift problem. Other functions, such as Leaky ReLU and Swish, also follow the same principle to tackle these problems.
A. ANALYSING BATCH NORMALIZATION OUTPUTS

The proposed approach by Chabanne et al. to train networks polynomial activation uses a batch normalization (BN) layer before the polynomial activation layer. The BN layer transforms the input data with an arbitrary range into a distribution with zero mean and unit variance. A distribution with zero mean and unit variance will have 99.73% of its data between the range [-3, 3] as shown in Fig. 10(b). Chabanne et al. used polynomial approximations of ReLU within this range and trained it on the MNIST dataset.

From the results in Table 2, we can observe that polynomial approximations between [-3, 3] achieve classification accuracies similar to those reported by Chabanne et al. However, when using the same activation function for FMNIST and CIFAR-10, we observe sub-optimal performance. To understand the effect of different ranges, we also tested two polynomial activations approximated between [-7, 7] and [-25, 25] on CIFAR-10.

From these experiments and their results, we make two observations. We observe an increase in classification accuracy using polynomials between a larger range from Table 2. This observation is contrary to the notion that polynomial approximations with lower error of approximations contribute to higher performance in CNNs. Simultaneously, it is also clear that too large a range (such as [-25, 25]) has a negative impact on performance. To understand this impact on performance, we investigate the inputs to the polynomial activation layer.

The inputs to the polynomial activation are the normalized outputs from the previous BN layer. Upon direct comparison of output distributions across datasets and layers, we observe the structures of these distributions to be similar. However, the range of the distributions differ, with a significantly larger range associated with the CIFAR-10 dataset. Table 3 lists the distribution ranges and standard deviations observed in the output distributions across different datasets.

<table>
<thead>
<tr>
<th>Range</th>
<th>Dataset</th>
<th>Accuracy</th>
<th>Approximation Error in [-3, 3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[-3, 3]</td>
<td>MNIST</td>
<td>99.45</td>
<td>4.187</td>
</tr>
<tr>
<td>[-3, 3]</td>
<td>FMNIST</td>
<td>97.75</td>
<td>4.187</td>
</tr>
<tr>
<td>[-3, 3]</td>
<td>CIFAR-10</td>
<td>81.32</td>
<td>4.187</td>
</tr>
<tr>
<td>[-7, 7]</td>
<td>CIFAR-10</td>
<td>88.25</td>
<td>4.508</td>
</tr>
<tr>
<td>[-25, 25]</td>
<td>CIFAR-10</td>
<td>83.45</td>
<td>5.392</td>
</tr>
</tbody>
</table>

From Table 3, we can see that while more than 99% of data lies between [-3, 3], the range of the distribution is...
much larger and increases with the complexity of the dataset. We suspect that the polynomial activations approximated between larger ranges in Table 2 perform better because they cover more data in the distribution. However, from the same experiment, we can also see that arbitrary large ranges can negatively impact performance. This is because extremely large ranges would have a high error of approximation, especially between [-3, 3]. Therefore, we can conclude that it is important to strike a balance between maintaining an acceptable error of approximation between [-3, 3], while covering the largest range possible. We examine this balance in detail using a grid search to discover where this optimal points lies in Section VI-C.

B. WEIGHTED POLYNOMIAL APPROXIMATION

In the previous section, we observe that using a polynomial approximation between a range larger than [-3, 3] contributes positively to CNN performance. However, a large increase in range does not help, but instead, negatively impacts the performance of the model. This is because the increase in approximation range results in an increase in approximation error. Moreover, around 99% of data that the polynomial function processes lies between the crucial range of [-3, 3]. We propose a simple method to generate polynomials that maintain an acceptable quality of approximation within the crucial range while covering a large range.

The standard approach to approximate a function using a polynomial with least squares approximation is to fit to linearly sampled set of data points. More formally, to approximate a non-linear function \( f \) between a range, we first sample \( f \) as follows:

\[
Y = \{ f(x) : x \in X \}
\]

where \( X = \{ L_{low}, \ldots, L_{high}\} \) is the set of linearly spaced points. Figure 11(a) depicts this procedure with a 100 sample points. Then, using a polynomial regression function like `polyfit` from MATLAB, a polynomial of degree \( n \) is fit to \( X \) and \( Y \) using an error measure such as least-squares.

We propose a weighted approximation technique to maintain an acceptable error of approximation between [-3, 3] in large range approximations. Our method takes advantage of the input data structure where around 99% of data lies between the range [-3, 3]. Instead of sampling the function linearly, we sample the range [-3, 3] at a relatively higher rate than the remaining sections using a weight. More formally, the activation function \( f \) is sampled between the range \([L_{low}, L_{high}]\) where

- \( X = [X_1 X_2 X_3] \)
- \( X_1 = \{ L_{low}, \ldots, -3 \} \) containing \( \frac{(1-R)n}{2} \) linearly spaced points,
- \( X_2 = \{ -3, \ldots, 3 \} \) containing \( Rn \) linearly spaced points,
- \( X_3 = \{ 3, \ldots, L_{high} \} \) containing \( \frac{(1-R)n}{2} \) linearly spaced points,
- \( R \in [0, 1] \) is the weight determining the rate of sampling between [-3, 3].

The sampling rates for each region in the method proposed above can be calculated as follows:

- \( \frac{1}{2} \) for the region \( X_2 \)
- \( \frac{2}{n} \frac{L_{high} - 3}{1 - R} \) for regions \( X_1 \) and \( X_3 \), given \( |L_{high}| = |L_{low}| \)

Figure 11(b) visualizes this sampling approach with \( n = 100, R = 0.7, L_1 = -30 \) and \( L_2 = 30 \). We have chosen to visualize the sampling on the Sigmoid activation function because the difference is visually more apparent than ReLU or Softplus. However, the effect of the approach will affect approximations of any activation function.

![FIGURE 11. Improving quality of approximations using our proposed method](image)

### TABLE 4. Comparison of approximation error and ranges according to type of approximation

<table>
<thead>
<tr>
<th>Range</th>
<th>Approximation method</th>
<th>Error in [-3, 3]</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>[-5, 5]</td>
<td>Non-weighted</td>
<td>4.376</td>
<td>82.17</td>
</tr>
<tr>
<td>[-5, 5]</td>
<td>Weighted</td>
<td>4.354</td>
<td>85.81</td>
</tr>
<tr>
<td>[-11, 11]</td>
<td>Weighted</td>
<td>4.785</td>
<td>89.91</td>
</tr>
</tbody>
</table>

Using `polyfit` with the above sampling rates for each region specifically minimizes the error between [-3, 3] while approximating through a large range. The benefit of this
approach over using a linear sample of points is the larger range covered for the same error of approximation between $[-3, 3]$. Table 4 shows the benefits of using weighted approximations on the CIFAR-10 dataset. Comparing with the performance of the conventional approach, we can see that the reduced error between $[-5, 5]$ helps improve classification accuracy. As an added benefit, larger approximation ranges can be accommodated which improves training optimization, consequently improving performance.

One thing to note in our proposed approach is the expense of weighting the approximation towards $[-3, 3]$. Due to a higher weight in this section, the region of approximation outside $[-3, 3]$ would have a high error of approximation. However, due to the sparsity of inputs in this region, it does not adversely affect the performance of the model.

VI. EXPERIMENTS AND RESULTS

A. EXPERIMENT SETUP

For our experiments we used a system with a Xeon Silver 4114 CPU at 2.20 GHz with 192 GB of RAM and an NVIDIA Tesla P100 GPU running Ubuntu 16.04. All CNN models were constructed and run using the PyTorch 1.02 library. We used MATLAB’s `polyfit` to generate polynomial approximations for both the weighted and non-weighted approaches. For Chebyshev approximations, we used the PyProximation library used by Hesamifard et al.

B. NETWORK ARCHITECTURE AND TRAINING PROCEDURE

To train on MNIST and FMNIST with polynomial activations, we adapt the Deep CNN model proposed by Chabanne et al. Figure 12(a) shows an overview of the model with 14 layers and 6 activation layers. Similar to Chabanne et al. we configure the network as follows –

- Block 1 contains two convolutional layers with with 32 filters of size 3 x 3 followed by an average pool.
- Block 2 and Block 3 follow are similar to block 1 but have 64 and 128 filters of size 3 x 3 for the convolutional layers respectively.
- Following these convolutional blocks is a fully connected layer with 256 neurons.
- After a dropout layer with $p=0.5$, the network results are parsed from a final fully connected layer of 10 neurons.
- All convolutional layers are followed by a batch normalization layer which is followed by an activation layer.

We also train on CIFAR-10 and use the architecture proposed by Springenberg et al. [50] shown in Fig. 12(b). Unlike the previous network, this architecture does not use pooling layers. Instead the last layers of Block 1 and 2 use convolutional layers with stride = 2 to operate like pooling layers. It contains 18 layers with 9 activation layers and the following details –

- Block 1 contains three convolutional layers with 96 filters of size 3 x 3. To emulate a pooling layer, the final convolutional layer has a stride = 2 which reduces the resolution of the outputs by half.
- Similar to block 1, block 2 also contains 3 convolutional layers but with 192 filters of size 3 x 3. The final convolutional layer has stride = 2.
- The next block contains a single convolutional layer with 192 filters of size 3 x 3. This is followed by another convolutional layer with 1 x 1 convolutions.
- The final convolutional layer reduces the number of outputs by only containing 10 filters. A final global average pooling function gathers the results of the network.

For both models, we use the training procedure proposed by Chabanne et al. explained in Section V. The model is initially trained using a traditional activation function like Softplus. The activation function is then replaced with its polynomial approximation, and the model is further trained with the pre-trained weights as a starting point. The training on MNIST using the original activation is conducted for 350
epochs with an initial learning rate of 0.01. Once the activations have been replaced, the training continues at \( lr = 10^{-6} \) for 200 epochs.

For CIFAR-10 however, the first phase of training is performed for 450 epochs with a learning rate scheduler scaling the learning rate by 0.1 at epochs 300, 350 and 400. After replacing the activation function with an approximation, the training continues at \( lr = 10^{-6} \) for 300 epochs with the scheduler scaling the learning rate by 0.1 at epochs 150, 200 and 250.

C. SEARCHING FOR THE OPTIMAL POINT

In table 2 we observe that a polynomial approximation with a range larger than \([-3, 3]\) yields better performance on CIFAR-10. Simultaneously, we also observe that model performance drops with significantly larger ranges. This shows that merely approximating a polynomial between arbitrary large ranges is not effective. There exists some range beyond \([-3, 3]\) at which the network would perform most optimally. It is also necessary to observe the consistency of this behaviour across different settings. Neural network instances performances can highly vary based on the dataset, network structure and the type of activation used. In the following paragraphs, we construct and test multiple experiments to observe the consistency of this results across the following parameters:

- Method of polynomial approximations
- High and low degree of approximations
- Type of activation function approximated
- Dataset for the CNN
- Structure of the CNN

To understand where the optimal range of approximation lies, we perform a grid search across ranges. Multiple polynomial candidates are approximated between the ranges of \([-L, L]\) where \(L \in \{3, 5, 7, \ldots, N\}\) and \(N\) is the observed absolute maximum in the BN output distribution for that dataset 3. As explained in VI-B, every network instance uses the pre-trained weights of the corresponding activation as a starting point. However, due to the large number of tests in this grid search, we restrict the number of epochs for the second stage of training to 50.

In our first set of experiments, we examine the effect of different activation functions, approximation methods and degrees. The polynomial candidates are approximated at two degrees, 4 and 7 and using three approximation methods:

- Fitting a polynomial to a linear set of points using the least-squares approach used by Chabanne et al.
- Chebyshev polynomial approximations proposed by Hesamifard et al. We use the measure \( \mu = e^{\frac{1}{10} \frac{1}{x^2 + 5 + x^2}} \) specified in CryptoDL to approximate the derivative of the activation. The integral of the generated approximation is then used as the polynomial activation in the network.
- Our proposed weighted polynomial approximation with \( R = 0.7 \)

We approximate 15 candidates of both Softplus and ReLU between the ranges \([-3, 3]\) and \([-31, 31]\). In total, we train 180 network instances on CIFAR-10 with the epochs in phase 2 restricted to 50. Figure 13 shows the performance achieved using each candidate against every corresponding range and approximation method. We considered testing approximations of Sigmoid and Tanh along with ReLU and Softplus. However, as mentioned in Section IV, these activation functions suffer from the vanishing gradient problem. For this reason, they perform significantly worse than ReLU and Softplus and hence, we choose not to explore their approximations.

We conduct another set of experiments to examine if our observations are consistent with different datasets and network structures. To reduce the number of networks to be trained, we approximate candidates of only the Softplus activation function using our proposed weighted approximation technique. These candidates are tested on three datasets, MNIST, FMNIST and CIFAR-10 using different network configurations described in Section VI-B. Due to space limitation, we summarize the results performed on CIFAR-10 in Table 5.

![Figure 13](image_url)

The first observation we make from Fig. 13 is that the optimal range of approximation for every candidate lies beyond \([-3, 3]\). Larger ranges of approximation yields a higher error of approximation between \([-3, 3]\). Despite this increase in error at the optimal range, we observe similar behaviour regardless of the approximation method, the base activation function, network structure or dataset. From these experiments, we understand that solely minimizing the error of approximation between \([-3, 3]\) does not yield the best performing candidates. Even though more than 99% of data lies between the range of \([-3, 3]\), it is necessary to take into account the range of the input distribution.

However, the performance trend rapidly changes after the optimal range of approximation. From Fig. 13 we can see a drop in performance beyond the optimal range, especially for lower degree candidates. Beyond the optimal range, the incentive to cover larger ranges is lost because the error of approximation between \([-3, 3]\) increases beyond an acceptable level. Table 4 lists error of approximations between \([-3, 3]\) for the candidates at each optimal range.
We also observe that our proposed weighted approximation technique performed the best among all the compared methods. By forcing the approximation to weight the region between \([-3, 3]\) over the remaining region, this approximation method achieves lower error of approximations while covering larger ranges. For example, Chabanne et al. achieved an accuracy of 97.91\% using a degree 4 polynomial activation approximated between \([-3, 3]\). At the optimal point, degree 4 weighted approximations were able to achieve a 99.59\% classification accuracy on the MNIST dataset.

While the Softplus activation performs slightly worse than ReLU, we observe that approximations of Softplus perform significantly better than the approximations of ReLU, regardless of the approximation method. This is because the approximations are able to fit the smooth curve of Softplus better than the sudden change in the slope of ReLU at \(x = 0\).

Candidates generated by our technique yield better performance because they cover a larger range of approximation while maintaining an acceptable level of error as seen in Fig. 13. This is especially observed in the more complex datasets, FMNIST and CIFAR-10, because they have a larger input distribution to the activation function.

### VII. CONCLUSION

In this work, we introduced a novel approach to generate HE-friendly non-linear activation functions using polynomial approximations of Softplus. We also showed weighted approximations of activation functions are effective regardless of the degree of approximation and the training method. Therefore, an improvement in classification accuracy can be achieved by first improving the performance of the network using Softplus – Changing the network structure or hyperparameter tuning can help in this case. In order to reduce the instability while training using a family of polynomials, the weights of the network could be fine-tuned by training each layer sequentially. In this process, we would start with replacing only the first activation layer and fine-tune the weights. After convergence, the weights before the first layer could be frozen and the next layers can be trained in the similar way, one by one. Another area to investigate is the misalignment of the approximated activation and the Softplus functions in the \(y\)-axis. While this can be easily fixed by the addition of a constant, we expect the bias to absorb this difference. We do not know, however, the effect of the presence of the constant on the learning process.

### REFERENCES


