

PRIVACY PRESERVING MACHINE LEARNING AS A SERVICE

Ehsan Hesamifard

Dissertation Prepared for the Degree of

DOCTOR OF PHILOSOPHY

UNIVERSITY OF NORTH TEXAS

May 2020

APPROVED:

Bill Buckles, Major Professor

Song Fu, Committee Member

Kirill Morozov, Committee Member

Mark Thompson, Committee Member

Barrett Bryant, Chair of the Department
of Computer Science and
Engineering

Hanchen Huang, Dean of the College of
Engineering

Victor Prybutok, Dean of the Toulouse
Graduate School

Hesamifard, Ehsan. *Privacy Preserving Machine Learning as a Service*. Doctor of Philosophy (Computer Science and Engineering), May 2020, 133 pp., 25 tables, 1 appendix, 97 numbered references.

Machine learning algorithms based on 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. We focus on training and classification of the well-known neural networks and convolutional neural networks. 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 neural networks with the approximation polynomials instead of original activation functions and analyze the performance of the models. Finally, we implement neural networks and convolutional neural networks over encrypted data and measure performance of the models.

Copyright 2020
by
Ehsan Hesamifard

ACKNOWLEDGEMENTS

I sincerely thank all my mentors in my whole academic life with special thanks to Dr. Bill Buckles, my brilliant, knowledgeable, patient and respectable advisor. He is a high-level professional with modesty of a humble person who worked with me as a team member while guiding me to find the right path. He is considerate while encouraging which gives you the ease of mind while pushing you forward. I cannot thank him enough for his valuable advice.

I would also like to appreciate all the other committee members Drs. Song Fu, Mark Thompson, and Kirill Morozov who helped me to improve my dissertation by giving precious advice. I am thankful for all the time and effort that they put in to this document. Im extremely grateful to the completion of my dissertation would not have been possible without the support and nurturing of Dr. Mehdi Ghasemi. To my lab-mate Shoaib Khan, thank you for your help and support. I would also like to thank my former committee members Drs. Hassan Takabi, Xiaohui Yuan and Eduardo Blanco.

I wish to thank Dr. Barrett R. Bryant, Diana Bergeman and Melanie Dewey from the Department of Computer Science and Engineering at the University of North Texas which was my second home for 5 years.

Words can not express how grateful I am to my mother and father for all of the sacrifices that youve made on my behalf. Special thank to my wife for being my best friend. I owe you everything. My sincere appreciation goes to my brother and my sisters, for their everyday support and love in my whole life.

I dedicate this work to my parents who have always believed in me and my loving wife, Hanieh Soleimanifar who has supported me unconditionally, and stood by me through all my absence and impatience. Without you, I would have stopped these studies a long time ago.

TABLE OF CONTENTS

	Page
ACKNOWLEDGEMENTS	iii
LIST OF TABLES	vi
CHAPTER 1 INTRODUCTION	1
1.1. Research Questions	5
1.2. Significance of the Research	5
CHAPTER 2 BACKGROUND	9
2.1. Homomorphic Encryption Schemes	9
2.2. Neural Networks	11
2.3. Convolutional Neural Networks (CNNs / ConvNets)	15
CHAPTER 3 RELATED WORK	20
3.1. Homomorphic Encryption	20
3.2. Secure Multiparty Computation	22
3.3. Other Related Work	25
3.4. Classic Machine Learning	28
CHAPTER 4 CHALLENGES AND SOLUTIONS	37
4.1. Challenges	38
4.2. Different Data Distribution Scenarios	42
CHAPTER 5 POLYNOMIAL APPROXIMATION	46
5.1. Polynomial Approximation	46
5.2. Polynomial Computation over Encrypted Values	51
CHAPTER 6 RESULTS	54
6.1. Homomorphic Encryption Performance	54
6.2. Neural Networks over Encrypted Data	55

6.3.	CNN Over Encrypted Data	72
6.4.	Comparison with the State-of-the-Art	86
6.5.	Conclusion	95
CHAPTER 7 CONCLUSION AND FUTURE WORK		97
7.1.	Conclusion	97
7.2.	Future Work	98
APPENDIX: TRAINING BASED ON POLYNOMIALS AS ACTIVATION FUNCTIONS		101
REFERENCES		123

LIST OF TABLES

		Page
5.1	Polynomial approximation for the Sigmoid function on the interval $[-10^3, 10^3]$. $p_1(x)$ generated by method 1 (Equation 5.9) and $p_2(x)$ generated by method 2 (Equation 5.10).	49
5.2	Polynomial approximation for the Sigmoid function on different intervals. $p_1(x)$ generated by method 1 (Equation 5.9) and $p_2(x)$ generated by method 2 (Equation 5.10).	50
5.3	Polynomial computation over encrypted values	53
6.1	Key Generation and Key Transfer time	55
6.2	Datasets (I, F and C represent Instances, Features and Classes respectively).	58
6.3	Accuracy vs. other parameters for Ovarian Dataset. Blue and red are polynomials from two different measure functions.	59
6.4	Accuracy vs. other parameters for Crab Dataset. Blue and red are polynomials from two different measure functions.	60
6.5	Accuracy vs. other parameters for Wine Dataset. Blue and red are polynomials from two different measure functions.	61
6.6	Performance of the neural network model based on $f_1(x) = \frac{1}{1+e^{-x}}$, $f_2(x) = \frac{2}{1+e^{-4x}} - 1$, $f_3 = x^2$ and the polynomial approximation in layer 2 only. a , i , d and n represent accuracy, number of iterations, polynomial degree and $\ \cdot\ _{2,\mu}$ respectively. Precision of the polynomial coefficient is 10, except Datasets 3 and 9 which are 20 and 15. We report the polynomial approximation with minimum degree. Refer to Table 6.2 for the list of datasets.	65
6.7	Neural Network over Encrypted Data	68
6.8	Training Neural Network over Encrypted Data (batch input size of 576 and $L = 20$; standard deviation is less than 2.9%). HL, NC, #C and NR represent Hidden Layer(s), Network Creation, Number of Communications and the Noise Reduction time.	69

6.9	Classification over Encrypted Data (batch size 576 and $L = 20$; standard deviation is less than 3%.) #HL, NR and #C represent for number of hidden layers, Noise Reduction times and number of communications.	71
6.10	Performance of the trained CNN using Different Approximation Methods	75
6.11	Performance of the model for different approaches based on the model from [22]	76
6.12	Performance of a CNN with one Conv layer for different approaches.	76
6.13	Comparing Performance of the Polynomial Approximation Method Papers.	83
6.14	Performance of the trained CNN with Different Degree Polynomials.	83
6.15	Performance of the trained CNN using different Activation Functions and their Replacement Polynomials	86
6.16	Comparing the number of communications in our approach and SMC-based approach for different number of hidden layers and different batch sizes for the input (HL represents for Hidden Layers).	88
6.17	Training Neural Network over the Encrypted MNIST Data (batch input size of 192 and $L = 20$).	91
6.18	Training different Neural Network architectures over the Encrypted Data ($L = \{5, 10, 20\}$).	91
6.19	Breakdown of Running Time of CNN Model 6.5) over Encrypted MNIST Dataset.	95
6.20	The performance of our solution(s) compare to [22]	95
6.21	Proposed solutions vs. Privacy concerns.	96
6.22	Comparison with the state-of-the-art Solutions	96

CHAPTER 1

INTRODUCTION

Machine learning algorithms based on Deep Neural Networks (DNNs) have attracted attention as a breakthrough in the advance of Artificial Intelligence (AI) and are the mainstream in current AI research. 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 [47, 51, 39]. However, training the models and classifying new instances requires access to the raw data which is often privacy sensitive and can create potential privacy risks.

Furthermore, with increasing growth of cloud services, machine learning algorithms can be run on the 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 include Microsoft Azure Machine Learning [63], Google Prediction API [29], GraphLab [87], and Ersatz Labs [48]. Machine learning algorithms typically consist of two phases as shown in Figure 1.1:

- (1) The training phase during which the algorithm \mathcal{A} learns a model w from a data set \mathcal{D} of labeled examples, $\mathcal{A}(\mathcal{D}) = \mathcal{C}$.
- (2) The classification phase that runs a classifier \mathcal{C} over a previously unseen feature vector i , using the model w to output prediction p , $\mathcal{C}(w, i) = p$.

Either or both of these steps could be outsourced to the cloud. In applications that handle sensitive data, it is important that the training data \mathcal{D} , the algorithm \mathcal{A} , the model w , and the feature vector x remain secret to one or some of the parties involved. The privacy of different components of a machine learning algorithm should be considered such as feature values, instance labels, final model, prediction results etc. *Feature values* contain sensitive information about the client, for example, Health information of a patient.

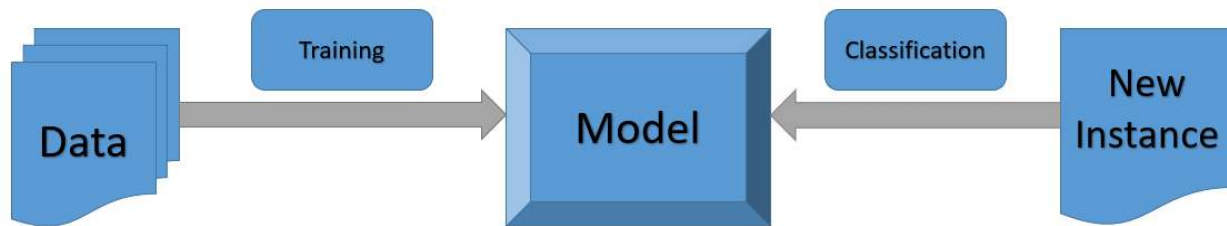


FIGURE 1.1. General model of a Machine Learning algorithm

The *machine learning algorithm processing the data* is one of assets and the owner of the model does not reveal information about it. The algorithm includes the method for building the model, the optimization method, and the architecture of the algorithm. *Final model* is also another component that counts as a sensitive asset from the server’s perspective. While the server provides the service for the client, it also wants to preserve the privacy of the final model’s parameters against clients. The *classification result* is another sensitive component from the client’s prospective. We explore the privacy of these components in details, Chapter 4.

Besides the privacy concerns from client’s prospective, there are also regulations which needed to follow for preserving privacy of data.

- Health Insurance Portability and Accountability Act (HIPPA) is one of the standards for preserving the privacy of health data of patients. Companies that provide medical services need to follow this standard, while processing health information [67].
- General Data Protection Regulation (GDPR) was introduced by EU for general data purposes [18].

Different techniques have been introduced in the literature to address this problem. To the best of our knowledge, proposed solutions are based on three different basic ideas: Differential Privacy (DP), Homomorphic Encryption (HE), and Secure Multiparty Computation (SMC).

This dissertation is presented towards the fulfillment of the University of North Texas, Computer Science and Engineering Department PhD program requirements. In this

dissertation, the goal is developing new techniques for preserving the privacy of data in deep neural network algorithms based on homomorphic encryption.

The aim of this research is providing a client-server solution to apply deep neural network algorithms on the encrypted data and allow the server or the client to provide/receive the service without having to reveal their sensitive data to each other. The main components of the proposed solution are **homomorphic encryption** and **neural networks**.

For designing the protocol, we aim to develop the theoretical foundation for adopting deep neural network within the limitations of current homomorphic encryption schemes and implementing them in the encrypted domain.

We consider a client-server model where the client owns the data and the server uses this data to build a model and provide prediction services to the client. In the *training phase*, the server gets the training dataset from the client in an encrypted format and can train a model on that data without learning anything about the training dataset. Since training is performed on the encrypted data, the server does not learn the parameters of the trained model or the final model.

The trained model is encrypted under the client's public key, and only this client can use the final model for classification of new instances, see Figure 1.2. Server cannot access the

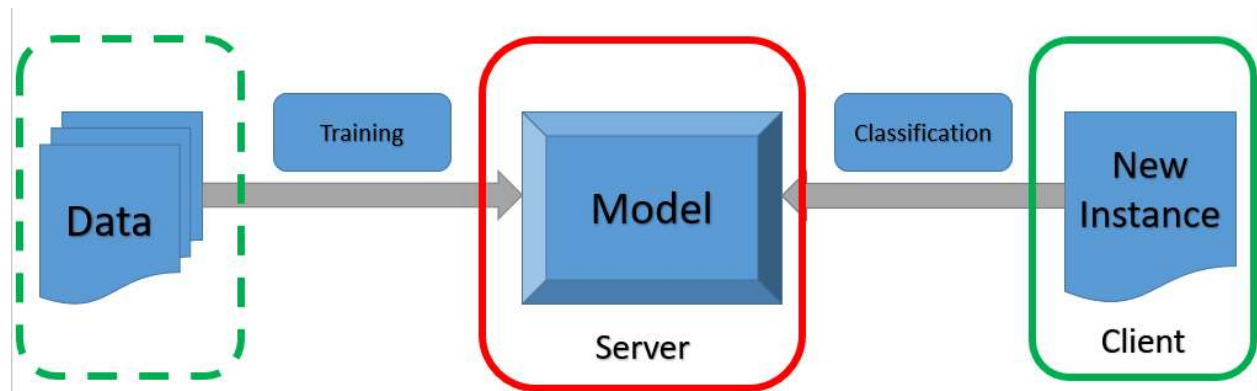


FIGURE 1.2. Training phase in a Machine Learning algorithm

input data or the prediction, and also the client cannot get information about the structure of the model and its parameters, see Figure 1.3.

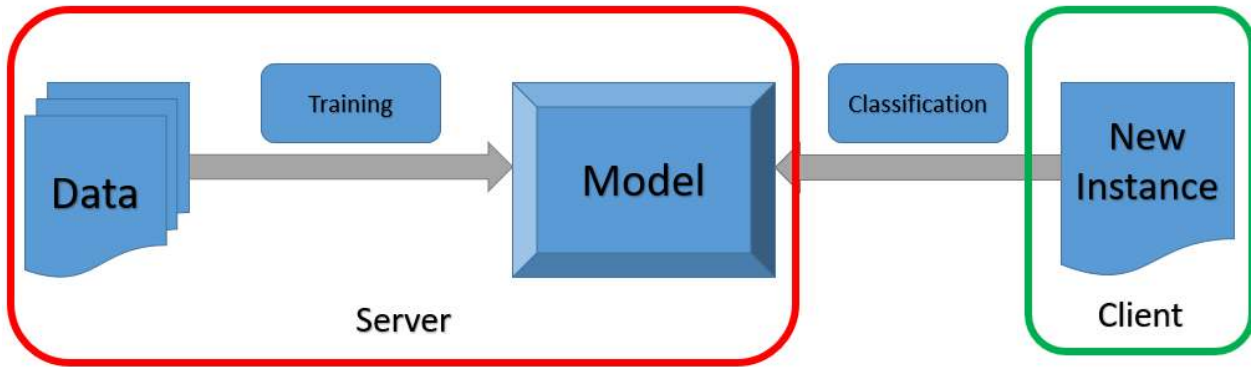


FIGURE 1.3. Classification phase in a Machine Learning algorithm

The original fully homomorphic encryption (FHE) scheme presented in Gentry’s seminal paper [26] was highly inefficient; since then, significantly more practical schemes have been developed. In order to have efficient and practical solutions for computations in the encrypted domain, we typically use somewhat homomorphic schemes or leveled homomorphic schemes instead of fully homomorphic encryption.

The most notable shortcoming of practical homomorphic encryption schemes is that the operations in practical schemes are limited to addition and multiplication. Another shortcoming is the limited number of operations on the encrypted data. We can only operate an algorithm to a specific depth on the encrypted data. Consequently, we need to adopt neural network algorithms within these limitations.

The computation performed over sensitive data by neural network algorithms is very complex and neural networks cannot simply be translated to encrypted versions without modification. For example, activation functions in neural networks are a sigmoid function ($\frac{1}{1+e^{-x}}$) and Rectified Linear Unit (ReLU, $f(x) = \max(x, 0)$) and they should be replaced by a function that only uses addition and multiplication such as polynomials.

However, a solution that builds upon homomorphic encryption schemes should be restricted to compute low degree polynomials in order to be practical [92]. Hence, presenting the desired computation as low-degree polynomials is an important task in making practical use of homomorphic encryption schemes for applying neural network algorithms to the

encrypted data. Polynomials of degree 2 are used to substitute the sigmoid function in neural networks [22] and polynomials of degree 3 are used to estimate the natural logarithm function [83].

Once the polynomial approximations are generated, we design and evaluate privacy-preserving training and classification of neural networks by replacing the activation functions with these polynomial approximations. The goal is to adopt neural networks within the practical limitations of homomorphic encryption, while keeping accuracy as close as possible to the original model.

1.1. Research Questions

The research questions that the dissertation is exploring are as follows:

- (1) How should we solve the issue about the plain-text domain (integers) in HE schemes?
How much does using integer values impact on the performance of the model?
- (2) How should we generate low degree polynomials with high performance for a general case with a deterministic algorithm?
- (3) How should we train a model based on a polynomial with integer coefficients?
- (4) What is the performance of the proposed solution for benchmark datasets such as the MNIST or CIFAR-10 for training and classification?
- (5) How can we generalize our solution for training and classifying data that is distributed among clients?

1.2. Significance of the Research

In this dissertation, the aim is to study and address challenges in HE based solutions. The first step is comparing two main cryptographic tools for preserving the privacy of data: Homomorphic Encryption (HE) and Secure Multiparty Computation (SMC).

The basic idea in SMC is based on distributing the computation among entities by using cryptographic techniques. In other words, SMC is a secure function evaluation over data for preserving the privacy of data. Oblivious Transfer (OT) and Garbled Circuits (GC) are two types techniques used in SMC protocols.

Our research approach in has several advantages over SMC-based approaches. Unlike SMC-based approaches, the structure of the client does not need to be changed in our approach when the architecture of the neural network is changed. The only operations on the client side are encryption and decryption and the server can perform different machine learning algorithms with the same client. Another advantage of our approach is that it preserves the privacy of the model’s architecture compared with the SMC-based solutions. In SMC-based solutions, the client participates in the computations and information about the model could possibly leak to the client.

For example, the client can learn information such as the number of layers in the neural network, the structure of each layer, and activation functions. Although there are some solutions such as adding more dummy layers to the model to mitigate this issue, these will add to the already large computation cost. In our approach, the only operations on the client side are encryption and decryption. Hence, the client cannot learn any information about the structure of the neural network.

In SMC protocols, interactions are needed between the client and the server for each operation, whereas in our approach the server does not need to communicate with the client unless the amount of noise reaches a threshold. In the training case, if the same neural network is implemented using SMC protocols, the number of communications is much higher compared with the HE based protocols. Also, as mentioned above, 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 Convolutional Neural Network (CNN), the structure of each layer and the activation functions.

Another significant point of our research is the polynomial approximation method. A solution that builds upon homomorphic encryption schemes should be restricted to compute low degree polynomials in order to be practical [92]. Hence, presenting the desired computation as low-degree polynomials is an important task in making practical use of homomorphic encryption schemes for applying neural network algorithms to the encrypted data. Polynomials of degree 2 are used to substitute the sigmoid function in neural networks [22] and polynomials

of degree 3 are used to estimate the natural logarithm function [83]. These are ad-hoc solutions which enable us to work around certain problems, but there is no generic solution to the problem of approximating a function with low degree polynomials. Our goal is to provide a framework capable of handling general cases. In [65], Mohassel and Zhang used Taylor Series for approximating the sigmoid function, as they mentioned, for achieving a high accuracy, the degree of polynomial should be 13 which is high, so that it imposes a significant communication overhead. Therefore, the authors used a linear piece-wise function which simulates the structure of the sigmoid function. However, we cannot use this method for encrypted data, due to the lack of comparison operation over encrypted data.

It is worth noting that polynomials with low degrees are applied not only in HE-based solutions, but also interested in solutions based on Secure Multiparty Computation. However, the use of the ReLU (max function) or a sigmoid (exponential function) functions impose a certain computation and communication cost for SMC based solutions, see [65, 58, 77].

In this chapter, we provided an introduction to privacy-preserving machine learning as a service. In the following chapter 2, we provide concepts, terms, theories, and ideas that will be useful for understanding the technical parts of this work. In particular, Homomorphic Encryption and Deep Neural Networks are the main topics which we focus in the next chapter. We provide the basic definitions of homomorphic encryption, main functions on an HE scheme, and also explain the difference between this kind of encryption scheme and other encryption schemes. Then, we provide more details in Deep Neural Networks and their main components such as fully connected or convolutional layers.

In Chapter 3, we perform a literature review of the proposed solutions to this problem. Two main techniques considered in chapter 3: Secure Multiparty Computation and Homomorphic Encryption. We also have one more section, 3.3, that includes other techniques including differential privacy.

Chapter 4 focuses on the main challenges in privacy-preserving computation on data by using homomorphic encryption. Besides the advantages of HE schemes, such as running operations over encrypted data, there are some shortcomings of using this kind of encryption

schemes, for instance lack of division operation. These shortcomings force us to revise the structure of the network architectures to adopt them within the limitations of HE schemes.

In Chapter 5, we propose our solution for addressing challenges from Chapter 4. We consider the limitations of HE schemes such as limited number of multiplications or lack of division and provide solutions for replacing or changing each module. We convert a Deep Neural Network to its adopted version while keeping the performance of the algorithm close to that of original one.

In Chapter 6, we provide the performance of our solution for the training and the classification of Deep Neural Network (DNN) over encrypted data and compare our results with similar solutions. We consider a wide variety of architectures and implement some of them over encrypted data and measure the performance by calculating the accuracy of the model and comparing it with the original architecture. The final chapter 7 discusses possible future research directions..

CHAPTER 2

BACKGROUND

In this chapter, we briefly describe HE schemes, their strengths and weaknesses which should be considered while using them for secure computation protocols.¹ We also give a brief description of neural networks and convolutional neural networks.

2.1. Homomorphic Encryption Schemes

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 $\{c_i\}$ for $i = \{1, \dots, n\}$. 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 (see [26])

$$(2.1) \quad Dec(sk, Eval(pk, C, c_1, \dots, c_n)) = C(m_1, \dots, m_n)$$

Homomorphic encryption (HE) was first introduced in 1978 by Rivest et al. [78]. Other researchers introduced several other HE schemes [24, 28, 71]. However, most of these encryption schemes have some constraints. Some of them, such as the Paillier encryption scheme [71], only support one operation (addition). If the encryption scheme only supports one operation, it is called *Partially Homomorphic Encryption* (PHE). In some work, researchers used the term *Somewhat Homomorphic Encryption* instead of Partially Homomorphic Encryption.

This question was answered in 2009 when first Fully Homomorphic Encryption scheme was designed by Gentry [26]. A Fully Homomorphic Encryption (FHE) scheme is a homomorphic encryption scheme that supports circuits with arbitrary depth. The idea behind

¹This section partially reproduced from Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi, *CryptoDL: Deep neural networks over encrypted data*, CoRR abs/1711.05189 (2017).

the encryption function in Enc is to add a small value, called *noise*, to the plaintext 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 increased. The Dec function works correctly if this amount is less than a threshold. If we add or multiply two ciphertext c_1 and c_2 , the level of the noise increases in the result of the computation. 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 the ciphertext and encrypt it again. For decryption, it needs the secret key, sk . For years, the community has been trying to find out if there is a way to decrease the noise without having the secret key. In [26] a technique introduced for handling an arbitrary depth of computations, called *bootstrapping*. In the bootstrapping technique, the amount of noise is decreased without needing to access sk . However, it has a huge computational cost and is a very slow process. This limitation makes FHE impractical for actual use.

Recent advances in homomorphic encryption have led to a faster HE scheme: Leveled Homomorphic Encryption (LHE), see [26]. LHE schemes do not support the bootstrapping function, so they only allow circuits with depths less than a specific threshold. If the number of operations is known before starting the computations, we can use LHE instead of FHE. The performance of LHE schemes is further improved using a Single-Instruction-Multiple-Data (SIMD) technique [31]. Halevi et al. in [31] use this technique to create a batch of ciphertexts. So, one single ciphertext has been replaced with an array of ciphertexts in computations. By using this functionality, we can improve the performance of the service by feeding more than one instance in each round of running the algorithm.

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; if this conversion is not done properly, it could lead to accuracy loss. The second limitation is ciphertext size. The size of the message increases considerably after encryption. Another important limitation is related to noise. After each operation, the amount of noise in the ciphertext increases. Multiplication increases

noise much more than addition. For HE schemes to work properly, the amount of noise should remain less than a predefined threshold. The last and most important limitation is lack of support for the division operation. See Chapter 4 for more details.

In summary, only a limited number of additions and multiplications could efficiently be performed over the encrypted data and complex functions such as sigmoid functions used in neural networks are not compatible with the current HE schemes.

2.2. Neural Networks

At a high level of abstraction, a neural network is a combination of neurons arranged in ordered layers. Each neuron gets an input, runs 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. We have three different types of layers and neurons in each layer have different structures, *input layer*, *hidden layer* and *output layer*.

The first type is the input layer and only one layer of this type exists in each neural network. Instances are fed for this layer, neurons in this layer have a simple structure. Neurons in the input layer only get the input and their outputs are the input values without any change. The second type is the hidden layer. Each neuron in the hidden layer has a vector of input (x_0, \dots, x_n) (the size of this vector is the number of neurons in the previous layer plus one for the bias neuron in the input layer), a vector of weights (w_0, \dots, w_n) (the size of this vector is equal to the size of the input vector), and an output y . The output is computed with this formula

$$(2.2) \quad y = f\left(\sum_{i=0}^n x_i * w_i\right).$$

Here, f is a function which is called an activation (or transfer) function. Several functions could be used as an activation function. Step, hyperbolic, or sigmoid functions are examples of activation functions. We use the sigmoid function as the activation function:

$$(2.3) \quad \sigma(x) = \frac{1}{1 + e^{-x}}$$

Another example of an activation function is Tanh $(2\sigma(2x) - 1$ or $\frac{e^x - e^{-x}}{e^x + e^{-x}})$.

The last layer is the output layer. Neurons in this layer are sometimes simple neurons (such as those in the input layer) and in some cases, they have complex structure (as those in the hidden layers). All layers have a bias neuron which is connected to all neurons in the next layer.

We consider fully connected feed-forward neural network and use such kind of structure for implementation. In this type of neural networks, all neurons in each layer are connected to all neurons in the next layer.

A dataset has a structure like a matrix, the number of rows in this matrix is the number of instances and the number of columns is equal to the number of features. We consider labeled data, the dataset also has another vector which holds labels. If we want to train a model based on an $m \times n$ matrix with c labels, our neural network has n neurons in the input layer and c neurons in the output layer. The number of neurons in the hidden layer depends on the problem and the dataset.

In the training phase, we have two main functions. The first function is *feed-forward* and the second is *back-propagation*. We initialize the algorithm by assigning random values to weights in our neural network. During the training process, we update these weights by running the back-propagation function.

The feed-forward function has the following steps. As mentioned above, an instance is a vector that consists of values $x_i, 1 \leq i \leq n$. Each neuron in the input layer receives one x_i as an input and results y_i as an output, and the bias neuron always gets a constant as an input. The second step is to prepare an input for the hidden layer. Each neuron in the input layer is connected with w_j 's weights, $1 \leq j \leq h + 1$ (h is the number of neurons in the hidden layer plus one for the bias neuron of input layer). Each neuron in the hidden layer gets a vector of x_i 's and it also has a vector of weights w_i 's. The output of each neuron in the hidden layer is

$$(2.4) \quad y = f\left(\sum_{i=1}^n x_i w_i\right)$$

where f is the activation function (for example, sigmoid function). All outputs of y 's are

inputs for the next layer.

Each neuron in the output layer gets a vector of y_i s and computes its outputs with this formula:

$$(2.5) \quad o = g\left(\sum_{i=1}^h y_i w'_i\right)$$

where w'_i s are the weights between all neurons in the last hidden layer and one neuron in the output layer.

The value of o is the actual value of the input instance. However, the value o is not always the same as the label related to this instance. The difference between the value of o and the label a is the error (different ways proposed for calculating the error); the neural network tries to minimize this error by updating the weights based on this error. In function back-propagation, we update all the weights based on the difference between the actual value and the target value. A common technique for updating weights is called *Gradient Descent* which we explain in detail.

For updating weights for each neuron, each neuron has another two values, *gradient* and Δ *weight*. The gradient for neurons in different layers can be computed in different ways. As mentioned above, for neurons in the input layer the output is the same as the input and neurons in the input layer have no gradients. For computing gradients, we start backward, we compute gradients for the neurons in the output layer. The gradient for one neuron g'_i in the output layer is computed with this formula:

$$(2.6) \quad g'_i = (o - a) \frac{df(o - a)}{dx}$$

where f is the activation function. The gradient of a neuron in the hidden layer has a different formula:

$$(2.7) \quad g_i = \left(\sum x_i w_i\right) \times \frac{df(\sum x_i w_i)}{dx}$$

After feeding one instance to the neural net, we update gradient values for all neurons. For updating g_i 's, we need to compute Δw for each neuron and then use this value to update the

g_i . When we want to update Δw_i for one neuron, we use g_i 's of neurons from the next layer. The formula for Δw_i is:

$$(2.8) \quad \Delta_{new} w_i = \eta \times o_i \times g_j + \alpha \times \Delta_{old} w_i$$

$$(2.9) \quad \Delta w_i = \Delta_{new} w_i$$

$$(2.10) \quad w_i = w_i + \Delta w_i$$

where g_j is the gradient of neurons in the next layer, o_i is the output value of the current neuron, and $\Delta_{old} w_i$ is the last value of Δw_i and $\Delta_{new} w_i$ is the new value. The integer j changes through all neurons in the next layer, and w_i is the weight of the current neuron. The values η and α are some technical parameters, we will omit their explanation for the sake of simplicity of this presentation.

Running one feed-forward and one back-propagation is called one *iteration*. In practice, we partition the dataset into smaller parts, each called are *batches*. In each iteration, we train the model based on one batch and update weights. Going through all instances in a dataset and running one iteration for each batch is called one *epoch*. We run more than one epoch for training the model. Two common methods are used for choosing the number of epochs and stop training phase: a threshold for error or a constant number of epochs. In the first case, we stop when the error is below a threshold or the performance of the model is not improved. In the second case, we run the training phase up to a constant number of epochs and then stop the training calculations and save the model for predictions.

There are two methods for feeding instances to the neural networks: online learning and batch learning. In the online training, the weights are updated after feeding one instance of the network whereas in batch training a batch of instances is fed to the network to update the weights in each step.

In the following section, we explain Convolutional Neural Networks, a generalized form of neural networks.

2.3. Convolutional Neural Networks (CNNs / ConvNets)

Convolutional Neural Networks (CNNs / ConvNets) are a specific type of feed-forward neural network. CNNs have proven to be very effective in areas such as image recognition and classification.²

CNNs commonly use several distinct kinds of layers as shown in Figure 2.1³⁴

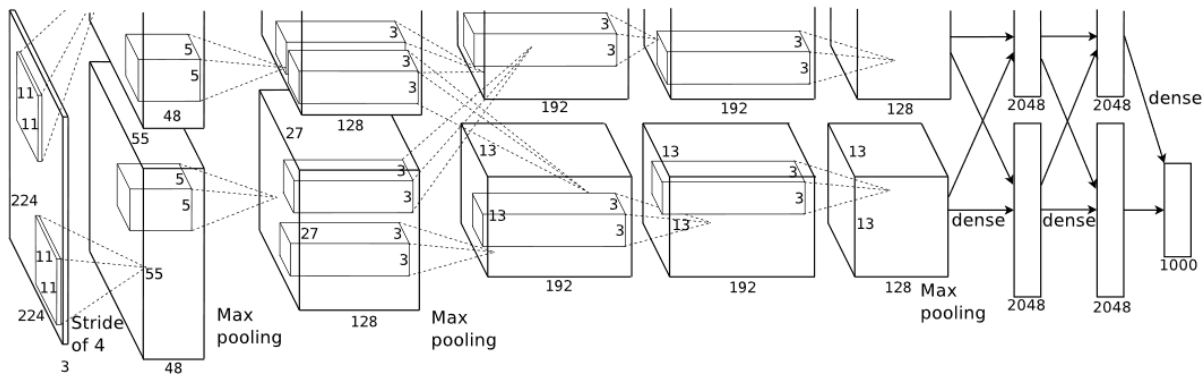


FIGURE 2.1. Convolutional Layer

The first layer in a CNN is 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. See Figure 2.2.

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

²Parts of this chapter have been previously published, either in part or in full, from Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi, *CryptoDL: Deep neural networks over encrypted data*, CoRR abs/1711.05189 (2017).

³Reproduced from <http://cs231n.github.io/convolutional-networks>

⁴Figure 2.2 reproduced from <https://developer.apple.com> and Figures 2.3, 2.4, 2.5 and 2.6 reproduced from <http://cs231n.github.io/convolutional-networks/>.

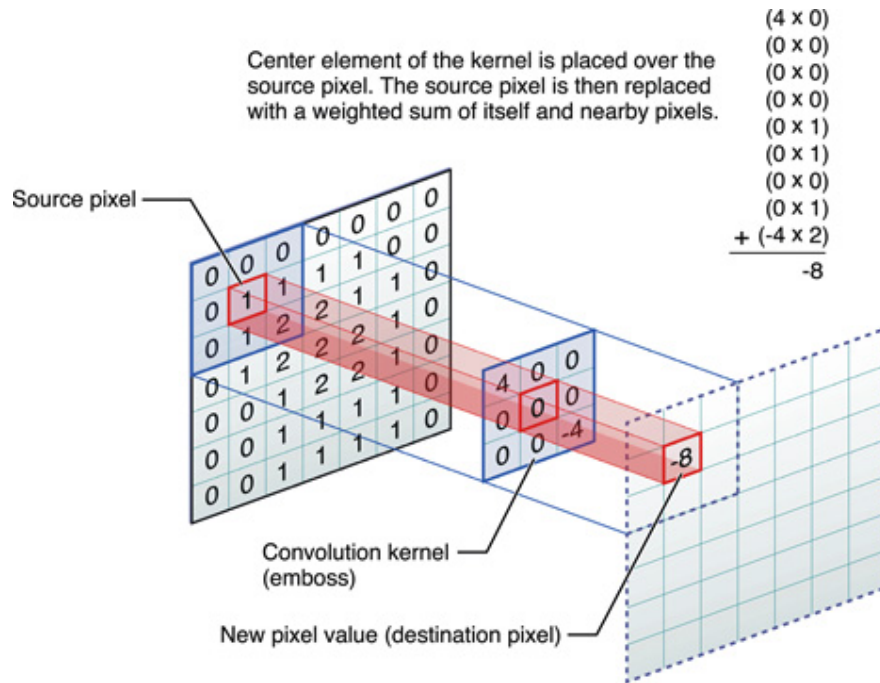


FIGURE 2.2. Convolutional Layer

mathematical operation on it. There are several activation functions we may encounter in practice, including Rectified Linear Unit (ReLU), sigmoid, and tanh functions. We cannot calculate these functions over encrypted values and we should find replacements for these functions that only include addition and multiplication operations. See Figure 2.3.

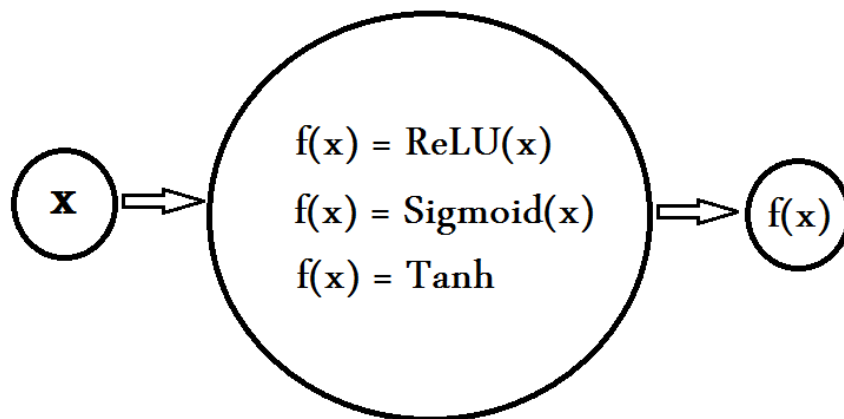


FIGURE 2.3. Activation Layer

After an activation layer, there is a pooling layer (or sub-sampling). This layer is for

sub-sampling of the data, it reduces the size of the data. Different kinds of pooling layer are introduced in the 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. In the case of integer domain for the plaintexts, we use a scaled-up version of the average pooling (proposed in [22]), and 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 an impact on the depth of the algorithm because the addition is cheap and almost noise-free. See Figure 2.4.

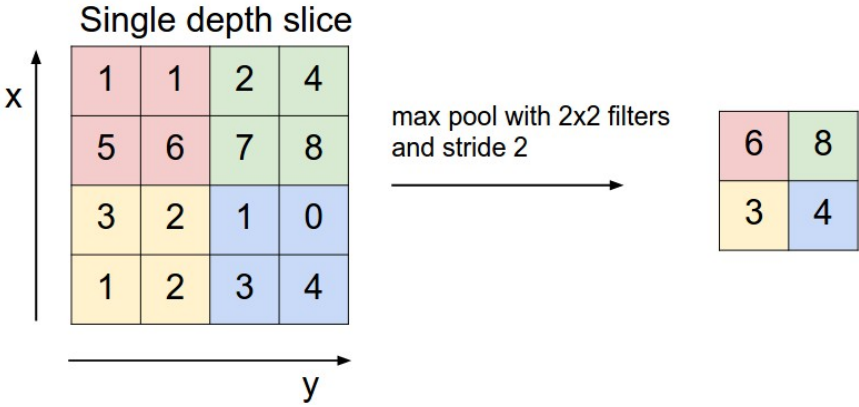


FIGURE 2.4. Max Pooling Layer

The last layer in a CNN is the fully connected layer. The fully connected layer has the same structure as 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 earlier layer, each connection is represented by a value which is called weight. The output of each neuron is the dot product of two vectors: output of neurons in the earlier layers and the related weight for each neuron. See Figure 2.5.

When we train a model over a training set, it is possible that the final model will be biased to the training set, and we will get high error over the test set. This issue is called over-fitting. For avoiding over-fitting during the training process, we use this specific type of layer 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 preventing over-fitting happen. We need this

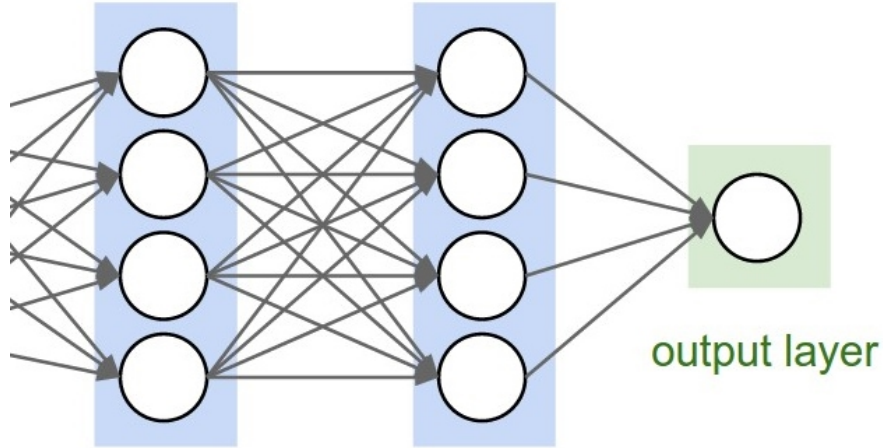


FIGURE 2.5. Fully Connected Layer

layer only for the training step and we can remove it in the classification step. See 2.6.

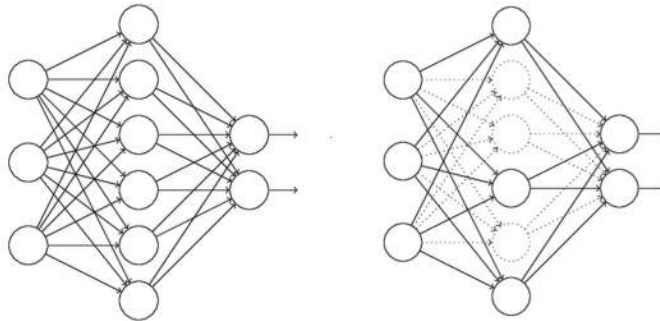


FIGURE 2.6. Dropout Layer

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 the 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, $[[Convolutional \rightarrow Activation]^n \rightarrow AveragePooling]^n$. As in the case of the first block, we can use the second block more than one time. Then, 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.

Figure 2.7 is an example of a CNN.

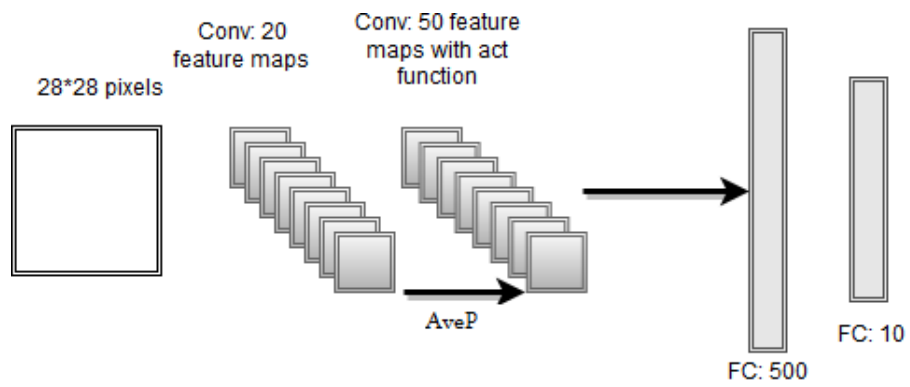


FIGURE 2.7. CNN Model 1 (AveP stands for Average Pooling).

CHAPTER 3

RELATED WORK

In this chapter, we review the literature on the privacy preserving machine learning as a service. Although a variety of techniques was used for addressing this problem, we focus on two main ones; Secure Multiparty Computation and Homomorphic Encryption and briefly review other techniques such as Differential Privacy.

3.1. Homomorphic Encryption

Graepel et al. [30] used a Somewhat HE scheme to train two machine learning classifiers: Linear Mean and Fisher’s Linear Discriminate (FLD). They proposed division-free algorithms to adapt within the 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. [11] used 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 Hyper-plane Decision, Naive Bayes, and Decision trees. Their approach is based on SMC, considers only classical machine learning algorithms and is only efficient for small data sets.

Xie et al. [92] discuss theoretical aspects of using polynomial approximation for implementing neural network in the encrypted domain. Building on this work, Dowlin et al. [22] present CryptoNets, a neural network classifier on the encrypted data. They use a leveled homomorphic encryption scheme which supports SIMD; Simple Encrypted Arithmetic Library (SEAL) ¹. The authors replace *max pooling* with scaled mean-pooling for solving the issue of having no division operation available for encrypted values. Since HE schemes do not support exp function, they replace the sigmoid function with $f(z) := z^2$ as the activation function. They build the model using plain data and then use this model for classifying

¹Available at <http://sealcrypto.codeplex.com>

encrypted data and can achieve accuracy of about 98.95% on MNIST dataset. The protocol reached a throughput of 48068 instances per hour.

Chabanne et al. improved accuracy of CryptoNets by combining the original ideas of CryptoNets' solution with the batch normalization principle [15]. They used ReLU as activation function and applied combination of Taylor series and batch normalization to approximate this function.

Kim et al. [44] proposed a solution for training over the encrypted data by using logistic regression and homomorphic encryption. They implement the least squares approximation of logistic regression by using a novel homomorphic encryption which is optimized for real numbers. The encryption scheme supports *approximate arithmetic* of ciphertexts which is more efficient, because the size of parameters do not grow too large during computation. Two main methods are used for training in logistic regression: Newton Raphson and the gradient descent. The first method includes matrix inversion which is complex over encrypted data. The authors use the second method for updating weights while training. The next part of the protocol is replacing the sigmoid function with an adopted one, the authors proposed their own method for approximating the sigmoid function instead of using Taylor series. As they mention, the Taylor series is not a good choice, because a high accuracy regression model needs a high degree polynomial which is not efficient over encrypted data.

Kim et al. [42] also proposed a solution for secure logistic regression based on HE schemes; this work is similar to [44] and optimizes the performance and running time of the learning algorithm. The proposed solution is the winner of iDASH 2017, task three . The proposed solution in [44] is dependent on the size of dataset (number of features) and for large scale data, it is not practical. The solution proposed in [42] has better performance and lower storage costs as compared to [44]. For decreasing the number of iteration, the authors adopted Nesterov's accelerated gradient within limitation of HE to speed up the convergence speed. The computation overhead of using this method, which is similar to a momentum method, is sending an extra ciphertext from the client to the server. The polynomial approximation method for approximating the sigmoid function over interval $[-8, 8]$ is similar to that of

[44]. This polynomial is chosen based on the input values to the activation function which is a weakness of the polynomial approximation method. Three different degrees used for approximation: three, five and seven.

3.2. Secure Multiparty Computation

Based on the number of entities participate in running the protocol, the efficiency of the protocol is different. If two entities run the protocol, we call it two-party computation. Yao’s garbled circuits [94] and Goldreich-Micali-Wigderson [27] (GMW) protocol are two common methods for a 2-party computation. Another technique in SMC is Oblivious Transfer, which is a two-party protocol [66]. The sender has a set, $M = (m_1, m_2, \dots, m_n)$, the receiver aims to choose one (or more) elements of these message, without revealing which elements of M were chosen by receiver. In 1-out-of- n protocol, the receiver only choose one of n messages.

Yao’s Garbled Circuits [94] is a cryptographic protocol for secure evaluation of function f over two inputs, x_1, x_2 . Running this protocol has five main steps: Garbling the algorithm, data transfer, oblivious transfer, evaluating and merging final results. Clearly, interaction between two parties is an essential part of running this protocol. For reducing this communication overhead, some optimizations were introduced in literature. For a brief description of this protocol and related optimization methods, see section 2.3 from [77]. For a series of work for computing deep neural network over encrypted data, see [65, 58, 79, 77, 38].

Mohassel and Zhang [65] proposed a solution for privacy preserving machine learning based on SMC and SMC/HE techniques. The idea is to run a protocol on the clients’ data (which could vertically or horizontally distributed among them without any limitations) for training and classification steps. Client(s) secret-share their data with two non-trusted non-colluded servers to run a 2PC protocol. These servers could be seen as a *Evaluator* and *Cryptography Service Provider* (CSP) in real scenarios. Three different algorithms are considered in this paper: linear regression, logistic regression and neural network. The protocol is partitioned into two phases: online (data independent) and offline (data-dependent). The training step in the online phase and calculating multiplication triplets run in the offline phase. The security of the protocol is shown in the semi-honest model (where the parties are

expected to follow the protocol), however, for improving the efficiency, the authors reduce the security of model to an *admissible adversary*. In this weaker security model, we assume that servers do not collude with clients. In the new structure, because of the large number of cryptographic operations in offline phase, clients compute the multiplication triplets to reduce the running time. Implementing the sigmoid function by using 2PC has some challenges, the authors choose different approaches, polynomial approximation (with numerical methods) and a secure computation friendly form of the activation function. Low degree polynomials (of degree 2 or 3) has low accuracy and low computation cost, and high degree polynomials (of degree ten) has high accuracy and high computation cost due to the number of multiplication. The second approach consists of a piece-wise linear activation function which is has high accuracy and low computation cost. However, for back-propagation step, instead of using the derivative of the new activation function, the authors suggest using the derivative of ReLU.

Rouhani et al. [79] proposed a solution for privacy preserving in the classification phase based on Yao’s Garbled Circuit. Based on their protocol, the privacy of both model and input data is preserved while classifying a new instance. The proposed approach is suitable for low rate classification requests from clients, when one client needs to dynamically analyze the data. These are the main steps of the proposed protocol:

- Pre-processing of the model: The server (model’s owner) processes the model and sends a netlist of the publicly known network architecture.
- The client generates the garbled tables of the network architecture.
- The server evaluates the client’s instance by executing garbled circuit and sends the encrypted result to the client.
- The client decrypts the result and inference the prediction.

Different activation functions and their error analyzed in the proposed solution, like sigmoid, tanh and ReLU functions. Different approximation methods analyzed for these activation functions by calculating the error and the computation overhead for each method.

Liu et al. [58] proposed a method for transferring a neural network to an *oblivious neural network*, the new structure preserves the privacy of the client’s data and the server’s

model. They consider a client-server model; the client owns the data and the server owns the model. MiniONN transforms any "common" neural network model into an oblivious neural network, this transformation does not affect the training phase and is applicable only for classification step. Their adversary model is based on the semi-honest model. The threat model is based on the assumption that the server S tries to learn information about the client's instance, X , and client C tries to learn about W and B values. The size of X , W , B and activation function is not protected here and both client and server have access to them.

Riazi et al. addressed the privacy-preserving classification in machine learning [77]. They design a framework based on the SMC, the proposed framework built upon ABY [20] and improved its performance by adding more features like integrating sequential GCs or fixed-point arithmetic. As a part of improvement, the offline computation is a three-party computation by using a Semi-honest Third Party (STP). STP is used for generating *correlated randomness* for two parties, the online phase only includes two parties and does not include STP. As the authors proposed, each layer implemented with a different method, therefore, they need to convert from one method like GC to GMW. The results show the running time and communication cost is more efficient compare to a design which is only based on one these techniques.

Juvekar et al. [38] addressed the secure inference for the neural network with a hybrid solution, a mixed of HE and SMC. They design GAZELLE, a framework that equipped with fast homomorphic encryption operations which supports SIMD. Their framework enhance the performance by switching from HE to GC or vice versa to reach low latency for preserving the privacy of data while classifying client's instance. GAZELLE supports *packed additively homomorphic encryption* (PAHE), using this HE scheme, speeds up the matrix-vector multiplication. Their solution is based on the 2-party computation without any need to a third party. They explore a trade-off between computation and communication cost by breaking an ML algorithm into sub-modules, and implementing each module by the technique with highest performance. The GAZELLE has three main components:

- *Gazelle Homomorphic Layer*. This component includes three basic operations,

namely: SIMD addition, SIMD scalar multiplication and automorphisms.

- *Gazelle Linear Algebra kernel*. This component includes homomorphic matrix-vector multiplications and homomorphic convolutions.
- *Gazelle Network Inference*. This component includes combination of GCs with linear algebra kernel for implementing preservation of the data privacy during classification in the neural network. Preprocessing of the network, creating the GC circuits, and implementing each layer with the efficient module are two main steps of this component.

As the authors mentioned, using HE schemes for implementing convolutional and fully-connected layers is more efficient than using GC because of the number of multiplications, this approach speeds up the performance of the protocol compare to similar solutions.

The privacy of weights values in fully-connected and convolutional layers, the size of filters and strides in convolutional layers are preserved while running the protocol. However, the number of layers and the size of layers are revealed to the other party. The size of the image is also revealed to the server during the protocol run.

Chandran et al. [16] designed a framework for transferring a code to an optimized mixed protocol of arithmetic or Boolean circuits. The new structure preserves the privacy of data during classification phase. By using this framework, programmers do not need to involve in details of 2PC computations and design, the developed code fed to EzPC and 2PC will use for privacy preserving classification. The framework built on top of the ABY, a popular framework for implementing SMC scenarios. EzPC considers the cost of cryptographic operations and generates efficient and scalable 2PC protocols from the target program. The program needs to be written in EzPC syntax which is similar to C++ or JAVA syntax. The output of the EzPC is based on C++ language.

3.3. Other Related Work

Aslett et al. [4] proposed methods for implementing statistical machine learning over encrypted data and implement extremely random forests and Naive Bayes classifiers over 20 datasets. 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 [60, 97, 43].

Shortell and Shokoufandeh [83] focused on secure signal processing. For computing the natural logarithm, they used the Taylor expansion of $\ln(x)$ to estimate this function by a polynomial with degree 5. Their results showed that for natural logarithms, estimation is accurate enough if proper values are chosen for parameters.

Shokri and Shmatikov [82] considered a solution for secure training based on the differential privacy. For a distributed dataset among clients, each client trains a model based on its data and only shares a part of model's parameters with a server. Each client before running an epoch, download a portion of weights, run one round of training and then shares only a portion of updates values to the server. At the end of protocol, each client has a model that trained based on not only its own data, but also trained on other clients. The performance of final model is always higher compare to the model that only trained based on each individual client's data (standalone case). By using this method, each client can train a model by considering its own learning objective and other client's data without revealing any information about its own data.

Phong et al. [73] proposed a solution for secure training from distributed data among clients based on a HE schemes. The basic idea is similar to [82], each client trains its own model individually, then uploads a part of gradient descent to a server. The solution proposed by Shokri et al. [82] is not secure against honest-but-curious server, they used a differential privacy technique against this kind of server, however, the performance of the model dropped. The main difference between these two solutions is the trade-off criteria, *accuracy/privacy* and *efficiency/privacy* for [82] and [73] respectively.

The solution proposed in [82] is not completely proved about data leakage to the server. Phong et al. show that in case of sharing even a small portion of gradient descent (3.89%), one can recover image from the client side [73].

Abadi et al. [1] proposed a solution for secure training with deep learning for non-convex objectives. In the proposed solution, the client stores the model on its own device and use it for future classification, storing model on the client has advantages such as power efficient, low-latency inference and preserves the privacy of client’s data, because the data is not transferred to a third party. Because of sharing the model with the client, the concern is about information leakage about the training data from analyzing the model. The authors control the influence of data during training the model for avoiding the information leakage. At each step of gradient descent updating, for a random subset of examples, a clip norm operated on values and then for protecting privacy a noise is added to gradient descents.

There is another line of research which is not directly related to our problem, but is worth mentioning here. Liu et al. propose a protocol for computing scalar product, Secure Scalar Product in MapReduce (S²PM) [56]. They use the BGN homomorphic encryption scheme.

A line of works studies the problem of adding division to homomorphic encryption schemes. Veugen [89] considered a semi-honest model where client A has some encrypted number $[x]$, and the server B has the decryption key K . Party A would like to divide the integer x by some integer d . Two algorithms are introduced for division in this scenario: exact division with public divisors and approximate division.

Chen et al. [17] analyzed the performance of integer arithmetics over encrypted data and propose an algorithm for the division operation. They use HELib for implementation and their results showed that division for large numbers cannot be supported without using bootstrapping or even SIMD technique.

Livni et al. [59] analyzed the performance of polynomial as an activation function in neural networks. 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.

There are also a few papers in image processing over encrypted images. Rane et al. [76] used HE schemes to calculate some special distortion functions in a secure way. They

implemented two different protocols: one for the Hamming distance between binary vectors and the other for squared error distortion between integer vectors. Their implementation is based on the Paillier encryption scheme. Puech et al. [75] reviewed challenges on image and video encryption in face detection and recognition, as well as matching biometric data. Islam et al. [37] proposed a framework for sharing images between clients based on HE schemes. They designed a protocol for the secret sharing problem by using homomorphic properties of RSA and Paillier encryption scheme.

3.4. Classic Machine Learning

Besides solutions for Neural Network, there is a series of works which focus on the classic machine learning. We briefly survey these works here to exhibit the state-of-the-art in privacy preserving machine learning as a service.

Beck and Kerschbaum [8] proposed a method for privacy-preserving of string matching. Their system gives a deterministic approximation instead of exact distance. Their design is non-interactive with linear complexity without involving any third party. They extend the protocol which reveals whether there is a match and not the exact distance. Their solution is based on Paillier encryption scheme.

Samet et al. [80] proposed a solution for the private clustering (k-means algorithm) of data for two or more data owners. The data could horizontally or vertically partitioned among parties.

Brickell and Shmatikov [13] proposed a solution for private distributed learning based on decision tree algorithm. They consider a client that wants to build a classifier based on the dataset on the server. Their aim is to preserve the privacy of data and privacy of the final model, against client and server respectively. The solution is based on oblivious classifier construction.

Catak [14] proposed a protocol for private learning from vertically partitioned data among parties by using Extreme Learning Machine (ELM) algorithm (a feed-forward NN with one hidden layer). The proposed approach preserves the privacy of numerical attributes.

Their protocol create the classification model without sharing data with other parties. The final model is built by an independent party constructs and use for private classification.

Lindell and Pinkas [55] discussed using MPC technique in privacy preserving data mining. They review the definitions and constructions for Secure Multiparty Computation and address the issues and difficulties in this area. They discuss about the relationship between SMC and privacy preserving data mining, which problems addresses and which problem need to be solved in this area. This is one the first attempts in solving the problem about learning from distributed data by using SMC techniques.

Bogdanov et al. [10] discussed about using SMC for performing a large-scale privacy-preserving statistical study on real government data. Their solution is based on a combination of cryptographic secure multi-party computation together with organizational measures and microdata release controls. They use Sharemind MPC platform and developed it by adding more features for it like data transformations, new attribute calculation, aggregations, custom merging procedures and visualization.

Aslett et al. [3] reviewed homomorphic encryption schemes for statistical and machine learning algorithms. They provide a background for homomorphic encryption and also explain of Fan and Vercauteren scheme in the paper. They review limitations of homomorphic limitations like message space and cipher text size.

Page et al. [70] proposed a technique for increasing efficiency and parallelism about using FHE for some algorithms. Their implementation with HELib leads to 20x speedup. Their scenario involves comparing sensor data with a threshold on a cloud environment. Instead of converting an algorithm to a circuit, they convert algorithm to a matrix, the determinant of the matrix is the output of the algorithm. Based on their implementation, the matrix form is 20x faster than the naive method of Long QT method.

Du and Zhan [23] addressed private learning from a dataset which is partitioned vertically between two parties. These two parties want to learn from the dataset by using decision tree classifier, at the same, each party does not want to share data to other party or any other third party. For solving this problem, they propose a protocol that based on

using an untrusted third party. They provide a protocol for scalar product for using in the proposed protocol and is efficient than any existing solution.

Lagendijk et al. [49] proposed a method for preserving the privacy of user's data in signal processing domains. Their solution is based on homomorphic encryption schemes and SMC. They focus on privacy preserving in these three signal processing problems: face recognition, user clustering, and content recommendation. There are some key concepts in signal processing like linear operations, inner products, distance calculation, dimension reduction, and thresholding which covered by their solution. Their protocol is based on honest-but-curious security model. They also assume that the client can not submit unlimited processing request to the server, by this assumption, the privacy of the model is also preserved. The Paillier encryption scheme used in encryption part of their protocol.

Lu et al. [61] proposed a framework for statistical analysis over encrypted data. They focused on histogram (count), contingency table (with cell suppression) for categorical data; k-percentile for ordinal data; and principal component analysis and linear regression for numerical data. They proposed a modularized design and build blocks for implementing more complex methods. They consider different types of attributes (categorical, ordinal and numerical) and proposed for each a method for presenting these types to the system. Their implementation is based on HELib.

Basilakis et al. [7] focused on applications of homomorphic encryption schemes for preserving the privacy of patient data. The aim of the proposer solution is analyze the confidential healthcare information for supporting Clinical Decision Support (CDS). The data is distributed in a cloud structure. We preserve the privacy of data in healthcare applications and not the methods for analyzing this data.

Kim et al. [45] proposed a method for matrix factorization based on secure 2-party computation and fully homomorphic encryption. Their aim is providing an efficiently perform matrix factorization to compute user and item profiles from the ratings given by users in a privacy-preserving way with full functionality. They use approximation techniques like gradient descent for solving this problem. Their proposed structure has three main parts:

users, Recommendation System (RecSys) and Crypto-Service Provider (CSP). RecSys tunes parameters for matrix factorization and these parameters considered as private data for RecSys. Their implementation is based on hash-ElGamal which supports addition operation and HELib.

Melis et al. [62] proposed a protocol for processing outsourced network functions by using homomorphic encryption scheme. Outsourcing network functions revealed sensitive information like firewall rules, preserving the privacy of this information is the aim of [62]. Their security model is honest-but-curious. The inbound traffic passes through a cloud (called cloud MiddleBox or cloud MB). The cloud MB has the network policies and performs network functions on them before sending the traffic to the client. The cloud MB processes the traffic in real-time and also minimum amount of the computations run on the client. Their focus is on inbound traffic in this paper. They consider two different adversary model, strong and weak. In the strong case, the adversary has access to the public key, description of the algorithm and also the network function. In weak case, the system is a black box for the adversary, the adversary can send a packet to the system and receives the output for that packet. They use two different HE schemes for preserving the privacy against these adversarial models. First, Boneh, Goh, and Nissim (BGN) encryption scheme for the strong case and second, and Public-key Encryption with Keyword Search (PEKS) for the weak case.

Barni et al. [6] proposed a protocol for classifying an Electro Cardio Gram (ECG) signal without learning any information about the ECG signal in the server side and the client is prevented from gaining knowledge about the classification algorithm used by the server. The solution is based on Linear Branching Programs (LBP) and a cryptographic protocol for secure evaluation of private LBPs. The proposed protocol is a kind of secure two-party computation. They present a system for privacy preserving classification of ECG signals and link the complexity of the system to the classification accuracy. They also showed the overall complexity of the system can be drastically reduced by tailoring the ECG classification algorithm to the 2PC scenario. At the end, they implemented the secure protocol with respect to different parameter sizes and security levels. Their protocol

only includes the classification step and is similar to Yaos garbled circuit (GC) and also based on Pialiari encryption scheme. The proposed protocol designed based on two different techniques: Garbled Circuits (GC) and on a hybrid combination of the homomorphic Paillier cryptosystem and GCs. They compared these two different designs and concluded that from communication complexity wise, the Hybrid solution works more efficient. They also concluded that by increasing the security parameters, the Hybrid protocol is more affected than the GC protocol.

Kenper et al. [40] introduced a technique -Computing on Masked Data (CMD) - for computations to be performed on masked data and only authorized recipients can unmask the data. Their technique is based on combining efficient cryptographic encryption methods with an associative array representation of big data to enable a low computation cost approach to both computation and query while revealing only a small amount of information about the underlying data.

Lindel and Pinkas [54] proposed a protocol for running a data mining algorithm on a dataset which distributed between two parties. They focus on a popular decision tree classifiers, ID3. They consider two main issues in their design: rounds of communication and reasonable bandwidth. The adversary model in their design is semi-honest. Their idea is based on Oblivious transfer, Oblivious polynomial evaluation and Yao's two-party protocol.

Aono et al. [2] built a secure system for preserving the privacy of data in logistic regression by using HE schemes. They show that only having an HE scheme which support addition operation is enough for preserving the privacy of the data. Their solution preserve the privacy of data in both training and predicting steps. By using function approximation, the original logistic regression converted to homomorphic-aware regression. They also show how to add differential privacy into the system. The solution is based on the honest-but-curious.

Xiao et al. [64] proposed several blocks for constructing an ID3 algorithm over horizontally partitioned data while preserving the privacy of each part. Their protocol can preserve the privacy of data in two parties or more. For implementing the algorithm, their design includes some blocks like Secure multi-party $x \log x$ protocol, secure two-party

multiplication protocol, secure two-party reverse multiplication protocol and secure multi-party reverse multiplication protocol. Their security model is based on semi-honest security model. Their protocol preserve the privacy of data for training and classification steps.

Togan et al. [85] proposed a protocol which run on a cloud (an untrusted server) for finding maximum/minimum of encrypted integers. In their design, running the algorithm could be done without collaboration with the client (data owner). At the end, the server does not know the maximum value or its index. They implement the protocol by using BGV encryption scheme and HELib. They also implement the same protocol by using FHEW library and the time is at least twice in compare of HELib.

Bent et al. [9] proposed a solution for storing encrypted data in a cloud and performing arbitrary operations on the encrypted data one behalf of the user. Ideally, coalition partners would wish to share data that can be used to compute specific results that are only relevant to a given operation, without revealing all of the shared information. They describe an example of a service that uses the proposed secure coalition cloud to perform a comparison between the planned routes of different coalition forces. This service determines if there are common points of intersection of the planned routes in space and time, without revealing any other information about the respective routes. They use HELib in their implementation.

Liu et al. [57] proposed a protocol for outsourcing SVM classification to a cloud environment. In their protocol, the server and users perform collaborative operations on encrypted and outsourced data while preserving the privacy of each user's data. The assumption about security model is honest but curious for both client and server parts. The only operations on user side are encryption and decryption. In their protocol, the most part of computation including addition and multiplication is done by the server. The number of communications is low in their protocol. In their implementation of SVM, the most times consuming part is kernel matrix generation. From operation wise, the most time-consuming operation is multiplication (around half of the running time). Their implementation is based on HELib library.

Suthampan and Maneewongvatana [84] addressed the problem for preserving of data

which distributed vertically among parties while running a decision tree on it. Their proposed could be generalized to use in an environment with more than two parties. Their solution includes both training and classification. They considered the ID3 algorithm, however, their solution can easily be adapted to other decision tree algorithms. The number of attributes in each party's node, the information gain for each party and the owner and cardinality of each node are revealed to other parties. By completing the protocol, all parties have the final model. At a high level description of their algorithm, each node computes the information gain of its own attributes and compares it to other party's information gain and decides about the attribute of the current level.

Hall et al. [32] proposed a protocol for statistical calculation over a data that is distributed between different parties. The focus of their solution is on computing linear regression and ridge regression estimates. Their method works for horizontally or vertically distributed data among parties. Their assumption about the security model is semi-honest. Their protocol is a sequence of steps, each party performs computations, and transmits results to other parties. Their idea is based on the combination of an extension to Yao's original idea and Paillier HE scheme.

Hall et al. [32] proposed a solution for performing statistical calculation on a distributed database among different parties without combining the datasets of each part. They use homomorphic encryption for regression analysis. By running their protocol, each party only shares the final results without sharing any other intermediate values during computations. They also implement their protocol to analyze the practicability of the proposed solution. Paillier homomorphic encryption scheme which supports multiplication of two ciphertexts and computing power of a ciphertext. Their algorithm has two main steps: computing covariance matrix and inverting the matrix. They implement the algorithm on a database with 51,016 cases and 23 covariates which are distributed between three parties in column-wise fashion. Each step of the algorithm takes one day for the computations.

Biswas et al. [80] proposed a protocol to build a model based on disjoint datasets which are distributed among parties by using k-means clustering algorithm. Their idea is based

on SMC and Paillier encryption scheme. The datasets of parties have the same format. All parties are supposed to be semi-honest in their design. One block of the algorithm is a Multi-Party Addition function, this function securely adds input from different process (one process for each party). Increasing number of parties, clusters, size of dataset in each party and the dimensionality will lead to multiplicative increases the run-time.

Kerschbaum [41] proposed a protocol for computing the intersection of two sets by using HE schemes. A client and a server have private sets. At the end of the protocol, the client computes the intersection of two sets without revealing information about its set and the server learns nothing about the intersection. The client and server submit their sets in a service provider, the service provider computes the intersection without learning information about the sets or the inputs. The security model is honest-but-curious. The solution is based on Goldwasser Micali encryption scheme and Sander Young Yung (SYY) techniques. SYY describe a technique for one logical-and operation on ciphertexts. For computing the intersection, the idea is based on Bloom Filter method. This method checks the inclusion of an element in a set. In their implementation, instead of using GM encryption scheme, Boneh, Goh, Nissim (BGN) encryption scheme is used.

Baardewijk[88] proposed a solution for comparing biometric encrypted data base on Hamming distance. The author implement iris biometric by using HELib library, the result shows the implementation works, however, the encryption, computation and decryption of an iris scan takes almost 10 minutes which is not practical for real applications.

Lauter et al. proposed a solution for some statistical evaluation over encrypted data by using HE schemes, [50]. Their implementation is based on a modified Lopez-Alt and Naehrig encryption scheme which supports addition and multiplication. They provide a method for encoding genome data and also changes in the structure of the algorithm for adopting them within the limitations of HE schemes.

The focus in this dissertation is neural network, therefore, we only consider papers with focus on neural networks or convolutional neural networks. We consider both SMC-based and HE-based solutions. More specifically, in Chapter 6, we will compare our solutions with

these papers, [16], [65], [58], [15], [77], [79] and [22]. These papers include more recent results for privacy preserving machine learning as a service.

CHAPTER 4

CHALLENGES AND SOLUTIONS

We aim to preserve the privacy of the main components of a Machine Learning as a Service (MLaaS) framework. We consider two different scenarios.

- First, a client-server model where the client owns the data; the server uses this data to build a model and provides prediction services to the client. In this scenario, both **training** and **classification** phases operate over encrypted data.
- Second, the server has a trained model and classifies the client's unseen instances. In this scenario, the **classification** phase only operates over encrypted data.

In the first scenario, the server gets the encrypted training dataset from the client and trains a model on the encrypted data. After building the model, the client sends new encrypted instances to the server, the server performs an inference on the data, and the client gets the result of prediction in the encrypted format. We aim to preserve the privacy of feature values, class labels, number of instances, predictions of unseen instances, and performance of the model against the server as well as machine learning algorithm and model against the client.

In the second scenario, the server has access to the training data and builds a model based on the plain data set, and the final model is in plaintext. The client sends instances to the server, the server classifies received instances and sends the classification results back to the client. We aim to preserve the privacy of feature values and prediction of unseen instances against the server, and machine learning algorithm and model against the client.

Then, we consider a dataset which is distributed among clients. Besides the challenges for a client-server architecture, we face new challenges for this scenario. The main challenge is computation on the data that encrypted with different private keys. We explain more details about this case in the following sections. Our design is based on these three principles:

- Since the original data is possessed by each data owner, no matter how the model is used, the data should be kept confidential to other entities. This is a learning privacy problem. Most existing works focus on solving this problem, where the learning data

is protected, but the learned models are published to everyone.

- The learned model is valuable and private to the learner, it should be kept secret in the predicting or comparing process. The leakage of a model results in the loss of interests of data owner or even to a breach of the original data. This is a model privacy problem. Without any protection, the predicting results can be utilized to reverse engineering the model and even retrieve the original data in recent adversarial studies [81] and [86].
- The unseen data is also private and should be protected properly in the predicting phase. Since the test data are predicted by the learned model, it also belongs to the model privacy problem. Little work pays attention to this part, but schemes described are either inefficient or lack practical applications.

4.1. Challenges

In the following section, we provide a list of components in the framework and discuss the privacy of each component in details.

- Feature values.
- Class labels.
- Number of instances.
- Number of features.
- Machine learning algorithm.
- Model.
- Predictions for unseen instances.
- Performance of the model.

We describe the security issues of each component separately, then explain our proposed protocol. We also describe, the privacy of each component in our proposed solution.

Feature values are one of the main concerns from the prospective of the data owner. These values contain sensitive information and revealing them to the server is an important security flaw in the system. One example of sensitive information which stores as feature

values are medical records. The Health Insurance Portability and Accountability Act (HIPPA) and the Health Insurance Technology for Economic Clinical Health (HITECH) Act are two standards for protecting the privacy of patient’s data. In all proposed scenarios, independent of the basic idea behind the protocol (like SMC, differential privacy or HE schemes), the goal is to preserve the privacy of feature values. This component is the main parameter for analyzing the security of the proposed protocol. Minimum (or no) information about the feature values should be revealed to a non-authorized party. In our protocol, the feature values are encrypted by the owner of the data before being transferred to another party or the cloud. Therefore, no information is leaked to the non-authorized party. Additionally, the SMC technique can be used to preserve the privacy of this component, however; in differential privacy we cannot assure that no information is leaked to the third party, because the noise in the feature values are accessible to non-authorized parties.

The next component is Class label. For classification problems, important information can be extracted from only having access to the class labels related to instance. For example, any non-authorized party can extract information regarding the distribution of the classes and the number of categories in the dataset only by having access to the class labels of the instances. We mention that, in the case of the training phase considered in the delegation of computation to a third party, the classes of the instances should be considered in the security analysis of the system. When the model is built based on a plain-text dataset, the assumption is that a third party has access to the dataset, the feature values, and the classes assigned to the instances. During the security analysis of our proposed protocol, we consider the privacy aspect of the labels of the instances during the training phase. We encrypt the class labels for all the instances before sending them to the server, so the server doesn’t have access to the labels and no information is leaked to the non-authorized third party.

Number of instances and number of features are not sensitive in most cases. If we want to preserve the privacy of the size of the dataset (which can be inferred from these two values), the protocol will be complex, especially in the training phase. For example, in a neural network, the model is fed each of the instances (or batch of instances) to run

the feed-forward step, compute the error of the output, and then update the weights based on this error. The server needs to have access to the number of instances for running the algorithm. Another example, in decision trees, information gain is calculated for each feature in each step and the most informative feature is selected for splitting in each level. The server needs to have access to the number of features to run this algorithm. In our protocol and all other proposed protocols these two components are not considered as sensitive information and non-authorized parties can have access to them.

The algorithm for learning from data is also important if the algorithm is not shared among parties. In MLaaS scenarios, the privacy of this component is considered, and it should be taken into consideration during the design phase of the protocol. Let us consider a company which provides data analysis services for other companies. The method for processing the data is one of their assets and they do not reveal information about how the model is built. They only send the result back to the client after the analysis.

Preserving the privacy of this component is important for the server's privacy. In our proposed protocol, the privacy of this component is preserved, as the server does not reveal any information about the steps of analyzing the data to the client. In solutions which are based on the SMC technique, the client and server take part through a protocol for building the model or classifying a new instance. The client needs to know the structure of the algorithm to run the computation. It is difficult in these cases to preserve the privacy of the learning algorithm. Some techniques, like differential privacy, the structure of the algorithm might also be revealed to other parties, because of the pattern noise added to the data is based on the computation which is supposed to run over the data. We note that, even though the client needs to build the encryption scheme based on the depth of the algorithm, revealing the depth of the algorithm to the client is not considered a security issue. For example, in the case of leveled homomorphic encryption, the depth of the algorithm is the number of multiplications and revealing the number of multiplications to a non-authorized party is not considered a security flaw of the system.

The privacy of the model is important for both the client and the server. It is important

for the client because the model contains the patterns of the dataset and knowledge extracted from instances. Therefore, the model should be kept confidential from the server. From the server-side perspective, such as the analysis algorithm, it is an asset of the server. As in the MLaaS scenario, the company provides a service to the client by classifying new instances. The model should be kept confidential from the client. Therefore, preserving the privacy of the model is a bit complicated. In our protocol, the server builds the model and doesn't transfer it to the client; therefore, the privacy of the model is preserved from the client. The model is in an encrypted format and the server does not have access to the model in the plain format, so it can't infer information of the dataset and the privacy of the model is preserved from the server.

If the security of the model is not considered for an application, the server sends the model back to the client, however, for future classifications, the client still needs the server for computation related to the classification part. Assuming that model is already trained, if the server has access to the training data and the model, which can be the case when an algorithm only considers the classification step, the privacy of the classification step is important, which we discuss in the next part. One recent domain research -General Adversarial Attack (GAN) see [81, 86, 25]- the attacker gains information about the training set that model built upon it. An attacker only by sending query and receive the prediction results can extract pattern from the training set which has sensitive information. Avoiding this attack is not completely possible, but by preserving the privacy of the model, we can limit the attacker's effect on the system.

The next component is predictions for unseen instances. The results from the classifier is another component and all the private classification protocols consider this component as private information of the client. For example, the result of analyzing medical records of a patient has sensitive information about the current health situation. Any party except the owner of the data should not have access to the results for an unseen data, independent of whether the privacy of the model has been considered or not. In our protocol, the results from the classifier are encrypted and the server does not have access to it and its privacy is

preserved. In solutions based on SMC, depending on which party should have access to the result, the protocol is different.

The performance of the model is not considered as a threat to privacy in most solutions. However, in some scenarios, this could lead to an information leakage. For example, if the server has access to the prediction error for each of the instances, there is some information leakage of the dataset and the model. As we've already discussed about the predictions for unseen instances in the earlier section, in our protocol, the server does not have access to the error value, because only the client can access the predictions and calculate the error of the model.

In conclusion, in our protocol only the number of instances and the number of features are shared among parties or with the cloud. For other components, based on the application, the privacy can be preserved. We suppose that the HE scheme does not support bootstrapping function and we use communication between client and server to overcome noise growth in the ciphertext during computation. The lack of bootstrapping functionality does not have any impact on the security of the model and only increases the running time because of the communication overhead.

4.2. Different Data Distribution Scenarios

In this section, we focus one training over distributed data problem. Consider a set of clients $\{c_1, c_2, \dots, c_n\}$. Each client possesses a dataset, D_i . We aim to learn a classifier from the data set \mathcal{C} which is defined as $\mathcal{C} = \{D_1 \cup D_2 \cup \dots \cup D_n\}$. This concept is different than distributed training which focuses on parallel computation for speeding up training computation. The aim of that problem is distributing computation among nodes for decreasing the training time. In the second scenario, the privacy of data is not a concern. The basic idea is to scale up the stochastic gradient descent method.

Some challenges for distributed training are race conditions, deadlocks, distributed state, and communication protocols while simultaneously developing mathematically complex models and algorithms.

We consider three distribution models for data: Horizontally Distributed Data, Vertically Distributed Data and Hybrid Distributed Data.

- **Horizontally Distributed Data:** The data is horizontally distributed, examples corresponding to a particular value of a particular attribute are scattered at different locations.
- **Vertically Distributed Data:** In vertically distributed datasets, we assume that each example has a unique index associated with it and belongs to one of the c_i 's.
- **Hybrid Distributed Data:** A combination of horizontally and vertically distributed data.

Several solutions proposed for the first two cases. For the third case, limited solutions are available, such as SecureML, [65]. Solutions based on differential privacy consider the vertically distributed case, the rows are distributed among and each clients has the same number of features.

As we said before, in the vertically distributed case, each client has a subset of instances and the number of features is the same in all instances in each client. In the horizontally distributed case, the features are distributed among clients. For the second case, we do not have any efficient solution based on the HE, because the server needs to do the computation in parallel over ciphertexts, which are encrypted under different keys. We assume that the data is distributed vertically. Each client has a subset of instances, and the number of instances might be different in each client, but the number of features is the same in all instances.

The first and the most important challenge is using encrypted data for learning. The client generates the secret/public keys and shares the public key with the server. The client and the server use this key for encryption and computation over encrypted data. In the distributed datasets scenario, the server receives different parts of data from different clients, each part is encrypted with a different public key. One solution is using *Multi-Key Homomorphic Encryption Schemes*. In such encryption schemes, each client generates its own public/private key and encrypts the data with the public key. The server executes over received encrypted data (with different public keys). Current Multi-Key Homomorphic

Encryption implementations are very slow and by increasing the number of clients, the error increases and the decryption function does not work correctly, therefore, we do not consider this scheme in our solution. Another theoretical solution for this challenge is using bootstrapping. In bootstrapping, the server can transfer the ciphertext from one public key to another without access to the secret key. The secret key of the client is encrypted with the public key and it sent to the server. The server by using the advantage of homomorphic encryption properties (running computation over encrypted data), runs an algorithm which transfers the ciphertext from one public key to the another. The shortcomings of using bootstrapping is the high computation cost of running the bootstrapping function. To convert the data from one client to another, the server only needs to run the bootstrapping function one time. It might be possible given high computation power on the server side. However, the server should run the bootstrapping several times. More often, the server needs to run the algorithm for more than one epoch. In each epoch, the server learns from the whole dataset by using the algorithm. Now, if we want to run n epochs and we have c clients, the server should run the bootstrapping function at least $n \times (c - 1)$. This is the minimum amount of running bootstrapping, because if the dataset is distributed among clients with the same pattern, each client has the same amount of data and the distribution of classes is the same in each client's data. If the data is not distributed normally, then after each iteration (batch input) the server should run the bootstrapping. In the worst case, the data needs to shuffle because of non-uniform distributed data. The server cannot use online learning and should train from one instance in each iteration, running the bootstrapping function after training from each instance. The second solution is using the adding noise technique to the data and using communications between the client and the server for running the algorithm. The server starts the algorithm by learning from client c_1 , after feeding the data to the algorithm and updating the weight. Before learning from client c_2 's data, the server adds noise to the weights, sends them to the client c_1 , who decrypts the noisy model and sends it back to the server which removes the noise and continues the algorithm by learning from the data of client c_2 . Using this solution reveals the model to the server. We can avoid

this risk by sharing the public keys among clients. The client c_1 after receiving the noisy model, decrypts it and encrypt with the client c_2 public key and sends it back to the server. Now, the server continues the algorithm by using the encrypted instances from client c_2 . It has a computation cost overhead over clients, but, the model is not revealed to the server. However, it has computation over cost and communication among clients which is not always available. We consider this solution not to be practical for our problem.

Besides all these challenges and proposed solutions, there is a potential technique for learning from distributed datasets is ensemble learning. The server can train different models based on each dataset from each client and then combine the model to build a general model from all trained model. The way we combine the models is important in the solution, if a technique like majority voting is used, we can implement it over encrypted data. If more complex techniques used by the server (e.g. train a strong model based on the weak models) for combining the models, then the same challenges exist during the calculations. We mention here this solution does not necessary give a good solution. we do not train a model from the whole dataset. The models from a part of the datasets have a low accuracy in comparison of a model, which is trained based on the whole dataset.

In conclusion, by using a solution only based on HE schemes, the training over distributed datasets is not practical. There is no solution proposed for this scenario. Therefore we consider this case as a part of future work for the application of an HE scheme in privacy-preserving MLaaS.

CHAPTER 5

POLYNOMIAL APPROXIMATION

In this chapter, we discuss the theoretical background of our proposed method for adopting a network within limitations of HE schemes.¹ One of the main challenges is finding a replacement for the activation functions like ReLU or Sigmoid. In the next section, we provide the details of theoretical background of approximating an activation function with a low degree polynomial. This chapter adapted from [36].

5.1. Polynomial Approximation

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 [93, 5].

Let us denote the family of all continuous real valued functions on a non-empty compact space X by $C(X)$. Suppose that among elements of $C(X)$, a subfamily A of functions are of particular interest. For simplicity, one can think of X as a closed, bounded interval $[a, b]$ in \mathbb{R} and A as the set of polynomials in a single variable with real coefficients. 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 $f \in C(X)$ 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 $f \in C(X)$ can be approximated by elements of A .

THEOREM 5.1 (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 $p(x) \neq p(y)$.*

¹Parts of this chapter have been previously published, either in part or in full, from Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Rebecca Wright, *Privacy-preserving machine learning as a service*, Proceedings on Privacy Enhancing Technologies 2018 (2018), 123–142.

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.

$\|f\|_\infty$ (the sup norm of f) of a given function $f \in C(X)$ is defined by

$$(5.2) \quad \|f\|_\infty = \sup_{x \in X} |f(x)|,$$

Then the above argument can be read as: *For every $f \in C(X)$ and every $\epsilon > 0$, there exists $p \in A$ such that $\|f - p\|_\infty < \epsilon$.* It is easy to see that $\|0\|_\infty = 0$, $\|\lambda f + g\|_\infty \leq |\lambda| \|f\|_\infty + \|g\|_\infty$ (subadditivity) and $\|f \times g\|_\infty \leq \|f\|_\infty \times \|g\|_\infty$. The function $\|\cdot\|_\infty$ on $C(X)$ is an instance of the *norm* on the function space $C(X)$, which also resembles the structure of inner product spaces which have nice geometry and one can define and develop intuitive concepts over them easily. Let V be an \mathbb{R} -vector space, an *inner product* on V is a function $\langle \cdot, \cdot \rangle : V \times V \rightarrow \mathbb{R}$ satisfying the following requirements:

- (1) $\langle f, f \rangle \geq 0$;
- (2) $\langle f, f \rangle = 0$ if and only if $f = 0$;
- (3) $\langle \alpha f + \beta g, h \rangle = \alpha \langle f, h \rangle + \beta \langle g, h \rangle \geq 0$ for every $\alpha, \beta \in \mathbb{R}$;
- (4) $\langle f, g \rangle = \langle g, f \rangle$.

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 . A basis $\{v_\alpha\}_{\alpha \in I}$ is called an orthonormal basis for V if $\langle v_\alpha, v_\beta \rangle = \delta_{\alpha\beta}$, where $\delta_{\alpha\beta} = 1$ if and only if $\alpha = \beta$ and is equal to 0 otherwise. Every given set of linearly independent vectors can be turned into a set of orthonormal vectors that spans the same sub vector space as the original. The following well-known theorem gives us an approach for producing such orthonormal vectors from a set of linearly independent vectors:

THEOREM 5.3 (Gram–Schmidt). *Let $(V, \langle \cdot, \cdot \rangle)$ be an inner product space. Suppose $\{v_i\}_{i=1}^n$ is a set of linearly independent vectors in V . Let $u_1 := \frac{v_1}{\|v_1\|}$ and (inductively) let $w_k := v_k - \sum_{i=1}^{k-1} \langle v_k, u_i \rangle u_i$ and $u_k := \frac{w_k}{\|w_k\|}$, then $\{u_i\}_{i=1}^n$ is an orthonormal collection, and for each*

k ,

$$(5.4) \quad \text{span}\{u_1, u_2, \dots, u_k\} = \text{span}\{v_1, v_2, \dots, v_k\}.$$

Note that in this theorem, we can even assume that $n = \infty$.

Let $B = \{v_1, v_2, \dots\}$ be an ordered basis for $(V, \langle \cdot, \cdot \rangle)$. For any given vector $w \in V$ and any initial segment of B , say $B_n = \{v_1, \dots, v_n\}$, there exists a unique $v \in \text{span}(B_n)$ such that $\|w - v\|$ is the minimum as shown in the below theorem.

THEOREM 5.5. *Let $w \in V$ and B be a finite orthonormal set of vectors (not necessarily a basis). Then for $v = \sum_{u \in B} \langle u, w \rangle u$*

$$(5.6) \quad \|w - v\| = \min_{z \in \text{span}(B)} \|w - z\|.$$

For proof see [93, 5].

Now, let μ be a finite measure on X and for $f, g \in C(X)$ define $\langle f, g \rangle = \int_X fg d\mu$. This defines an inner product on the space of functions. The norm induced by the inner product is denoted by $\|\cdot\|_{2,\mu}$. It is evident that

$$(5.7) \quad \|f\|_{2,\mu} \leq \|f\|_{\infty} \mu(X)^{1/2}, \quad \forall f \in C(X),$$

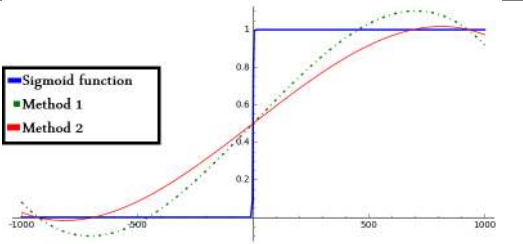
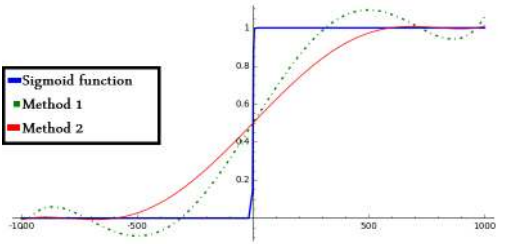
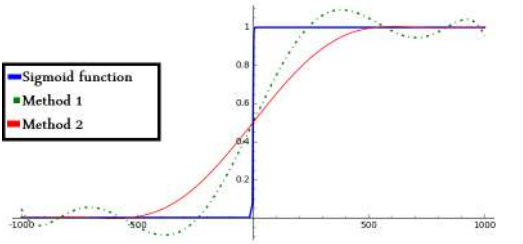
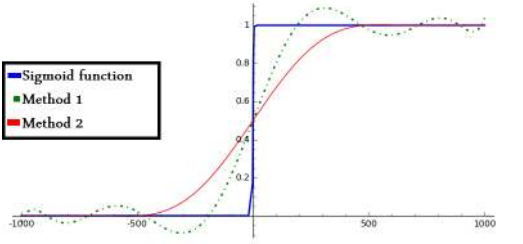
which implies that any good approximation in $\|\cdot\|_{\infty}$ gives a good $\|\cdot\|_{2,\mu}$ -approximation. But generally, our interest is the other way around. Employing Gram–Schmidt procedure, we can find $\|\cdot\|_{2,\mu}$ within any desired accuracy, but this does not guarantee a good $\|\cdot\|_{\infty}$ -approximation. The situation is favorable in finite dimensional case. Take $B = \{p_1, \dots, p_n\} \subset C(X)$ and $f \in C(X)$, then there exists $K_f > 0$ such that for every $g \in \text{span}(B \cup \{f\})$,

$$(5.8) \quad K_f \|g\|_{\infty} \leq \|g\|_{2,\mu} \leq \|g\|_{\infty} \mu(X)^{1/2}.$$

Since X is assumed to be compact, $C(X)$ is separable, i.e., $C(X)$ admits a countable dimensional dense subvector space (e.g. polynomials for when X is a closed, bounded interval). Thus for every $f \in C(X)$ and every $\epsilon > 0$, one can find a big enough finite B , such that (5.8) holds. In other words, “good enough $\|\cdot\|_{2,\mu}$ -approximations of f give good $\|\cdot\|_{\infty}$ -approximations”, as desired. This is particularly useful, since the Gram–Schmidt

procedure provides a concrete algorithm to compute best $\|\cdot\|_2$ -approximations, while just computing $\|\cdot\|_\infty$ in general is an NP-complete task.

TABLE 5.1. Polynomial approximation for the Sigmoid function on the interval $[-10^3, 10^3]$. $p_1(x)$ generated by method 1 (Equation 5.9) and $p_2(x)$ generated by method 2 (Equation 5.10).

Polynomial Approximations	Sigmoid function, $p_1(x)$ and $p_2(x)$
$p_1(x) = -(8.6^{-10}) * x^3 + \dots + 0.499$ $p_2(x) = -(4.8^{-10}) * x^3 - \dots + 0.500$ $\ f - p_1\ _2 = 0.276, \ f - p_2\ _2 = 0.270$	
$p_1(x) = (2.06^{-15}) * x^5 - \dots + 0.50$ $p_2(x) = (6.65^{-16}) * x^5 + \dots + 0.5$ $\ f - p_1\ _2 = 0.22, \ f - p_2\ _2 = 0.09$	
$p_1(x) = -(5.91^{-21}) * x^7 + \dots + 0.5$ $p_2(x) = -(1.18^{-21}) * x^7 - \dots + 0.5$ $\ f - p_1\ _2 = 0.2, \ f - p_2\ _2 = 0.03$	
$p_1(x) = (1.84^{-26}) * x^9 - \dots + 0.5$ $p_2(x) = (2^{-27}) * x^9 + \dots + 0.5$ $\ f - p_1\ _2 = 0.17, \ f - p_2\ _2 = 0.01$	

In practice, $X = [a, b]$ and the countable dimensional subspace is the algebra of polynomials which satisfies the assumption of the Stone–Weierstrass theorem and the set of monomials is admissible in the Gram–Schmidt process. Different choices of μ , gives different systems of orthogonal polynomials. Two of the most popular measures are $d\mu = dx$ and

TABLE 5.2. Polynomial approximation for the Sigmoid function on different intervals. $p_1(x)$ generated by method 1 (Equation 5.9) and $p_2(x)$ generated by method 2 (Equation 5.10).

Polynomial Approximations	Sigmoid, $p_1(x)$ and $p_2(x)$
<p>$Interval = [-10, 10]$</p> <p>$p_1(x) = 10^{-5} * x^5 - \dots + 0.5$</p> <p>$p_2(x) = (6.37^{-6}) * x^5 \dots + 0.5$</p> <p>$\ f - p_1\ _2 = 0.06, \ f - p_2\ _2 = 0.008$</p>	
<p>$Interval = [-10^2, 10^2]$</p> <p>$p_1(x) = (2.06^{-10}) * x^5 + \dots + 0.5$</p> <p>$p_2(x) = (6.65^{-11}) * x^5 + \dots + 0.5$</p> <p>$\ f - p_1\ _2 = 0.20, \ f - p_2\ _2 = 0.03$</p>	
<p>$Interval = [-10^3, 10^3]$</p> <p>$p_1(x) = (2.06^{-15}) * x^5 + \dots + 0.5$</p> <p>$p_2(x) = (6.65^{-16}) * x^5 + \dots + 0.5$</p> <p>$\ f - p_1\ _2 = 0.22, \ f - p_2\ _2 = 0.09$</p>	
<p>$Interval = [-10^4, 10^4]$</p> <p>$p_1(x) = (2.06^{-20}) * x^5 - \dots + 0.5$</p> <p>$p_2(x) = (6.7^{-21}) * x^5 + \dots + 0.5$</p> <p>$\ f - p_1\ _2 = 0.23, \ f - p_2\ _2 = 0.3$</p>	

$d\mu = \frac{dx}{\sqrt{1-x^2}}$. By using $d\mu = dx$ on $[-1, 1]$, the generated polynomials are called Legendre polynomials and by using $d\mu = \frac{dx}{\sqrt{1-x^2}}$ on $[-1, 1]$ the generated polynomials are 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. For more details about these polynomials, see [93, 5].

Now, we experiment with polynomial approximations of the Sigmoid function $\frac{1}{1+e^{-x}}$ 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

$$(5.9) \quad d\mu = \frac{dx}{l\sqrt{1 - (x/l)^2}}.$$

Our second choice comes from the measure.

$$(5.10) \quad d\mu = e^{-(l/x)^2} dx.$$

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. Figure 5.1 shows the output of our approximating algorithm with the polynomial form of the activation function.

5.2. Polynomial Computation over Encrypted Values

There are some challenges related to homomorphic encrypted values. First is that all values must be integers. We used a vector of integer values as an input for polynomials. Coefficients in polynomial approximations are small. We truncated coefficients and multiplied the polynomial with a power of 10 to make all coefficients integer. The output of new polynomials for large integers like 10^4 was out of the acceptable range. In our implementation, we put some small values as coefficients.

We used HELib for implementation. For generating schemes in HELib, we set some values for input parameters, $k = 80$, $s = 1$, $w = 64$ (for details about these parameters

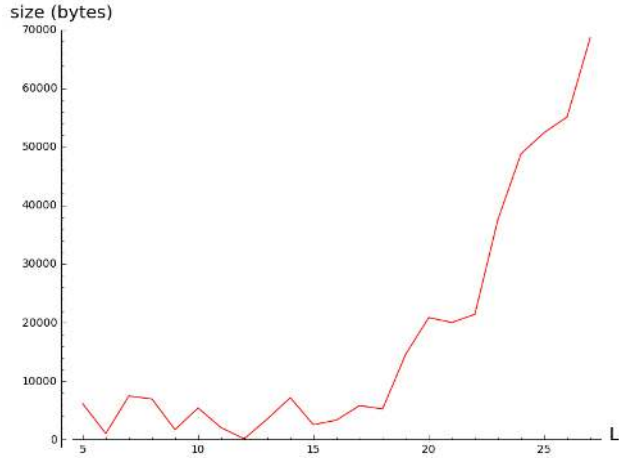


FIGURE 5.1. Ciphertext size and input values for L

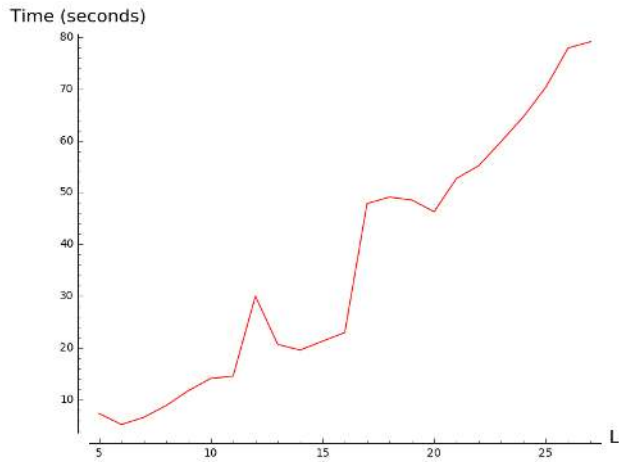


FIGURE 5.2. Time for generating encryption scheme and input values for L

see [31]). The other important value is L . Small values for L decreases the computation time, however, the degree of polynomial also decreases. Large values of L lead to slower computations and larger degrees for polynomials. Size of ciphertext increased by increasing the value for L , see figure 5.1. The time for generating encryption scheme also increases by increasing the value for L , see figure 5.2.

We used polynomials from degree 2 to 9 and gave integers less than 100 as inputs to these polynomials. By growing the degree of polynomials, we increased value of L from 5 to 25. All the computations are in mod $10^{18} + 37$ which is a prime number. For each degree, running time for polynomial computation and the minimum value for L are showed in table

5.3. Now, we approximated activation function over encrypted data, our algorithm is adapted

TABLE 5.3. Polynomial computation over encrypted values

Degree	Running Times	L
2	0.122829	9
3	0.305238	15
4	1.81771	21
5	2.19533	21
6	2.26061	25
7	2.71306	25
8	3.27217	25
9	4.59763	27

within the limitations of HE schemes, in the next chapter, we use our adapted form of the algorithm over encrypted data and provide experiments' results.

CHAPTER 6

RESULTS

In this chapter, we provide our experimental results for private preserving MLaaS. We compare our results with the state of the art solutions.¹ At the first section, we evaluate the performance of the underlying HE library. In the following, we provide our results for NN and CNN over encrypted data, both classification and training steps. .

For the training, we use a virtual machine with 48GB RAM, 12 CPU cores and Ubuntu 14.04 for the training and for the classification step, we use a virtual machine with 16GB RAM, Intel Xeon E5-2640, 2.4GHz and Ubuntu 16.04.

6.1. Homomorphic Encryption Performance

We use the HELib for implementation as the underlying homomorphic encryption library that implements BGV scheme, see [12]. Before running the protocol for both NN and CNN, we need to run some operations independent of the learning algorithm. This step is necessary for all HE-based solutions, *Key Generation* and *Key Sharing*. Homomorphic Encryption scheme that we use as a part of our solution is public/private encryption scheme. The owner of the data runs the key generation function, this function generates a private key and a public key. Private key is for decryption and public key is for the encryption. All the entities in the protocol (for example, Service Provider), have access to the public key.

In our protocol, client needs to generate the public key and the private key, and shares (sends) the public key to the server. This step is one time computation and need one round of communication. The client shares the public key with the server, the key is valid until its expiration date. For having a better estimation of running time of key generation and

¹Parts of this chapter have been previously published, either in part or in full, from Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Catherine Jones, *Privacy-preserving machine learning in cloud*, Proceedings of the 2017 on Cloud Computing Security Workshop (New York, NY, USA), CCSW 17, Association for Computing Machinery, 2017, p. 3943, Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Rebecca Wright, *Privacy-preserving machine learning as a service*, Proceedings on Privacy Enhancing Technologies 2018 (2018), 123–142 and Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi, *Deep neural networks classification over encrypted data*, Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy (New York, NY, USA), CODASPY '19, ACM, 2019, pp. 97–108. Reproduced with permission from the Association for Computing Machinery.

sharing, we provide details about the key generation and key transfer in Table 6.1. Now, the

TABLE 6.1. Key Generation and Key Transfer time

p	L	Key Generation(s)	Key Transfer(s)
1009	5	3.1e-05	2.59
1002263	10	3.1e-05	8.86
10000030013	15	4.1e-05	18.56
1000000034057	10	3.4e-05	10.40
1000000034057	15	3.2e-05	19.34
19900000085257	20	3.5e-05	32.71
19900000083773	20	3.2e-05	32.06
1000000034057	17	3.3e-05	18.82
1000000034057	19	3.4e-05	47.65
1013329	5	3.2e-05	2.48
1013401	10	3.5e-05	10.33
10000030471	15	3e-05	16.26
10000024429	15	4.3e-05	13.40
19900000085753	20	3.3e-05	32.51

client and the server are ready to run the protocol. The next step is running algorithm over encrypted data.

6.2. Neural Networks over Encrypted Data

In this section, we focus on privacy preserving neural networks. At first, we implement NN over plaintext by considering HE limitations. Then, we transfer our learning algorithm over encrypted data and provide the performance of the algorithm, both running time and accuracy of the final model.

For implementing NN over Encrypted Data in C++, we focus on fully connected feed-forward back-propagation neural networks. Current implementations which work with

the plain data, usually have three classes: *net*, *layer* and *neuron*. Each neuron stores input values, output values, weights and other required parameters. A layer is an array of neurons and neural network is a combination of these layers in a specific order. During the feed-forward or back-propagation functions, we go through all layers and in each layer, we process all the neurons in that layer.

Depending on the method that feeds inputs to the neural network, online learning or batch learning, we use two different implementations to estimate the running time and memory. In the following sections, we explain each of them in details.

For the online learning, we can use two different approaches. When we use HELib or other SIMD-enabled HE libraries, we can assign one ciphertext for w and one ciphertext for $\Delta weight$, one ciphertext for output, one ciphertext for the input and so on. The size of ciphertexts in HE is big and this structure is not efficient memory wise. WE implemented this approach and we concluded it's not practical, because by increasing the size of the network, the memory usage increasing drastically.

We designed neural network over encrypted data in a more efficient way for online learning. Each layer has an input and an output, and a set of weights connects each layer to the next layer. We keep input and output of each layer in two different ciphertexts, and we keep set of weights in one ciphertext. We only have two classes, *network* and *layer*. We need some changes in the way that we store values inside vectors which we leave these details for the implementation. This solution is more efficient compare to the previous one and we use it in our implementation. However, we need to mention that online learning is not common compare to the batch learning. As our focus is on deep learning algorithms and almost in all cases, we use batch learning in those scenarios, our next step is implementing NN over encrypted data for the batch learning case.

For implementing batch learning, we follow the same structure as the plaintext by using three classes: *net*, *layer* and *neuron*. Each neuron stores input values, output values, weights and other required parameters in one ciphertext separately. As all operation on neurons are the same and our HE scheme supports SIMD, the implementation is similar to

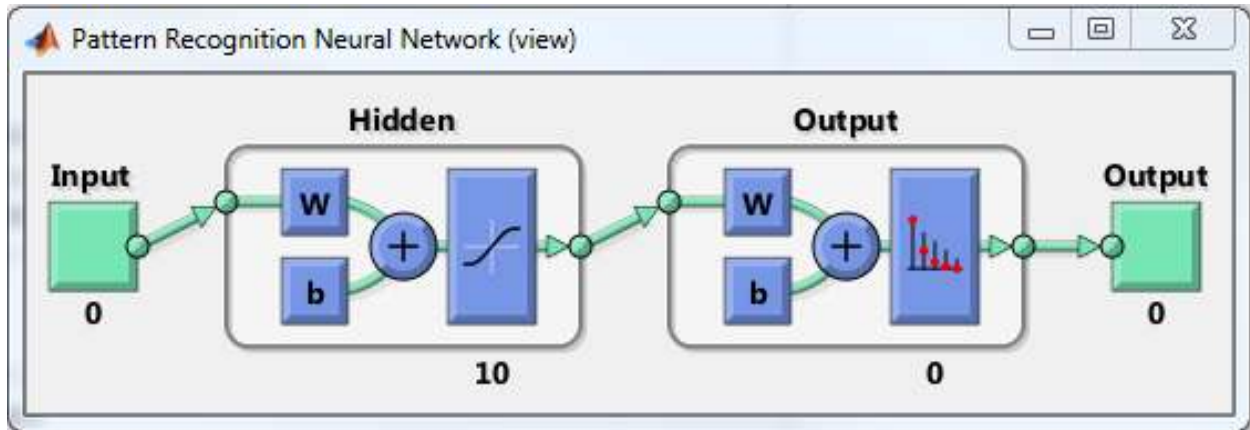


FIGURE 6.1. Neural Network

the plaintext format, we do not give details of implementation.

To measure the Performance of Neural Networks with Polynomial Approximations over plaintext, first, we train the models based on the encrypted data and measure the running time for the training process. Then, we use the trained model to classify encrypted instances and measure the accuracy of the model as well as the running time. In all experiments, for generating encryption schemes in the HELib, we set values for input parameters as $k = 80$ (security level which is equivalent to AES-128) and $s = 0$ (number of slots in the ciphertext) which allows the HELib to automatically pick the best number of slots. Another important value is L which is set to 20 in our experiments. We report the average running time from different rounds of algorithms and the standard deviation of running times is less than 3.0% in all cases.

For our experiments, we choose datasets from [53]. We choose different datasets with wide range of instances and feature values to have a better estimation of the performance of our solution. Table 6.2 includes the list of datasets and their specifications. At first step, we need to run experiments for the performance of our solution over plaintext before running them on encrypted data.

We trained models by using a polynomial as an activation function in the NN with two hidden layers, see 6.1. We used NN Toolbox [74] in our implementation for the plaintext format and datasets from Table 6.2. We compute polynomial approximations for the Sigmoid function

TABLE 6.2. Datasets (I, F and C represent Instances, Features and Classes respectively).

	Name	#I	# F	#C
1	Arcene	900	10000	2
2	Banknote Authentication	1372	4	2
3	Blood Transfusion Service Center	748	4	2
4	Breast Tissue	106	9	6
5	Cardiotocography (3 classes)	2126	21	3
6	Cardiotocography (10 classes)	2126	21	10
7	Climate Model Simulation Crashes	540	17	2
8	CNAE-9	1080	857	9
9	Connectionist Bench (Sonar, Mines vs. Rocks)	208	60	2
10	Connectionist Bench (Vowel Recognition)	528	10	2
11	Crab	200	6	2
12	Daphnet Freezing of Gait	1048576	10	2
13	Fertility	100	10	2
14	First-Order Theorem Proving	6118	51	6
15	Ovarian	216	100	2
16	Wine	178	13	3
17	Sensorless Drive Diagnosis	58509	49	11

based on four different parameters: degree, error, intervals and precision of coefficients. We change the degree of polynomials from 2 to 9. Our approximation method gets an error values as the input. we consider the following values for the error parameter: $\{0.1, 0.01, 0.001, 1e - 07, 1e - 11\}$. We also choose intervals as follows: $[-10^i, 10^i]$, $0 \leq i \leq 5$. The coefficients are small and we have to truncate them. For this reason, we truncate coefficients with different precisions: 10, 20, 30 and 40 digits. We also calculate the $\|\cdot\|_{2,\mu}$ for each polynomial for a more precise analysis. We generate 1920 polynomials (by assigning all the possible values to

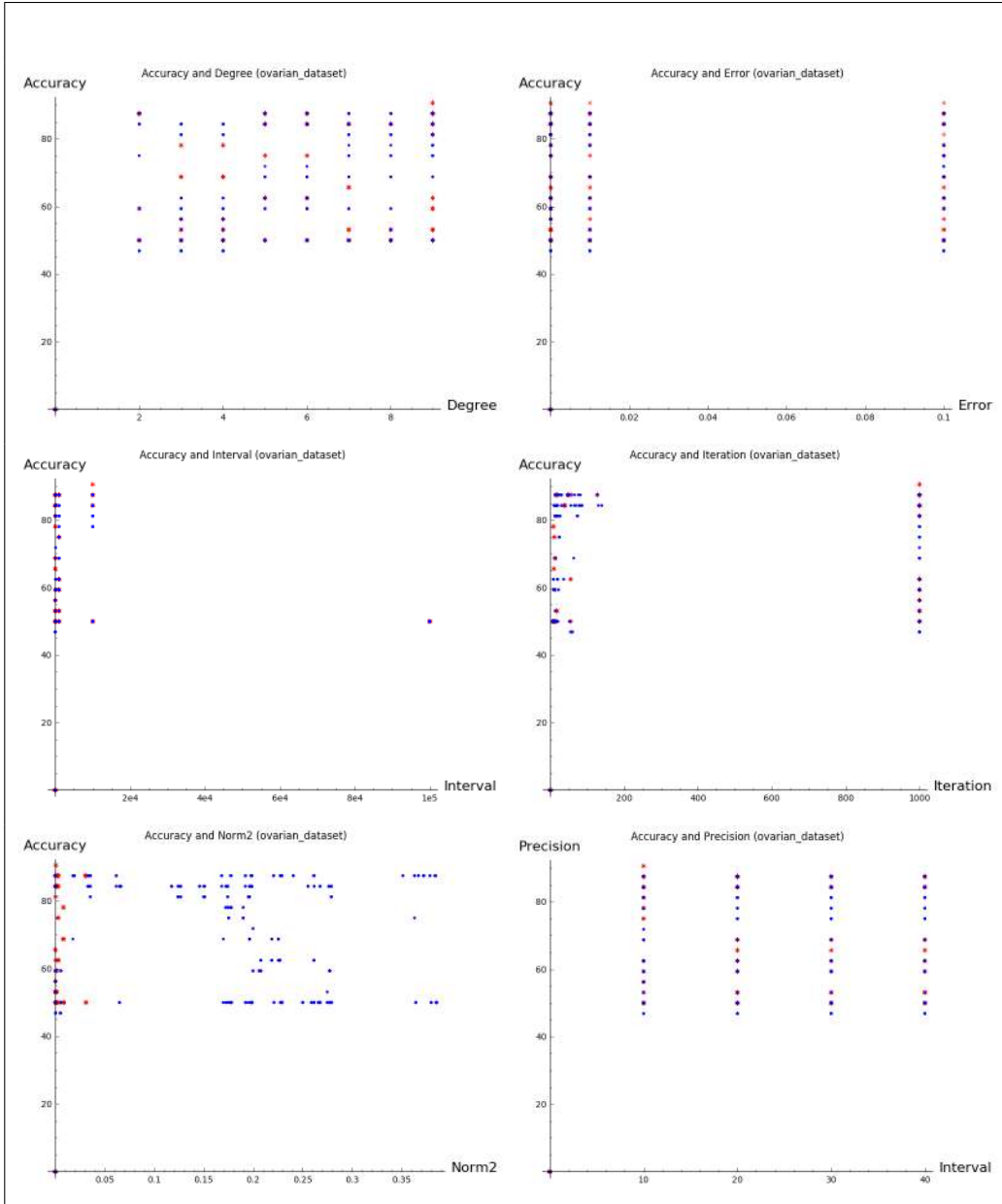


TABLE 6.3. Accuracy vs. other parameters for Ovarian Dataset. Blue and red are polynomials from two different measure functions.

all the parameters) and train the neural network using each polynomial. In Tables 6.3, 6.4 and 6.5, we draw the results for three of the datasets and all these polynomials. Based on our results, our polynomial approximation performance had the best accuracy in 10 out of 13 datasets. In wine_dataset, the accuracy of f_x is better than our accuracy and the polynomial approximation is 96.29%. The average feature values in wine_dataset is 5000. We generated

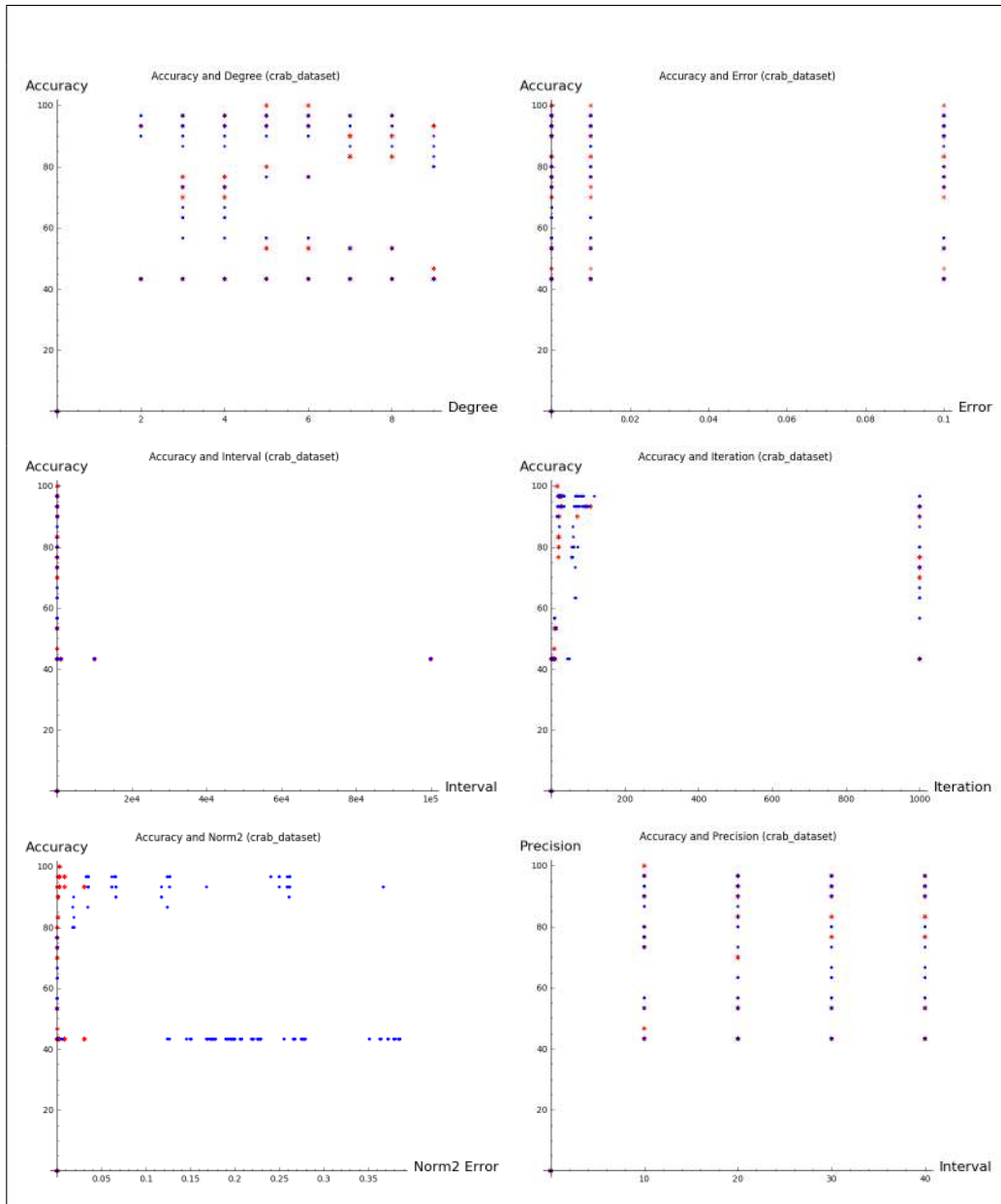


TABLE 6.4. Accuracy vs. other parameters for Crab Dataset. Blue and red are polynomials from two different measure functions.

a set of polynomial approximation on the interval $[-5000, 5000]$. We got a 100% accuracy by a polynomial from degree 2, precision 20 digits in 43 iterations. Based on our results, we concluded these remarks.

Remarks:

- The impact of approximation interval on the accuracy depends on feature values

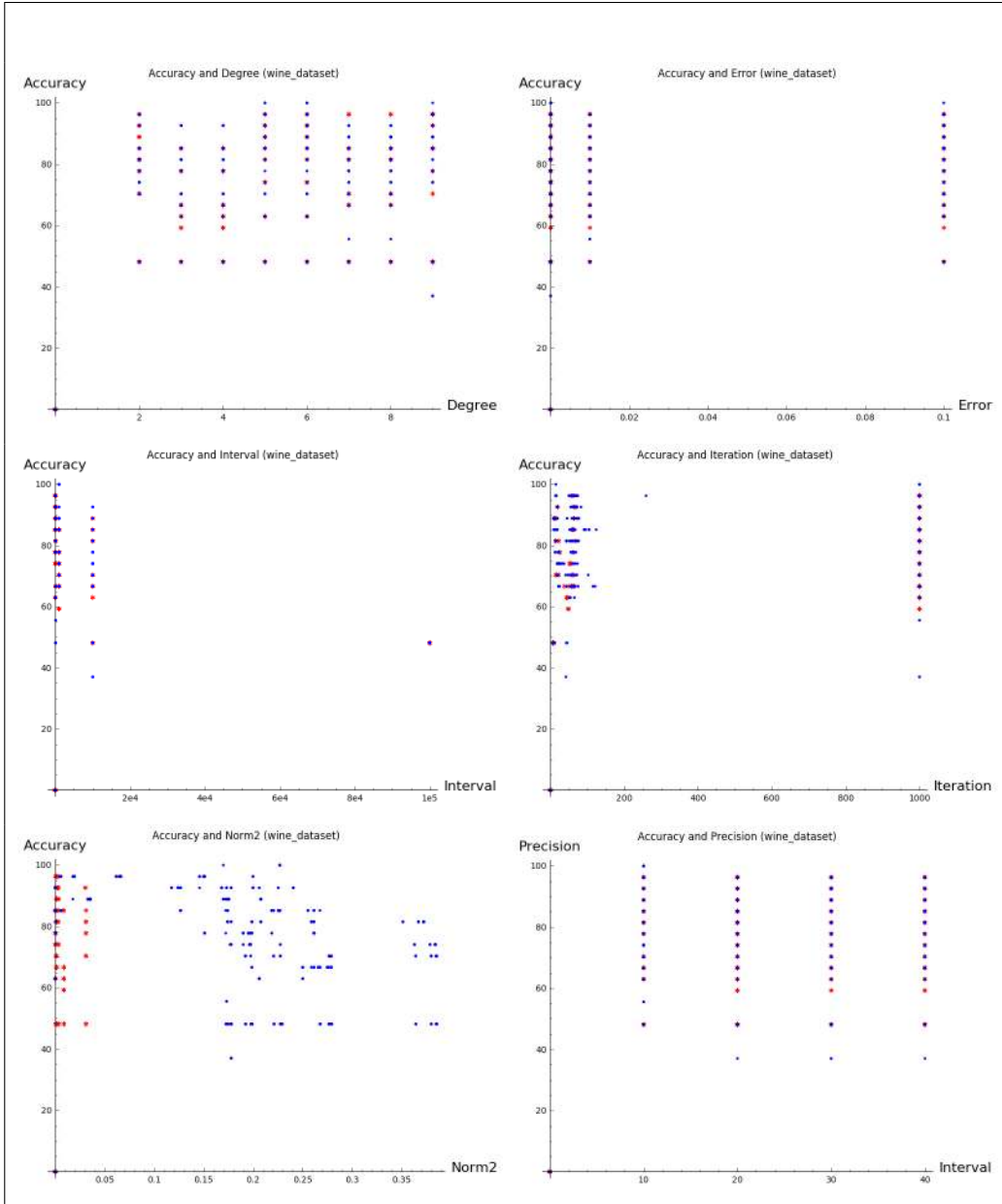


TABLE 6.5. Accuracy vs. other parameters for Wine Dataset. Blue and red are polynomials from two different measure functions.

in dataset. For example, in crab_dataset, the values are less than 100 and we got the best accuracy on interval $[-100, 100]$. For other intervals, the accuracy was reduced. For wine_dataset, feature values are less than around 5,000 and we got the best accuracy for interval $[-10^3, 10^3]$. For ovarian_dataset, feature values are more scattered, from 0 to 10^4 . We got 87.5% accuracy for different intervals from

$[-10^2, 10^2]$ to $[-10^4, 10^4]$. Therefore, we should use a suitable interval based on feature values.

- The error parameter does not show a direct impact on the accuracy and this parameter is not a good factor for choosing the polynomial approximation.
- The precision results were interesting. Our results showed that increasing the precision, in most cases, reduces the accuracy. We should truncate coefficients for a better accuracy.
- If the value of L^2 -norm increases, the accuracy decreases. Therefore, if we want a good accuracy, we should choose a polynomial that approximate the sigmoid function more accurate based on this norm.
- The degree of polynomial depends on the dataset. For example, in `crab_dataset` and `ovarian_dataset` the best accuracy was related to degree 5 while for `win_dataset`, the best accuracy was related to degree 9.
- We also recorded the number of iteration for training the model. It did not show a reasonable impact on the accuracy; however, the interesting point is that in some cases, the number of iteration decreases considerably in compare to using sigmoid function. In `crab_dataset`, the number of iteration was 16 for polynomial approximation and the number of iteration for sigmoid function was 45.

There is a shortcoming about this method of polynomial approximation. The set of polynomials for choosing the best polynomial is big and it's time consuming to choose the best polynomial. Based on mentioned remarks, we optimize the polynomial approximation solution which is more efficient and polynomial set includes less number of polynomials.

In order to find the interval that gives the best accuracy, we extract three values from each dataset. We first build a vector for each dataset where each item in the vector is the mean of the feature values. We train the neural network for each interval and the results show that the interval $[-\text{meanMean}, \text{meanMean}]$ has the best accuracy. As it can be seen in Table 6.6, in our experiments, neural networks with polynomial approximation as activation function achieve the best accuracy in all cases. Algorithm 1 shows the process of choosing

the best polynomial approximation based on the dataset and its features. Informally, we first extract the mean of values from features (meanMean) and then generate the polynomial approximation in the $[-\text{meanMean}, \text{meanMean}]$ interval. Based on our extensive experiments, this approach always lead to the best polynomial approximation [36]. To compare the

Input: Dataset, MaxDegree, ActivationFunction

Output: A set of Polynomials

MeanArray \leftarrow {};

Mean = 0;

PolynomialSet \leftarrow {};

for $i \leftarrow 1$ **to** #Features **do**

| *mean* \leftarrow mean of *Feature*[i];
 | *MeanArray* \leftarrow *MeanArray* \cup {*mean*};

end

Mean \leftarrow mean of *MeanArray*;

for $i \leftarrow 1$ **to** MaxDegree **do**

| *poly* \leftarrow Approximate the ActivationFunction in the interval $[-\text{Mean}, \text{Mean}]$
 | with precision 10 and degree i ;
 | *PolynomialSet* \leftarrow *PolynomialSet* \cup {*poly*};

end

return *PolynomialSet*;

Algorithm 1: Generating Polynomial Approximations

performance of our polynomial approximations with other studies, we use several activation functions for training: the Sigmoid function $f_1(x) = \frac{1}{1+e^{-x}}$, another variation of the Sigmoid function $f_2(x) = \frac{2}{1+e^{-4x}} - 1$, and the square function $f_3 = x^2$ which was proposed in [59] and [22]. We use these four different activation functions and if we choose the interval properly

based on the dataset, our model could achieve the best accuracy among all four activation functions. Our neural networks had two different activation functions in layer 2 and 3. We trained neural network in two different ways. First we used our four different activation functions in layer 2. For layer 3, we used softmax as an activation function, table 6.6 shows results.

In conclusion, our results show if we choose the interval properly based on our dataset, our model has the best accuracy among all four activation functions. **Error Rate of Neural Networks with Polynomial Approximations:** In the previous section, we showed that current activation functions in neural networks could be replaced with polynomial approximation without causing any accuracy loss on the model. To further confirm the suitability of the polynomial approximations, we also compare the error rate of neural networks with the original activation function (Sigmoid function) and with polynomial approximations as replacement. To provide a precise and fair comparison, we calculate the error rate over the same test set for all activation functions. This approach was previously used in [68] for comparing performances of different activation functions. We first split the dataset into training set and test set. We then train the neural network using the training set phase by phase. We start with the first 10% of the training set to build the model and calculate the error rate, then add another 10% of the training data and repeat the process until we have used the entire training data. The test set is kept the same during all the corresponding phases for all the activation functions. We report the results for four datasets, and as shown in Figure 6.2, the error rate in the case of the polynomials is almost the same as in the case of other activation functions. Now, we conclude with higher confidence that approximation of an activation function with a polynomial using our proposed method is a practical replacement for the currently used activation functions in neural networks.

For implementing neural network over encrypted data, due to limited operations over ciphertexts, we should consider the growing noise in ciphertexts during computations. As mentioned before, one technique to deal with the noise is bootstrapping. However, bootstrapping comes with heavy computation cost. To address this problem, we use an

TABLE 6.6. Performance of the neural network model based on $f_1(x) = \frac{1}{1+e^{-x}}$, $f_2(x) = \frac{2}{1+e^{-4x}} - 1$, $f_3 = x^2$ and the polynomial approximation in layer 2 only. a , i , d and n represent accuracy, number of iterations, polynomial degree and $\|\cdot\|_{2,\mu}$ respectively. Precision of the polynomial coefficient is 10, except Datasets 3 and 9 which are 20 and 15. We report the polynomial approximation with minimum degree. Refer to Table 6.2 for the list of datasets.

Nom	$f_1(x)$		$f_2(x)$		$f_3(x)$		Polynomial			
	$a(\%)$	i	$a(\%)$	i	$a(\%)$	i	$a(\%)$	d	i	n
1	86.67	18	56.67	7	30.00	1000	90.00	2	40	0.005
2	100	69	100	85	40.29	6	100	3	34	0.118
3	79.46	15	81.25	25	83.93	13	83.93	3	38	0.005
4	67.00	30	68.75	16	18.75	1000	75.00	2	23	0.001
5	98.58	30	89.34	43	79.31	1000	99.37	2	42	0.001
6	82.45	55	83.07	74	15.67	1000	89.97	2	45	0.001
7	95.37	21	91.36	11	87.65	11	96.30	2	23	0.001
8	96.30	179	66.05	107	7.40	1000	97.53	2	106	0.001
9	77.42	15	83.87	22	54.84	1000	93.55	3	27	0.000
10	86.67	33	75.84	66	12.08	52	96.30	2	23	0.001
11	100	45	100	31	66.66	1000	100	2	30	0.001
12	90.14	448	90.19	243	90.27	49	90.26	3	63	0.000
13	86.67	24	86.67	13	40.00	8	86.67	2	7	0.001
14	100	48	100	51	16.45	1000	100	2	48	0.001
15	93.75	25	96.87	1000	56.25	41	96.87	2	13	0.001
16	96.29	37	96.29	45	44.44	7	96.29	2	20	0.001
17	86.29	359	86.29	368	42.22	1000	89.87	2	373	0.001

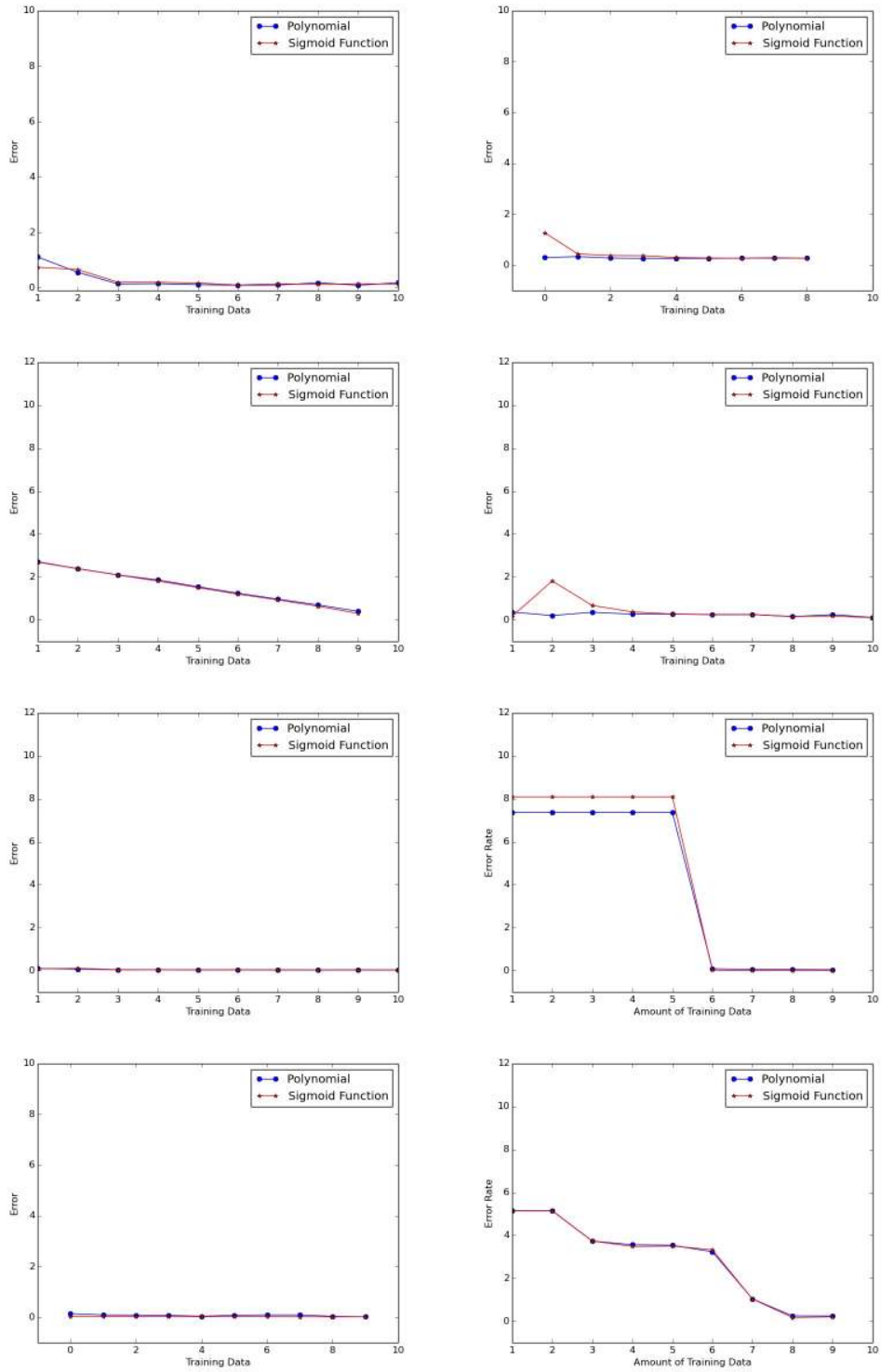


FIGURE 6.2. Comparing the error rate of the polynomial approximation and the Sigmoid function.

alternative approach where the server checks the level of noise in the ciphertext after each operation. If the noise level is lower than the threshold (i.e. two), the server sends the ciphertext to the client. The client decrypts and encrypts it again and sends the fresh ciphertext back to the server. Therefore, in each round of communication, one cipher refreshed by the client for reducing the noise.

The number of operations allowed on the ciphertext depends on the input value of L . If we set a small value for L , we need more communications between the client and the server whereas for higher values of L , less communications are required. However, by increasing the value of L , both the size of ciphertexts and the amount of transferred data increase.

We performed experiments for learning from datasets with different numbers of features and different number of classes. We measured the computation time on the server and the client. We also recorded the number of interactions and the amount of data transferred during the process. Based on our results shown in Table 6.7, communication between the client and the server is the most time consuming part of the process which accounts for about 90% of the training time. Therefore, we should choose our parameters in the encryption schemes in a way to have fewer communications. Two values that have impact on the number of interactions between the client and the server are L and p . Our experimental results show that instead of using large values for L which increases the size of ciphertext and time of computations, parameter p should be chosen properly. If we choose a proper prime number, we could use smaller values for L instead of larger values. The next experiment is finding the relation between number of hidden layers and performance of our solution. We implement neural networks over encrypted data with different numbers of hidden layers (1, 2, 3, 4 and 5). We implement the neural networks and use three different datasets: Crab, Fertility and Climate Model to perform the experiments. The results show that if we use batch learning, the performance of the training phase is acceptable. In the above experiments, we only run the experiment for one iteration. The HELib supports SIMD feature and the running time for one instance is the same as the time for a batch of instances. We also train the neural network with different sizes of batch as an input (282, 576, 1420, 3668 and 6144) and calculate

TABLE 6.7. Neural Network over Encrypted Data

Dataset	p	L	Data(s)	Training(s)	#C	Reduce Noise(s)
crab	1009	5	8	276.39s	2198	269
crab	1002263	10	25	652	170	470
crab	10000030013	15	48	1054	170	1011
crab	1000000034057	10	26	676	169	651
crab	1000000034057	15	47	1033	170	988
crab	$199 * 10^{11} + 85257$	20	86	4043	1860	3961
crab	19900000083773	20	86	3991	1860	3910
crab	1000000034057	17	55	2527	1860	2476
crab	1000000034057	19	89	4811	1860	4730
crab	19900000085041	20	94	4121	1860	4035
pendigits	1013329	5	8	358	1015	350
pendigits	1013401	10	28	1061	170	1037
pendigits	10000030471	15	57	4411	3550	4358
pendigits	10000024429	15	44	3554	170	3513
pendigits	$199 * 10^{11} + 85257$	20	99	9040	3550	8947

the running time for feed-forward, back-propagation and noise reduction for one, two, and five hidden layers. Figure 6.3 shows the result for one, two and five hidden layers respectively. As it can be seen, when we increase the size of the batch by 30 times, the running time only doubles. Our empirical results show that training over encrypted data is efficient when batch learning is used, and the network performance is acceptable. For example, we reach the training rate of 0.68 seconds per instance (6 features) for a batch size of 576 and two hidden layers (see Table 6.8). By increasing the batch size to 6144, the training rate decreases to 0.10 second per instance for a neural network with two hidden layers. It is worth mentioning that although larger batch sizes lead to faster training, they also increase the size of the network. Hence, a trade-off between the memory and running time is needed and we should

TABLE 6.8. Training Neural Network over Encrypted Data (batch input size of 576 and $L = 20$; standard deviation is less than 2.9%). HL, NC, #C and NR represent Hidden Layer(s), Network Creation, Number of Communications and the Noise Reduction time.

#HL(s)	NC(s)	Forward(s)	Back(s)	#C	NR(s)	Total Time(s)
Crab Dataset						
1	42.33	59.80	109.81	77	149.03	217.07
2	56.97	130.26	202.12	192	312.27	394.38
3	96.22	289.88	412.85	452	682.82	803.95
4	119.22	383.72	548.42	600	912.07	1056.5
5	125.63	407.64	582.13	640	969.58	1120.70
Fertility Dataset						
1	55.24s	65.30	121.83	77	160.17	247.89
2	72.42s	138.17	220.69	192	331.60	436.99
3	112.66s	305.51	440.44	452	718.36	864.60
4	131.20s	390.96	562.10	600	926.35	1089.86
5	140.52s	419.80	602.63	640	995.61	1168.47
Climate dataset						
1	92.73	86.69	170.85	77	210.98	357.40
2	11.34	161.42	273.74	192	388.69	553.61
3	112.66	333.65	497.64	452	780.07	989.26
4	165.77	416.37	638.44	600	1011.63	1228.38
5	175.25	430.35	843.84	640	1229.50	1456.7

choose a proper batch size based on the size of the dataset.

Now, we consider the classification phase and provide the performance of our solution for this phase. In the classification phase, we only run the feed-forward step and can perform the computation in parallel for a batch of unseen instances. Note that we implement

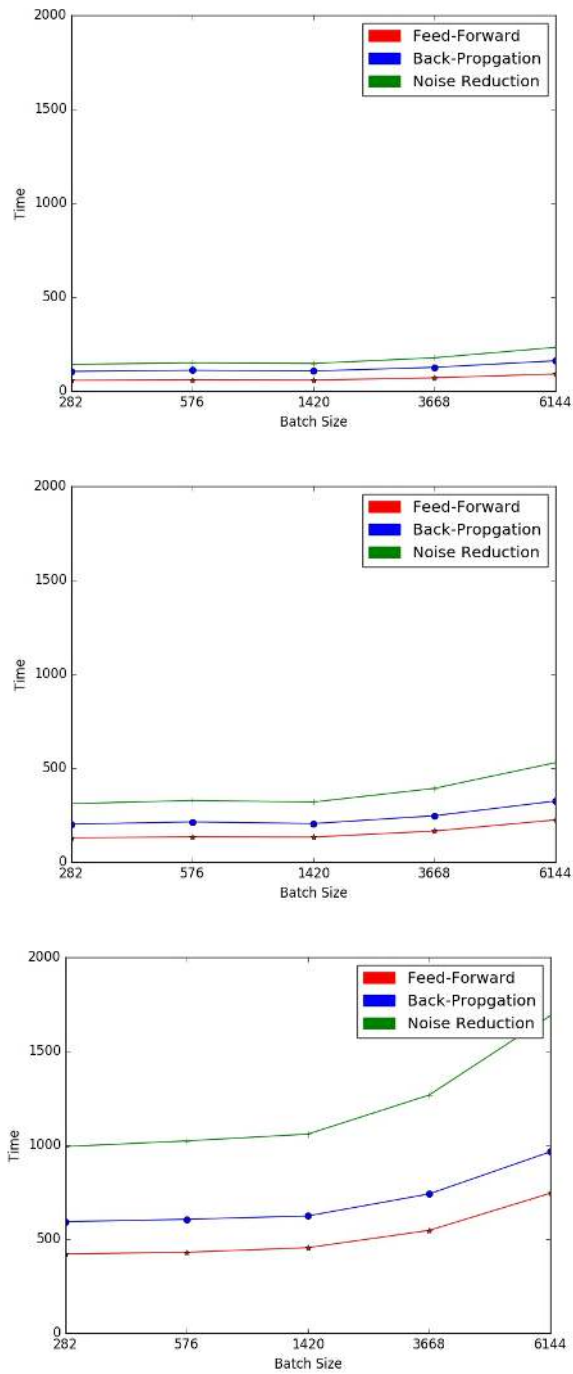


FIGURE 6.3. Neural network with one, two and five hidden layers

classification over encrypted data while the model is also encrypted. In this phase, noise growth is not a big challenge and the run-time is more important. The results are shown in Table 6.9. For the classification with a batch size of 576, we reach 0.04 second per instance

TABLE 6.9. Classification over Encrypted Data (batch size 576 and $L = 20$; standard deviation is less than 3%.) #HL, NR and #C represent for number of hidden layers, Noise Reduction times and number of communications.

#HL(s)	Classification(s)	#C	Noise Reduction(s)
Crab Dataset			
1	28.62	0	0
2	61.17	0	0
3	154.20	0	0
4	229.28	10	155.87
5	234.65	21	171.20
Fertility dataset			
1	35.94	0	0
2	68.86	0	0
3	160.49	0	0
4	236.72	10	152.34
5	255.70	21	169.84
Climate dataset			
1	55.85	0	0
2	61.78	0	0
3	178.25	0	0
4	249.54	10	149.10
5	269.39	21	166.89

for one hidden layer and 0.1 second per instance for two hidden layers. If we choose a batch of size of 6144, we reach 0.014 second per instance for one hidden layer and 0.036 second per instance for two hidden layers. The model is encrypted in all training and classification experiments and also we did not use any parallelization techniques in our implementation.

6.3. CNN Over Encrypted Data

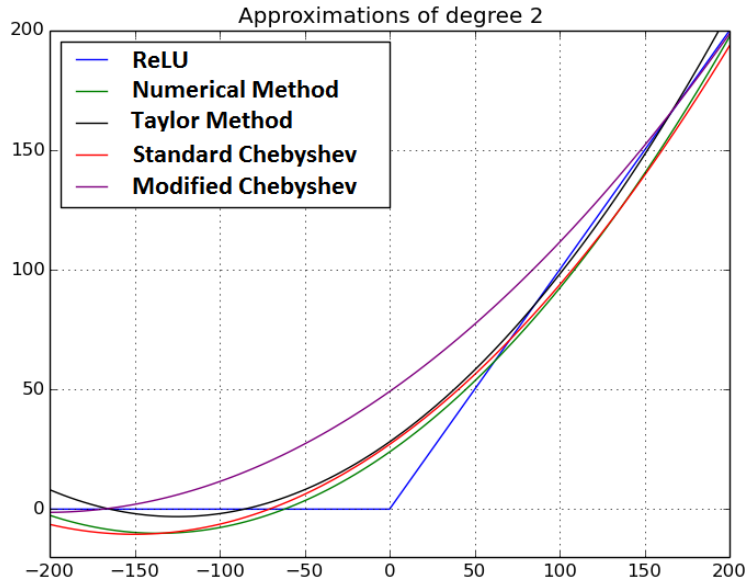
In this section, we consider Convolutional Neural Network over encrypted data and preserve the privacy of data for this case. By considering the performance of current HE schemes, number of epochs for training and size of data, training over encrypted data for CNN are not practical.

For neural networks, we use the Sigmoid function as the activation function. For CNNs, ReLU is more popular as the activation function. In the next section, we explain details of our solution for approximating the ReLU function.

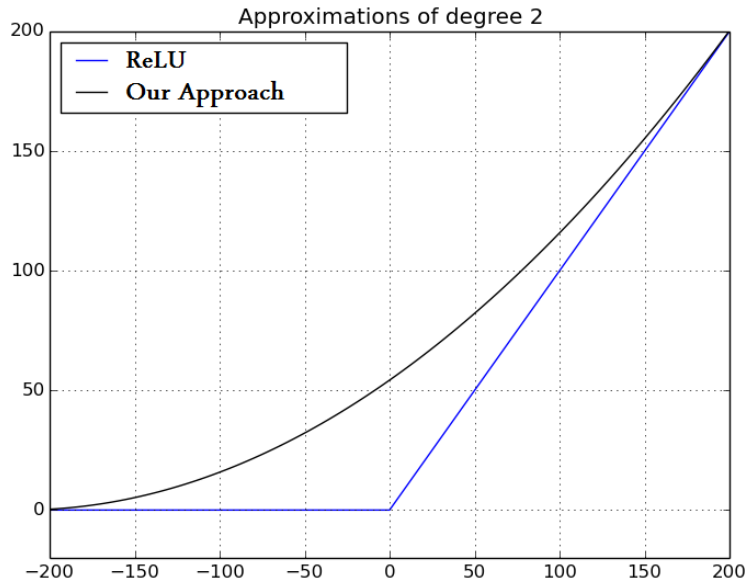
Several methods have been proposed in the literature for polynomial approximation including Taylor series, Chebyshev polynomials, etc. [93, 5]. We first try these approaches for finding the best approximation for the ReLU function and then present our proposed approach that provides better approximation than all these methods. We investigate the following methods for approximating the ReLU function. We use SAGE for implementing our polynomials approximation method, [21].

- (1) Numerical approximation
- (2) Taylor series
- (3) Standard Chebyshev polynomials
- (4) Modified Chebyshev polynomials
- (5) Our approach based on the derivative of ReLU function

The first method is **Numerical analysis 1**. In 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. 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. Another method is **Taylor series 2**. In this method, we use Taylor series [93, 5], a popular method for approximating functions. We use different degrees for approximating the ReLU function and trained the model using polynomials of



(A) Approximation of ReLU using different methods



(B) Approximation of ReLU based on our approach

FIGURE 6.4. Polynomial Approximation of ReLU

different degrees. Two main issues make this method inefficient. The first issue is the high degree of polynomial approximation and the second and more important issue is the interval of approximation.

We use another method which we call is **Standard Chebyshev polynomials** 3.

Chebyshev polynomials approximate a function in an interval instead of a small neighborhood of a point. In the standard Chebyshev polynomials, we use $d\mu = \frac{dx}{\sqrt{1-x^2}}$ as the standard norm and approximate the ReLU function. As shown in Table 6.10, the accuracy is still not a good performance in comparison with the original activation function. We modify the measure function based on the structure of the ReLU function which is done in the next method.

We adopt method 3 to reach better accuracy and call it **Modified Chebyshev polynomials** 4. In order to simulate the structure of ReLU function, we changed the standard norm used in Chebyshev polynomials to $e^{\left(\frac{-1}{1+e^{-5+(x)^2}}\right)}$ and were able to achieve much better results compared to all the previous methods. This 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. Since our goal is to approximate the ReLU function, 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.

We improve method 4 based on the **the derivative of ReLU function** 5. 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. The derivative of ReLU function is like a Step function and is non-differentiable in point 0. 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 6.10, this method achieves the best approximation of the ReLU function and we will use this method for approximation in this dissertation.

Figure 6.4a shows the structure of the functions generated by all the above approximations methods in comparison with the ReLU function whereas Figure 6.4b 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 6.4a, for some values less than zero, the structure of the polynomial goes up and

TABLE 6.10. 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%

down. This behavior has impact on the performance of the model that is trained based on these polynomials. However, in Figure 6.4b, 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, we only work with degree 2 and degree 3 polynomials. After finding the best polynomial approximation of ReLU function, we measure the performance of our solution for training adapted form of CNNs. Before this step, we provide a discussion about tune hyper parameters and its impact on the performance of our solution.

As we can see, different approximation methods result in widely different accuracy values and as expected, our proposed method based on the derivative of the ReLU achieves the best accuracy among all the methods. In the rest of this dissertation, we only use this method for approximation of the ReLU function, and whenever approximation method is mentioned, it refers to the last method.

All the above experiment runs without any parameter tuning to analyze the performance of the polynomials. In machine learning after finalizing the architecture of the network, we need to tune the hyper parameters. Different hyper parameters needed to be tuned, in our experiment, we only tune two parameters: learning rate and momentum. We tune two different CNNs with two convolutional layers and one fully connected layer (similar to the CNN proposed in [22]) for training over the MNIST dataset. We approximate the ReLU with

TABLE 6.11. Performance of the model for different approaches based on the model from [22]

ReLU	Tuned/ReLU	Polynomial	Tuned/Polynomial	Polynomial/Pretrain
98.87%	98.94%	98.35%	98.94%	98.94%

TABLE 6.12. Performance of a CNN with one Conv layer for different approaches.

ReLU	Tuned/ReLU	Polynomial	Tuned/Polynomial	Polynomial/Pretrain
98.32%	98.83%	98.18%	98.46%	98.82%

our polynomial approximation and replace it in the architecture of the CNN. We use grid search method for parameter tuning. In this method, we define one set for each parameters and then, train models for all possible pairs of these two parameters up to a constant number of epochs. As Table 6.11 shows, tuning parameters impacts on the performance of the model, specially for polynomial based architectures. Without parameter tuning, the accuracy of the model is 98.35% whereas the accuracy of the original model is 98.94%. After tuning parameter, the accuracy of the model based on the polynomial is the same as original model, 98.94%. For a better analysis, we run the same experiment with a different architecture. As shown in Table 6.12, the accuracy of the model is 98.32% and the tuned polynomial form of the network is 98.46%. We observe a gap between these two accuracies. The reason of this gap is related to the weight initialization values which has impact on the accuracy of the model.

Since we run each experiment different times, we observe that in some cases, the model does not train and we need to use a different initial weights for training a model. For avoiding this situation, we proposed to use knowledge transferability technique from machine learning which we explain in the next section.

Transfer learning is one of the popular **Transfer Learning** All the above solutions are based on training a model from scratch which its architecture is adopted within limitations of HE or SMC techniques. For preserving the privacy of data during training and classification steps, the basic approach which introduced in papers is training the network from scratch

after changes in the structure of the network for adopting the protocol within limitations of HE or SMC based solutions. However, changing the whole structure has a side effect which is computation cost. We introduce a more simple and efficient solution which is based on Knowledge Transferability in machine learning, [72]. The Transfer Learning in machine learning means that we can use pre-trained features in CNN for building a network for a different object. For example, suppose that we want to train a network based on the CIFAR-10, we can use extracted features from a model which is trained based on CIFAR-100 or ImageNet and only train a portion of network's parameters. We use a similar idea and use a pre-trained model based on a non-adoptive algorithm and adopt it within the limitations of HE or SMC for reducing the computation cost which comes from the training.

We can use transfer learning for both steps: training or classification. For the training problem, consider a client which owns a data set and its learning object is image recognition like CIFAR-10. Based on the proposed solutions, we need to build a network and trained in a private manner based on the client's data. Instead of using this approach, we can train a network based on a similar problem and extract features by taking advantage of transfer learning in machine learning. After training the network, we freeze convolutional layers and only train weights which related to changed layers. This approach has a much lower computation cost for both techniques: HE and SMC.

In the classification case, at first, we train a model based on non adoptive architectures and then replace the activation function, freeze some layers and only train weights related to changed layers.

We run experiment for evaluating this idea by using pre-trained models based on MNIST and adopting them within limitations of HE schemes. As shown in Tables 6.11 and 6.12, the performance of the model by using a pre-trained model is higher than training a model from scratch. For two different CNNs, the original accuracy values are 98.94% and 98.83%, using the same models and only train activation functions layers, the accuracy values are 98.84% and 98.82% relatively. From computation wise, the number of epochs for reaching these accuracy values decreases by 65% in compare of training all layers. Therefore, for

training a network over encrypted data, because of high cost of computations over encrypted data, using a pre-trained model is more efficient in compare of training from scratch.

Now, we analyze the relation between the approximation interval and approximation error with the model performance. First, we approximate the ReLU function over different intervals. We start from $[-10, 10]$ interval and go up to $[-590, 590]$ with a 10 unit increment in each step, calling it $\mathcal{P} = \{p_i\}_1^{58}$. The approximation interval must be symmetric. Then, we consider these architectures in different experiments:

- (1) A CNN with one convolutional layer, a polynomial as the activation function, an average pooling layer and a fully connected layer with 100 neurons with the ReLU as the activation function. We increase the number of filters from 1 to 100, calling it $\{Arc_i\}_1^{100}$.
- (2) A CNN with one convolutional layer, an average pooling layer, a polynomial as the activation function, and a fully connected layer with 100 neurons and ReLU as the activation function. We increase the number of filters in the convolutional layer from 1 to 100, calling it $\{Arc_i\}_1^{100}$.
- (3) A CNN with one convolutional layer, and activation function, an average pooling layer and a fully connected layer with 100 neurons and ReLU as the activation function. We increase the number of filters in the convolutional layer from 1 to 100, calling it $\{Arc_i\}_1^{100}$.
- (4) A CNN with one convolutional layer, a polynomial as the activation function, an average pooling layer and a fully connected layer with 100 neurons and polynomial as the activation function. We increase the number of filters in the convolutional layer from 1 to 100, calling it $\{Arc_i\}_1^{100}$.
- (5) A CNN with two convolutional layers, a polynomial as the activation function, an average pooling layer and a fully connected layer with 100 neurons and polynomial as the activation function. We increase the number of filters in convolutional layers from 1 to 100 (the first convolutional layer has i filters and the second one has $2 * i$), calling it $\{Arc_i\}_1^{100}$.

- (6) A CNN with one convolutional layer with 32 filters, a polynomial as the activation function, a fully connected layer. We increase the number of neurons in the hidden layer from 1 to 100, calling it $\{Arc_i\}_1^{100}$.
- (7) A neural network with only one hidden layer, an activation function, and increase the number of neurons from 1 to 100, calling it $\{Arc_i\}_1^{100}$.

Finally, we train a model for each polynomial and each architecture up to 50 epochs. At first step of experiment, we use different initial weights for each architecture. However, the standard deviations of accuracy values were high ($\geq 12\%$) due to randomized initial weights. To solve this problem, we run the experiment again using fixed initial weights. The new standard deviations are less than $\leq 3\%$ which makes our results more reliable. Running up to 50 epochs for each architecture is time consuming, for this reason and based on our results from Section 6.3, we decided to use another method which is more efficient in terms of the performance and the running time. First, we train a model up to 50 epochs based on the ReLU function. Then, we use the trained model's weights as the initial weights and train our models using polynomials, see Algorithm 2. The accuracy of these new models is higher and we only need to train up to 5 epochs to reach a stable performance. So, the running time is much lower compared to the training for 50 epochs.

For each case, we draw three different diagrams which we listed in here:

- Accuracy vs. Architecture. See Figures A.1, A.4, A.7, A.10, A.13, A.16, A.17 for cases 1 to 7 respectively.
- Accuracy vs. Error. See Figures A.19, A.2, A.5, A.8, A.11, A.14, A.20 for cases 1 to 7 respectively.
- Accuracy vs. Polynomials. See Figures A.3, A.6, A.9, A.12, A.15, A.18, A.21 for cases 1 to 7 respectively.

We define the error between the polynomial and the ReLU function based on the L^2 -norm. We concluded:

- Higher error values do not result in a lower accuracy. The reason is that the structure of the polynomial is similar to the ReLU function. This gives a good enough accuracy

Input: $\mathcal{P} = \{p_i\}_{i=1}^{i=58}$, MNIST

Output: \mathcal{A}

$\mathcal{M} \leftarrow \{\}, \mathcal{A} \leftarrow \{\};$

CreateModel *CreateModel*(*nomNeu*, *actFunc*):

$m \leftarrow$ neural network with *nomNeu* neurons in the hidden layer and the ReLU
 function;
 return M ;

for *nomNeus* $\leftarrow 1$ **to** 100 **do**

$m_{neurons} \leftarrow$ CreateModel(*nomNeu*, *ReLU*);
 Train model m_{nomNeu} up to 50 epochs;
 Save the model m_{nomNeu} ;
 Add m_{nomNeu} to the Array \mathcal{M} ;

end

for *polyIndex* $\leftarrow 1$ **to** 58 **do**

for *nomNeu* $\leftarrow 1$ **to** 100 **do**
 $m_{ReLU} \leftarrow$ *nomNeu*th of \mathcal{M} ;
 $m \leftarrow$ CreateModel(*nomNeu*, *p_{polyInd}*);
 Initial weights of $m \leftarrow m_{ReLU}$;
 Train model m up to 5 epochs;
 Add Accuracy of m to \mathcal{A} ;
 end

end

return \mathcal{A} ;

Algorithm 2: Training based on Polynomials

even though the error of the approximation increases.

- The accuracy of the model decreases by increasing the number of filters or neurons for a fixed interval. To improve the accuracy, we need to approximate the activation function over a larger interval. Note that for a larger approximation interval, the

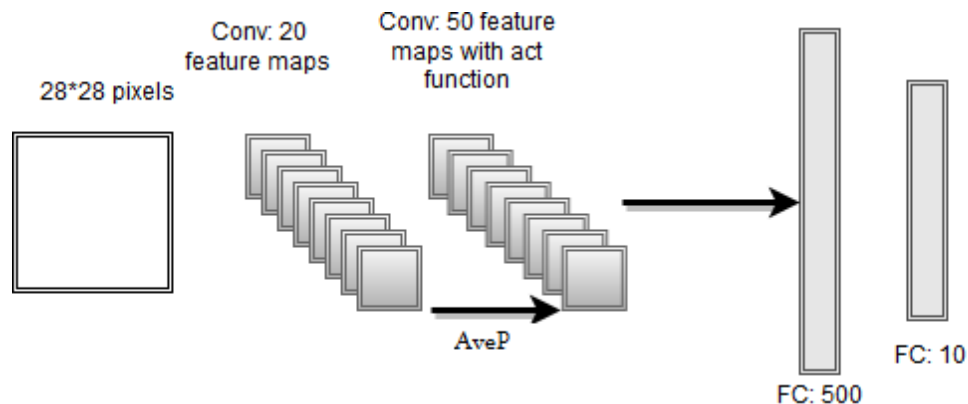


FIGURE 6.5. CNN Model 1 (AveP and BN stand for Average Pooling and Batch normalization).

model performance for a small number of neurons is almost the same as the original form of the activation function. Accuracy vs. error shows a similar trend. It clearly shows that as the number of neurons increases from 15 to 95, the accuracy over the smaller intervals is deteriorating.

- For each polynomial which approximates the activation function on a different intervals, by increasing the number of filters or neurons, the performance of the trained model increases which shows a similar trend like previous items.

In conclusion, for training the model, we need to choose the initial weights properly in order to get a higher performance and a lower running time. For choosing the best polynomial, we need to approximate the activation function over a large interval to cover all feature values during training the model.

In order to evaluate effectiveness of our polynomial approximation method, we use a CNN and the MNIST dataset [52] 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 includes the following components as shown in Figure 6.5.

- (1) Convolutional layer 1 has 20 filters and filter size 5×5 .

- (2) An average pooling layer with window size 2×2 .
- (3) Batch normalization layer.
- (4) Convolutional layer 2 has 50 filters and filter size 5×5 with polynomial as the activation function.
- (5) An average pooling layer with window size 2×2 .
- (6) Batch normalization layer.
- (7) Fully connected layer with 500 outputs and polynomial as the activation function.
- (8) Fully connected layer with 10 outputs

This CNN has similar architecture to the CNN used in [22] and the light CNN used in [15]. This will allow us to provide direct comparison with those works. We were able to achieve 98.94% accuracy while their accuracy was 97.95%. The main difference comes from the polynomial approximation method used; the approach in [15] 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 compare to Taylor Series.

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.

In Table 6.13, we provide the results for the accuracy of our method, the method from [15] and [22]. Our method works better than the method from [15] and close to the accuracy of [22]. We mention here that the structure of the method in [22] is more complex than our structure. We intend to train based on more convolutional layers, however before that, we need more analysis about the performance of our method in compare of ReLU function.

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 new CNN structure consists of:

- (1) Convolutional layer 1 has 30 filters and filter size 5×5 .
- (2) An average pooling layer with window size 2×2 .

TABLE 6.13. Comparing Performance of the Polynomial Approximation Method Papers.

Method	Accuracy (%)
Our solution	98.94
[15]	97.95
[22]	98.94

Degree	Accuracy
2	98.38%
4	98.32%
6	99.21%

TABLE 6.14. Performance of the trained CNN with Different Degree Polynomials.

- (3) Convolutional layer 2 has 15 filters and filter size 3×3 .
- (4) An average pooling layer with window size 2×2 .
- (5) Fully connected layer with 128 outputs and polynomial as the activation function.
- (6) Batch normalization layer.
- (7) Fully connected layer with 10 outputs

The accuracy of the model based on the 6.3 with polynomial of degree 2 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 2 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. The results are shown in Table 6.14, and as expected, the higher degree polynomials results in higher performance.

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 start with a

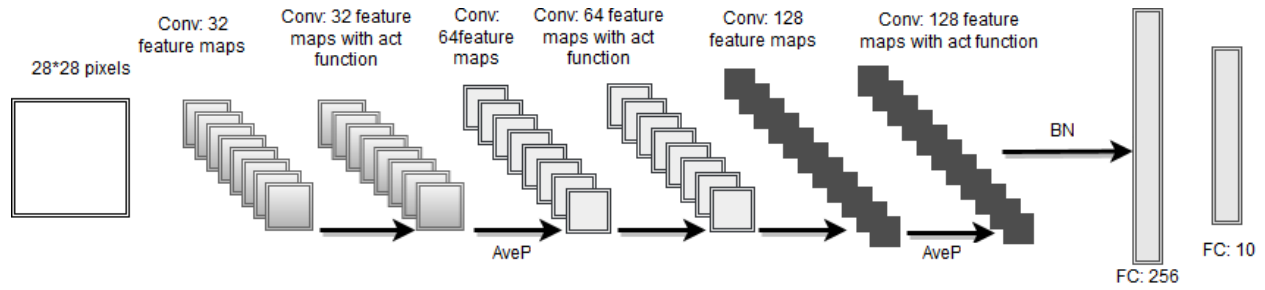


FIGURE 6.6. CNN Model 2 (AveP and BN stand for Average Pooling and Batch normalization).

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 6.3, the depth of the CNN in this case is 8 (number of multiplication used in implementation of the CNN which is an important limiting factor when we deal with homomorphic encryption). 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%.

- (1) Convolutional layer with 32 filters and filter size 3×3 .
- (2) Batch normalization layer.
- (3) Convolutional layer with 64 filters and filter size 3×3 .
- (4) An average pooling layer.
- (5) Batch normalization layer.
- (6) Convolutional layer with 128 filters and filter size 3×3 with polynomial.
- (7) An average pooling layer.
- (8) Batch normalization layer.
- (9) Dropout layer with probability 0.5.
- (10) Fully connected layer with 256 outputs with polynomial.
- (11) An average pooling layer.
- (12) Fully connected layer with 10 outputs.

Next, we add an activation function to the first convolutional layer and the accuracy increases to 99.16%. We add two more convolutional layers to the CNN and train the new model 6.3.

- (1) Convolutional layer with 32 filters and filter size 3×3 .

- (2) Batch normalization layer.
- (3) Convolutional layer with 32 filters and filter size 3×3 with polynomial.
- (4) Batch normalization layer.
- (5) Convolutional layer with 64 filters and filter size 3×3 .
- (6) An average pooling layer.
- (7) Batch normalization layer.
- (8) Convolutional layer with 64 filters and filter size 3×3 with polynomial.
- (9) An average pooling layer.
- (10) Batch normalization layer.
- (11) Convolutional layer with 128 filters and filter size 3×3 with polynomial.
- (12) An average pooling layer.
- (13) Batch normalization layer.
- (14) Dropout layer with probability 0.5.
- (15) Fully connected layer with 256 outputs with polynomial.
- (16) Fully connected layer with 10 outputs.

The accuracy of the trained model based on 6.3 increases to 99.29%, while the depth goes up to 14. Then, we add an activation layer after layer 6 and another activation function after layer 8, the accuracy changes to 99.28%. We go one step further and add an activation function after each convolutional layer. The accuracy for this CNN is 99.32%. The depth of the CNN in this case is 19. This number of multiplication is high for HE schemes and makes the classification over encrypted data inefficient. Finally, we try the same structure as the deep CNN used in [15] shown in Figure 6.6, to have the same depth as the deep CNN used in [15].

- (1) Convolutional layer with 32 filters and filter size 3×3 .
- (2) Batch normalization layer.
- (3) Convolutional layer with 32 filters and filter size 3×3 with polynomial.
- (4) An average pooling layer.
- (5) Batch normalization layer.

- (6) Convolutional layer with 64 filters and filter size 3×3 with polynomial.
- (7) Batch normalization layer.
- (8) Convolutional layer with 64 filters and filter size 3×3 with polynomial.
- (9) Batch normalization layer.
- (10) Convolutional layer with 128 filters and filter size 3×3 .
- (11) An average pooling layer.
- (12) Batch normalization layer.
- (13) Convolutional layer with 128 filters and filter size 3×3 with polynomial.
- (14) Fully connected layer with 256 outputs with polynomial.
- (15) Fully connected layer with 10 outputs.

The accuracy of this model, shown in Figure 6.6, is 99.52% which is very close to the accuracy of the same CNN that uses the ReLU function, 99.56%. This is achieved with the depth 14. Our accuracy is higher than both methods in [22] (98.95%) and [15] (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 6.6) and only change the activation functions to Sigmoid and Tanh instead of the ReLU function. The results shown in Table 6.15.

TABLE 6.15. 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%

6.4. Comparison with the State-of-the-Art

In this section, we compare the performance of our solution with previous solutions based on SMC and HE.

At first we consider SMB-based technique and evaluate our approximation polynomial for training and classification scenarios compare to this technique. Our aim is to show that our approximation method generates low degree polynomials with enough good performance which make them a good replacement for most common activation functions for both SMC based and HE based solutions.

Although our approach is based on homomorphic encryption, it does not use bootstrapping due to huge computation cost and instead requires some communications between the client and the server. One may argue that why not use secure multi-party computation (SMC) instead of homomorphic encryption. Although our approach requires some communications, it has several advantages over SMC-based approaches as discussed below.

While privacy-preserving machine learning based on secure multi-party computation techniques have been studied in the literature, those approaches focus on traditional machine learning algorithms such as linear regression [91], decision trees [11, 95, 4] and linear classifiers [30, 11]. There are a few recent works that focus on neural networks and aim to develop privacy-preserving training [65] and prediction [79, 58, 77].

To the best of our knowledge, secureml [65] is the only work based on SMC techniques that considers both training and classification phases. The others only focus on classification phase. In the following, we provide a detail comparison of our proposed approach with these efforts.

The recent work of secureml and Zhang [65] aims to develop privacy-preserving training and classification of neural networks using SMC techniques. In their proposed approach which uses HE as one building block of SMC, a data owner shares the data with two servers and the two servers run the machine learning algorithm using two-party computation (2PC) technique. They focus on three different algorithms: linear regression, logistic regression and neural network. Their protocol is divided into two phases: online and offline. In the offline phase, they use oblivious transfer (OT) for generating multiplication triplets and in the online phase, they securely compute the activation function in logistic regression and neural network training. We only look at their neural network algorithm which is closely related

TABLE 6.16. Comparing the number of communications in our approach and SMC-based approach for different number of hidden layers and different batch sizes for the input (HL represents for Hidden Layers).

	1 HL		2 HLs		3 HLs		4 HLs		5 HLs	
Batch Size	Ours	[65]	Ours	[65]	Ours	[65]	Ours	[65]	Ours	[65]
282	77	6042	192	12216	452	29940	600	77148	640	129144
576	77	7507	192	15151	452	37095	600	95543	640	159919
1420	77	10452	192	21036	452	51420	600	132348	640	221484
3668	77	12778	192	25714	452	62850	600	161762	640	270706
6144	77	13924	192	28036	452	68548	600	176452	640	295300

to our work. The SMC-based approach of [65] implements two solutions based on linearly homomorphic encryption (LHE) and oblivious transfer (OT) and results are provided for both LAN and WAN settings. In their LHE-based solution, the running time for LAN and WAN is almost the same. However, the accuracy of the model is very low. To improve the accuracy, higher degree polynomials should be used which will introduce significant computation and communication overhead. Alternatively, they present an OT-based solution which achieves better accuracy. However, there is a huge gap between communication overhead of LAN and WAN settings (e.g., 0.86s for LAN and 43.2s for WAN, see Table 2 from [65]). As mentioned by the authors, the reason for this gap is the communication overhead since the number of communications increases drastically as the neural networks become more complex. Since we assume a client-server model, the WAN setting is more important and we use the number of communications as one of the metrics for comparison.

To provide a comparison for the number of communications, we look at the approach proposed in [65]. The number of communications is $O(t \cdot \sum_{i=1}^m (|B| \cdot d_{i-1} + d_{i-1} \cdot d_i))$ where t , m , $|B|$ and d_i are the number of iterations, number of layers, size of the batch input and the number of neurons in layer i relatively (refer to section III-D of [65]). It is not as straightforward in our approach to calculate the number of communications, because the

number of communications depends on the level of the noise in the ciphertext. However, it is possible to estimate the number of communications based on the architecture of the neural network.

In our approach, when the level of the noise gets close to the threshold, the server sends the ciphertext to the client and receives the refreshed ciphertext. Therefore, one communication needed between client and server for reducing noise in each ciphertext. This ciphertext could be a weight or a variable in the process of the algorithm. Suppose the network has l layers and d_i is the number of neurons in layer i . D is the degree of the polynomial used as the activation function and based on the input parameters for the HE scheme, the number of allowed multiplication is \mathcal{L} . Each iteration (one feed-forward and one back-propagation) includes $(l - 1) + (2D - 1)(l - 2)$ multiplications (call it \mathcal{M}). Therefore, $\frac{\mathcal{M}}{\mathcal{L}}$ is the number of communications for each variable to reduce the noise in one iteration. On the other hand, the number of weights in a network is $\sum_{i=1}^{l-1} d_i \cdot d_{i+1}$ (call it \mathcal{W}). We can estimate the total number of communications for t iterations with this equation: $t \cdot \frac{\mathcal{M}}{\mathcal{L}} \cdot \mathcal{W}$. In our solution, the number of communications is independent of the batch size whereas in [65], the size of the batch contributes to the number of communications. Table 6.16 shows the number of communications for one iteration of our approach and SMC-based approach for different number of hidden layers and different batch sizes. As shown, there is a huge difference in the number of communications in our solution and the proposed solution in [65].

Table 6.16 shows the number of communications for one iteration of our approach and SMC-based approach for different number of hidden layers and different batch sizes. As shown, there is a huge difference in the number of communications in our solution and the proposed solution in [65].

In addition to the the number of communications, we compare the amount of data transferred between the client and the server. When we have simpler neural networks, SMC-based approach has smaller communication overhead but our approach is much better when the neural network becomes more complex. For example, for a neural network with 2 hidden layers and batch size of 512, the amount of data transferred in [65] is 12MB whereas

for our approach, it is 33MB. However, for a neural network with 5 hidden layers and batch size of 512, the amount of data in [65] is 159MB while in our solution, it is 155MB. If we use larger batch sizes (i.e., 3668), our approach outperforms [65] in neural networks with 4 hidden layers and higher (151 MB for our approach v.s. 161 MB for [65]). These results show that [65] has much higher communication overhead compared to our approach and the main reason is that the number of communications in [65] grows significantly as the neural networks become more complex.

We implemented a neural network similar to the one used in [65] and performed experiment using the encrypted MNIST dataset. The results shown in Table 6.17 for one iteration of training and indicate that the communication has a huge overhead on the running time. Besides the number of communications, the size of data transferred in each communication impacts the running time. The size of data in each communication is the same as the size of a ciphertext. Although we can decrease the size of the ciphertext by decreasing the value of L , decreasing the value of L results in higher number of communications. Table 6.18 shows the result for $L = \{5, 10, 20\}$ and different topologies of the neural network for one iteration of training. As it can be seen, although we have higher number of communications for $L = 5$, the whole run time is less than $L = 10$. The reason is that the size of the ciphertext is larger when $L = 10$. To improve the run time, we can run the code in parallel or using Graphics Processing Unit (GPU) which is part of our future work. Note that we did not use any parallelization techniques and the performance can be further improved if we use parallel implementation.

The main advantage of our approach over SMC-based solutions is the communication overhead which consists of the number of communications between the client and the server and the amount of data transferred in the communications. In SMC protocols, we need a number of communications between the client and the server for each operation whereas in our approach, the server does not need to communicate with the client unless the amount of noise reaches a threshold. If the same neural network is implemented using SMC protocols, the number of communications is much higher compared with the HE based protocols.

TABLE 6.17. Training Neural Network over the Encrypted MNIST Data (batch input size of 192 and $L = 20$).

#HL(s)	Forward(s)	Back(s)	#C	Noise Reduction(s)	Total Time(s)
2	2884.31	6301.26	6740	8939.40	10476.29

TABLE 6.18. Training different Neural Network architectures over the Encrypted Data ($L = \{5, 10, 20\}$).

Topology	L	Forward(s)	Back(s)	N/R(s)	#C	Total Time(s)
[6 1 1 10]	5	9	24	34	164	37
[12 2 2 10]	5	17	46	63	314	67
[24 4 4 10]	5	44	113	157	788	165
[48 8 8 10]	5	134	331	465	2432	488
[93 16 16 10]	5	654	1357	2012	8504	2086
[6 1 1 10]	10	23	40	63	106	71
[12 2 2 10]	10	42	71	113	190	125
[24 4 4 10]	10	105	160	265	448	289
[48 8 8 10]	10	315	464	779	1324	842
[93 16 16 10]	10	1157	1631	2788	4516	2999
[6 1 1 10]	20	28	60	86	34	118
[12 2 2 10]	20	58	105	158	62	204
[24 4 4 10]	20	127	214	322	130	419
[48 8 8 10]	20	608	882	1339	314	1860
[93 16 16 10]	20	1527	1927	3020	874	4469

In terms of the accuracy, the model trained by our proposed approach can reach 95.15% using polynomial as the activation function over plain-text MNIST dataset while the model trained by the protocol in [65] can only reach 93.4% accuracy. The running time for privacy preserving prediction in our model is better compared with the solution in [65].

Another advantage of our approach is the polynomial approximation method we used for approximating the Sigmoid function. In our approach, the degree of polynomial is 2 or 3 in all cases which allows the implementation to be efficient. However, in [65], the performance of the model drops considerably when low degree polynomials are used and it requires a high degree polynomial (i.e. 10) to achieve a reasonable accuracy (refer to Table 1 in [65]). As discussed before, computing the high degree polynomial over encrypted data is very inefficient.

The approach in [65] needs two different servers in addition to the client and assumes that the servers do not collude with each other. This is a weaker security assumption compared to ours and if the servers collude with each other, the privacy of the client’s data is not preserved anymore. In addition, to improve the performance of their model, they proposed a client aided approach where the client operates part of the computation. In this scenario, if a subset of clients collude with one of the servers, their approach leaks information about the data from honest clients (see [65], Section V).

Darvish et al. [79] present DeepSecure that enables distributed clients (data owners) and cloud servers to jointly evaluate a deep learning network on their private assets. It uses Yao’s Garbled Circuit (GC) protocol to securely perform deep learning. Liu et al. [58] propose a privacy-preserving classification protocol based on the Oblivious Transfer (OT) and HE schemes. Riazi et al. [77] propose a solution based on Garbled Circuits (GC) and Additive Secret Sharing (SS). They all perform experiments on the MNIST dataset and report the results.

The accuracy of the models trained with the MNIST dataset in these approaches is 98.95%, 98.95%, and 99% respectively whereas in our approach, the accuracy is 98.95%. Table 6.22 shows details of comparison between our solution and previous solutions.

In addition to having a much better performance, our approach has several advantages over SMC-based approaches. Unlike SMC-based approaches, the structure of the client does not need to be changed in our approach when the architecture of the neural network is changed. The only operations in the client side are encryption and decryption and the

server can perform different machine learning algorithms with the same client. Another advantage of our approach is that it preserves the privacy of the model compared with the SMC-based solutions. In SMC-based solutions, the client participates in the computations and information about the model could possibly leak to the client. For example, the client can learn information such as the number of layers in the neural network, the structure of each layer and the activation functions. Although there are some solutions like adding more dummy layers to the model to mitigate this issue, these will add to the already large computation cost. On the other hand, in our approach, the only operations in the client side are encryption and decryption (and possibly division in the case of networks with high number of layers). Hence, the client cannot learn any information about the structure of the neural network.

Now, we compare our results with the state-of-the-art privacy-preserving neural networks based on homomorphic encryption. CryptoNets [22] is the closest work to ours where it aims to implement neural networks classification using HE.

The main difference between our approach and CryptoNets is that our approach performs both training and classification phases over encrypted data whereas CryptoNets only considers the classification phase which is much simpler compared to the training. The authors of CryptoNets assume that the model is trained based on plain data and then use this model for classifying encrypted instances. In our approach, the model is built using encrypted data and the final model is encrypted. When the model is trained using plain data, the final model is also in plain format and works much faster than an encrypted model.

In order to provide a fair comparison, we change our implementation discussed earlier and follow the same process as the CryptoNets: train a model based on the plain data and then classify encrypted instances. In CryptoNets, the authors implement a Convolutional Neural Network (CNN) with two convolutional layers and two fully connected layers. They replace the Max Pooling layer with Average Pooling layer and replace the activation function (i.e., Sigmoid function) with the square function. We design a CNN with the same depth (the number of multiplication required) and similar structure. we use an average pooling layer

similar to CryptoNets.

We train the same model with ReLU as the activation function instead of the Sigmoid function. In this case, the accuracy of the trained model is 99.02% and when we replace the activation function (i.e., RELU) with polynomial approximation, the accuracy is 98.94% whereas the accuracy of the similar model in CryptoNets is 98.94%. When we used more complex neural network, we were able to achieve 99.25% accuracy which is close to the accuracy of the same model that uses the ReLU function (99.56%) and significant improvement over CryptoNets. The accuracy of this model with the Sigmoid function as activation function is 99.10% and when we replace the Sigmoid function with the polynomial approximation, we achieve 99.15% accuracy. For the Tanh function as the activation function, the original accuracy is 97.27% and the accuracy with polynomial approximation is 98.15%.

In terms of throughput, our approach significantly outperforms CryptoNets. When we classify ciphertext with the batch size of 8192 (the same batch size used in CryptoNets), our approach can make 163840 predictions per hour whereas CryptoNets makes only 51739 predictions per hour. The running time for classifying this batch input size is 570 for CryptoNets whereas in our approach, the running time is 320 seconds. The size of the data transferred from the client to the server in CryptoNets is 570Mb and in our approach is 336.7MB. 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.

To further show applicability of our proposed approach for more complicated network architectures, we use the CIFAR-10 [46] which is one of the widely used benchmark datasets for deep learning, to train a neural network and implement it over encrypted data. The CIFAR-10 dataset consists of 60000 32×32 colour images categorized in 10 classes, with 6000 images per class. There are 50000 training and 10000 test images. We trained convolutional neural networks using the CIFAR-10 dataset and achieved 91.5% accuracy with polynomials as the activation functions whereas the accuracy with the original activation function (ReLU) is 94.2%. As expected, the CIFAR-10 is much slower compared to the MNIST since both

TABLE 6.19. Breakdown of Running Time of CNN Model 6.5) over Encrypted MNIST Dataset.

Layer	Time (seconds)
Conv layer(20 filters)	13.078
Average Pooling Layer	7.630
Conv layer (50 filters)	77.642
Activation layer	9.763
Average Pooling Layer	6.543
2 Fully Connected (500 and 10 neurons)	34.32

TABLE 6.20. The performance of our solution(s) compare to [22]

Layer	Our Approach	CryptoNets [22]
Encryption	62.8	122
Communication	320	570
Decryption	3.9	5

the dataset and the CNN are much more complex. We need to note that for CIFAR-10, the depth of algorithm is high and we cannot implement it over encrypted data.

6.5. Conclusion

Table 6.21 shows the comparison all proposed solutions from privacy wise. We provide details about each solution addresses the privacy of which component.

Table 6.22 provides details of all solutions, the learning algorithm used in each one, communication and computation cost, accuracy and the methodology used.

Solution	[65, 90]	[79, 58, 16]	[77]	[38]	[22, 34]	[42, 19]	[96]	[69]	[1]	[82]	[73]
Privacy Training	✓	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓
Privacy Testing	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Privacy Network Parameters	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Privacy Network Architecture	✗	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗
Privacy Training Data	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Privacy Test Data	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Batch Input	✓	✗	✓	✗	✓	✓	✓	✓	✓	✓	✓
Distributed Data	✓	✓	✓	✗	✗	✗	✗	✗	✓	✓	✗
Client update	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓
Server update	✓	✓	✓	✓	✗	✗	✗	✓	✗	✗	✗
Client/Server	✗	✗	✗	✗	✓	✓	✓	✓	✓	✗	✗

TABLE 6.21. Proposed solutions vs. Privacy concerns.

TABLE 6.22. Comparison with the state-of-the-art Solutions

Dataset	Criteria	Ours	[22]	[79]*	[65]*	[16]*	[58]*	[77]*
MNIST	Accuracy	99.25%	99%	99%	93.4%	99%	99%	99%
	Time (s)	320	570	6328	759	41779	76349	12095
	Data Transfer	436.7MB	595.5MB	6479GB	N/A	4104GB	5386GB	86GB
	# of P/H	163840	51739	379	738	706	386	1333

*: The values are extrapolated.

CHAPTER 7

CONCLUSION AND FUTURE WORK

In this chapter, we wrap up this dissertation via two sections. In first section, we briefly review this dissertation. In the second section, we suggest future work for privacy preserving MLaaS.

7.1. Conclusion

This dissertation includes these phases:

- Review current solutions for data privacy in machine learning:

In this step we reviewed the current solution for preserving the privacy of data in machine learning as a service. As we mentioned, three main ideas introduced: differential privacy, secure multiparty computation and homomorphic encryption. Each solution has pros and cons. By analyzing each solution and the security model for each solution, we decided to use HE schemes for solving this problem.

- Analyzing the limitations of HE schemes:

In this phase we analyzed the limitations of HE schemes. For running an algorithm, there are some challenges that we should solve, for example, lack of division operation and complex functions such as exponential function. Basically, HE schemes support only two operations: addition and multiplication, see [12].

- Polynomial approximation for approximating infinite differentiable functions like sigmoid function:

we need to approximate such the activation function with polynomials. We developed an approximation method based on Chebyshev polynomials and adopted it to improve the approximation method.

- Adopting neural network within limitation of HE schemes:

In this phase we considered all limitations of HE schemes and designed an adopted form of neural networks over encrypted data.

- Implementing Neural Networks within limitations of HE schemes:

we implement neural networks over encrypted data in a client-server architecture. We measured the performance of implementation and compared it with similar solution based on SMC technique.

- Polynomial approximation for approximating not infinitely differentiable functions like ReLU function:

In this phase we changed the approach for approximating the ReLU function. our primary results show that approximating ReLU function with polynomials does not have high performance.

- Adopting Convolutional Neural Networks within limitations of HE schemes:

In this phase we considered all limitations of HE schemes and used the approximation of ReLU with high performance.

- Implementing Convolutional Neural Network within limitations of HE schemes:

we implemented CNN over encrypted data and run it for the MNIST dataset. Our results show that the performance of the model is acceptable, however, there are some new challenges that we found, for example, running a deep CNN over encrypted data is not practical with HELib.

- Improving the performance of CNN over encrypted data:

In this phase we found techniques from machine learning to improve the performance of the algorithm over encrypted data.

At the end, we concluded that training Deep Neural Networks over encrypted data based on the performance of current HE libraries is not practical. Although, our experiments over plaintext show that we can train a model which is adapted within the limitations of HE schemes, due to noise growth during computation, we cannot train models based on the encrypted data. However, for shallower networks, such the training is possible.

Regarding the classification of encrypted data, our results show that we are able to do it for enough deep neural networks and convolutional neural networks. Our results also show that solutions based on HE schemes are more efficient compared to SMC-based solutions in terms of communication cost.

7.2. Future Work

The first problem is finding the best polynomial with a deterministic algorithm. Current methods employ heuristic algorithms for finding the best polynomial (with the highest accuracy) from a set of polynomials. We implemented the code for the MNIST and extracted the results. The final algorithm gets architecture of the network, error of the polynomials and some other parameters

and outputs the best polynomial without need for training. The second problem is about training based on polynomials as the activation functions. The question is **Which problems are covered with polynomials in ML domain?** Using polynomials as the activation proposed in papers, see [59], however, they are no not as general as current activation functions and using them has some limitations.

The next step is supporting distributed data (vertically/horizontally). The current solution is for one client and one server. For training based on distributed data among clients, we need to change the HE scheme. We need a Multi-Key homomorphic encryption. Theoretically, all HE schemes are Multi-Key HE schemes, however, no implementation has yet been developed (to the best of our knowledge) for this kind of HE schemes and only theoretical background has been explored. The last proposed future work is about Hybrid solution based on HE/SMC and HE/DP. Such a hybrid solution would be an alternative to solution to our approach. It is out scope of this work, however, it is an interesting approach which has already been explored in some papers, c.f. [38] and [77].

Finally, let us discuss the design of a framework for adopting a Machine Learning algorithm within the limitations of HE schemes. Such the framework should have some extra features that enhance usability.. We list these features and propose our architecture for it. The first function is that the framework should be independent of the HE library. We suppose that the HE scheme supports basic functions: key generation, encryption, decryption, noise estimation, addition, and multiplication over encrypted data. We consider the basic module in the machine learning algorithms and develop a separate function for each one. For example, basic modules in a neural network are weighted sum, activation function, measure for calculating error, etc. We consider CPU hardware in our implementation and develop the framework to support the CPU, however, our framework should also support Graphical Processing Unit (GPU) and Field-Programmable Gate Array if it is supported by the HE library. The framework should also support client-server functionality. Figure 7.1 shows a high-level description of our framework.

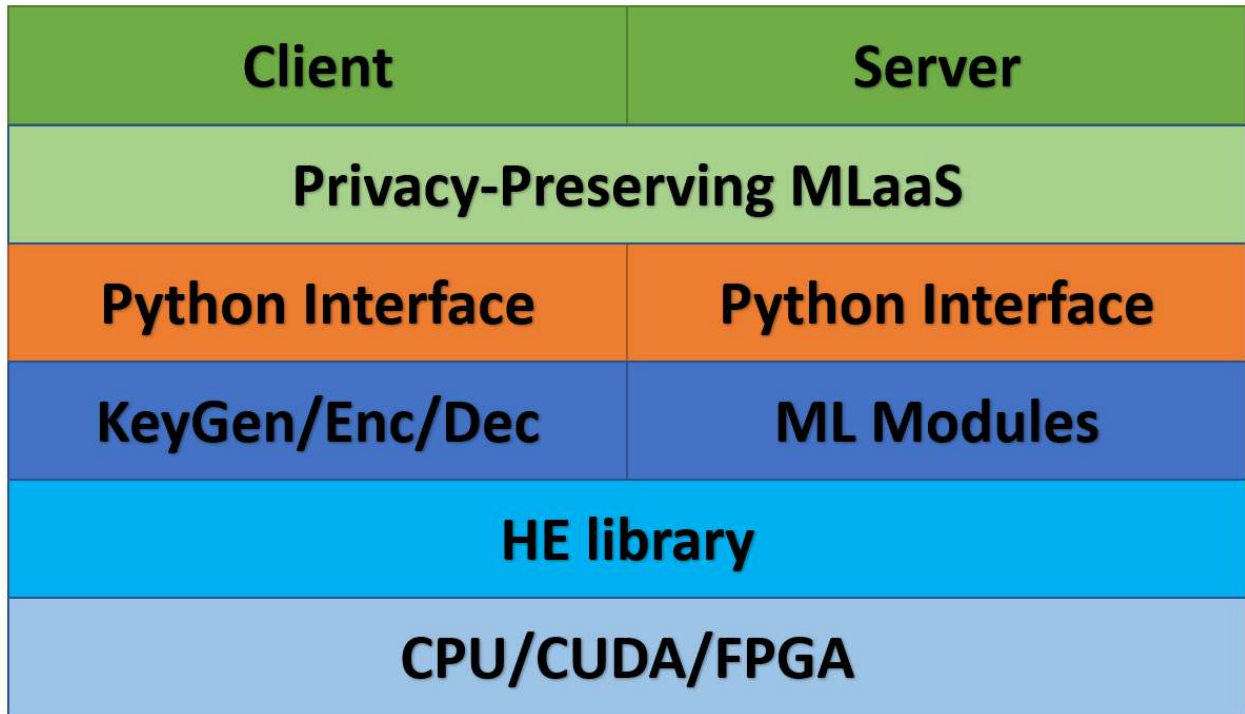


FIGURE 7.1. High Level Architecture of the Framework

APPENDIX

TRAINING BASED ON POLYNOMIALS AS ACTIVATION FUNCTIONS

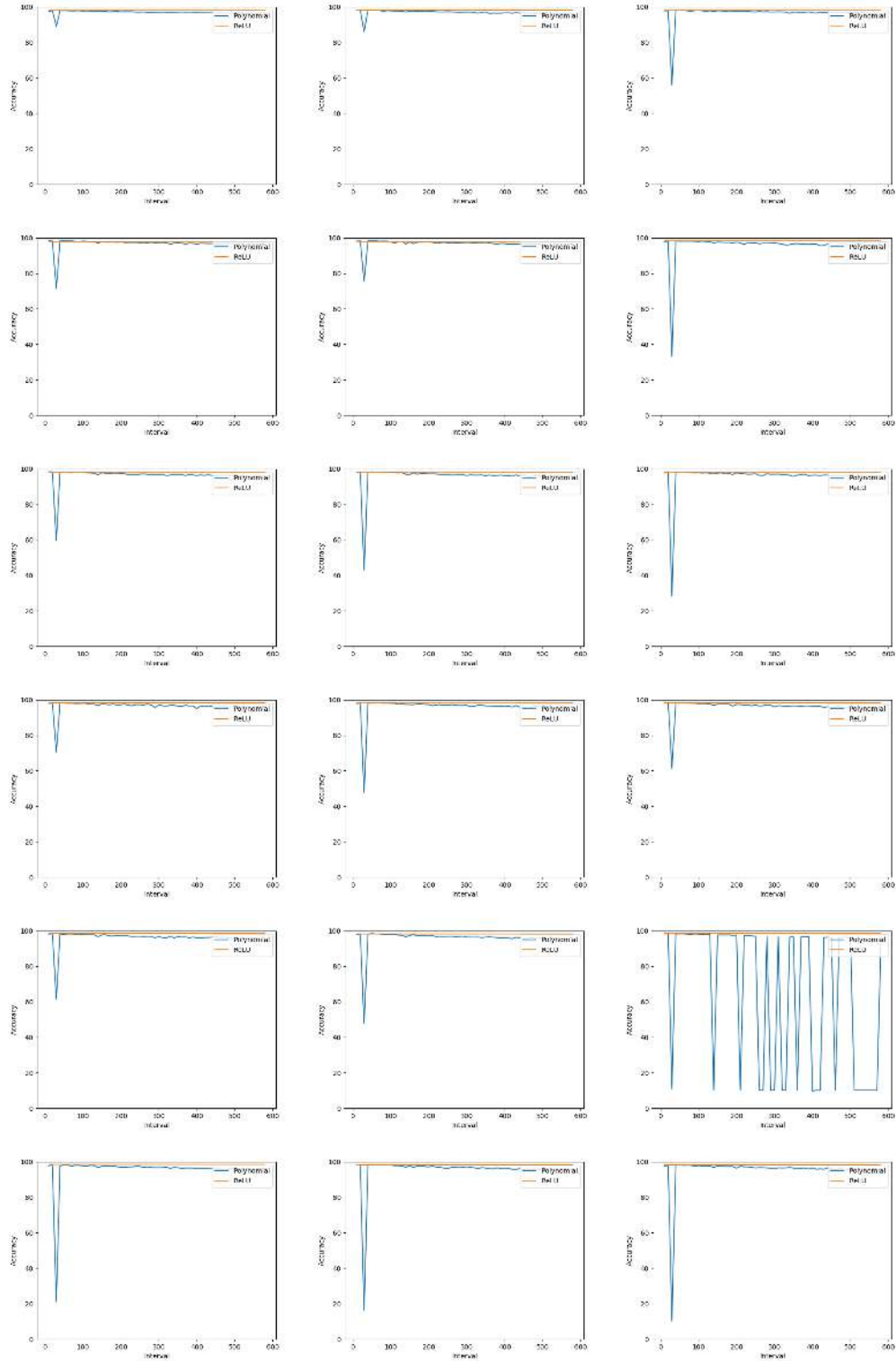


FIGURE A.1. Case 1 from 6.3, Accuracy vs. Architecture

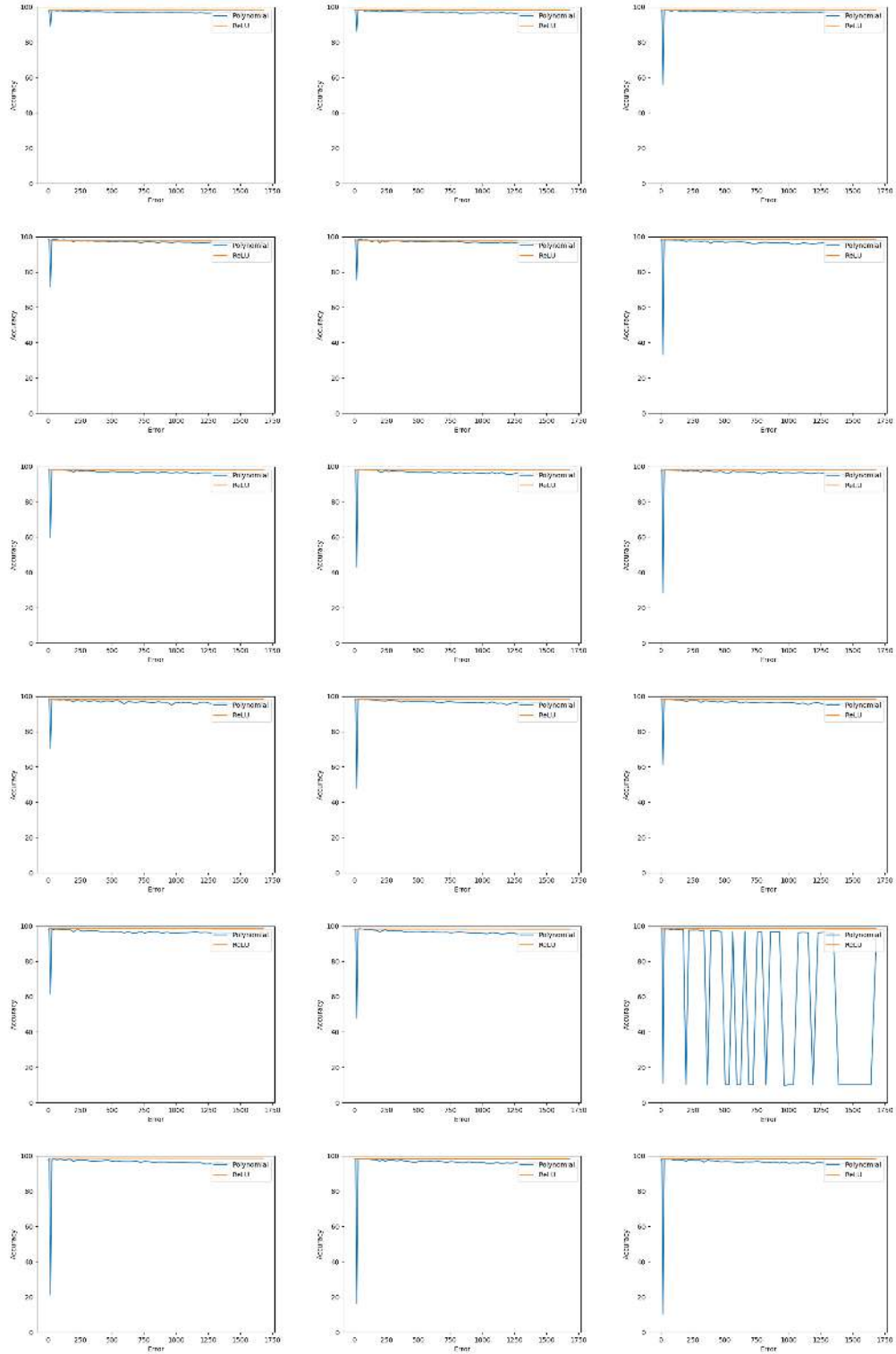


FIGURE A.2. Case 1 from 6.3, Accuracy vs. Error

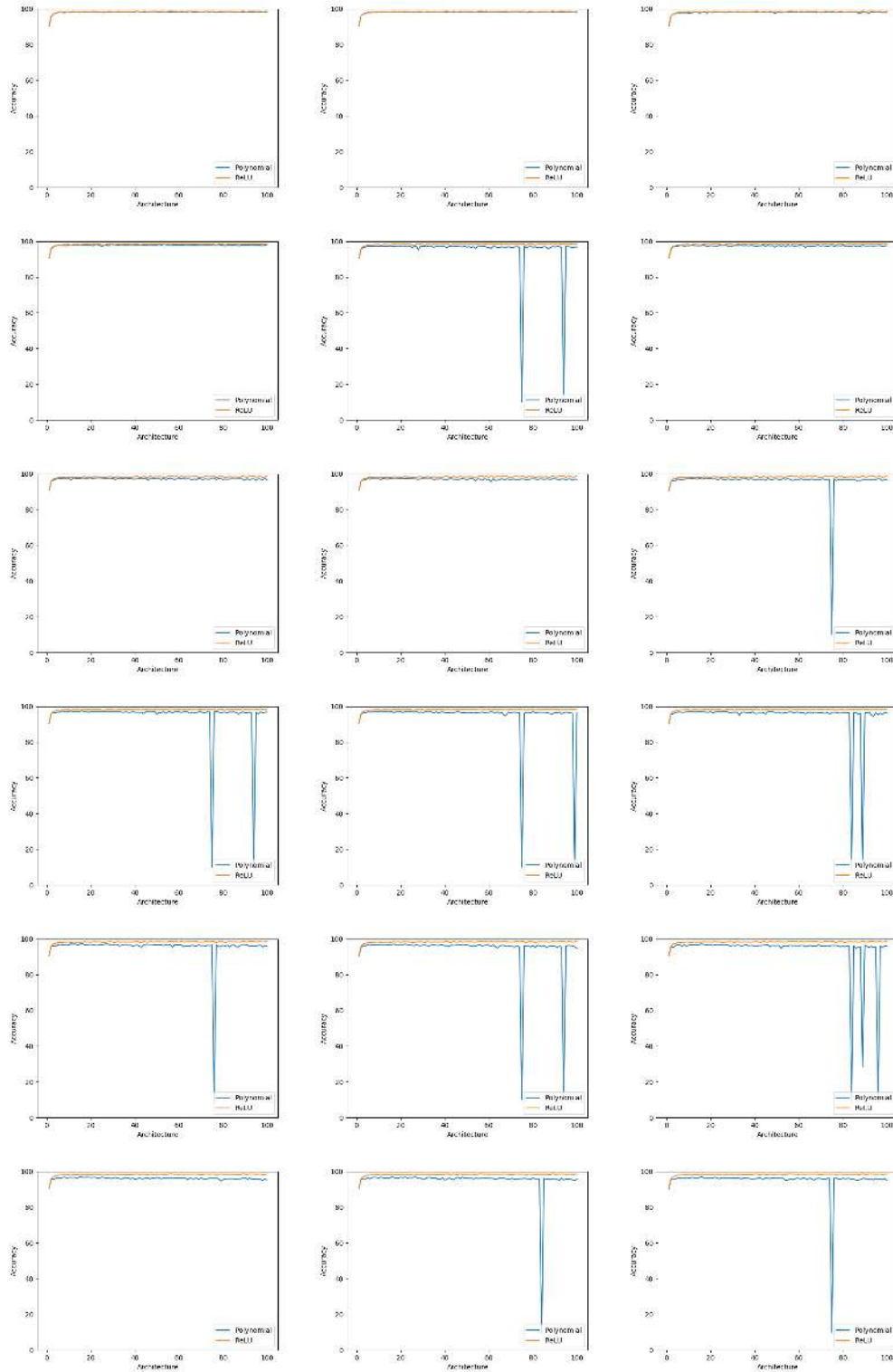


FIGURE A.3. Case 1 from 6.3, Accuracy vs. Polynomials

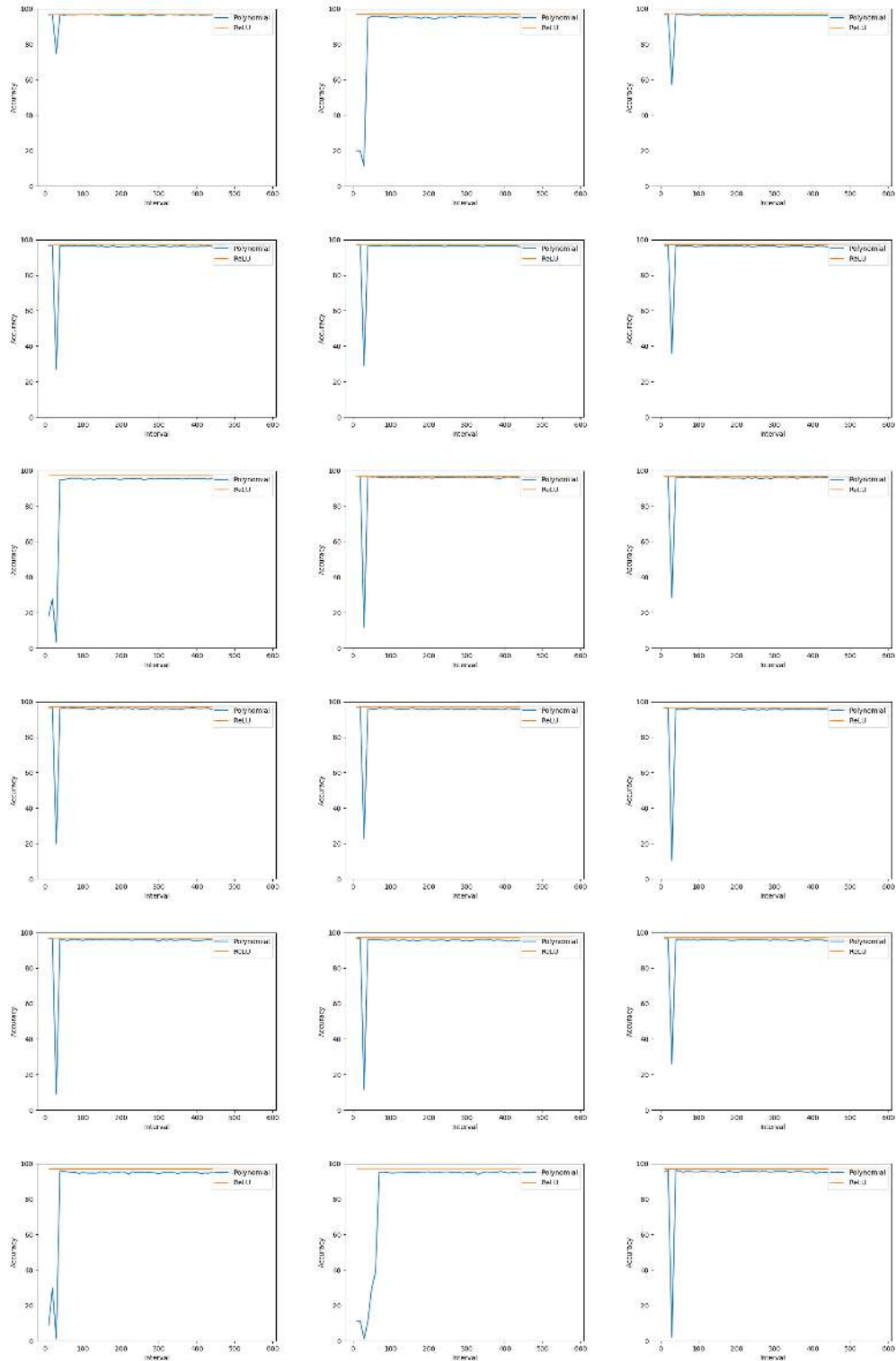


FIGURE A.4. Case 2 from 6.3, Accuracy vs. Architecture

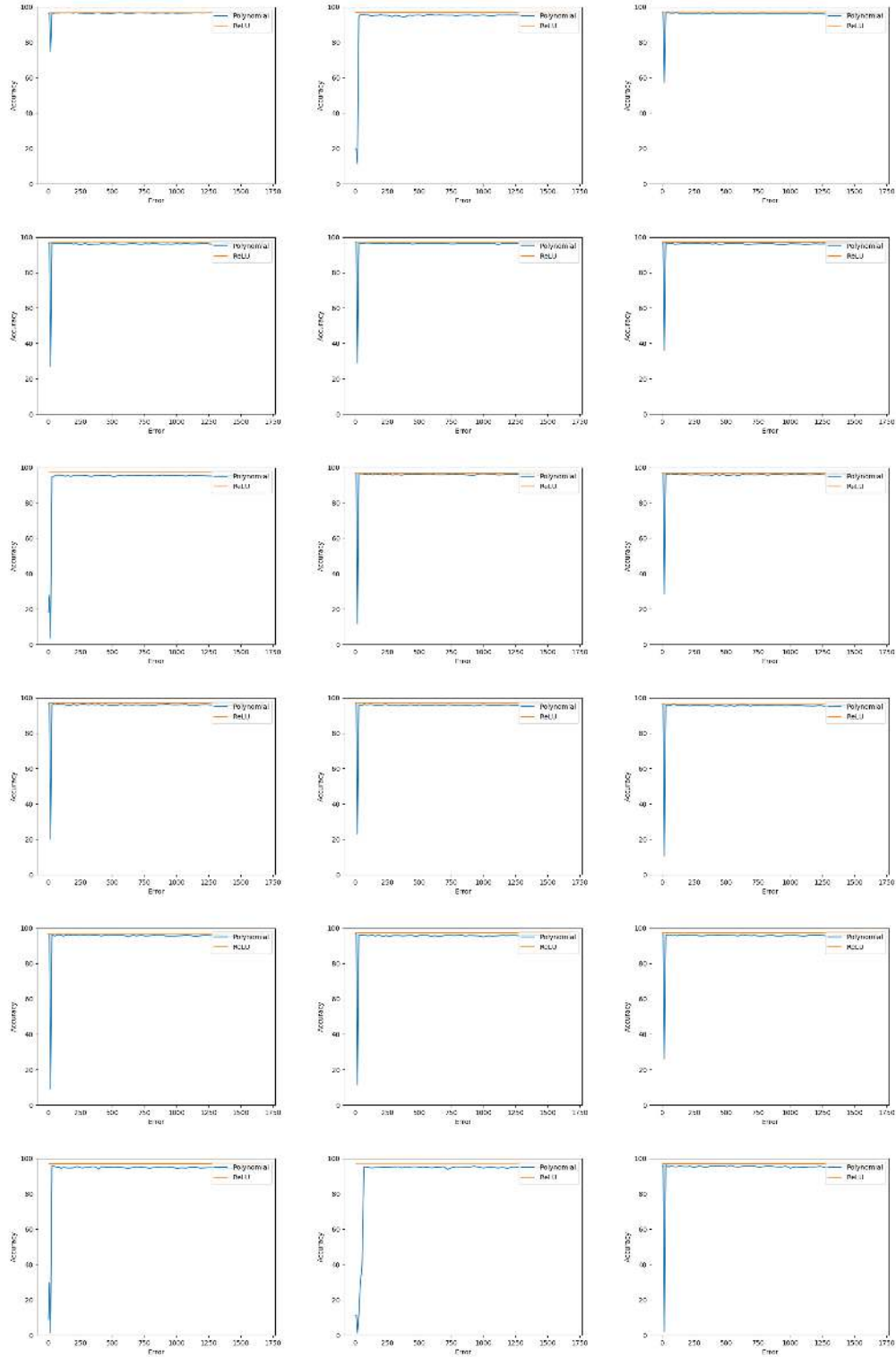


FIGURE A.5. Case 2 from 6.3, Accuracy vs. Error

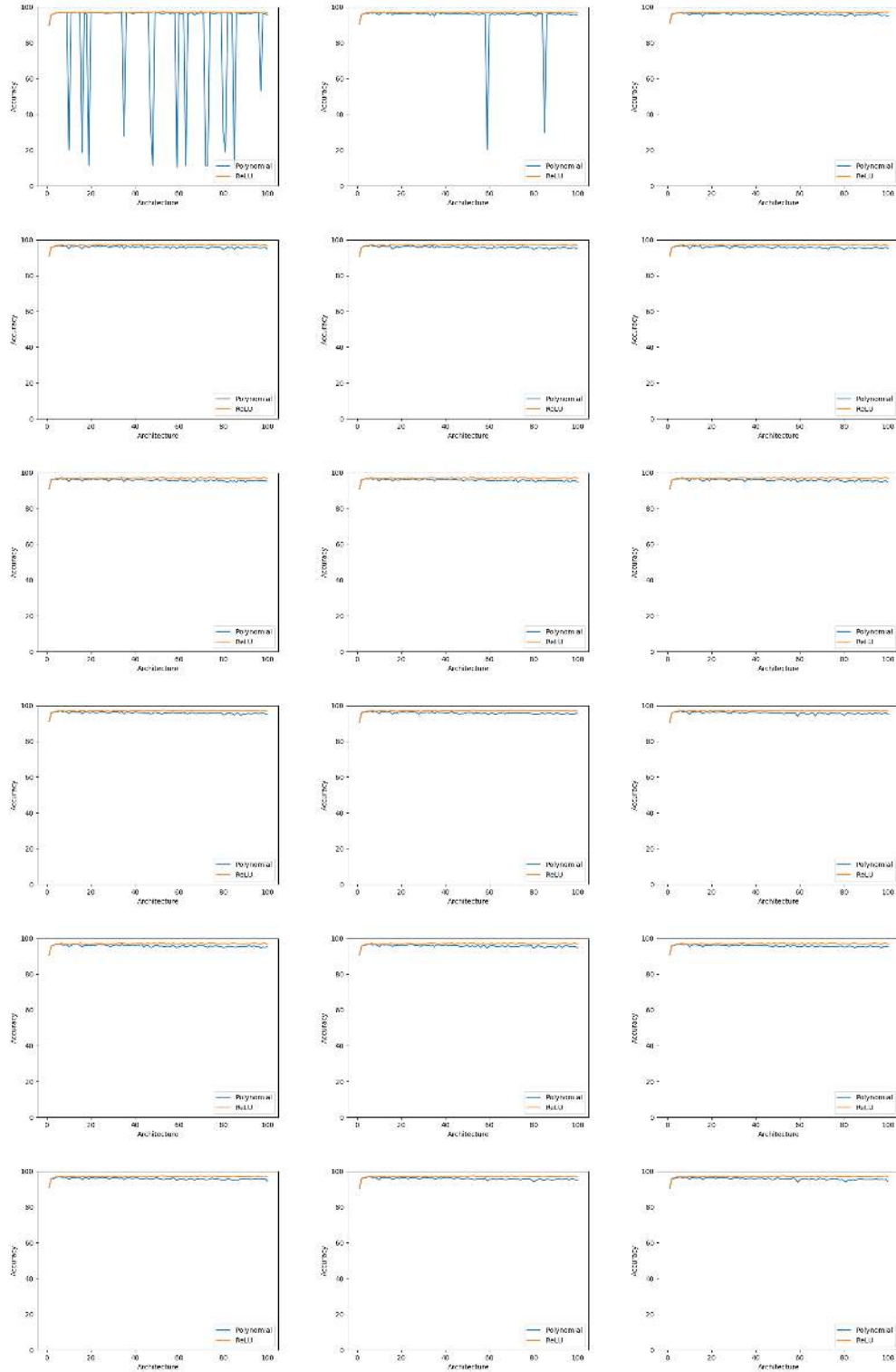


FIGURE A.6. Case 2 from 6.3, Accuracy vs. Polynomials

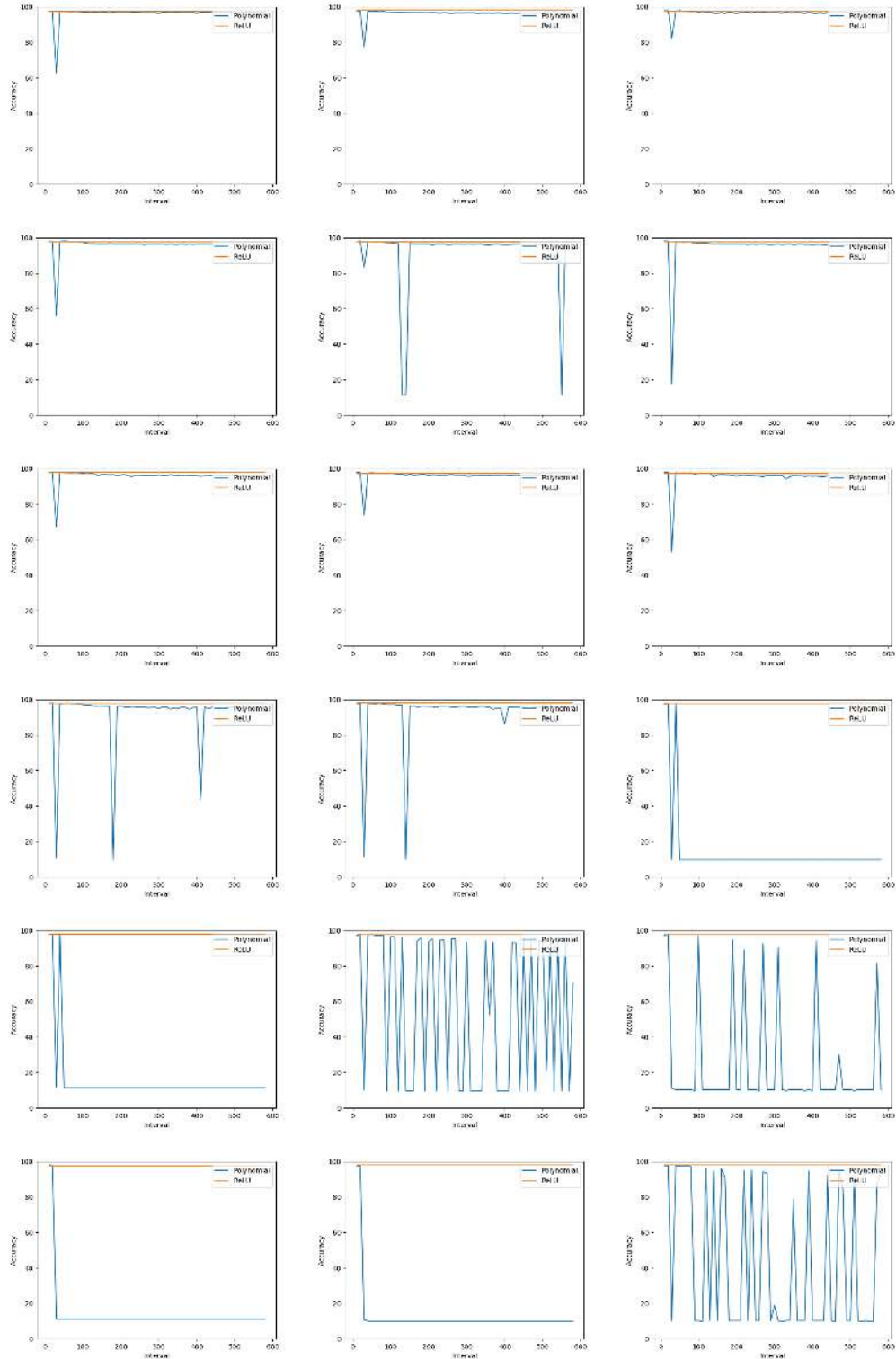


FIGURE A.7. Case 3 from 6.3, Accuracy vs. Architecture

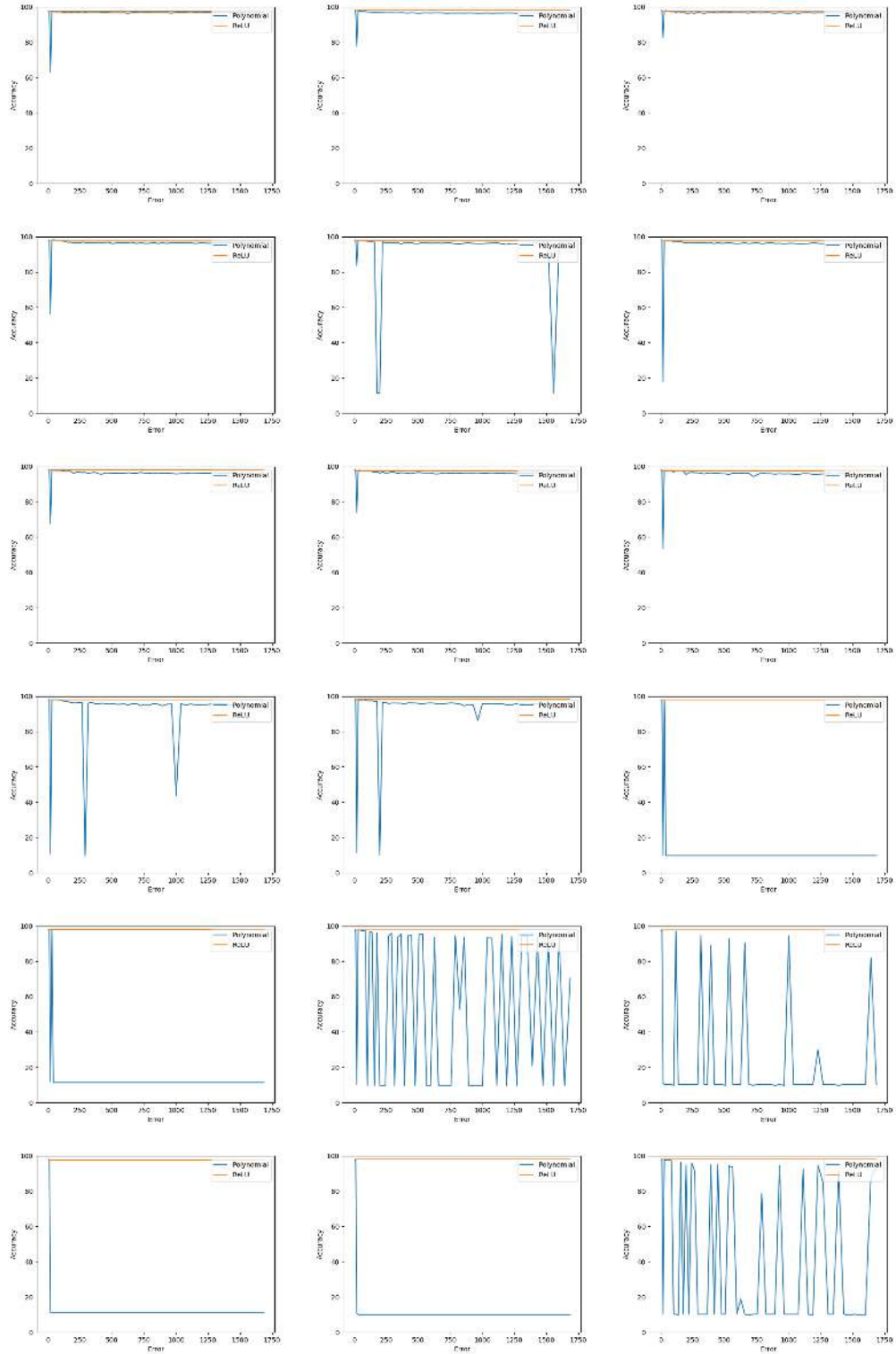


FIGURE A.8. Case 3 from 6.3, Accuracy vs. Error

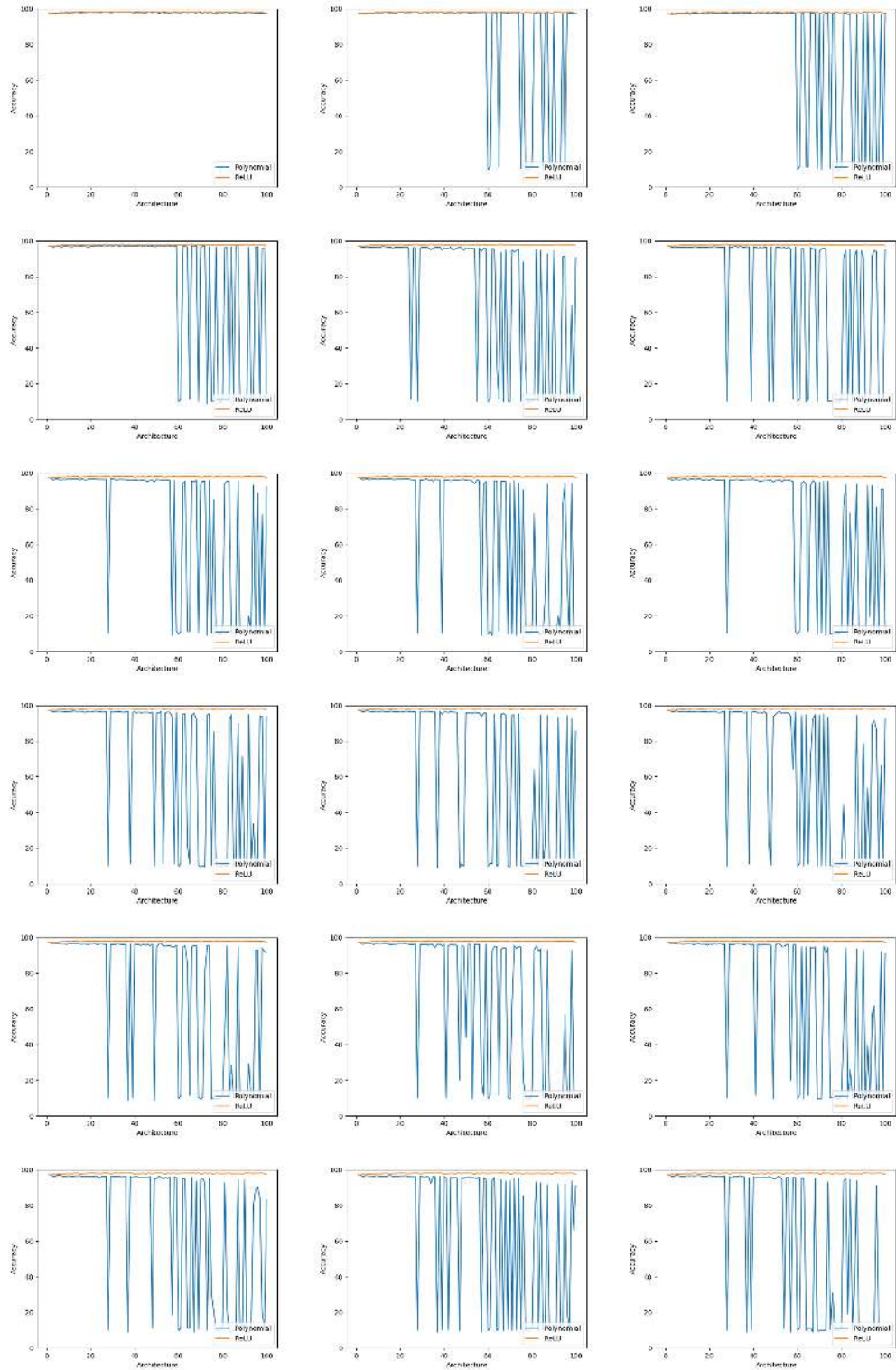


FIGURE A.9. Case 3 from 6.3, Accuracy vs. Polynomials

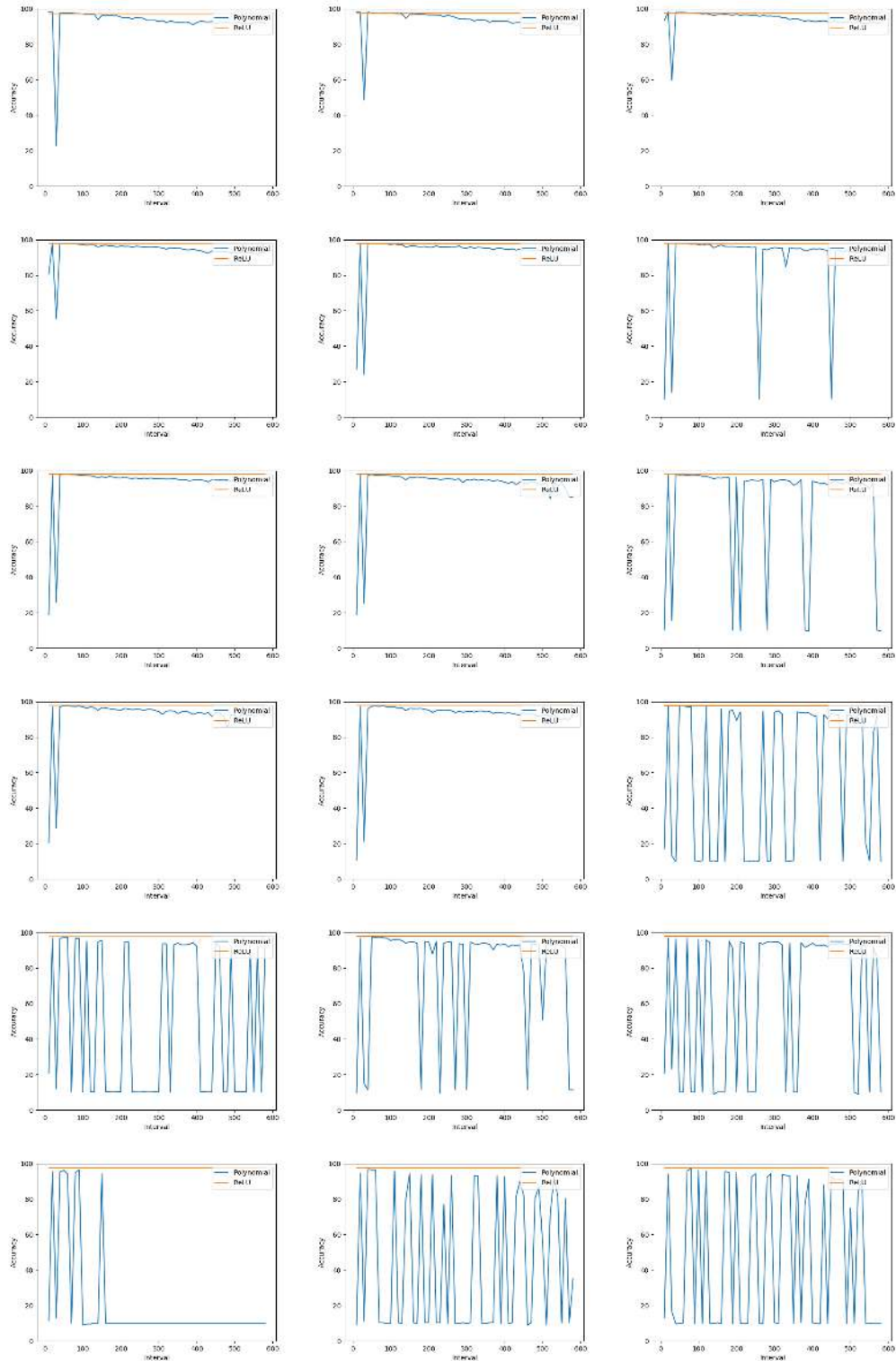


FIGURE A.10. Case 4 from 6.3, Accuracy vs. Architecture

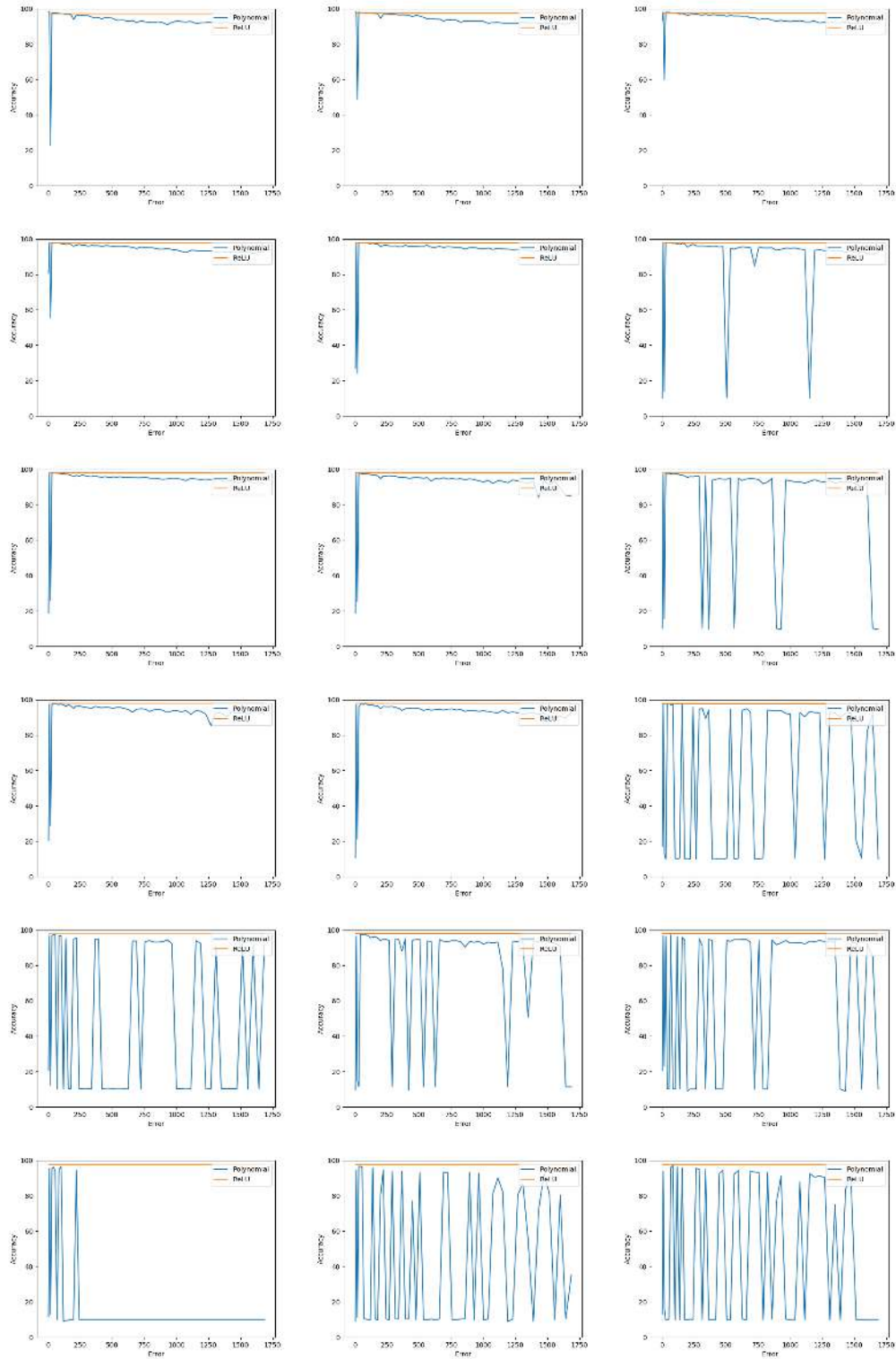


FIGURE A.11. Case 4 from 6.3, Accuracy vs. Error

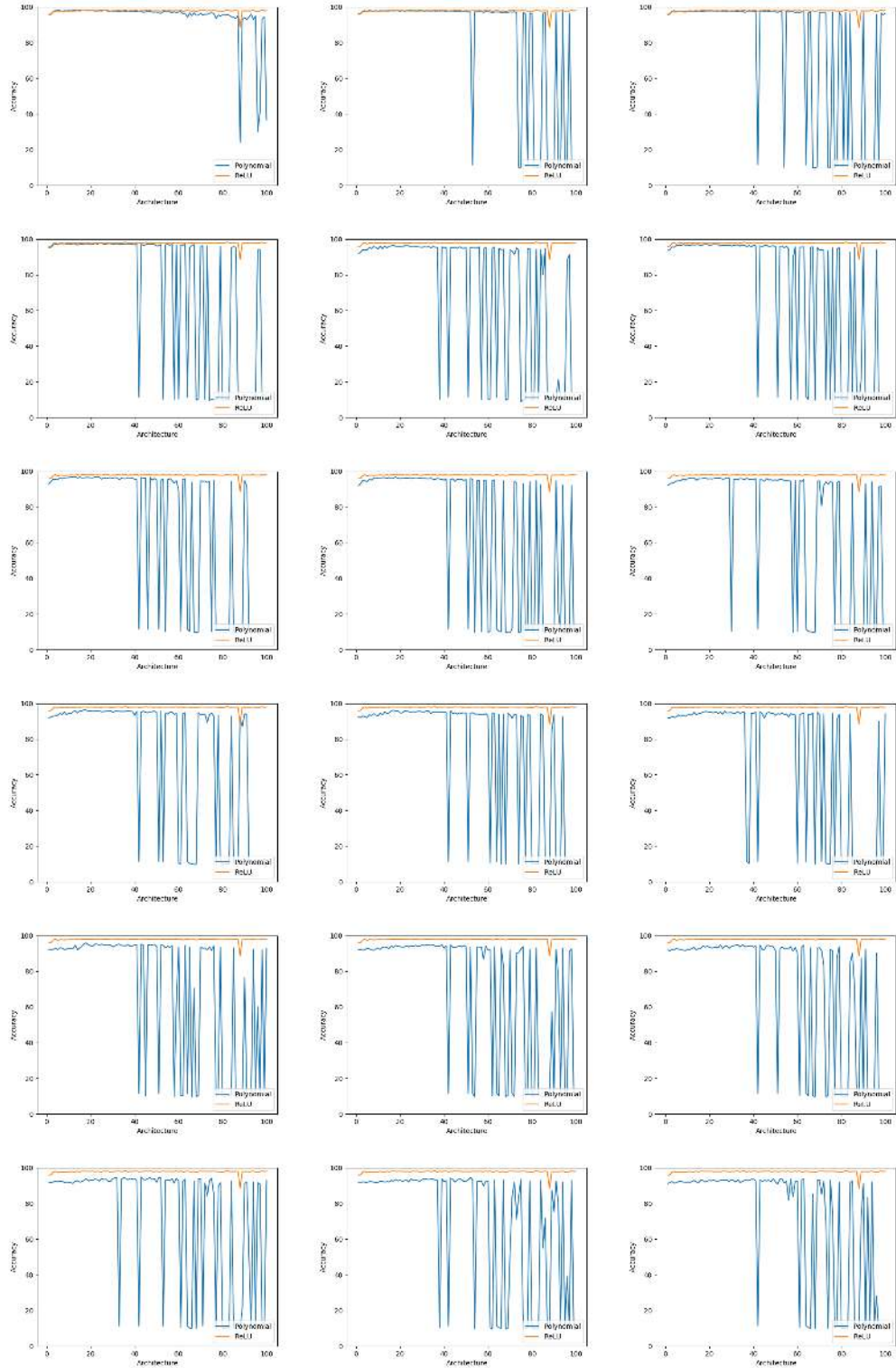


FIGURE A.12. Case 4 from 6.3, Accuracy vs. Polynomials

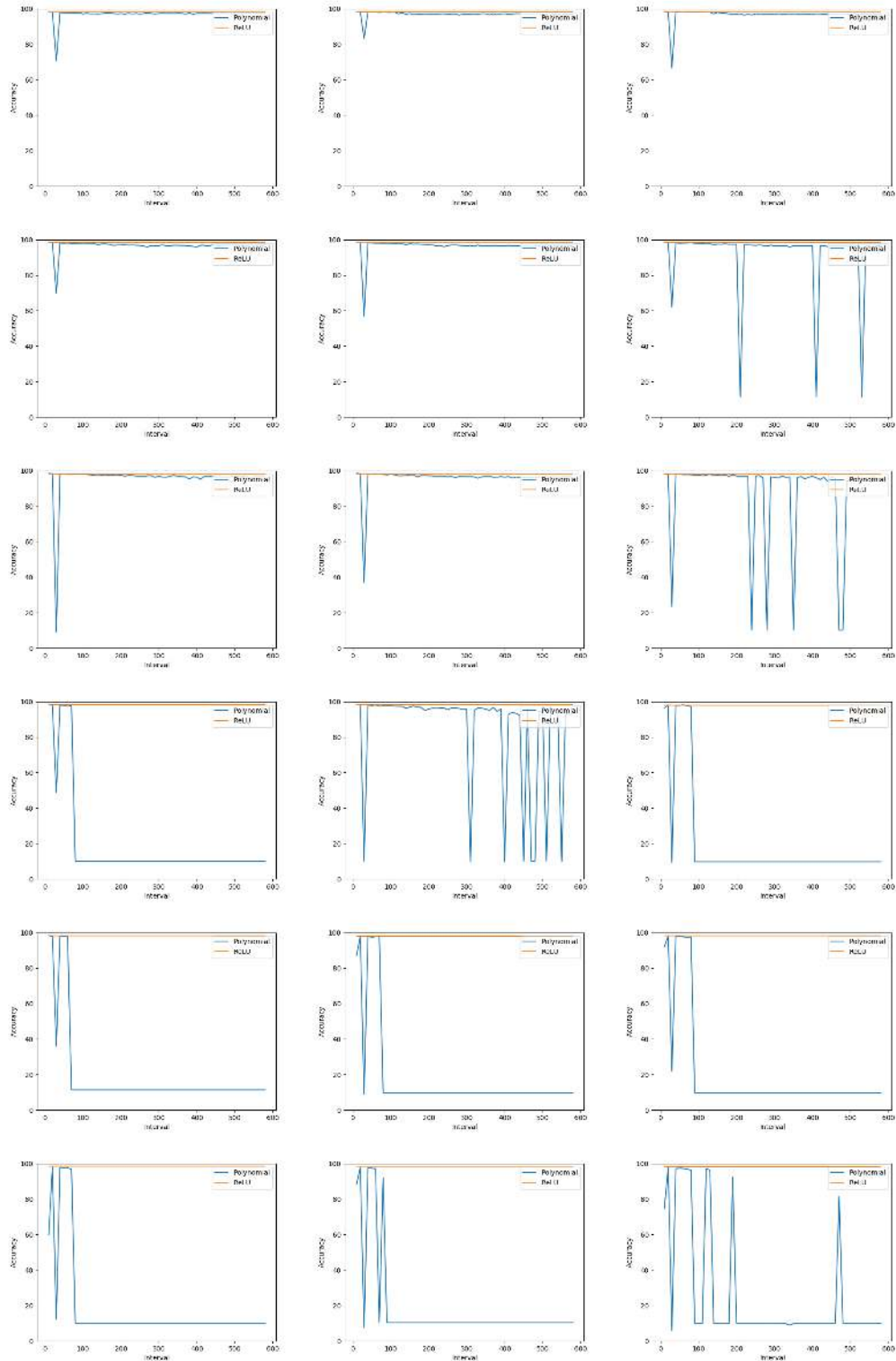


FIGURE A.13. Case 5 from 6.3, Accuracy vs. Architecture

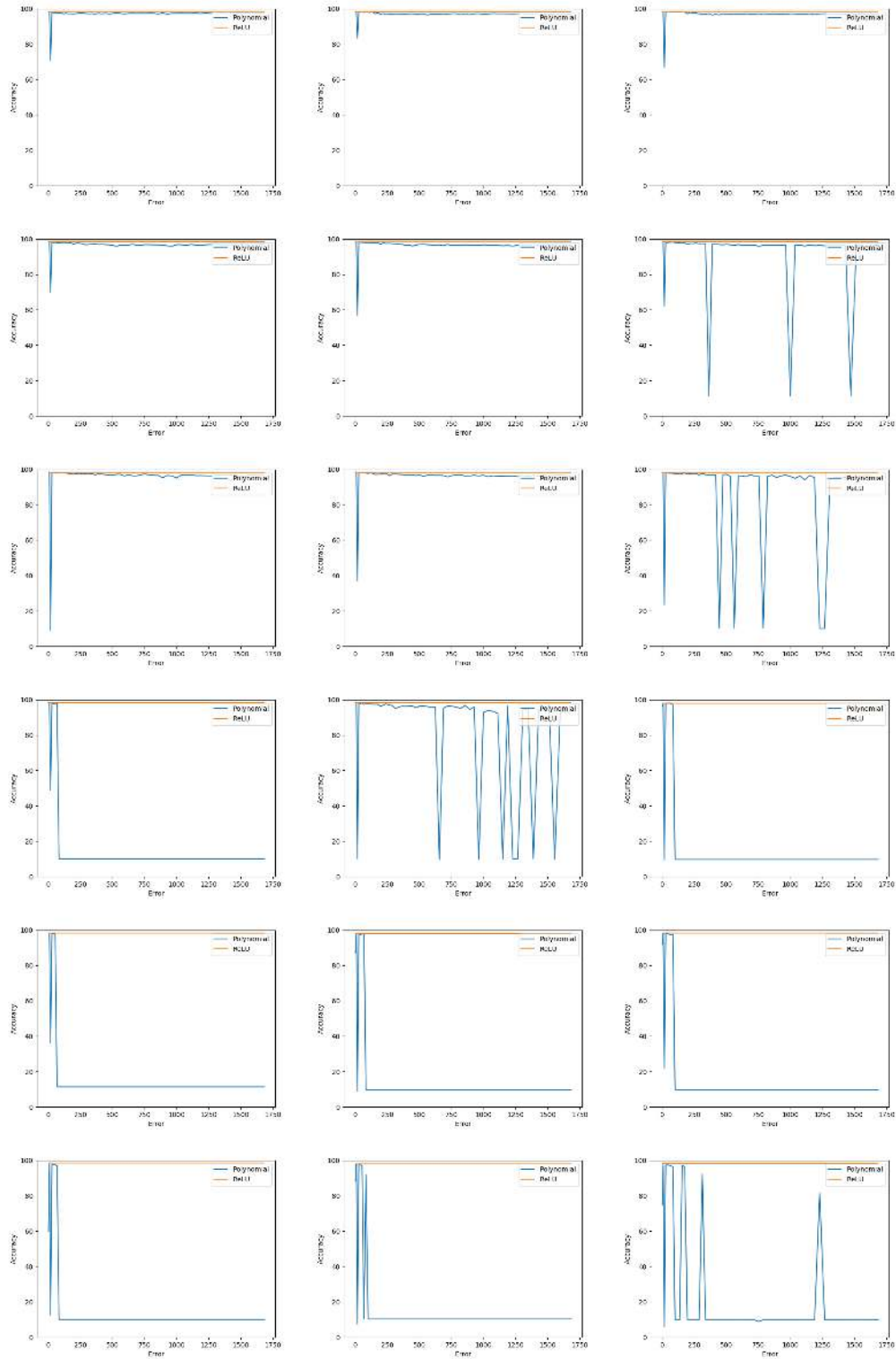


FIGURE A.14. Case 5 from 6.3, Accuracy vs. Error

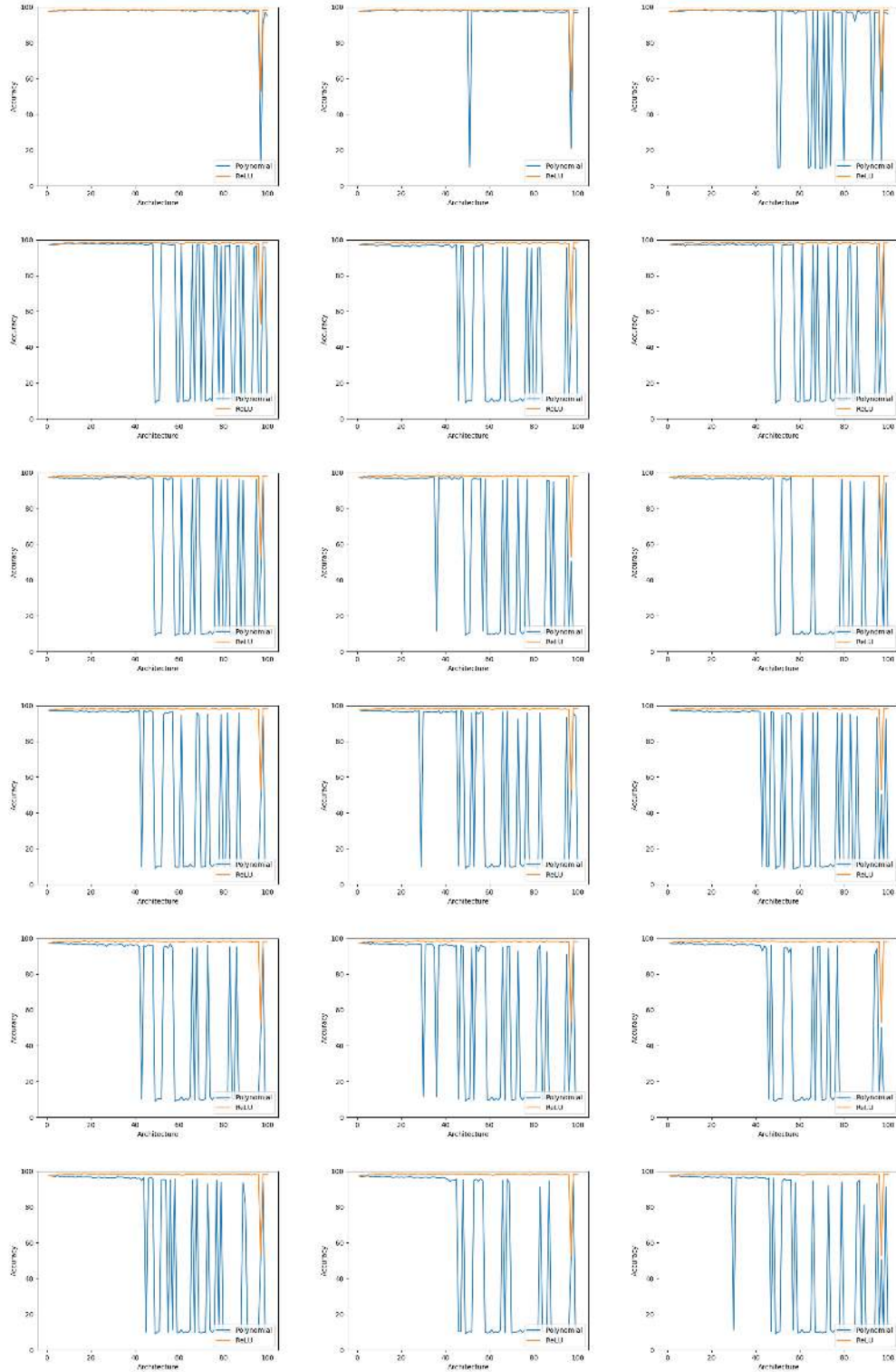


FIGURE A.15. Case 5 from 6.3, Accuracy vs. Polynomials

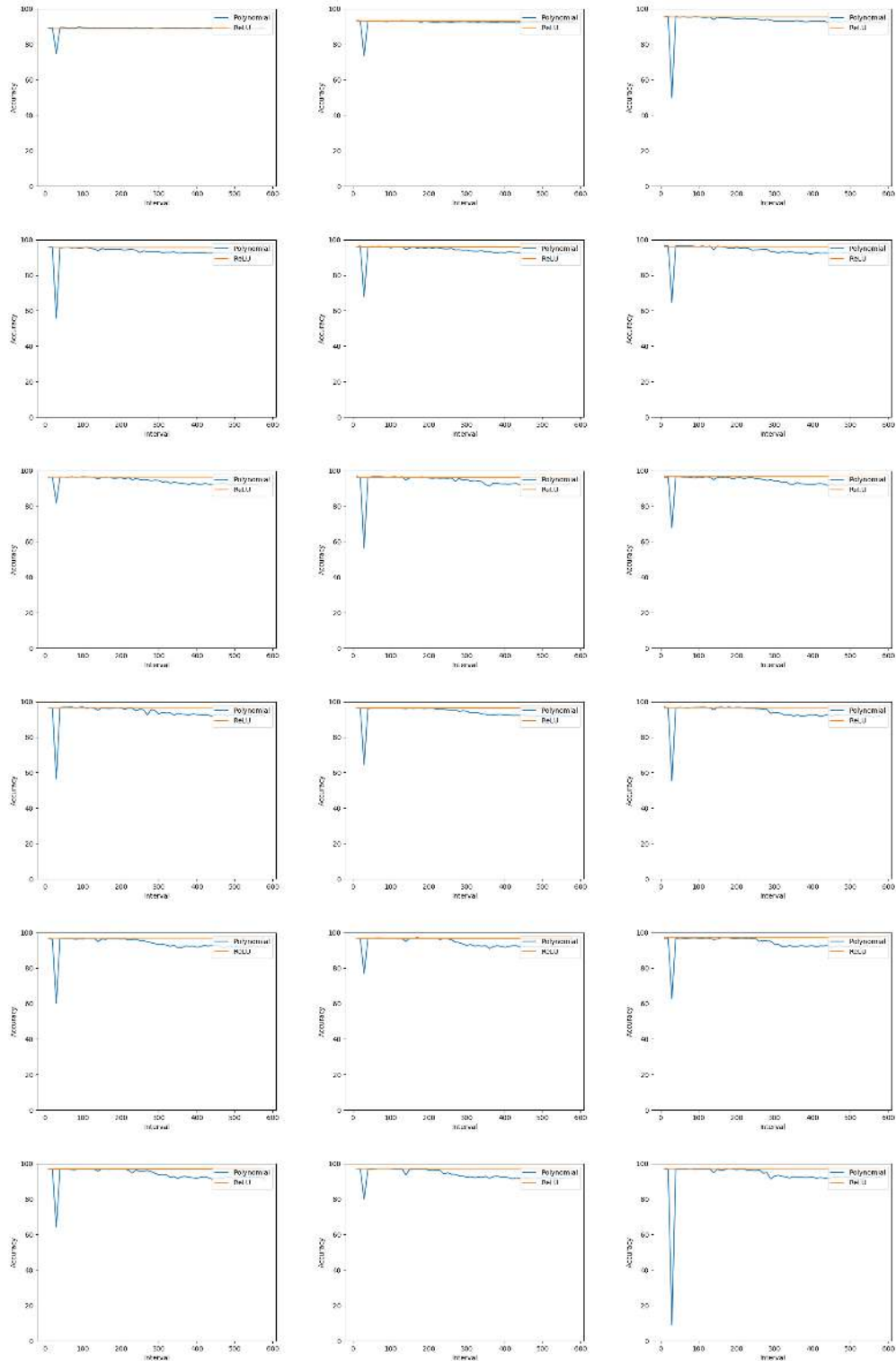


FIGURE A.16. Case 6 from 6.3, Accuracy vs. Architecture

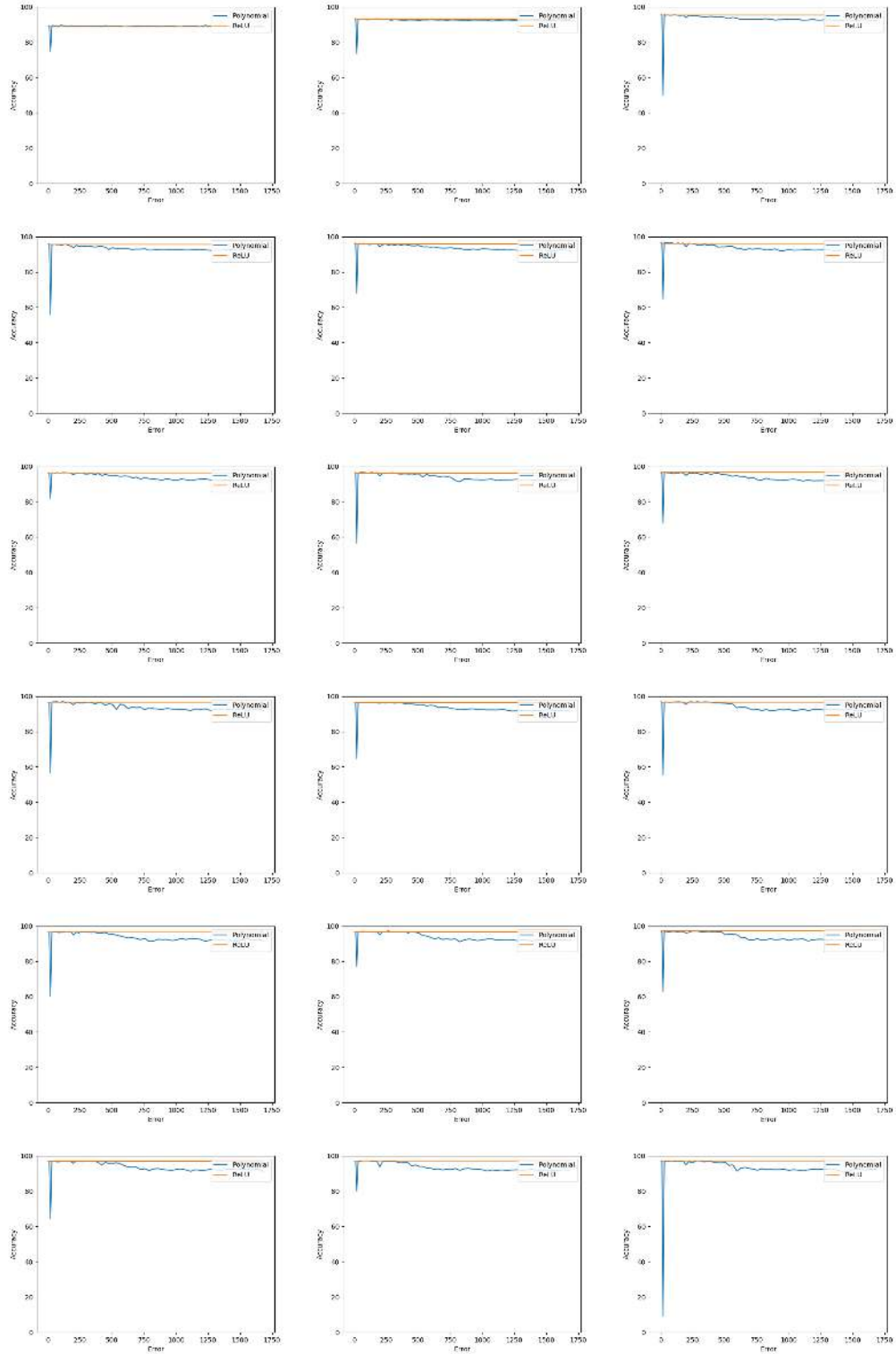


FIGURE A.17. Case 6 from 6.3, Accuracy vs. Error

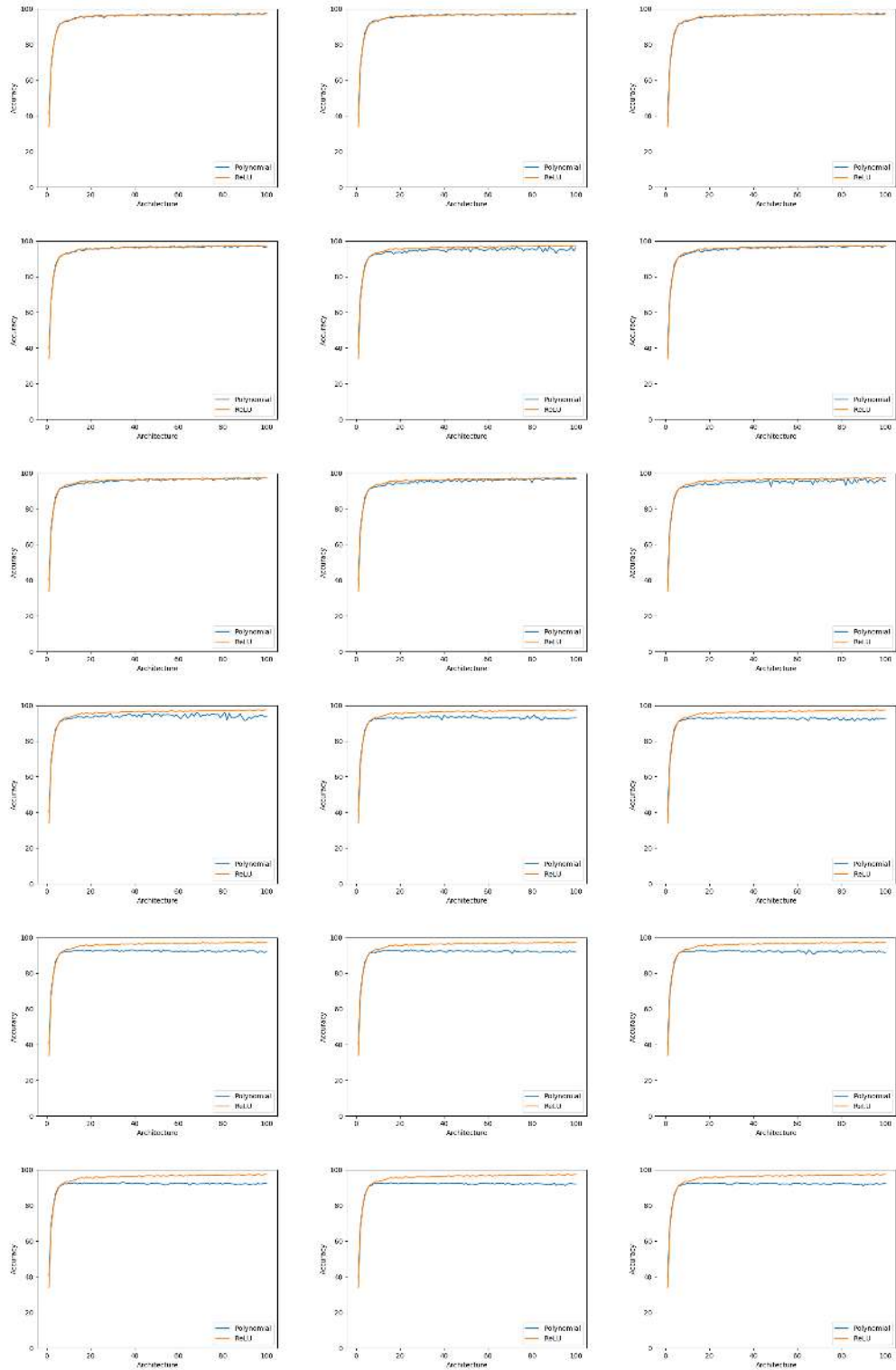


FIGURE A.18. Case 6 from 6.3, Accuracy vs. Polynomials

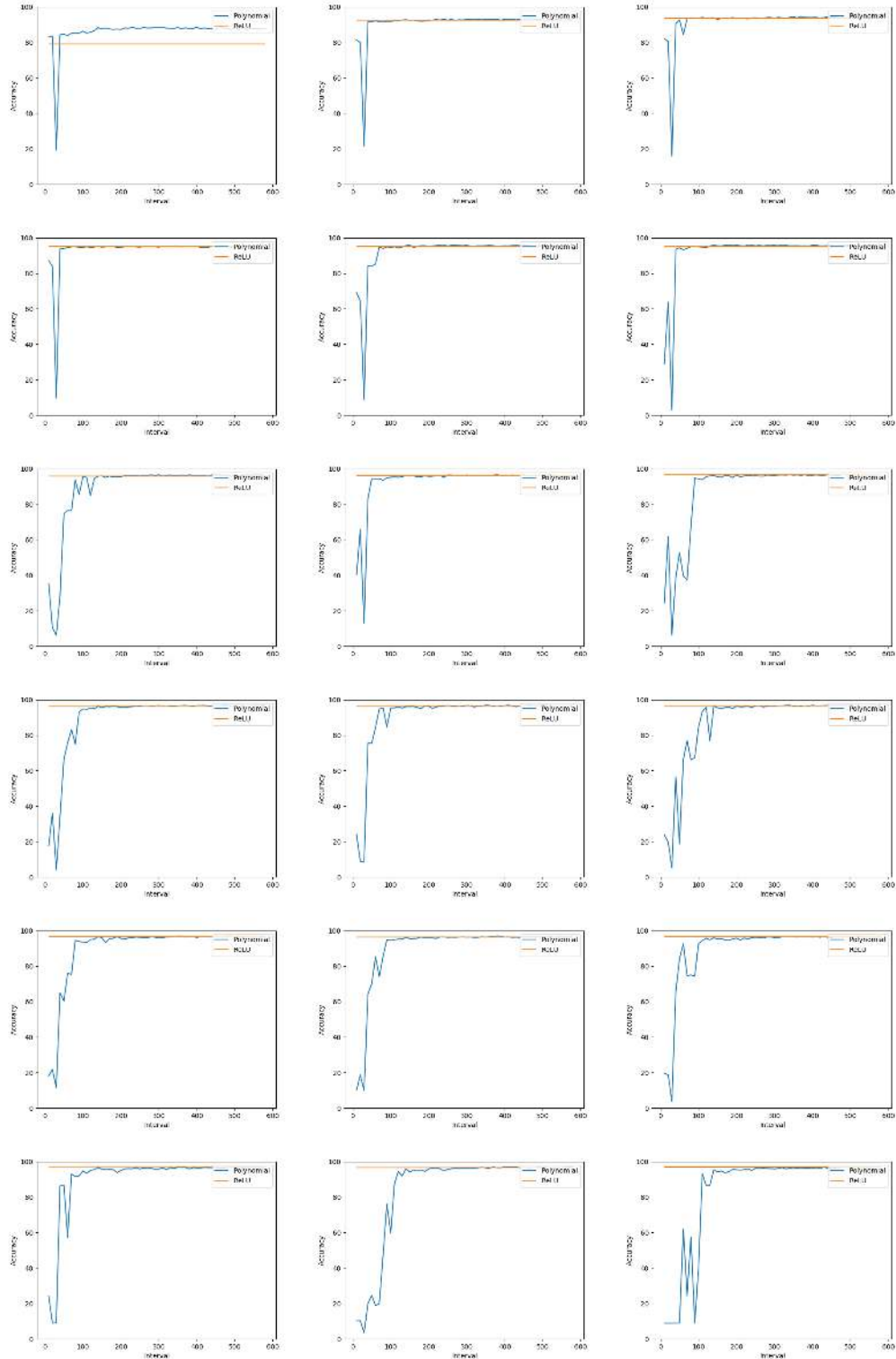


FIGURE A.19. Case 7 from 6.3, Accuracy vs. Architecture

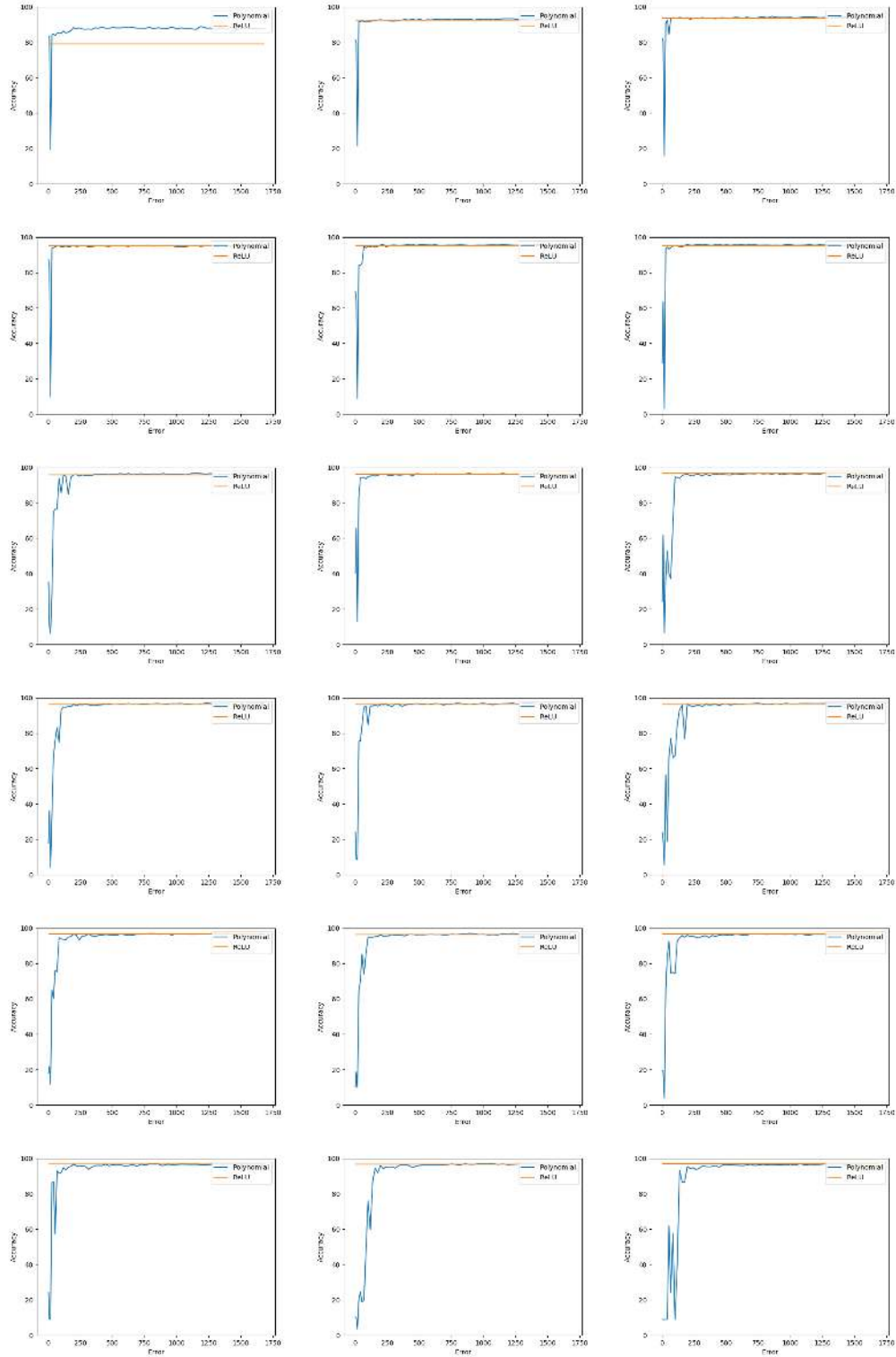


FIGURE A.20. Case 7 from 6.3, Accuracy vs. Error

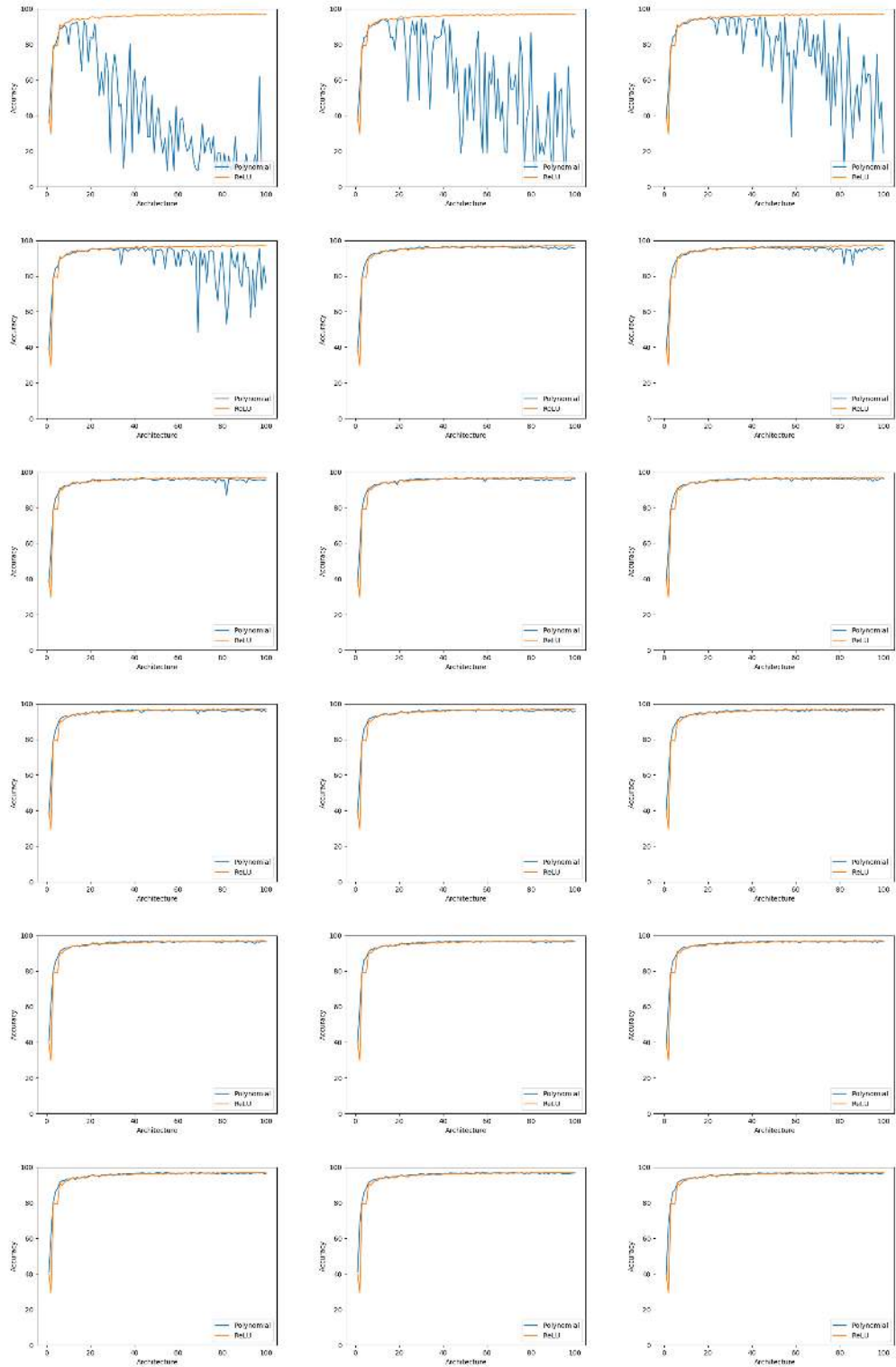


FIGURE A.21. Case 7 from 6.3, Accuracy vs. Polynomials

REFERENCES

- [1] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang, *Deep learning with differential privacy*, Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (New York, NY, USA), CCS '16, ACM, 2016, pp. 308–318.
- [2] Yoshinori Aono, Takuya Hayashi, Le Trieu Phong, and Lihua Wang, *Scalable and secure logistic regression via homomorphic encryption*, Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy (New York, NY, USA), CODASPY '16, ACM, 2016, pp. 142–144.
- [3] L. J. M. Aslett, P. M. Esperança, and C. C. Holmes, *A review of homomorphic encryption and software tools for encrypted statistical machine learning*, Tech. report, University of Oxford, 2015.
- [4] Louis J. M. Aslett, Pedro M. Esperança, and Chris C. Holmes, *Encrypted statistical machine learning: new privacy preserving methods*, CoRR (2015).
- [5] K. Atkinson and W. Han, *Theoretical numerical analysis: A functional analysis framework*, Texts in Applied Mathematics, Springer New York, 2009.
- [6] M. Barni, P. Failla, R. Lazzeretti, A. Paus, A. Sadeghi, T. Schneider, and V. Kolesnikov, *Efficient privacy-preserving classification of ecg signals*, 2009 First IEEE International Workshop on Information Forensics and Security (WIFS), Dec 2009, pp. 91–95.
- [7] Jim Basilakis, Bahman Javadi, and Anthony John Maeder, *The potential for machine learning analysis over encrypted data in cloud-based clinical decision support - background and review*, Workshop on Health Informatics and Knowledge Management (HIKM 2015), 2015, pp. 3–13.
- [8] Martin Beck and Florian Kerschbaum, *Approximate two-party privacy-preserving string matching with linear complexity*, Proceedings of the 2013 IEEE International Congress on Big Data (Washington, DC, USA), BIGDATAACONGRESS '13, IEEE Computer Society, 2013, pp. 31–37.

- [9] Flavio Bergamaschi and Graham Bent, *Computing on encrypted data and its applicability to a coalition operations environment*, 05 2015.
- [10] Dan Bogdanov, Liina Kamm, Baldur Kubo, Reimo Rebane, Ville Sokk, and Riivo Talviste, *Students and taxes: a privacy-preserving study using secure computation*, Proceedings on Privacy Enhancing Technologies 2016 (2016).
- [11] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser, *Machine learning classification over encrypted data*, 22nd Annual Network and Distributed System Security Symposium, NDSS, San Diego, California, USA, 2015.
- [12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan, *(leveled) fully homomorphic encryption without bootstrapping*, Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (New York, NY, USA), ITCS '12, ACM, 2012, pp. 309–325.
- [13] Justin Brickell and Vitaly Shmatikov, *Privacy-preserving classifier learning*, 02 2009, pp. 128–147.
- [14] Ferhat Ozgur Catak, *Secure multi-party computation based privacy preserving extreme learning machine algorithm over vertically distributed data*, 02 2016.
- [15] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff, *Privacy-preserving classification on deep neural network*, Cryptology ePrint Archive, Report 2017/035, 2017.
- [16] Nishanth Chandran, Divya Gupta, Aseem Rastogi, Rahul Sharma, and Shardul Tripathi, *Ezpc: Programmable, efficient, and scalable secure two-party computation for machine learning*, Cryptology ePrint Archive, Report 2017/1109, 2017, <https://eprint.iacr.org/2017/1109>.
- [17] Yao Chen and Guang Gong, *Integer arithmetic over ciphertext and homomorphic data aggregation*, Communications and Network Security (CNS), IEEE Conference on, Sept 2015, pp. 628–632.
- [18] European Commission, *General data protection regulation (gdpr)*, <https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/>

- `data-protection/2018-reform-eu-data-protection-rules_en`, accessed August 2018, accessed August 2018.
- [19] Jack L.H. Crawford, Craig Gentry, Shai Halevi, Daniel Platt, and Victor Shoup, *Doing real work with fhe: The case of logistic regression*, Cryptology ePrint Archive, Report 2018/202, 2018, <https://eprint.iacr.org/2018/202>.
 - [20] Daniel Demmler, Thomas Schneider, and Michael Zohner, *ABY - A framework for efficient mixed-protocol secure two-party computation*, 22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015, 2015.
 - [21] The Sage Developers, *Sagemath, the Sage Mathematics Software System (Version 7.1)*, 2016, <http://www.sagemath.org>.
 - [22] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing, *Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy*, Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, ICML16, JMLR.org, 2016, p. 201210.
 - [23] Wenliang Du and Zhijun Zhan, *Building decision tree classifier on private data*, Proceedings of the IEEE International Conference on Privacy, Security and Data Mining - Volume 14 (Darlinghurst, Australia, Australia), CRPIT '14, Australian Computer Society, Inc., 2002, pp. 1–8.
 - [24] Taher El Gamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, Proceedings of CRYPTO 84 on Advances in Cryptology (New York, NY, USA), Springer-Verlag New York, Inc., 1985, pp. 10–18.
 - [25] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart, *Model inversion attacks that exploit confidence information and basic countermeasures*, Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security (New York, NY, USA), CCS '15, ACM, 2015, pp. 1322–1333.

- [26] Craig Gentry, *A fully homomorphic encryption scheme*, Ph.D. thesis, Stanford University, Stanford, CA, USA, 2009, AAI3382729.
- [27] O. Goldreich, S. Micali, and A. Wigderson, *How to play any mental game*, Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing (New York, NY, USA), STOC '87, ACM, 1987, pp. 218–229.
- [28] Shafi Goldwasser and Silvio Micali, *Probabilistic encryption & how to play mental poker keeping secret all partial information*, Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing (New York, NY, USA), STOC '82, ACM, 1982, pp. 365–377.
- [29] Google, *Google prediction api*, (2017).
- [30] Thore Graepel, Kristin Lauter, and Michael Naehrig, *ML confidential: Machine learning on encrypted data*, Proceedings of the 15th International Conference on Information Security and Cryptology, ICISC'12, Springer-Verlag, 2013.
- [31] Shai Halevi and Victor Shoup, *Algorithms in HElib*, Advances in Cryptology - CRYPTO - 34th Annual Cryptology Conference, CA, USA, Proceedings, 2014.
- [32] Rob Hall, Stephen Fienberg, and Yuval Nardi, *Secure multiple linear regression based on homomorphic encryption*, Journal of Official Statistics 27 (2011).
- [33] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi, *CryptoDL: Deep neural networks over encrypted data*, CoRR abs/1711.05189 (2017).
- [34] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi, *Deep neural networks classification over encrypted data*, Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy (New York, NY, USA), CODASPY '19, ACM, 2019, pp. 97–108.
- [35] Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Catherine Jones, *Privacy-preserving machine learning in cloud*, Proceedings of the 2017 on Cloud Computing Security Workshop (New York, NY, USA), CCSW 17, Association for Computing Machinery, 2017, p. 3943.
- [36] Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Rebecca Wright, *Privacy-*

- preserving machine learning as a service*, Proceedings on Privacy Enhancing Technologies 2018 (2018), 123–142.
- [37] Naveed Islam, William Puech, Khizar Hayat, and Robert Brouzet, *Application of Homomorphism to Secure Image Sharing*, Optics Communications 284 (2011), no. 19, 4412–4429.
- [38] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan, *Gazelle: A low latency framework for secure neural network inference*, Proceedings of the 27th USENIX Conference on Security Symposium (Berkeley, CA, USA), SEC’18, USENIX Association, 2018, pp. 1651–1668.
- [39] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom, *A convolutional neural network for modelling sentences*, CoRR abs/1404.2188 (2014).
- [40] J. Kepner, V. Gadepally, P. Michaleas, N. Schear, M. Varia, A. Yerukhimovich, and R. K. Cunningham, *Computing on masked data: a high performance method for improving big data veracity*, 2014 IEEE High Performance Extreme Computing Conference (HPEC), Sep. 2014, pp. 1–6.
- [41] Florian Kerschbaum, *Outsourced private set intersection using homomorphic encryption*, Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security (New York, NY, USA), ASIACCS ’12, ACM, 2012, pp. 85–86.
- [42] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon, *Logistic regression model training based on the approximate homomorphic encryption*, Cryptology ePrint Archive, Report 2018/254, 2018, <https://eprint.iacr.org/2018/254>.
- [43] Miran Kim and Kristin E. Lauter, *Private genome analysis through homomorphic encryption*, IACR Cryptology ePrint Archive 2015 (2015), 965.
- [44] Miran Kim, Yongsoo Song, Shuang Wang, Yuhou Xia, and Xiaoqian Jiang, *Secure logistic regression based on homomorphic encryption: Design and evaluation*, Cryptology ePrint Archive, Report 2018/074, 2018, <https://eprint.iacr.org/2018/074>.
- [45] Sungwook Kim, Jinsu Kim, Dongyoung Koo, Yuna Kim, Hyunsoo Yoon, and Junbum Shin, *Efficient privacy-preserving matrix factorization via fully homomorphic encryption:*

- Extended abstract*, Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security (New York, NY, USA), ASIA CCS '16, ACM, 2016, pp. 617–628.
- [46] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton, *Cifar-10 (canadian institute for advanced research)*, accessed on 2016.
- [47] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, *Imagenet classification with deep convolutional neural networks*, Advances in Neural Information Processing Systems 25 (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), Curran Associates, Inc., 2012.
- [48] Ersatz Labs, *Ersatz*, (2017).
- [49] R. L. Lagendijk, Z. Erkin, and M. Barni, *Encrypted signal processing for privacy protection: Conveying the utility of homomorphic encryption and multiparty computation*, IEEE Signal Processing Magazine 30 (2013), no. 1, 82–105.
- [50] Kristin Lauter, Adriana López-Alt, and Michael Naehrig, *Private computation on encrypted genomic data*, Progress in Cryptology - LATINCRYPT 2014 (Cham) (Diego F. Aranha and Alfred Menezes, eds.), Springer International Publishing, 2015, pp. 3–27.
- [51] S. Lawrence, C. L. Giles, Ah Chung Tsoi, and A. D. Back, *Face recognition: a convolutional neural-network approach*, IEEE Transactions on Neural Networks (1997).
- [52] Yann LeCun, Corinna Cortes, and Christopher Burges, *MNIST handwritten digit database*, accessed June 2015.
- [53] M. Lichman, *UCI machine learning repository*, 2013.
- [54] Yehuda Lindell and Benny Pinkas, *Privacy preserving data mining*, Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology (London, UK, UK), CRYPTO '00, Springer-Verlag, 2000, pp. 36–54.
- [55] ———, *Secure multiparty computation for privacy-preserving data mining.*, IACR Cryptology ePrint Archive 2008 (2008), 197.
- [56] F. Liu, W. K. Ng, and W. Zhang, *Secure scalar product for big-data in mapreduce*, Big

- Data Computing Service and Applications (BigDataService), IEEE First International Conference on, March 2015, pp. 120–129.
- [57] Fang Liu, Wee Keong Ng, and Wei Zhang, *Encrypted svm for outsourced data mining*, Proceedings of the 2015 IEEE 8th International Conference on Cloud Computing (Washington, DC, USA), CLOUD '15, IEEE Computer Society, 2015, pp. 1085–1092.
- [58] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan, *Oblivious neural network predictions via minionn transformations*, Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (New York, NY, USA), CCS '17, ACM, 2017, pp. 619–631.
- [59] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir, *On the computational efficiency of training neural networks*, Advances in Neural Information Processing Systems 27 (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), Curran Associates, Inc., 2014, pp. 855–863.
- [60] Wen-Jie Lu, Yoshiji Yamada, and Jun Sakuma, *Privacy-preserving genome-wide association studies on cloud environment using fully homomorphic encryption*, BMC Medical Informatics and Decision Making 15 (2015), no. 5, 1–8.
- [61] Wenjie Lu, Shohei Kawasaki, and Jun Sakuma, *Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data*, IACR Cryptology ePrint Archive 2016 (2016), 1163.
- [62] Luca Melis, Hassan Jameel Asghar, Emiliano De Cristofaro, and Mohamed Ali Kaafer, *Private processing of outsourced network functions: Feasibility and constructions*, Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization (New York, NY, USA), SDN-NFV Security '16, ACM, 2016, pp. 39–44.
- [63] Microsoft, *Microsoft azure machine learning*, 2017.
- [64] Ming-Jun Xiao, Liu-Sheng Huang, Yong-Long Luo, and Hong Shen, *Privacy preserving id3 algorithm over horizontally partitioned data*, Sixth International Conference on

- Parallel and Distributed Computing Applications and Technologies (PDCAT'05), Dec 2005, pp. 239–243.
- [65] P. Mohassel and Y. Zhang, *Secureml: A system for scalable privacy-preserving machine learning*, IEEE Symposium on Security and Privacy (SP), May 2017.
- [66] Moni Naor and Benny Pinkas, *Computationally secure oblivious transfer*, Journal of Cryptology 18 (2005), no. 1, 1–35.
- [67] U.S. Department of Health & Human Services, *Health insurance portability and accountability act (hipaa)*, <https://www.hhs.gov/hipaa/index.html>, accessed August 2018, accessed August 2018.
- [68] A Olgac, *Performance analysis of various activation functions in generalized mlp architectures of neural networks*, International Journal of Artificial Intelligence And Expert Systems 1 (2011), 111–122.
- [69] C. Orlandi, A. Piva, and M. Barni, *Oblivious neural network computing via homomorphic encryption*, EURASIP Journal on Information Security 2007 (2007), no. 1.
- [70] A. Page, O. Kocabas, S. Ames, M. Venkitasubramaniam, and T. Soyata, *Cloud-based secure health monitoring: Optimizing fully-homomorphic encryption for streaming algorithms*, 2014 IEEE Globecom Workshops (GC Wkshps), Dec 2014, pp. 48–52.
- [71] Pascal Paillier, *Public-key cryptosystems based on composite degree residuosity classes*, 17th International Conference on Theory and Application of Cryptographic Techniques (Berlin, Heidelberg), EUROCRYPT'99, Springer-Verlag, 1999.
- [72] Sinno Jialin Pan and Qiang Yang, *A survey on transfer learning*, IEEE Trans. on Knowl. and Data Eng. 22 (2010), no. 10, 1345–1359.
- [73] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, *Privacy-preserving deep learning via additively homomorphic encryption*, IEEE Transactions on Information Forensics and Security 13 (2018), no. 5, 1333–1345.
- [74] Paul Pilotte, *Neural Network Toolbox*, 2016.
- [75] William Puech, Zekeriya Erkin, Mauro Barni, S. Rane, and Reginald L. Lagendijk,

- Emerging cryptographic challenges in image and video processing.*, ICIP, IEEE, 2012, pp. 2629–2632.
- [76] Shantanu Rane, Wei Sun, and Anthony Vetro, *Secure distortion computation among untrusting parties using homomorphic encryption.*, ICIP, IEEE, 2009, pp. 1485–1488.
- [77] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar, *Chameleon: A hybrid secure computation framework for machine learning applications*, Proceedings of the 2018 on Asia Conference on Computer and Communications Security (New York, NY, USA), ASIACCS '18, ACM, 2018, pp. 707–721.
- [78] R.L. Rivest, L. Adleman, and M.L. Dertouzos, *On data banks and privacy homomorphisms*, Foundations on Secure Computation, Academia Press, 1978.
- [79] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, *Deepsecure: Scalable provably-secure deep learning*, 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), June 2018, pp. 1–6.
- [80] Saeed Samet, Ali Miri, and Luis Orozco-Barbosa, *Privacy preserving k-means clustering in multi-party environment.*, 01 2007, pp. 381–385.
- [81] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, *Membership inference attacks against machine learning models*, 2017 IEEE Symposium on Security and Privacy (SP), May 2017, pp. 3–18.
- [82] Reza Shokri and Vitaly Shmatikov, *Privacy-preserving deep learning*, Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security (New York, NY, USA), CCS '15, ACM, 2015, pp. 1310–1321.
- [83] Thomas Shortell and Ali Shokoufandeh, *Secure signal processing using fully homomorphic encryption*, Advanced Concepts for Intelligent Vision Systems - 16th International Conference, ACIVS, Italy, Proceedings, 2015.
- [84] Eakalak Suthampan and Songrit Maneewongvatana, *Privacy preserving decision tree in multi party environment*, Proceedings of the Second Asia Conference on Asia Information Retrieval Technology (Berlin, Heidelberg), AIRS'05, Springer-Verlag, 2005, pp. 727–732.

- [85] M. Togan and C. Pleca, *Comparison-based computations over fully homomorphic encrypted data*, 2014 10th International Conference on Communications (COMM), May 2014, pp. 1–6.
- [86] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart, *Stealing machine learning models via prediction apis.*, USENIX Security Symposium, 2016, pp. 601–618.
- [87] Turi, *Graphlab*, 2017.
- [88] Jan Ubbo van Baardewijk, *Privacy-enhanced biometric hamming distance classification*, 2015.
- [89] Thijs Veugen, *Encrypted integer division and secure comparison*, Int. J. Appl. Cryptol. 3 (2014), no. 2, 166–180.
- [90] Sameer Wagh, Divya Gupta, and Nishanth Chandran, *Securenn: Efficient and private neural network training*, Privacy Enhancing Technologies Symposium, (PETS 2019), February 2019.
- [91] David Wu and Jacob Haven, *Using homomorphic encryption for large scale statistical analysis*, 2012.
- [92] Pengtao Xie, Misha Bilenko, Tom Finley, Ran Gilad-Bachrach, Kristin E. Lauter, and Michael Naehrig, *Crypto-nets: Neural networks over encrypted data*, CoRR (2014).
- [93] Yuan Xu, *Orthogonal polynomials of several variables*, Encyclopedia of Mathematics and its Applications 81 (2001).
- [94] Andrew Chi-Chih Yao, *How to generate and exchange secrets*, Proceedings of the 27th Annual Symposium on Foundations of Computer Science (Washington, DC, USA), SFCS '86, IEEE Computer Society, 1986, pp. 162–167.
- [95] J. Zhan, *Using homomorphic encryption for privacy-preserving collaborative decision tree classification*, 2007 IEEE Symposium on Computational Intelligence and Data Mining, March 2007, pp. 637–645.
- [96] Q. Zhang, L. T. Yang, and Z. Chen, *Privacy preserving deep computation model on*

cloud for big data feature learning, IEEE Transactions on Computers 65 (2016), no. 5, 1351–1362.

- [97] Yuchen Zhang, Wenrui Dai, Xiaoqian Jiang, Hongkai Xiong, and Shuang Wang, *Foresee: Fully outsourced secure genome study based on homomorphic encryption*, BMC Medical Informatics and Decision Making 15 (2015), no. 5, S5.