CryptoDL: Deep Neural Networks over Encrypted Data

Ehsan Hesamifard*[1], Hassan Takabi†[1], and Mehdi Ghasemi‡[2]

[1]Department of Computer Science and Engineering, University of North Texas
[2]Department of Mathematics and Statistics, University of Saskatchewan

November 15, 2017

Abstract

Machine learning algorithms based on deep neural networks have achieved remarkable results and are being extensively used in different domains. However, the machine learning algorithms requires access to raw data which is often privacy sensitive. To address this issue, we develop new techniques to provide solutions for running deep neural networks over encrypted data. In this paper, we develop new techniques to adopt deep neural networks within the practical limitation of current homomorphic encryption schemes. More specifically, we focus on classification of the well-known convolutional neural networks (CNN). First, we design methods for approximation of the activation functions commonly used in CNNs (i.e. ReLU, Sigmoid, and Tanh) with low degree polynomials which is essential for efficient homomorphic encryption schemes. Then, we train convolutional neural networks with the approximation polynomials instead of original activation functions and analyze the performance of the models. Finally, we implement convolutional neural networks over encrypted data and measure performance of the models. Our experimental results validate the soundness of our approach with several convolutional neural networks with varying number of layers and structures. When applied to the MNIST optical character recognition tasks, our approach achieves 99.52% accuracy which significantly outperforms the state-of-the-art solutions and is very close to the accuracy of the best non-private version, 99.77%. Also, it can make close to 164000 predictions per hour. We also applied our approach to CIFAR-10, which is much more complex compared to MNIST, and were able to achieve 91.5% accuracy with approximation polynomials used as activation functions. These results show that CryptoDL provides efficient, accurate and scalable privacy-preserving predictions.

1 Introduction

Machine learning algorithms based on deep neural networks have attracted attention as a breakthrough in the advance of artificial intelligence (AI) and are the mainstream in current AI research.

*ehsanhesamifard@my.unt.edu
†takabi@unt.edu
‡mehdi.ghasemi@usask.ca

These techniques are achieving remarkable results and are extensively used for analyzing big data in a variety of domains such as spam detection, traffic analysis, intrusion detection, medical or genomics predictions, face recognition, and financial predictions. Furthermore, with increasing growth of cloud services, machine learning services can be run on cloud providers' infrastructure where training and deploying machine learning models are performed on cloud servers. Once the models are deployed, users can use these models to make predictions without having to worry about maintaining the models and the service. In a nutshell, this is Machine Learning as a Service (MLaaS), and several such services are currently offered including Microsoft Azure Machine Learning [32], Google Prediction API [33], GraphLab [34], and Ersatz Labs [35]. However, machine learning algorithms require access to the raw data which is often privacy sensitive and can create potential security and privacy risks. In recent years, several studies have investigated the privacy protection of this sensitive data in different machine learning algorithms such as linear regression [39], linear classifiers [5, 13], decision trees [5, 44] or neural networks [9, 49].

In this paper, we propose CryptoDL, a solution to run deep neural network algorithms on encrypted data and allow the parties to provide/ receive the service without having to reveal their sensitive data to the other parties. In particular, we focus on classification phase of deep learning algorithms. The main components of CryptoDL are convolutional neural networks (CNNs) and homomorphic encryption (HE). To allow accurate predictions we propose using neural networks, specifically CNNs which are extensively used in the machine learning community for a wide variety of tasks. Recent advances in fully homomorphic encryption (FHE) enable a limited set of operations to be performed on encrypted data [11, 55]. This will allow us to apply deep neural network algorithms directly to encrypted data and return encrypted results without compromising security and privacy concerns.

However, due to a number of constraints associated with these cryptographic schemes, designing practical efficient solutions to run deep neural network models on the encrypted data is challenging. The most notable shortcoming of HE is that operations in practical schemes are limited to addition and multiplication. Consequently, we need to adopt deep neural network algorithms within these limitations. The computation performed over sensitive data by deep neural network algorithms is very complex and makes it hard to support efficiently and deep neural network models cannot simply be translated to encrypted versions without modification. For example, in neural networks activation functions such as Rectified Linear Unit (ReLU) and Sigmoid are used as an activation function and we have to replace these functions with another function that only uses addition and multiplication such as polynomials.

In order to have efficient and practical solutions for computations in encrypted domain, we typically need to use leveled HE schemes instead of FHE. However, a solution that builds upon these encryption schemes has to be restricted to computing low-degree polynomials in order to be practical and efficient. Approximating a function with low-degree polynomials is an important issue for running deep CNN algorithms on encrypted data when we use HE, see [53, 54].

1.1 Threat Model and Problem Statement

In this paper, we focus on the CNN, one of the most popular deep learning algorithms. We assume that the training phase is done on the plaintext data and a model has already been built and trained.

Problem: *Privacy-preserving Classification on Deep Convolutional Neural Networks*

The client has a previously unseen feature vector x and the server has an already trained deep CNN model w. The server runs a classifier C over x using the model w to output a prediction

$C(x, w)$. To do this, the client sends an encrypted input to the server, server performs encrypted inference, and the client gets the encrypted prediction. The server must not learn anything about the input data or the prediction and the classification must not reveal information about the trained neural network model w.

A practical solution to this problem for real-world applications should be both **accurate** (the prediction performance should be close to the prediction performance of the plaintext) and **efficient** (the running time to obtain the prediction result should be low).

1.2 Contributions

In this paper, we design and evaluate a *privacy-preserving classification for deep convolutional neural networks*. Our goal is to adopt deep convolutional neural networks within practical limitations of HE while keeping accuracy as close as possible to the original model.

The most common activation functions used in CNNs are ReLU, Sigmoid, and Tanh. In order to achieve our goal, these functions should be replaced by HE friendly functions such as low-degree polynomials.

The main contributions of this work are as follows.

- We provide theoretical foundation to prove that it is possible to find lowest degree polynomial approximation of a function within a certain error range.
- Building upon the theoretical foundation, we investigate several methods for approximating commonly used activation functions in CNNs (i.e. ReLU, Sigmoid, and Tanh) with low-degree polynomials to find the best approximation.
- We utilize these polynomials in CNNs and analyze the performance of the modified algorithms.
- We implement the CNNs with polynomial approximations as activation functions over encrypted data and report the results for two of the widely used datasets in deep learning, MNIST and CIFAR-10.
- Our experimental results of MNIST show that CryptoDL can achieve 99.52% accuracy which is very close to the original model's accuracy of 99.56%. Also, it can make close to 164000 predictions per hour. These results show that CryptoDL provides efficient, accurate, and scalable privacy-preserving predictions.

The rest of this paper is organized as follows: In Section 2, we provide a brief overview about the structure of HE schemes and deep CNNs. In Section 3, we describe our theoretical foundation and proposed solution for polynomial approximation in details. Section 4 provides experimental results for CNN models over encrypted datasets followed by a discussion. In Section 5, we review related work. In Section 6, we conclude the paper and discuss future work.

2 Overview and Background Information

In this section, we provide a brief introduction to HE schemes, their strengths and weaknesses which should be considered while using them for secure computation protocols. We also briefly describe deep CNNs and modifications that are required for adopting them within HE schemes.

2.1 Homomorphic Encryption

Homomorphic encryption (HE) schemes preserve the structure of the message space such that we can perform operations such as addition and multiplication over the ciphertext space. Like other types of encryption schemes, an HE scheme has three main functions, *Gen*, *Enc*, and *Dec*, for key generation, encryption, and decryption, respectively. However, an HE scheme also has an evaluation function, *Eval*. Suppose we have a set of plaintext messages $\{m_i\}$ and relative ciphertexts $\{c_i\}$. Now, consider a circuit C. The evaluation function processes the public key pk, a set of ciphertexts $\{c_i\}$ and a circuit C such that

$$Dec(sk, Eval(pk, C, c_1, \cdots, c_n)) = C(m_1, \cdots, m_n)$$

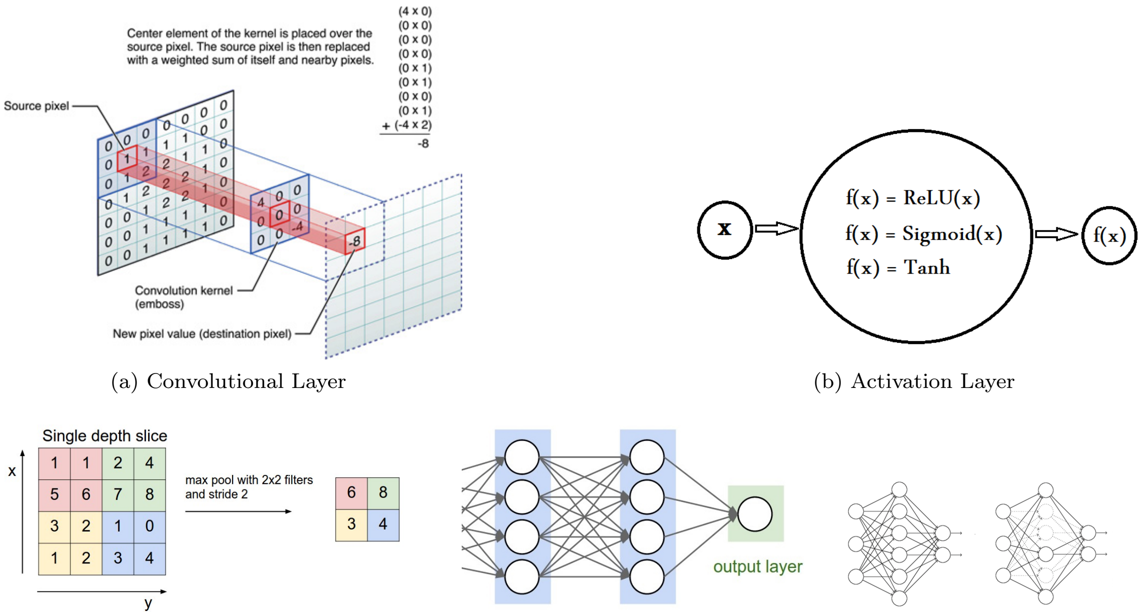
HE was first introduced in 1978 by Rivest et al. [25]. Other researchers followed to introduce several other HE schemes [22]. However, most of these encryption schemes have some constraints. Some of them, such as the Paillier cryptosystem [22], only support one operation (addition). If the encryption scheme only supports one operation, it is called Somewhat Homomorphic Encryption (SHE).

The idea behind encryption function *Enc* is to add a small value, called *noise*, to m for encrypting. Therefore, each ciphertext has a small amount of noise. When we add two ciphertexts c_1 and c_2, the result is also a ciphertext, but with noise that has grown. The *Dec* function works correctly if this amount is less than a threshold. This threshold leads to a bound on the number of computations that can be performed over encrypted data. If an entity wants to decrease the noise, it should decrypt and encrypt the ciphertext, for decryption, it needs the secret key sk. For years, the community was trying to find out if there is a way to decrease the noise without having the secret key.

This question was answered in 2009 when first Fully Homomorphic Encryption (FHE) scheme was designed by Gentry [11]. Am FHE scheme is an HE scheme that supports circuits with arbitrary depth. In his dissertation, Gentry introduced a technique for handling an arbitrary depth of computations, called *bootstrapping*. In bootstrapping technique, the amount of noise is decreased without needing to access sk [11]. However, it has a huge computational cost and is a very slow process. This limitation makes FHE impractical for actual use.

Recent advances in HE have led to a faster HE scheme: Leveled Homomorphic Encryption (LHE). LHE schemes do not support the bootstrapping step, so they only allow circuits with depth less than a specific threshold. If we know the number of operations before starting the computations, we can use LHE instead of FHE. The performance of LHE schemes is further improved using Single-Instruction-Multiple-Data (SIMD) techniques. Halevi et al. in [14] use this technique to create a batch of ciphertexts. So, one single ciphertext has been replaced with an array of ciphertexts in computations.

Despite the advantages of using HE schemes, they have some limitations. The first one is message space. Almost all HE schemes work with integers. Therefore, before encrypting data items, we need to convert them to integers. The second limitation is ciphertext size. The size of the message increases considerably by encryption. Another important limitation is related to the noise. After each operation, the amount of noise in ciphertext increases. Multiplication increases noise much more than addition. We should always keep the amount of noise less than the predefined threshold. The last and most important limitation is lack of division operation. In summary, only a limited number of additions and multiplications are allowed over encrypted data and therefore complex functions such as activation functions used in neural networks are not compatible with HE schemes.


Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.
(4 x 0)
(0 x 0)
(0 x 0)
(0 x 0)
(0 x 1)
(0 x 1)
(0 x 0)
(0 x 1)
+ (-4 x 2)
-8
Source pixel
Convolution kernel (emboss)
New pixel value (destination pixel)
x
f(x) = ReLU(x)
f(x) = Sigmoid(x)
f(x) = Tanh
f(x)
Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters and stride 2
6 8
3 4
output layer

(a) Convolutional Layer

(b) Activation Layer

(c) Max Pooling Layer

(d) Fully Connected Layer

(e) Dropout Layer

Figure 1: Different Layers in a Convolutional Neural Network

2.2 Deep Learning: Convolutional Neural Networks (CNNs / ConvNets)

At a high level of abstraction, a neural network is a combination of neurons arranged in ordered layers. Each neuron gets an input, operates a function on it and outputs the result of the function. The structure of this function depends on the layer to which the neuron belongs. Besides the first layer (input layer) and the last layer (output layer), there is at least one middle layer, called hidden layer. In fully feed-forward neural networks, each neuron has a weighted connection to all neurons in the next layer. Neurons in different layers are of different types. For example, neurons in the input layer only get one input and output which is the same value. Neurons in hidden layers are more complex; they get inputs, compute the weighted summation of inputs, operate a function on the summation and then output the value of the function. These functions could be Sigmoid, Max or Mean functions and are called activation functions (or transfer functions).

Convolutional Neural Networks (CNNs / ConvNets) are a specific type of feed-forward neural networks in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex. They have proven to be very effective in areas such as image recognition and classification. CNNs commonly use several distinct kinds of layers as shown in Figure 1 (adopted from http://cs231n.github.io/convolutional-networks) and described in the followings.

2.2.1 Convolutional Layer

The first layer in a CNN is always a convolutional layer. A convolutional layer is a set of filters that operates on the input points. For the first layer, the input is the raw image. The idea behind using convolutional layers is learning features from the data. Each filter is a $n \times n$ square (for example, $n = 3 or 5$) with a stride. We convolve the pixels in the image and calculate the dot product of the filter values and related values in the neighbor of the pixel. This step only includes addition and multiplication and we can use the same computation over the encrypted data. The stride is a pair of two numbers, for example $(2, 2)$, in each step we slide the filter two units to the left or down.

2.2.2 Activation Layer

After each convolutional layer, we use an activation layer which is a non-linear function. Every activation function takes a single number and performs a certain fixed mathematical operation on it. There are several activation functions we may encounter in practice including ReLU ($ReLU(x) = max(0, x)$), Sigmoid ($\sigma = \frac{1}{1+e^{-x}}$), and Tanh ($2\sigma(2x) - 1$) functions. We cannot calculate these functions over encrypted values and we should find replacements for these functions that only include addition and multiplication operations.

2.2.3 Pooling Layer

After an activation layer, a pooling layer (or sub-sampling). This layer if for sub-sampling from the data and reduces the size of data. Different kind of pooling layers are introduced in literature, two of the most popular ones are max pooling and average pooling. We cannot use max pooling because of the lack of the max operation over encrypted data. We use a scaled up version of average pooling (proposed in [9]), calculate the summation of values without dividing it by the number of values. We implement average pooling with addition only, and it does not have impact on the depth of the algorithm.

2.2.4 Fully Connected Layer

Fully connected layer has the same structure like hidden layers in classic neural networks. We call this layer fully connected because each neuron in this layer is connected to all neurons in the previous layer, each connection represents by a value which called weight. The output of each neuron is the dot product of two vectors: output of neurons in the previous layers and the related weight for each neuron.

2.2.5 Dropout Layer

When we train a model over training set, it's possible the final model be biased to the training set, and we get high error over the test set. This problem called over-fitting. For avoiding over-fitting during the training process, we use this specific type of layers in the CNN. In this layer, we drop out a random set of connections and set them to zero in each iteration. This dropping of values does not let the over-fitting happens in our training process and the final model is not completely fit to the training set. We need this layer only for the training step and we can remove it in the classification step.

2.2.6 Architecture of CNNs

There are different ways to use the above-mentioned layers in a CNN for training a model. However, there is a common pattern for creating a CNN. The first layer is a convolutional layer, after a convolutional layer, we add an activation layer. One block of a CNN is $[Convolutional \rightarrow Activation]$ and we can use this block more than one time, $[Convolutional \rightarrow Activation]^n$. After this series of block, we use an average pooling layer. This is the second block which is a combination of the first block plus an average pooling layer, $[[Convolutional \rightarrow Activation]^n \rightarrow AveragePooling]^n$. Like the first block, we can use the second block more than one times. Then, the we have one or more fully connected layers after the second block and the CNN ends with an output layer, the output of this layer is the number of classes in the dataset. This is a common pattern for creating a CNN. Figures 3 and 4 are two examples of CNNs with different architectures.

3 The Proposed Privacy-preserving Classification for Deep Convolutional Neural Networks

Since our goal is to adopt a CNN to work within HE constraints, our focus is on operations inside the neurons. Besides activation functions inside the neurons, all other operations in a neural network are addition and multiplication, so they can be implemented over encrypted data. It is not possible to use activation functions within HE schemes. Hence, we should find compatible replacement functions in order to operate over encrypted data.

The basic idea of our solution to this problem is to approximate the non-compatible functions with a compatible form so they can be implemented using HE. In general, most functions including activation functions used in CNNs can be approximated with polynomials which are implemented using only addition and multiplication operations. Hence, we aim to approximate the activation functions with polynomials and replace them with these polynomials when operating over encrypted data. We investigate polynomial approximation of activation functions commonly used in CNNs, namely ReLU, Sigmoid, and Tanh and choose the one that approximates each activation function the best.

Polynomials of degree 2 have been used to substitute the Sigmoid function in neural networks [9] and polynomials of degree 3 are used to estimate the natural logarithm function [26]. However, these are specific solutions which enable us to work around certain problems, but there is no generic solution to this problem yet. Generally, we can approximate activation functions with polynomials from different degrees. The higher degree polynomials provide a more accurate approximation and when they replace the activation function in a CNN, lead to a better performance in the trained model. However, when operations are performed over encrypted data, a higher degree polynomial results in very slow computations. Therefor, a solution that builds upon HE schemes should be restricted to computing low-degree polynomials in order to be practical [29]. We need to find a trade-off between the degree of the polynomial approximation and the performance of the model. In the following, we first provide theoretical foundation and prove that it is possible to find lowest degree polynomial approximation of a function within a certain error range. Next, we propose a solution for polynomial approximation of several activation functions (i.e. ReLU, Sigmoid, Tanh). Then, we train CNN models using these polynomials and compare the performance with the models with the original activation functions.

3.1 Polynomial Approximation: Theoretical Foundation

Among continuous functions, perhaps polynomials are the most well-behaved and easiest to compute. Thus, it is no surprise that mathematicians tend to approximate other functions by polynomials. Materials of this section are mainly folklore knowledge in numerical analysis and Hilbert spaces. For more details on the subject refer to [4, 30].

Let us denote the family of all continuous real valued functions on a non-empty compact space X by $\mathrm{C}(X)$. Since linear combination and product of polynomials are also polynomials, we assume that A is closed under addition, scalar multiplication and product and also a non-zero constant function belongs to A (This actually implies that A contains all constant functions).

We say an element $f \in \mathrm{C}(X)$ can be approximated by elements of A, if for every $\epsilon > 0$, there exists $p \in A$ such that $|f(x) - p(x)| < \epsilon$ for every $x \in X$. The following classical results guarantee when every $f \in \mathrm{C}(X)$ can be approximated by elements of A.

Theorem 1 (Stone–Weierstrass)**.** *Every element of $C(X)$ can be approximated by elements of A if and only if for every $x \neq y \in X$, there exists $p \in A$ such that $p(x) \neq p(y)$.*

Despite the strong and important implications of the Stone-Weierstrass theorem, it leaves computational details out and does not give a specific algorithm to generate an estimator for f with elements of A, given an error tolerance ϵ. We address this issue here.

For every $f \in C(X)$ and every $\epsilon > 0$, there exists $p \in A$ such that $\|f - p\|_\infty < \epsilon$. Let V be an $\mathbb{R}$-vector space, an *inner product* on V ($\langle \cdot, \cdot \rangle : V \times V \to \mathbb{R}$, see [30] for definition). The pair $(V, \langle \cdot, \cdot \rangle)$ is called an inner product space and the function $\|v\| = \langle v, v \rangle^{\frac{1}{2}}$ induces a norm on V. Every given set of linearly independent vectors can be turned into a set of orthonormal vectors (see [30] for definition) that spans the same sub vector space as the original. The Gram–Schmidt well-known theorem gives us an approach for producing such orthonormal vectors from a set of linearly independent vectors.

Now, let μ be a finite measure on X and for $f, g \in \mathrm{C}(X)$ define $\langle f, g \rangle = \int_X fgd\mu$. This defines an inner product on the space of functions. Any good approximation in $\| \cdot \|_\infty$ gives a good $\| \cdot \|_{2,\mu}$-approximation. But generally, our interest is the other way around. Employing Gram–Schmidt procedure, we can find $\| \cdot \|_{2,\mu}$ within any desired accuracy, but this does not guarantee a good $\| \cdot \|_\infty$-approximation. In other words, *"good enough $\| \cdot \|_{2,\mu}$-approximations of f give good $\| \cdot \|_\infty$-approximations"*, as desired.

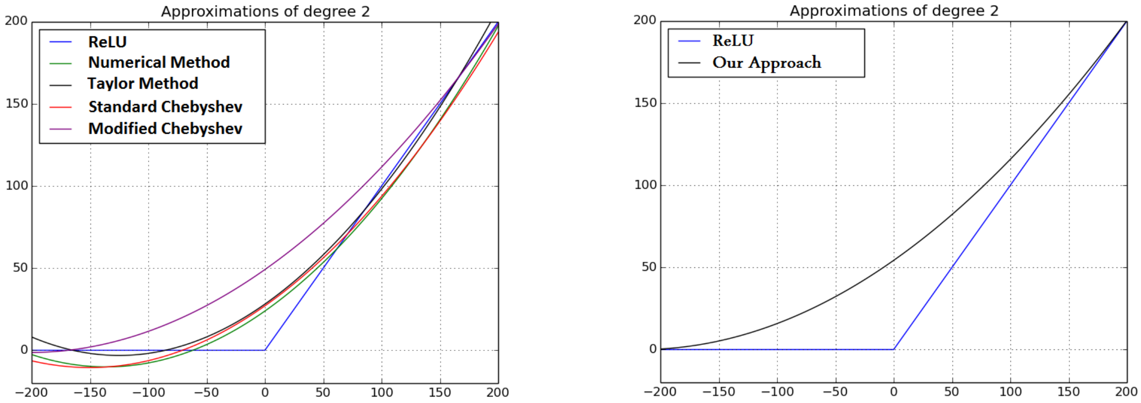
Different choices of μ, gives different systems of orthogonal polynomials. Two of the most popular measures are $d\mu = dx$ and $d\mu = \frac{dx}{\sqrt{1-x^2}}$. By using $d\mu = dx$ on $[-1, 1]$, the generated polynomials called Legendre polynomials and by using $d\mu = \frac{dx}{\sqrt{1-x^2}}$ on $[-1, 1]$ the generated polynomials called Chebyshev polynomials.

These two polynomial sets have different applications in approximation theory. For example, the nodes we use in polynomial interpolation are the roots of the Chebyshev polynomials and the Legendre polynomials are the coefficient of the Taylor series [4, 30].

3.2 Polynomial Approximation: ReLU

Several methods have been proposed in the literature for polynomial approximation including Taylor series, Chebyshev polynomials, etc. [4, 30]. We first try these approaches for finding the best approximation for the ReLU function and then present our proposed approach that provides better approximation than all these methods. We investigate the following methods for approximating the ReLU function.

1. Numerical analysis
2. Taylor series
3. Standard Chebyshev polynomials
4. Modified Chebyshev polynomials



(a) Approximation of ReLU using different methods

(b) Approximation of ReLU based on our approach

Figure 2: Polynomial Approximation of ReLU

5. Our approach based on the derivative of ReLU function

Due to space limits, we do not provide detailed results for all these methods and only explain our conclusion for each method.

Method 1: Numerical analysis: For this method, we generate a set of points from ReLU function and give this set of inputs to the approximation function and a constant degree for the activation function. The main shortcoming of this method is poor performance. We experimented with polynomials of degree 3 to 13 and for lower degree polynomials, the accuracy drops considerably. For achieving a good accuracy, we have to increase the degree which makes it inefficient when we are working with encrypted data. Our investigation showed that the method 1 is not a good approach for approximating the ReLU function.

Method 2: Taylor series: In this method, we use Taylor series [4, 30], a popular method for approximating functions. We used different degrees for approximating the ReLU function and trained the model using polynomials of different degrees. Two main issues make this method inefficient. The first issue is the high degree of polynomial approximation, although it's lower than the method 1, the degree is still high to be used with HE schemes. The second and more important issue is the interval of approximation. Basic idea of Taylor series is to approximate a function in a neighbor of a point. For the points that are not included in the input interval, the approximation error is much higher than points included in this interval. For example, in the MNIST dataset, pixel values are integers in the interval $[0, 255]$ and this method cannot cover this interval. If we can approximate the ReLU function in a large interval, we can avoid using extra layers and this can be done using Chebyshev polynomials [30] as explained below.

Method 3: Standard Chebyshev polynomials: Chebyshev polynomials are not as popular as the previous methods (i.e., 1 and 2). However, they have a specific feature that makes them more suitable for our problem. In this method, we approximate a function in an interval instead of a small neighborhood of a point. HE schemes are over integers with message space $\mathbb{Z}$; therefore, we extend the interval to be able to cover integers. In the standard Chebyshev polynomials, we use $d\mu = \frac{dx}{\sqrt{1-x^2}}$ as the standard norm. We approximate the ReLU function with this method and train the model based on that. As shown in Table 1, the accuracy is much better than methods 1 and 2, however, it is still not a good performance in comparison with the original activation function. One

Table 1: Performance of the trained CNN using Different Approximation Methods

Method	Accuracy
Numerical analysis	56.87%
Taylor series	40.28%
Standard Chebyshev	68.98%
Modified Chebyshev	88.53%
Our Approach	98.52%

way to improve the accuracy and find better approximation is to modify the measure function based on the structure of the ReLU function which is done in the next method.

Method 4: Modified Chebyshev polynomials: In order to simulate the structure of ReLU function, we changed the standard norm used in Chebyshev polynomials to $e^{(\frac{-1}{(1e-5+(x)^2)})}$ and were able to achieve much better results compared to all the previous methods. The idea behind this method is that the measure for Chebyshev polynomials mainly concentrates at the end points of the interval which causes interpolation at mostly initial and end points with two singularities at both ends. While the second measure evens out through the whole real line and puts zero weight at the center. This behavior causes less oscillation in the resulting approximation and hence more similarities of derivatives with Sigmoid function. However, this improvement in performance is still not good enough.

Let us look at why the accuracy drops significantly in the above methods. Our goal is to approximate the ReLU function, however, note the derivative of the ReLU function is more important than the structure of the ReLU function. So, we changed our approach to the problem and focused on approximating the derivative of the ReLU function instead of approximating the ReLU function as explained below.

Method 5: Our approach based on the derivative of ReLU function: All the above methods are based on simulating the activation function with polynomials. In this method, however, we use another approach and consider the derivative of the activation function because of its impact on the error calculation and updating the weights. Therefore, instead of simulating the activation function, we simulate the derivative of the activation function. The derivative of ReLU function is like a Step function and is non-differentiable in point 0. If the function is continuous and infinitely derivative, we can approximate it more accurately than a non-continuous or non-infinitely differentiable function. Instead of approximating the ReLU function, we simulate the structure of derivation of the ReLU function, a Step function. Sigmoid function is a bounded, continuous and infinitely differentiable function, it also has a structure like derivative of the ReLU function in the large intervals. We approximate the Sigmoid function with the polynomial, calculate the integral of the polynomial, and use it as the activation function. As shown in Table 1, this method achieves the best approximation of the ReLU function and we will use this method for approximation in this paper.

Figure 2a shows the structure of the functions generated by all the above approximations methods in comparison with the ReLU function whereas Figure 2b shows only the method 5 in comparison with the ReLU function. These two figures show that the last method simulates the structure of the ReLU function considerably better than other methods. In Figure 2a, for some values less than zero, the structure of the polynomial goes up and down. This behavior has impact on the performance of the model that is trained based on these polynomials. However, in Figure 2b, the structure of function is almost same as the ReLU function, and for this reason, we expect to have a performance close to the ReLU function. We try to keep the degree of the polynomial as low as possible, therefore,

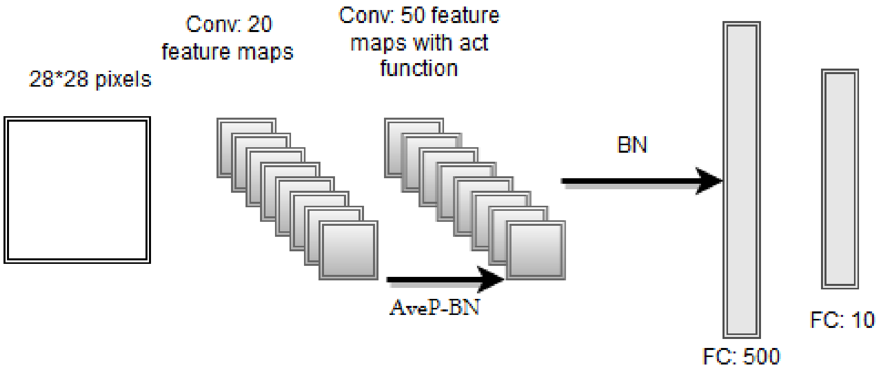


Figure 3: CNN Model 1 (AveP and BN stand for Average Pooling and Batch normalization).

we only work with degree 2 and degree 3 polynomials.

3.3 Polynomial Approximation: Sigmoid and Tanh

In addition to the ReLU function, we also approximate two other popular activation functions: Sigmoid and Tanh. Approximating these two functions are more straight forward compared with the ReLU, because they are infinitely derivative. In this paper, we experiment with polynomial approximations of the Sigmoid function $\frac{1}{1+e^{-x}}$ and the Tanh function $Tanh(x)$, over a symmetric interval $[-l, l]$ using two different orthogonal system of polynomials. As the first choice, we consider Chebyshev polynomials on the stretched interval which come from the measure $d\mu = \frac{dx}{l\sqrt{1-(x/l)^2}}$. Our second choice comes from the measure $d\mu = e^{-(l/x)^2}dx$.

We note that the measure for Chebyshev polynomials mainly concentrates at the end points of the interval which causes interpolation at mostly initial and end points with two singularities at both ends. While the second measure evens out through the whole real line and puts zero weight at the center. This behavior causes less oscillation in the resulting approximation and hence more similarities of derivatives with Sigmoid function.

Now that we have found polynomial approximations, the next step is training CNN models using these polynomials and comparing the performance of the model with the original models.

3.4 CNN Model 1 with Polynomial Activation Function

In order to evaluate effectiveness of different approximation methods, we use a CNN and the MNIST dataset [43] for our experiments. The MNIST dataset consists of 60,000 images of hand written digits. Each image is a 28x28 pixel array, where value of each pixel is a positive integer in the range $[0, 255]$. We used the training part of this dataset, consisting of 50,000 images, to train the CNN and the remaining 10,000 images for testing.

The architecture of the CNN we use is shown in Figure 3. This CNN has similar architecture to the CNN used in [9] and the light CNN used in [49]. This will allow us to provide direct comparison with those work.

We train the CNN using Keras library [52] on the MNIST dataset. We train different models using each approximation method discussed above. We use polynomials of degree 2 to replace the ReLU function in the first four methods and polynomial of degree 3 for the last method. Table 1 shows accuracy of the trained model using each approximation method. As we can see, different approximation methods result in widely different accuracy values and as expected, our proposed method 5 achieves the best accuracy among all the methods. In the rest of this paper, we only use this method for approximation of the ReLU function, and whenever approximation method is mentioned, it refers to the method 5.

We train the model based on a CNN structure similar to the one used in [49] and [9]. We were able to achieve 98.52% accuracy while their accuracy was 97.95% . The main difference comes from the polynomial approximation method used; the approach in [49] used Taylor series to approximate the ReLU function whereas we used our proposed method 5. These results show that our polynomial approximates the ReLU function better.

However, both results are still far from the state-of-the-art digit recognition problem (99.77%). This is due to small size and simple architecture of the CNN used here. We use larger and more complex CNNs with more layers that are able to achieve accuracy much closer to the state-of-the-art as explained in section 3.5.

To ensure that the polynomial approximation is independent of the structure of the CNN and works well with different CNN structures, we change the structure of the CNN and train models based on the new one. The accuracy of the model based on this CNN with polynomial of degree 3 as the activation function is 98.38% and close to the first structure. This shows that the behavior of the polynomial approximation of the ReLU function is robust against the changes in the structure. In addition, to analyze the relation between the degree of the polynomial and the performance of the model, we change the degree of the polynomial from 3 to 8 and calculate the accuracy of the model. As expected, the higher degree polynomials results in higher performance and we were able to achieve 99.21% accuracy.

3.5 CNN Model 2 with Polynomial Activation Function

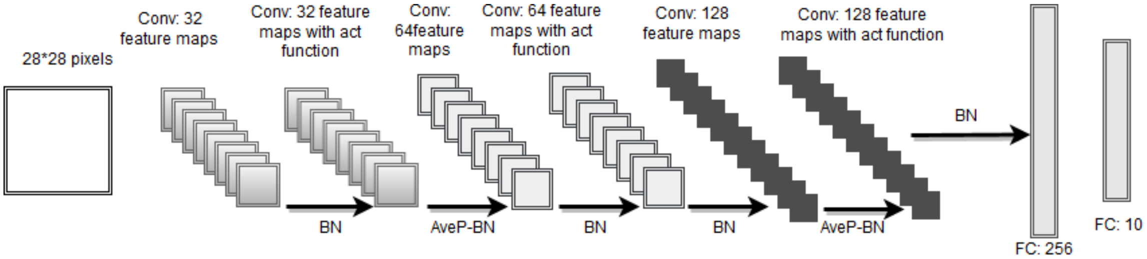

28*28 pixels
Conv: 32 feature maps
Conv: 32 feature maps with act function
Conv: 64feature maps
Conv: 64 feature maps with act function
Conv: 128 feature maps
Conv: 128 feature maps with act function
BN
AveP-BN
BN
BN
AveP-BN
BN
FC: 256
FC: 10

Figure 4: CNN Model 2 (AveP and BN stand for Average Pooling and Batch normalization).

Next, we implement several CNNs with more layers and more complex structures to improve performance of the model and get closer to the state-of-the-art. We implement each model in Keras library [52] and measure the performance to find a relationship between the depth of the CNN and the accuracy of the model.

In order to provide a comparison, we use a CNN with similar structure to the one used in [49]. However, we start with a simpler CNN and gradually add layers to the CNN to check how the performance of the model changes as the number of layers increases and the model becomes more complex.

First, we implement a CNN with 3 convolutional layers The accuracy of the trained model is 99.10% which is very close to the same CNN with the ReLU as activation function, 99.15%. Next, we add an activation function to the first convolutional layer and the accuracy increases to 99.16%. We then add two more convolutional layers to the CNN and train the new model. The accuracy of the trained model increases to 99.29%. We go one step further and add an activation function after each convolutional layer. The accuracy for this CNN is 99.32%. Finally, we try the same structure as the deep CNN used in [49]. However, the degree of our polynomial is 3 while the polynomial used in [49] is of degree 2. Therefore, we design our CNN, shown in Figrue 4, to have the same depth as the deep CNN used in [49].

Table 2: Performance of the trained CNN using different Activation Functions and their Replacement Polynomials

Activation Function	Original Model	Model with Polynomial
ReLU	99.56%	99.52%
Sigmoid	98.85%	98.94%
Tanh	97.27%	98.15%

The accuracy of this model, shown in Figure 4, is 99.52% which is very close to the accuracy of the same CNN that uses the ReLU function, 99.56%. Our accuracy is higher than both methods in [9] (98.95%) and [49] (99.30%) over plaintext for a CNN with the same structure. We also train CNNs with Sigmoid and Tanh as activation functions. To provide a comparison, we use the CNN model 2 (Figure 4) and only change the activation functions to Sigmoid and Tanh instead of the ReLU function. The results shown in Table 2.

4 Experimental Results: Deep Convolutional Neural Network over Encrypted Data

In this section, we present results of implementing adopted version of CNNs (ReLU is replaced with polynomial approximation) over encrypted data. We train the models using plaintext data and measure the accuracy of the built model for classification of encrypted data. We used HELib [14] for implementation and all computations were run on a computer with 16GB RAM, Intel Xeon E5-2640, 2.4GHz and Ubuntu 16.04.

We use the CNN models trained in the previous section. We give encrypted inputs to the trained networks and measure the accuracy of the outputs. We implement the CNN Model 1 (Figure 3). Thanks to the SIMD feature in the HELib, in each round of classification, we can classify a batch of encrypted images. We measure the running time for encryption and sending data from the client to the server. We also measure the running time for classifying this encrypted batch as well as amount of the data transferred in the process.

First, the encryption scheme is initiated in the client side using the the plaintext base p a 16-digit prime number, $L = 6$ and $k = 80$ (security level which is equivalent to AES-128). Then, the input images were encrypted. As the HELib supports SIMD operations, the images were encrypted in batches. So, for each batch of images - each having the size of 28×28 pixels, we obtained one set of ciphertext representation of size 28×28. Pixel values from each location for all the images in the batch were encrypted to one ciphertext.

Table 3: Breakdown of Running Time of CNN Model 1 (Figure 3) over Encrypted MNIST Dataset

Layer	Time (seconds)
Conv layer(20 feature maps)	13.078
Average Pooling Layer	7.630
Conv layer (50 feature maps)	77.642
Average Pooling Layer	6.543
Activation layer	9.763
2 Fully Connected (256 and 10 neurons)	34.32

Table 4: Running Time for Data Transfer (seconds)

Layer	Our Approach (s)	CrytoNets [9] (s)
Encryption	15.7	122
Communication	320	570
Decryption	1	5

Table 5: Comparison with the state-of-the-art Solutions

Dataset	Criteria	Our Approach	CryptoNets [9]	[49]*	DeepSecure [50]	SecureML [39]
MNIST	Accuracy	99.52%	98.95%	99.30%*	98.95%	93.4%
	Run Time (s)	320	697	N/A	10649**	N/A
	Data Transfer	336.7MB	595.5MB	N/A	722GB**	N/A
	# p/h	163840	51739	N/A	2769**	N/A
CIFAR-10	Accuracy	91.5%***	N/A	N/A	N/A	N/A
	Run Time (s)	11686	N/A	N/A	N/A	N/A
	Data Transfer	1803MB	N/A	N/A	N/A	N/A
	# p/h	2524	N/A	N/A	N/A	N/A

*: The model is implemented over plaintext and results over encrypted data is not reported.
**: The values are extrapolated.
***: The model is trained over plaintext with polynomials as activation function.

Then, the encrypted images along with the encryption parameters and the public key were sent to the server side, where the server runs the CNN over the encrypted data. In our experiments, we classify a batch of ciphertext with size 8192 (the same batch size used in [9]) and provide the running time for classification. Table 3 shows breakdown of the time it takes to apply CrytoDL to the MNIST dataset using the CNN Model 1 of Figure 3. We also provide time required for encryption, transferring and decryption time as shown in Table 4. As it can be seen, our approach is much faster than [9].

4.1 Comparison with the state-of-the-art Solutions

In this section, we compare our results with the state-of-the-art privacy-preserving classification of neural networks. This includes approaches based on the HE as well as secure multi-party computation (SMC) as shown in Table 5.

The two closest work to our approach are CryptoNets [9] and [49]. CryptoNets uses HE and implements a CNN with two convolutional layers and two fully connected layers. It assumes Sigmoid is used as the activation function and replaces the max pooling with scaled mean-pooling and the activation function with square function. As it can been seen in Table 5, our approach significantly outperforms CryptoNets in all aspects. Note that to provide a fair comparison, we use machines with similar configuration (Intel Xeon E5-1620 CPU running at 3.5GHz with 16GB of RAM in CryptoNets and Intel Xeon E5-2640, 2.4GHz with 16GB RAM in our case) for the experiments. Chabanne et al. [49] use Taylor series for approximating the ReLU function and also use batch normalization layers for improving the performance of the model. They don't provide any results over the encrypted data. So, we can't provide comparison w.r.t to the performance measures (e.g., run time, amount of data transferred, number of predictions) over encrypted data but our approach provide much better accuracy.

DeepSecure [50] and SecureML [39] are two recent works based on SMC techniques. Darvish et al. [50] present DeepSecure that enables distributed clients (data owners) and cloud servers, jointly evaluate a deep learning network on their private assets. It uses Yao's Garbled Circuit (GC) protocol to securely perform deep learning. They perform experiments on MNIST dataset and report the results. As shown in Table 5, our approach significantly outperforms DeepSecure in all aspects. Note that in [50], authors provide the communication and computation overhead for one instance, and the proposed protocol classifies one instance at each prediction round. Our approach, on the other hand, can classify a bath of instances in each round with size 8192 or larger. To provide a fair comparison, we extrapolate the running time and number of communications reported accordingly. Mohassel and Zhang [39] present SecureML that aims to develop privacy-preserving training and classification of neural networks using SMC techniques. In their proposed approach, a data owner shares the data with two servers and the servers run the machine learning algorithm using two-party computation (2PC) technique. However, they can only implement a very simple neural network with 2 hidden layers with 128 neurons in each layer, without any convolutional layers and as shown in the Table 5, the accuracy is very low. Implementing a CNN with their approach is not practical and hence we cannot provide comparison w.r.t to CNNs. To provide a comparison, we implemented the same neural network (2 hidden layers with 128 neurons in each layer and without any convolutional layers) using our approach. They report 14 seconds as the running time for 100 instances, and the running time of our approach is 12 seconds. It is also worth noting that by increasing the size of the batch input, the running time increases sub-linearly in [39] whereas in our solution, the running time remains the same for larger sizes of the batch input. Additionally, unlike [39] our solution does not need any communications between the client and the server for providing privacy-preserving predictions.

Generally, the SMC-based solutions have a big shortcoming which is the very large number of communications since we need interactions between client and server for each operation. For example, DeepSecure has a huge communication cost of 722GB for a relatively small network (CNN Model 1) whereas CryptoNets's communication cost is 595.5MB and ours is only 336.7MB for the same network. Also, since the client participates in the computations, information about the model could possibly leak. For example, the client can learn information such as the number of layers in the CNN, the structure of each layer and the activation functions.

4.2 CIFAR-10 Results

To further show applicability of our proposed approach for more complicated network architectures, we use CIFAR-10 [51] which is one of the widely used benchmark dataset for deep learning, to train a CNN and implement it over encrypted data. The CIFAR-10 dataset consists of 60000 32×32

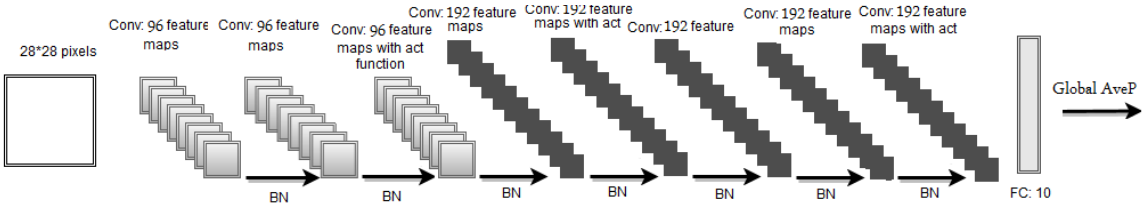


Figure 5: The architecture of CNN for CIFAR-10 classification

colour images categorized in 10 classes, with 6000 images per class. There are 50000 training and 10000 test images. We train the CNN shown in Figure 5 using CIFAR-10 dataset and we achieved 91.5% accuracy with polynomials as the activation functions whereas the accuracy with the original activation function is 94.2%. As shown in Table 5, CIFAR-10 is much slower compared to the MNIST. This was expected since both the dataset and the CNN are much more complex.

4.3 Discussion

Additional Datasets: We used MNIST to be able to provide a comparison with related work which only report results on the MNIST. Additionally, we reported results on CIFAR-10 for the first time. Our solution is independent of the dataset and can be applied to other datasets such as CIFAR-100 and ImageNet. However, these datasets usually require GPU for efficient implementation and take very long time to train. This is left to our future work.

Training CNN over encrypted data: Although our focus in this paper is on classification phase of CNNs, it is possible to train neural networks over encrypted data. If we replace all the activation functions and the loss function with polynomials, back-propagation can be computed using additions and multiplications. However, there are several challenges in doing so as stated in [9]. One major challenge is computational complexity. Even when we deal with plaintext, CNNs are slow to train and the deep learning community is putting a lot of effort towards increasing performance of this training process by using sophisticated hardware such as graphics processing units (GPUs) or field programmable gate arrays (FPGAs). However, adding HE to the process will make the process much slower, and therefore using leveled HE does not seem to be practical. Another challenging aspect in training while using HE is designing the architecture of the CNNs. An essential part of designing efficient CNNs is the ability of data scientists to inspect the data and the trained models, and to tune the network by correcting mislabeled items, and adding features when required. This ability will not be available when encryption is used.

5 Related Work

Graepel et al. use a somewhat HE scheme to train two machine learning classifiers: Linear Mean and Fisher's Linear Discriminate (FLD) [13]. They proposed division-free algorithms to adapt to limitations of HE algorithms. They focus on simple classifiers such as the linear means classifier, and do not consider more complex algorithms. Also, in their approach, the client can learn the model, and they consider a weak security model. Bost et al. use a combination of three homomorphic systems (Quadratic Residuosity, Piallier, and BGV schemes), and garbled circuits to provide privacy-preserving classification for three different machine learning algorithms, namely Hyperplane Decision, Naive Bayes, and Decision trees [5]. Their approach is based on SMC, considers only classical machine learning algorithms and is only efficient for small data sets. Our proposed approach is based only on HE, focuses on the deep learning algorithms, and is efficient for large datasets.

Xie et al. discuss theoretical aspects of using polynomial approximation for implementing neural network in encrypted domain [29]. Building on this work, Dowlin et al. present CryptoNets, a neural network classifier on encrypted data [9]. Chabanne et al. improve accuracy of CryptoNets by combining the ideas of Cryptonets' solution with the batch normalization principle [49]. These two are the closest to our work and were discussed in section 4.1.

[50] and [39] are two recent works based on SMC techniques and were discussed in section 4.1. Aslett et al. propose methods for implementing statistical machine learning over encrypted data and implement extremely random forests and Naive Bayes classifiers over 20 datasets [44]. In these algorithms, the majority of operations are addition and multiplication and they show that performing algorithms over encrypted data without any multi-party computation or communication is practical. They also analyze HE tools for use in statistical machine learning [3]. Several methods have been proposed for statistical analysis over encrypted data, specifically for secure computation of a χ^2-test on genome data [21]. Shortell et al. use the Taylor expansion of $ln(x)$ to estimate the natural logarithm function by a polynomial of degree 5 [26]. Livni et al. analyzed the performance of polynomial as an activation function in neural networks [20]. However, their solution cannot be used for our purpose because they approximate the Sigmoid function on the interval [-1,1] while the message space of HE schemes is integers. Our methods is able to generate polynomial approximation for an arbitrary interval.

There are also a few recent work that look at privacy issues in training phase, specifically for back-propagation algorithm [47, 48]. Bu et al. propose a privacy-preserving back-propagation algorithm based on BGV encryption scheme on cloud [48]. Their proposed algorithm offloads the expensive operations to the cloud and uses BGV to protect the privacy of the data during the learning process. Zhang et al. also propose using BGV encryption scheme to support the secure computation of the high-order bakc-propagation algorithm efficiently for deep computation model training on cloud [47]. In their approach, to avoid a multiplicative depth too big, after each iteration the updated weights are sent to the parties to be decrypted and re-encrypted. Thus, the communication complexity of the solution is very high. Unlike these papers, our focus is on privacy-preserving classification problem.

6 Conclusion and Future Work

In this paper, we developed new solutions for running deep neural networks over encrypted data. In order to implement the deep neural networks within limitations of the HE schemes, we introduced new techniques to approximate the activation functions with the low degree polynomials. We then used these approximation to train several deep CNNs with the polynomial approximation as the activation function over the encrypted data and measured the accuracy of the trained models. Our results show that polynomials, if chosen carefully, are the suitable replacements for activation functions to adopt deep neural networks within the HE schemes limitations. We were able to achieve 99.52% accuracy and make close to 164000 predictions per hour when we applied our approach to the MNIST dataset. We also reported results on CIFAR-10 (for the first time to the best of our knowledge) and were able to achieve 91.5% accuracy. These results show that our proposed approach provides efficient, accurate, and scalable privacy-preserving predictions and significantly outperforms the state-of-the-art solutions.

For future work, we plan to implement more complex models over GPU and also study privacy-preserving training of deep neural networks in addition to the classification.

References

1. Removed for anonymization.
2. L. J. M. ASLETT, P. M. ESPERANÇA, C. C. HOLMES, Encrypted statistical machine learning: new privacy preserving methods. CoRR abs/1508.06845 (2015).
3. L. J. M. ASLETT, P. M. ESPERANCA, C. C. HOLMES, A review of homomorphic encryption and software tools for encrypted statistical machine learning. Tech. rep., University of Oxford, 2015.
4. K. ATKINSON, W. HAN, Theoretical Numerical Analysis: A Functional Analysis Framework. Texts in Applied Mathematics. Springer New York, 2009.
5. R. BOST, R. A. POPA, S. TU, S. GOLDWASSER, Machine learning classification over encrypted data. In 22nd Annual Network and Distributed System Security Symposium, NDSS, San Diego, California, USA (2015).
6. Y. CHEN, G. GONG, Integer arithmetic over ciphertext and homomorphic data aggregation. In Communications and Network Security (CNS), IEEE Conference on (Sept 2015), pp. 628–632.
7. T. S. DEVELOPERS, SageMath, the Sage Mathematics Software System (Version 7.1), 2016. http://www.sagemath.org.
8. N. DOWLIN, R. GILAD-BACHRACH, K. LAINE, K. LAUTER, M. NAEHRIG, J. WERNSING, Manual for using homomorphic encryption for bioinformatics. Tech. Rep. MSR-TR-2015-87, November 2015.
9. N. DOWLIN, R. GILAD-BACHRACH, K. LAINE, K. L. M. NAEHRIG, J. WERNSING, Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. Tech. Rep. MSR-TR-2016-3, 2016.
10. T. ELGAMAL, A public key cryptosystem and a signature scheme based on discrete logarithms. In Proceedings of CRYPTO 84 on Advances in Cryptology (New York, NY, USA, 1985), Springer-Verlag New York, Inc., pp. 10–18.
11. C. GENTRY, A Fully Homomorphic Encryption Scheme. PhD thesis, Stanford, CA, USA, 2009. AAI3382729.
12. S. GOLDWASSER, S. MICALI, Probabilistic encryption & how to play mental poker keeping secret all partial information. In Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing (New York, NY, USA, 1982), STOC '82, ACM, pp. 365–377.
13. T. GRAEPEL, LAUTER, M. NAEHRIG, ML confidential: Machine learning on encrypted data. In Proceedings of the 15th International Conference on Information Security and Cryptology (Berlin, Heidelberg, 2013), ICISC'12, Springer-Verlag, pp. 1–21.

14. S. HALEVI, V. SHOUP, Algorithms in HElib. In Advances in Cryptology - CRYPTO - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, Proceedings, Part I (2014), pp. 554– 571.

15. N. ISLAM, W. PUECH, K. HAYAT, R. BROUZET, Application of Homomorphism to Secure Image Sharing. Optics Communications 284, 19 (Sept. 2011), 4412–4429.

16. B. KARLIK, A. V. OLGAC, Performance analysis of various activation functions in generalized mlp architectures of neural networks. International Journal of Artificial Intelligence and Expert Systems 1, 4 (2011), 111–122.

17. M. KIM, K. E. LAUTER, Private genome analysis through homomorphic encryption. IACR Cryptology ePrint Archive 2015 (2015), 965.

18. M. LICHMAN, UCI machine learning repository, 2013.

19. F. LIU, W. K. NG, W. ZHANG, Secure scalar product for big-data in MapReduce. In Big Data Computing Service and Applications (BigDataService), IEEE First International Conference on (March 2015), pp. 120–129.

20. R. LIVNI, S. SHALEV-SHWARTZ, O. SHAMIR, On the computational efficiency of training neural networks. CoRR abs/1410.1141 (2014).

21. W. -J. LU, Y. YAMADA, J. SAKUMA, Privacy-preserving genome-wide association studies on cloud environment using fully homomorphic encryption. BMC Medical Informatics and Decision Making 15, 5 (2015), 1–8.

22. P. PAILLIER, Public-key cryptosystems based on composite degree residuosity classes. In Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques (Berlin, Heidelberg, 1999), EUROCRYPT'99, Springer-Verlag, pp. 223–238.

23. P. PILOTTE, Neural Network Toolbox, 2016.

24. S. RANE, W. SUN, A. VETRO, Secure distortion computation among untrusting parties using homomorphic encryption. In ICIP (2009), IEEE, pp. 1485–1488.

25. R. RIVEST, L. ADLEMAN, M. DERTOUZOS, On data banks and privacy homomorphisms. In Foundations on Secure Computation, Academia Press (1978), pp. 169–179.

26. T. Shortell, A. Shokoufandeh, Secure signal processing using fully homomorphic encryption. In Advanced Concepts for Intelligent Vision Systems - 16th International Conference, ACIVS, Catania, Italy, October 26-29, Proceedings (2015), pp. 93–104.

27. M. Togan, C. Plesca, Comparison-based computations over fully homomorphic encrypted data. In Communications (COMM), 10th International Conference on (May 2014), pp. 1–6.

28. T. Veugen, Encrypted integer division and secure comparison. Int. J. Appl. Cryptol. 3, 2 (June 2014), 166–180.

29. P. Xie, M. Bilenko, T. Finley, R. Gilda-Bachrach, K. E. Lauter, and M. Naehrig, Crypto-nets: Neural networks over encrypted data. CoRR abs/1412.6181 (2014).

30. Y. Xu, Orthogonal polynomials of several variables. Encyclopedia of Mathematics and its Applications 81 (2001).

31. Y. Zhang, W. Dai, X. Jiang, H. Xiong, S. Wang, Foresee: Fully outsourced secure genome study based on homomorphic encryption. BMC Medical Informatics and Decision Making 15, 5 (2015), 1–11. Yuchen Zhang, Wenrui Dai contributed equally to this work.

32. Microsoft Azure Machine Learning, https://azure.microsoft.com/en-us/services/machine-learning/, Accessed: 2017-02-20.

33. Google Prediction API, https://cloud.google.com/prediction/, Accessed: 2017-02-20.

34. GraphLab, http://www.select.cs.cmu.edu/code/graphlab/, Accessed: 2017-02-20.

35. Ersatz Labs, http://www.ersatzlabs.com/, Accessed: 2017-02-20.

36. OpenMP, http://www.openmp.org/, Accessed: 2017-02-22.

37. M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, Deep Learning with Differential Privacy. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16), Vienna, Austria. pp 308–318.

38. R. Shokri, V. Shmatikov, Privacy-Preserving Deep Learning. Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15). Denver, Colorado, USA. pp 1310–1321.

39. P. Mohassel, Y, Zhang, SecureML: A System for Scalable Privacy-Preserving Machine Learning. IACR Cryptology ePrint Archive, 2017.

40. F. Tramèr and F. Zhang, A. Juels, M.K. ReiterT. Ristenpart, Stealing Machine Learning Models via Prediction APIs. 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016. 601–618.

41. M. Fredrikson, S. Jha, T. Ristenpart, Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. ACM Conference on Computer and Communications Security 2015, Denver, CO, USA, October 12-6, 2015, 1322-1333.

42. R. Shokri, M. Stronati, V. Shmatikov. Membership Inference Attacks against Machine Learning Models. arXiv preprint. arXiv: abs/1610.05820 (2016).

43. L. Yann, C. Corinna. MNIST handwritten digit database. 2010.

44. L. J. M. Aslett, P. M. Esperanç, C. C. Holmes, Encrypted statistical machine learning: new privacy-preserving methods. CoRR, abs/1508.06845. 2015

45. D. Boneh, Wu. David, H Jacob, Using homomorphic encryption for large scale statistical analysis. 2012.

46. J. Zhan, Using Homomorphic Encryption For Privacy-Preserving Collaborative Decision Tree Classification. 2007 IEEE Symposium on Computational Intelligence and Data Mining, 637-645.

47. Q. Zhang, L. T. Yang, Z. Chen, Privacy Preserving Deep Computation Model on Cloud for Big Data Feature Learning. IEEE Transactions on Computers, 2016.

48. F. Bu, Y. Ma, Z. Chen, H. Xu, Privacy Preserving Back-Propagation Based on BGV on Cloud, 17th IEEE International Conference on High Performance Computing and Communications, HPCC 2015, 7th IEEE International Symposiumon Cyberspace Safety and Security, CSS 2015, and 12th IEEE International Conference on Embedded Software and Systems, ICESS 2015, New York, NY, USA, August 24-26, 2015, 1791–1795, 2015.

49. H. Chabanne, A. Wargny, J. Milgram, C. Morel, E. Prouff, Privacy-Preserving Classification on Deep Neural Network, Cryptology ePrint Archive, Report 2017/035, 2017.

50. B. D. Rouhani, M. S. Riazi, F. Koushanfar, DeepSecure: Scalable Provably-Secure Deep Learning, CoRR, abs/1705.08963, 2017.

51. A. Krizhevsky, V. Nair, G. Hinton, CIFAR-10 (Canadian Institute for Advanced Research), www.cs.toronto.edu/ kriz/cifar.html.

52. C. François, Keras. 2017, https://github.com/fchollet/keras

53. H. Takabi, E. Hesamifard, M. Ghasemi, Privacy Preserving Multi-party Machine Learning with Homomorphic Encryption, Private Multi-Party Machine Learning, NIPS 2016 Workshop, 2016.

54. E. Hesamifard, H Takabi, M. Ghasemi, CryptoDL: Towards Deep Learning over Encrypted Data, Annual Computer Security Applications Conference (ACSAC), 2016.

55. Z. Brakerski, C. Gentry, V. Vaikuntanathan, (Leveled) Fully Homomorphic Encryption Without Bootstrapping, Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, ITCS '12, 2012, 978-1-4503-1115-1, Cambridge, Massachusetts, 309–325, 17, ACM, New York, NY, USA,