On Polynomial Approximations for Privacy-Preserving and Verifiable ReLU Networks

Abstract

Outsourcing deep neural networks (DNNs) inference tasks to an untrusted cloud raises data privacy and integrity concerns. While there are many techniques to ensure privacy and integrity for polynomial-based computations, DNNs involve non-polynomial computations. To address these challenges, several privacy-preserving and verifiable inference techniques have been proposed based on replacing the non-polynomial activation functions such as the rectified linear unit (ReLU) function with polynomial activation functions. Such techniques usually require polynomials with integer coefficients or polynomials over finite fields. Motivated by such requirements, several works proposed replacing the ReLU activation function with the square activation function. In this work, we empirically show that the square function is not the best degree-2 polynomial that can replace the ReLU function even when restricting the polynomials to have integer coefficients. We instead propose a degree-2 polynomial activation function with a first order term and empirically show that it can lead to much better models. Our experiments on the CIFAR and Tiny ImageNet datasets on various architectures such as VGG16 show that our proposed activation function improves the test accuracy by up to 10.4% compared to the square function.

1 Introduction

Offloading computationally-demanding learning and inference tasks to the cloud has become a necessity, but this presents several privacy and integrity risks (Amazon 2021; Microsoft 2021; Google 2021). Privacy-sensitive user's data such as medical images must not be revealed to the cloud, hence it should be first encrypted and the inference can be performed on the encrypted data. In addition, the cloud also may wish to keep its model confidential from the clients. Furthermore, since an untrusted or unreliable cloud may return incorrect inference results, the user must also be able to verify the correctness of the results.

While there are many efficient privacy-preserving (Rivest et al. 1978; Gentry 2009) and verifiable computing techniques (Lund et al. 1992; Bos et al. 2013) for polynomial-based computations in the literature, neural networks involve non-polynomial functions such as the rectified linear unit (ReLU) activation function, the Sigmoid activation function and the max-pooling layers. Moreover, many techniques even require polynomial with *integer coefficients* or polynomials over a *finite field* (Lund et al. 1992; Thaler 2013).

Several works (Xie et al. 2014; Gilad-Bachrach et al. 2016; Ghodsi, Gu, and Garg 2017; Mohassel and Zhang; Liu et al. 2017) address these challenges by replacing the non-polynomial functions in neural networks with a polynomial functions. Then, the polynomial-based techniques for privacy-preserving and verifiable machine learning can be readily applied. Specifically, the ReLU function $\sigma_r(x) = \max(x, 0)$ is usually replaced with the square activation function

$$\sigma_{\text{square}}(x) = x^2$$

and the max-pooling layers are usually replaced with sum-pooling layers.

The rationale behind choosing the square function in particular as pointed out in (Gilad-Bachrach et al. 2016) is that it is a lowest-degree non-linear polynomial function. In addition, some prior works on overparameterized polynomial networks suggested that neural networks with square activations are as expressive as networks with threshold activations (Livni, Shalev-Shwartz, and Shamir 2014; Gautier, Nguyen, and Hein 2016). Furthermore, several experiments illustrated that the square activation function based network yields an accuracy that is comparable with the corresponding ReLU networks. These experiments, however, were performed on simple datasets such as the MNIST dataset (LeCun 1998) with networks that have a small number of activation layers.

While the square function works well for some experiments with small number of layers and it is commonly used in many privacy-preserving and verifiable frameworks, it is not clear if it is the best function that can replace the ReLU activation function. In fact, the error resulting from approximating the activation functions by polynomials in deep neural networks (DNNs) grows with the number of layers (Petrushev and Popov 2011; Telgarsky 2017). Hence, experiments with deeper networks and more realistic datasets are necessary to better assess the accuracy of the square activation function.

Contributions. In this work, we empirically show that replacing the ReLU function with the square function in DNNs may result in a severe degradation in the accuracy. Indeed, we empirically illustrate that the square function is not the best second-degree polynomial that can replace the ReLU function even when dealing with polynomials over finite fields. Specifically, our contributions are as follows.

1. We study the problem of approximating the ReLU function with polynomials with integer coefficients and polynomial over finite fields. We show that the ReLU function $\sigma_{\rm r}(x)$ cannot be uniformly approximated with a polynomial with integer coefficients on the interval $I = [-1, 1]$. In contrast, we show that the scaled ReLU function $\sigma_{\rm sr}(x; c) = c\ \sigma_{\rm r}(x)$ can be uniformly approximated on the interval $I = [-1, 1]$ when the constant c is even.
2. Motivated by this, we propose to replace the ReLU activation function in DNNs with a polynomial that uniformly approximates $\sigma_{\rm sr}(x; 2) = 2\ \sigma_{\rm r}(x)$. This results into a polynomial activation function that is given by
$$\sigma_{\rm poly}(x) = x^2 + x.$$
For large intervals, since the ReLU and the scaled ReLU functions are not uniformly approximable by polynomials with integer coefficients as we show later, we propose to uniformly approximate $\sigma_{\rm sr}(x; c)$ with a polynomial with real coefficients and round the resulting coefficients. When $I = [-a, a]$, our polynomial activation function is given by
$$\sigma_{\rm poly}(x) = x^2 + ax.$$
3. We empirically show that our polynomial function can lead to better models. Our experiments on the CIFAR-10, CIFAR-100, Tiny-ImageNet-10 and Tiny-ImageNet-200 datasets show significant accuracy improvement compared to the square activation function.
Specifically, our experiments on the DNN considered in (Liu et al. 2017) show that our polynomial activation function improves the test accuracy by 5.6% on CIFAR-10 and by 4.0% on CIFAR-100 compared to the square function. Moreover, on the "Network In Network (NIN)" architecture of (Lin, Chen, and Yan 2014) show that our polynomial function improves the test accuracy by 7.7% on CIFAR-10 and by 9.4% on CIFAR-100 compared to the square function.
In addition, our polynomial activation function improves the test accuracy by 5.8% on LeNet (LeCun 1998) and Tiny-ImageNet-10, and by 10.4% on VGG16 (Simonyan and Zisserman 2014) and Tiny-ImageNet-200.

Organization. The rest of this paper is organized as follows. In Section 2, we discuss the closely-related works. In Section 3, we discuss the feasibility approximating the ReLU function with a polynomial with integer coefficients and discuss the rationale behind choosing our activation function. We empirically show that our polynomial function leads to much better accuracy compared to the square function in Section 4. Finally, concluding remarks are discussed in Section 5.

2 Related Work

Numerous works considered privacy-preserving and verifiable inference for deep ReLU neural networks which can benefit from a better polynomial activation function than the square activation function and that is our goal in this work. In this section, we briefly review the closely-related works.

A straightforward approach to deal with the non-polynomial functions while keeping the user's data private is to use an interactive approach such that the user performs these non-polynomial computations as proposed in (Barni, Orlandi, and Piva 2006). This interactive approach, however, incurs significant communication and computation costs. More importantly, this approach leaks information about the cloud's model.

In order to avoid such costs, CryptoNets (Gilad-Bachrach et al. 2016) proposed a privacy-preserving inference technique for DNNs that keeps the user's data and also the cloud's model confidential. Such neural networks are known as oblivious neural networks (ONNs). Specifically, CryptoNets uses leveled homomorphic encryption techniques (Bos et al. 2013), replaces the ReLU function with the square function and the max-pooling layers with sum-pooling layers. The empirical evaluation of CryptoNets resulted in a training accuracy of 99% on the MNIST dataset using a 5-layer network, with only two square activation layers.

Several privacy-preserving and verifiable inference frameworks also focused on reducing the latency of the computations such as (Liu et al. 2017; Sanyal et al. 2018; Chou et al. 2018; Brutzkus, Gilad-Bachrach, and Elisha 2019; Mishra et al. 2020; Ghodsi et al. 2020). For instance, MiniONN (Liu et al. 2017) has considered the privacy-preserving inference problem for neural networks while requiring no changes to the training phase. Specifically, the goal of MiniONN is to transform an already trained neural network to an ONN without changing the training phase. Unlike CryptoNets which does not leak any information about the cloud's model, MiniONN reveals the architecture of the cloud's neural network in terms of the number of layers, number of nodes in each layer and the operations used in each layer.

Another line of work also focused on the integrity issue of the inference problem (Ghodsi, Gu, and Garg 2017; Chen et al. 2018; Zhao et al. 2021). In particular, SafetyNets (Ghodsi, Gu, and Garg 2017) proposed a verifiable inference approach for neural networks that can be represented as arithmetic circuits based on the sum-check protocol (Lund et al. 1992; Thaler 2013; Goldwasser, Kalai, and Rothblum 2015). Since such techniques require *polynomials over a finite field*, SafetyNets also replaces the ReLU function with the square function and the max-pooling layers with sum-pooling layers. The square activation function was shown to work well in a few experiments with three-layer and four-layer neural networks on the simple MNIST, MNIST-Back-Rand and the TIMIT speech recognition datasets (Garofolo 1993).

The closest-work to our work is CryptoDL (Hesamifard, Takabi, and Ghasemi 2017), which considered the problem of designing better polynomial approximations of the ReLU function over reals. Our work, however, is different from CryptoDL in the following aspects.

1. We consider *polynomials over finite fields* which is necessary for some works as SafetyNets (Ghodsi, Gu, and Garg 2017). This a novel aspect of our work which, to the best of our knowldege, has not been considered before beyond the square activation function.
2. The ReLU function in CryptoDL is approximated using a degree-3 polynomial. Specifically, the Sigmoid function is first approximated with a degree-2 polynomial. This

degree-2 polynomial is then integrated to get a degree-3 polynomial that approximates the ReLU function. Instead, we focus on polynomial approximations of degree-2 for a fair comparison with the square activation function.

3. Finally, when comparing between the different activation functions, (Hesamifard, Takabi, and Ghasemi 2017) changes the original network architecture by adding more layers to get closer to the performance of the baseline ReLU network. In contrast, our work does not change the network architecture while comparing between the different activation functions.
   This is an important feature of our work that allows for using the same baseline architecture without searching for a new architecture that is compatible with the polynomial functions. In addition, adding more layers to the network complicates the training and the inference further.

Finally, it is worth mentioning that some recent works also considered approximating the ReLU function through rational functions as they are more efficient (Telgarsky 2017; Boullé, Nakatsukasa, and Townsend 2020). Specifically, while an ϵ-close polynomial requires a degree of $\Omega(\text{poly}(1/\epsilon))$, rational functions only require a degree of $O(\text{poly}(\ln(1/\epsilon)))$. However, it is not clear how to use rational functions to develop privacy-preserving and verifiable frameworks.

3 Polynomial Approximations of the ReLU Function

In this section, we discuss the feasibility of uniformly approximating the ReLU function with a polynomial with integer coefficients and discuss the minimax polynomial approximation approach.

3.1 Can We Approximate the ReLU Function with a Polynomial with Integer Coefficients?

Since our goal is to replace the ReLU function with a polynomial with integer coefficients or a polynomial over finite field, we start by discussing the feasibility of doing so.
First, we note that uniform polynomial approximation when restricting the polynomial to have integer coefficients is not possible when the interval I is of length four or more (Ferguson 2006). We now recall this result from (Ferguson 2006).

Lemma 1. *If the interval I is of length four or more, then the only functions that can be uniformly approximated by polynomials with integer coefficient are those polynomials themselves.*

We next focus on approximations over smaller intervals. Based on Lemma 1, the only functions that can be uniformly approximated by polynomials with integer coefficient are those polynomials themselves on $I = [-2, 2]$. The natural question that we ask next then is whether we can approximate a real-valued continuous function f with a polynomial of integer coefficients on the interval $I = [-1, 1]$. It turns out that this is possible if and only if two conditions are satisfied as provided in Lemma 2 (Ferguson 2006).

Lemma 2. *For a continuous real-valued function f on the interval $I = [-1, 1]$ to be uniformly approximable by polynomials with integer coefficients it is necessary and sufficient that*

i. f is integer-valued at -1, 0, and 1, and
ii. the integers $f(-1)$ and $f(1)$ have the same parity.

Next, we show that the ReLU function cannot be uniformly approximated by a polynomial with integer coefficients as it does not satisfy the conditions of Lemma 2. However, scaling the ReLU function with an even number c leads to a function that is uniformly approximable by polynomials with integer coefficients.

Theorem 1. *(Uniform Approximation with Integer Coefficients of the ReLU Function)*

- *The ReLU function $\sigma_{\text{r}}(x) = \max(x, 0)$ is not uniformly approximable by polynomials with integer coefficients on the interval $I = [-1, 1]$.*
- *The scaled ReLU function $\sigma_{\text{sr}}(x; c) = \max(cx, 0)$, where c is an even number, is uniformly approximable by polynomials with integer coefficients on the interval $I = [-1, 1]$. Moreover, the degree-2 interpolating polynomial is given by*

$$q(x) = \frac{c}{2}\left(x^2 + x\right). \tag{1}$$

We provide the proof of Theorem 1 in Appendix A.

Since it is impossible to uniformly approximate the ReLU function with polynomial with integer coefficients even on $I = [-1, 1]$, we instead propose to approximate the scaled ReLU function on this interval. For instance, for $c = 2$, this results in the polynomial activation function

$$\sigma_{\text{poly}}(x) = x^2 + x, \tag{2}$$

which is shown in Fig. 1.

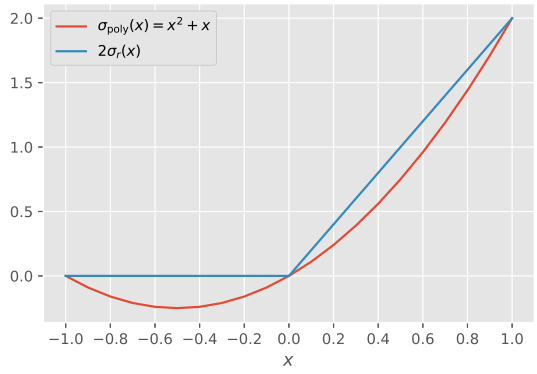

Figure 1: Our activation function $\sigma_{\text{poly}}(x) = x^2 + x$ is shown versus the scaled ReLU function $\sigma_{\text{sr}}(x; 2) = 2\sigma_{\text{r}}(x)$.

We next consider larger intervals than the interval $[-1, 1]$. For this case, we recall the following result (Ferguson 2006).

Lemma 3. *A continuous function f on an interval I of length strictly less than four is uniformly approximable by polynomials with integer coefficients if and only if its interpolating polynomial on $J(I)$ has integer coefficients, where $J(I)$ denotes the algebraic kernel of I*[1].

For instance, the algebraic kernel of the interval $I = [-\sqrt{2}, \sqrt{2}]$ is given by (Ferguson 2006)

$$J([-\sqrt{2}, \sqrt{2}]) = \{0, \pm 1, \pm\sqrt{2}\}. \tag{3}$$

Since the ReLU and the scaled ReLU functions do not satisfy the condition of Lemma 3 on $I = [-\sqrt{2}, \sqrt{2}]$, then it follows that they are not uniformly approximable on that interval. More generally, the algebraic kernel of any sub-interval $I = [-\alpha, \alpha]$, where $\alpha \leq 1.563$, contains 0 and whichever of ± 1 and $\pm\sqrt{2}$ that are in I (Ferguson 2006). Hence, it follows from Lemma 3 that the ReLU and the scaled ReLU functions are not uniformly approximable on any such interval with $\sqrt{2} \leq \alpha \leq 1.563$.

Given the impossibility of uniformly approximating the ReLU function and the scaled ReLU function over large intervals, we propose to approximate the ReLU function over reals and scale and round the resulting coefficients as we discuss in the next subsection.

3.2 Minimax Approximation

In this subsection, we discuss approximating the ReLU function through the minimax approximation technique over reals. In this approach, the goal is to approximate a function $f(x)$ over an interval $I = [a, b]$ through a polynomial $p_n(x)$ of degree at most n that minimizes

$$\begin{aligned} E &\triangleq \|f - p_n\|_\infty \\ &= \max_{a \leq x \leq b} |f(x) - p_n(x)|. \end{aligned} \tag{4}$$

Chebyshev showed that if $f(x)$ is a continuous function in $[a, b]$, then the polynomial $p_n^*(x)$ is a minimax polynomial of degree at most n if and only if $\exists n + 2$ points $a \leq x_0, x_1, \cdots, x_{n+1} \leq b$ such that

$$f(x_i) - p_n^*(x_i) = (-1)^i E^*. \tag{5}$$

That is, the error function has the same magnitude at these points with alternating signs. This is known as the equioscillation theorem (Carothers 1998). The Remez algorithm (Veidinger 1960) is an efficient iterative algorithm which solves for the coefficients of the minimax polynomial. We use the minimax approximation approach to approximate the ReLU function in the interval $I = [-a, a]$ and we get the following polynomial

$$p_2(x) = \frac{1}{2a}x^2 + \frac{1}{2}x + \frac{a}{16}. \tag{6}$$

This polynomial, however, has real-valued coefficients and our goal is to construct polynomials over $\mathbb{F}_p$. In order to do so, we scale this polynomial as follows

$$\sigma_{\mathrm{M}}(x) = x^2 + ax + \frac{a^2}{8}. \tag{7}$$

[1]We refer the reader to (Ferguson 2006) for the definition of the algebraic kernel and some illustrating examples.

Hence, for an integer a and aside from the bias term, this suggests the polynomial activation function

$$\sigma_{\mathrm{poly}}(x) = x^2 + ax. \tag{8}$$

We now discuss some important remarks.

Remark 1. (Zero Constant Term). We observe that our activation function has a zero constant term. In fact, many prior works as (Le Cun, Kanter, and Solla 1991; LeCun et al. 2012) illustrated that pushing the mean activations to zero decreases the bias shift effect and speeds up the learning. This motivated the development of several activation functions such as leaky ReLUs (LReLUS) (Maas, Hannun, and Ng 2013), parametric ReLUS (PReLUs) (He et al. 2015) and exponential linear units (ELUs) (Clevert, Unterthiner, and Hochreiter 2016).

Remark 2. (Bounded Interval). The assumption that the interval I is bounded is a typical assumption in approximation theory and in prior works that apply approximation theory in deep learning as (Telgarsky 2017; Boullé, Nakatsukasa, and Townsend 2020). The normalization layers can help in ensuring so, although this is not strictly ensured.

Remark 3. (Trainable Activation Function). In this work, we have derived a polynomial activation function based on our theoretical results in Theorem 1 rather than having a trainable activation function for various reasons. First, the polynomial must have integer coefficients or more precisely the polynomial must be over a finite field which complicates the training. Second, our polynomial function already achieves substantial accuracy gains compared to the square activation function as shown in our experiments, and also avoids us the extra computation cost of having trainable activation functions.

4 Empirical Evaluation

In this section, we compare between the various activation functions. To better assess the performance of such activation functions and to show the limitations of the square activation function, we consider several networks with a large number of activation layers compared to the prior works. A common problem in our work and the prior works is that the finite field size can be quite large due to using polynomial activations instead of ReLU activations and sum-pooling layers instead of max-pooling layers (Ghodsi, Gu, and Garg 2017). This prevented us from considering more complicated networks than the networks considered in this section and in the Appendix.

We consider image classification problems on the CIFAR-10, CIFAR-100, Tiny-ImageNet-10 and Tiny-ImageNet-200 datasets. The CIFAR datasets have RGB images of size $3 \times 32 \times 32$ of everyday objects classified into 10 and 100 classes, respectively. In CIFAR-10, the training set has 50000 images while the test set has 10000 images. CIFAR-100 has 100 classes, each contains 600 images with 500 training images and 100 test images. The Tiny-ImageNet-200 datset has RGB images of size $3 \times 64 \times 64$ classified into 200 classes, a training dataset of $100,000$ images, a validation dataset of $10,000$ images, and a test dataset of $10,000$ images. The Tiny-ImageNet-10 dataset contains the first 10 classes out of the 200 classes. Each class has 500 images, which are split into a training, validation, and test set with a ratio of

$8:1:1$. We normalize the images with the mean and the standard deviation of the pixels of each RGB channels as in (Krizhevsky, Sutskever, and Hinton 2012). Specifically, the mean and standard deviation of the RGB channels are $[0.485, 0.456, 0.406]$ and $[0.229, 0.224, 0.225]$, respectively for the preprocessing.

4.1 CNN of (Liu et al. 2017)

We first consider the convolutional neural network (CNN) architecture of (Liu et al. 2017). This network has 7 ReLU activation layers and is described in Table 1.

Input Size	Layer
32×32	Convolutional 3×3, 64, $/1$
32×32	Convolutional 3×3, 64, $/1$
32×32	Mean-pooling 2×2
16×16	Convolutional 3×3, 64, $/1$
16×16	Convolutional 3×3, 64, $/1$
16×16	Mean-pooling 2×2
16×16	Convolutional 3×3, 64, $/1$
8×8	Convolutional 1×1, 64, $/1$
8×8	Convolutional 1×1, 16, $/1$
1024×1	Fully Connected
10 or 100	Softmax

Table 1: The CNN considered in (Liu et al. 2017). Each convolutional layer is followed by a ReLU activation layer.

To investigate the performance of the various activation functions, we have implemented the following three schemes.

1. **CNN-ReLU**. For a baseline performance, we implement the CNN using ReLU activations and max-pooling layers where all computations are carried out in the real domain.
2. **CNN-Poly**. In this CNN, we use our proposed polynomial activation function $\sigma_{\text{poly}}(x) = x^2 + x$ and sum-pooling layers. The training is carried out in the real domain, while the inference is carried out in the finite field $\mathbb{F}_p$.
3. **CNN-Quad**. In this CNN, we use the square function activation $\sigma_{\text{square}}(x) = x^2$ and sum-pooling layers. The training is carried out also in the real domain and the inference is in the finite field $\mathbb{F}_p$.

In Fig. 2 and Fig. 3, we compare between the different activation functions on the CIFAR-10 and CIFAR-100 datasets, respectively. As we can see, the accuracy of CNN-Poly using our polynomial function significantly outperforms the accuracy of CNN-Quad. Moreover, CNN-Poly has comparable accuracy to CNN-ReLU while CNN-Poly involves quantization errors to preserve the privacy and/or to allow verifiable inference. We summarize this comparison in Table 2.

Activation	CIFAR-10	CIFAR-100
CNN-ReLU	84.6%	54.7%
CNN-Poly	83.0%	55.3%
CNN-Quad	77.4%	51.3%

Table 2: Test accuracy for the various activation functions for the CNN considered in (Liu et al. 2017).

4.2 Network In Network (NIN) (Lin, Chen, and Yan 2014)

To further investigate the performance of the various activation functions, we implemented the architecture of (Lin, Chen, and Yan 2014), known as "Network In Network (NIN)". This network has 9 ReLU activation layers as described in Table 3.

Input Size	Layer
32×32	Convolutional 5×5, 192
32×32	Convolutional 1×1, 160
32×32	Convolutional 1×1, 96
32×32	Max-pooling 3×3, /2
16×16	Dropout, 0.5
16×16	Convolutional 5×5, 192
16×16	Convolutional 1×1, 192
16×16	Convolutional 1×1, 192
16×16	Average-pooling 3×3, $/2$
8×8	Dropout, 0.5
8×8	Convolutional 3×3, 192
8×8	Convolutional 1×1, 192
8×8	Convolutional 1×1, 10
8×8	Global Average-pooling 8×8, $/1$
10 or 100	Softmax

Table 3: The Network In Network (NIN) architecture of (Lin, Chen, and Yan 2014). Each convolutional layer is followed by a ReLU activation layer.

We also implemented with the three activation schemes, referred to as *NIN-ReLU*, *NIN-Poly*, and *NIN-Quad*, respectively. For both CIFAR-10 and CIFAR-100 dataset, the accuracy of NIN-Poly significantly outperforms the accuracy of NIN-Quad as shown in Fig. 4 and Fig. 5. We also summarize this comparison in Table 4.

Activation	CIFAR-10	CIFAR-100
NIN-ReLU	88.5%	64.2%
NIN-Poly	88.7%	55.4%
NIN-Quad	81.0%	46.0%

Table 4: Test accuracy for the various activation functions for the NIN architecture (Lin, Chen, and Yan 2014).

4.3 LeNet (LeCun 1998) on Tiny-ImageNet-10 Dataset

To investigate the performance of the various activation functions with higher resolution images, we implemented CNN on the Tiny-ImageNet-10 dataset where each image consists of 64×64 pixels with 3 RGB channels. We implement the LeNet in (LeCun 1998), and modify the size of fully connected layers in order to accommodate differences between MNIST and Tiny-ImageNet images. This network has 3 ReLU activation layers as described in Table 5.
We also implemented with the three activation schemes, referred to as *LeNet-ReLU*, *LeNet-Poly*, and *LeNet-Quad*, respectively.

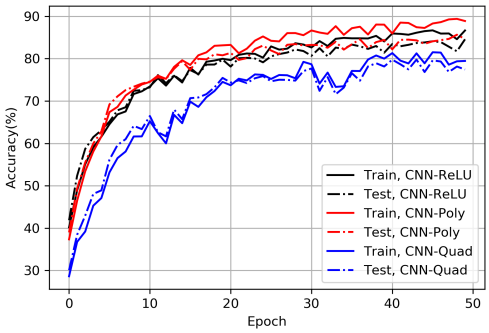

Figure 2: Accuracy of the CNN architecture in (Liu et al. 2017) on the CIFAR-10 dataset.

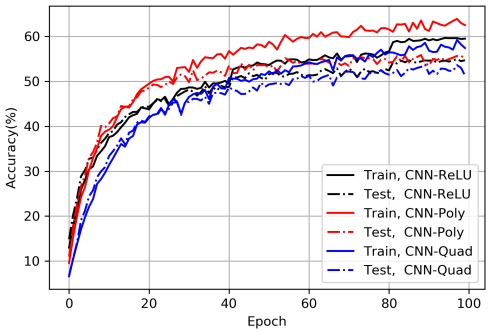

Figure 3: Accuracy of the CNN architecture in (Liu et al. 2017) on the CIFAR-100 dataset.

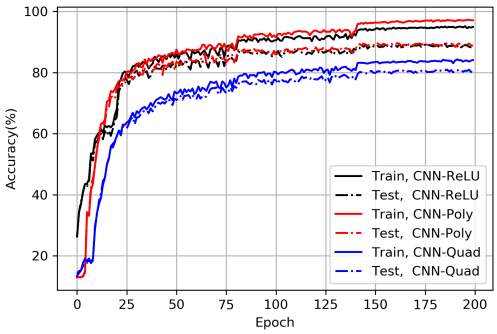

Figure 4: Accuracy of the NIN architecture in (Lin, Chen, and Yan 2014) on the CIFAR-10 dataset.

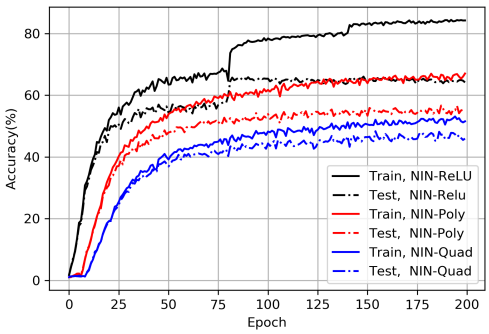

Figure 5: Accuracy of the NIN architecture in (Lin, Chen, and Yan 2014) on the CIFAR-100 dataset.

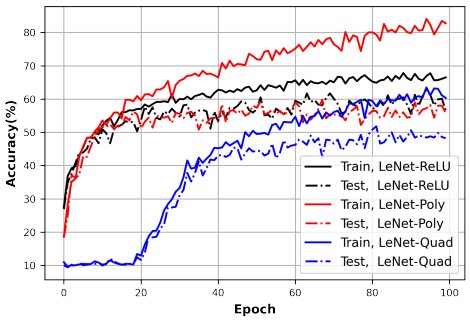

Figure 6: Accuracy of the LeNet architecture in (LeCun 1998) on the Tiny-ImageNet-10 dataset.

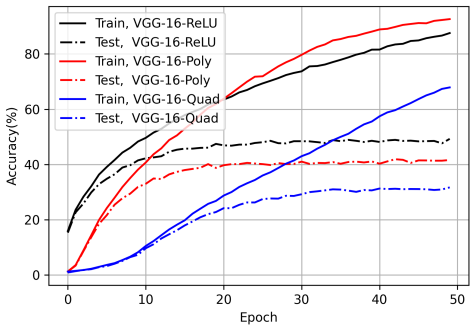

Figure 7: Accuracy of the VGG16 architecture in (Simonyan and Zisserman 2014) on the Tiny-ImageNet-200 dataset.

The accuracy of LeNet-Poly significantly outperforms the accuracy of LeNet-Quad as shown in Fig. 6. We also summarize this comparison in Table 6.

4.4 VGG (Simonyan and Zisserman 2014) on Tiny-ImageNet-200 Dataset

To study the performance of the various activation functions with more challenging image classification task, we implemented VGG-16 (Simonyan and Zisserman 2014) on the Tiny-ImageNet-200 dataset. This network has 9 ReLU activation layers as described in Table 7.

We also implemented with the three activation schemes, referred to as *VGG-16-ReLU*, *VGG-16-Poly*, and *VGG-16-Quad*, respectively. The accuracy of VGG-16-Poly significantly outperforms the accuracy of VGG-16-Quad as shown in Fig. 7. This comparison is summarized in Table 8.

Input Size	Layer
64×64	Convolutional 5×5, 10
60×60	Max-pooling 2×2
26×26	Convolutional 5×5, 20
26×26	Max-pooling 2×2
26×26	Dropout, 0.5
3380×1	Fully Connected
512×1	Fully Connected
64×1	Fully Connected
10	Softmax

Table 5: LeNet architecture of (LeCun 1998). Each convolutional layer and fully connected layer (except for the last fully connected layer) is followed by a ReLU activation layer.

Activation	Tiny-ImageNet-10
LeNet-ReLU	56.2%
LeNet-Poly	55.8%
LeNet-Quad	50.0%

Table 6: Test accuracy for the various activation functions for the LeNet architecture (LeCun 1998).

Input Size	Layer
64×64	Convolutional 3×3, 64
64×64	Convolutional 3×3, 64
64×64	Max-pooling 2×2, /2
32×32	Convolutional 3×3, 128
32×32	Convolutional 3×3, 128
32×32	Max-pooling 2×2, /2
16×16	Convolutional 3×3, 256
16×16	Convolutional 3×3, 256
16×16	Convolutional 3×3, 256
16×16	Max-pooling 2×2, /2
8×8	Convolutional 3×3, 512
8×8	Convolutional 3×3, 512
8×8	Convolutional 3×3, 512
8×8	Max-pooling 2×2, /2
4×4	Average-pooling 1×1, /1
8192×1	Fully Connected
200	Softmax

Table 7: The VGG-16 architecture of (Simonyan and Zisserman 2014). Each convolutional layer is followed by a batch normalization and a ReLU activation layer.

Activation	Tiny-ImageNet-200
VGG-ReLU	49.3%
VGG-Poly	41.6%
VGG-Quad	31.2%

Table 8: Test accuracy for the various activation functions for the VGG-16 architecture (Simonyan and Zisserman 2014).

Next, we discus the hyperparameters.

Hyperparameters. For a fair comparison between three activation functions, we find the best learning rate from $\{0.1, 0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001\}$ for each scheme. Given the choice of the best learning rate η, η is decayed to 0.4η every 80 and 140 rounds in the NIN architecture while η is not decayed in the CNN, LeNet, VGG-16 architectures.

We set the mini batch-size to 125 for both CIFAR-10 and CIFAR-100 datasets, and 100 for Tiny-Imagenet-10 and Tiny-Imagenet-200 datasets. We use L_2 regularization parameter $\lambda = 5 \cdot 10^{-4}$ for the CNN architecture, and use $\lambda = 3 \cdot 10^{-4}$ for the NIN, LeNet, VGG-16 architectures.

We also report an additional experiment on AlexNet (Krizhevsky, Sutskever, and Hinton 2012) in Appendix B. Finally, we discuss some important remarks.

Remark 4. (More Complicated Networks). Our goal in this work is not to achieve or outperform the state-of-the-art results on the CIFAR and the Tiny-ImageNet datasets. Instead, we show that the square function is not good enough to replace the ReLU function and our polynomial activation improves the accuracy significantly. In fact, prior works as (Liu et al. 2017; Ghodsi, Gu, and Garg 2017; Hesamifard, Takabi, and Ghasemi 2017) performed experiments on even simpler architectures compared to our work. The main challenges in performing more experiments on complicated architectures while using polynomial activation functions is the finite field size and the gradient explosion problem (Ghodsi, Gu, and Garg 2017; Ghodsi 2021).

Remark 5. (Degree-2 Polynomials). We have focused on degree-2 polynomials to have an inference scheme of low complexity, to keep the field size small as possible and for a fair comparison with the square activation function. Nevertheless, our approach can be extended to develop polynomials of higher degrees.

5 Conclusions

In this work, we have considered the problem of designing polynomial activation functions with integer coefficients or over finite field for privacy-preserving and verifiable inference for ReLU networks. While most prior works replace the ReLU activation function with the square activation function $\sigma_{\text{square}}(x) = x^2$, we have empirically shown that the square function can result in a severe degradation in the accuracy. Indeed, we have empirically shown that the square activation function is not the best function to replace the ReLU function even if the coefficients are restricted to be integers. In particular, we have proposed the activation function $\sigma_{\text{poly}}(x) = x^2 + x$ and empirically shown that it significantly outperforms the square function by up to 10.4% improvement in the test accuracy through several experiments on the CIFAR and Tiny ImageNet datasets for several network architectures.

References

Amazon. 2021. Amazon AWS AI. https://aws.amazon.com/machine-learning/. Last accessed: May 2021.

Barni, M.; Orlandi, C.; and Piva, A. 2006. A privacy-preserving protocol for neural-network-based computation. In *Proceedings of the 8th workshop on Multimedia and security*, 146–151.

Bos, J. W.; Lauter, K.; Loftus, J.; and Naehrig, M. 2013. Improved security for a ring-based fully homomorphic encryption scheme. In *IMA International Conference on Cryptography and Coding*, 45–64. Springer.

Boullé, N.; Nakatsukasa, Y.; and Townsend, A. 2020. Rational neural networks. *arXiv preprint arXiv:2004.01902*.

Brutzkus, A.; Gilad-Bachrach, R.; and Elisha, O. 2019. Low latency privacy preserving inference. In *International Conference on Machine Learning*, 812–821. PMLR.

Carothers, N. L. 1998. A short course on approximation theory. *Department of Mathematics and Statistics, Bowling green State University*.

Chen, X.; Ji, J.; Yu, L.; Luo, C.; and Li, P. 2018. Securenets: Secure inference of deep neural networks on an untrusted cloud. In *Asian Conference on Machine Learning*, 646–661. PMLR.

Chou, E.; Beal, J.; Levy, D.; Yeung, S.; Haque, A.; and Fei-Fei, L. 2018. Faster cryptonets: Leveraging sparsity for real-world encrypted inference. *arXiv preprint arXiv:1811.09953*.

Clevert, D.-A.; Unterthiner, T.; and Hochreiter, S. 2016. Fast and accurate deep network learning by exponential linear units (ELUS). *International Conference on Learning Representations (ICLR)*.

Ferguson, L. B. O. 2006. What can be approximated by polynomials with integer coefficients. *The American Mathematical Monthly*, 113(5): 403–414.

Garofolo, J. S. 1993. TIMIT acoustic phonetic continuous speech corpus. *Linguistic Data Consortium*.

Gautier, A.; Nguyen, Q. N.; and Hein, M. 2016. Globally Optimal Training of Generalized Polynomial Neural Networks with Nonlinear Spectral Methods. In *Advances in Neural Information Processing Systems*, volume 29, 1687–1695.

Gentry, C. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 169–178.

Ghodsi, Z. 2021. *Secure Frameworks for Outsourced Deep Learning Inference*. Ph.D. thesis, New York University Tandon School of Engineering.

Ghodsi, Z.; Gu, T.; and Garg, S. 2017. Safetynets: Verifiable execution of deep neural networks on an untrusted cloud. In *Advances in Neural Information Processing Systems*, 4672–4681.

Ghodsi, Z.; Veldanda, A.; Reagen, B.; and Garg, S. 2020. Cryptonas: Private inference on a relu budget. *arXiv preprint arXiv:2006.08733*.

Gilad-Bachrach, R.; Dowlin, N.; Laine, K.; Lauter, K.; Naehrig, M.; and Wernsing, J. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, 201–210.

Goldwasser, S.; Kalai, Y. T.; and Rothblum, G. N. 2015. Delegating computation: interactive proofs for muggles. *Journal of the ACM (JACM)*, 62(4): 1–64.

Google. 2021. Google Cloud AI. https://cloud.google.com/products/machine-learning/. Last accessed: May 2021.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, 1026–1034.

Hesamifard, E.; Takabi, H.; and Ghasemi, M. 2017. Cryptodl: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25: 1097–1105.

Le Cun, Y.; Kanter, I.; and Solla, S. A. 1991. Eigenvalues of covariance matrices: Application to neural-network learning. *Physical Review Letters*, 66(18): 2396.

LeCun, Y. 1998. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

LeCun, Y. A.; Bottou, L.; Orr, G. B.; and Müller, K.-R. 2012. Efficient backprop. In *Neural networks: Tricks of the trade*, 9–48. Springer.

Lin, M.; Chen, Q.; and Yan, S. 2014. Network in network. *ICLR*.

Liu, J.; Juuti, M.; Lu, Y.; and Asokan, N. 2017. Oblivious neural network predictions via minionn transformations. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 619–631.

Livni, R.; Shalev-Shwartz, S.; and Shamir, O. 2014. On the computational efficiency of training neural networks. In *Advances in neural information processing systems*, 855–863.

Lund, C.; Fortnow, L.; Karloff, H.; and Nisan, N. 1992. Algebraic methods for interactive proof systems. *Journal of the ACM (JACM)*, 39(4): 859–868.

Maas, A. L.; Hannun, A. Y.; and Ng, A. Y. 2013. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, 3. Citeseer.

Microsoft. 2021. Azure Machine Learning Studio. https://azure.microsoft.com/en-us/services/machine-learning-studio/. Last accessed: May 2021.

Mishra, P.; Lehmkuhl, R.; Srinivasan, A.; Zheng, W.; and Popa, R. A. 2020. Delphi: A cryptographic inference service for neural networks. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2505–2522.

Mohassel, P.; and Zhang, Y. ???? Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, 19–38.

Petrushev, P. P.; and Popov, V. A. 2011. *Rational approximation of real functions*, volume 28. Cambridge University Press.

Rivest, R. L.; Adleman, L.; Dertouzos, M. L.; et al. 1978. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11): 169–180.

Sanyal, A.; Kusner, M.; Gascon, A.; and Kanade, V. 2018. TAPAS: Tricks to accelerate (encrypted) prediction as a service. In *International Conference on Machine Learning*, 4490–4499. PMLR.

Simonyan, K.; and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Telgarsky, M. 2017. Neural networks and rational functions. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 3387–3393. JMLR. org.

Thaler, J. 2013. Time-optimal interactive proofs for circuit evaluation. In *Annual Cryptology Conference*, 71–89. Springer.

Veidinger, L. 1960. On the numerical determination of the best approximations in the Chebyshev sense. *Numerische Mathematik*, 2(1): 99–105.

Xie, P.; Bilenko, M.; Finley, T.; Gilad-Bachrach, R.; Lauter, K.; and Naehrig, M. 2014. Crypto-nets: Neural networks over encrypted data. *arXiv preprint arXiv:1412.6181*.

Zhao, L.; Wang, Q.; Wang, C.; Li, Q.; Shen, C.; and Feng, B. 2021. Veriml: Enabling integrity assurances and fair payments for machine learning as a service. *IEEE Transactions on Parallel and Distributed Systems*.