KU LEUVEN

An Investigation of Polynomial Activation Functions in Neural Networks

Yingshuo Xi

Thesis submitted for the degree of Master of Science in Artificial Intelligence, option Engineering and Computer Science

Supervisors:
Prof. Matthew Blaschko
Prof. Frederik Vercauteren

Assessors:
Dr. Wouter Castryck
Ir. Dusan Grujicic

Assistant-supervisors:
Ir. Junyi Zhu
Ir. Jiayi Kang
Ir. Robin Geelen

Academic year 2022 – 2023

Preface

Combining cryptography and machine learning, this thesis project was exciting but challenging. I am very grateful to everyone who has guided and helped me along the way.

I am deeply thankful to my supervisors, Prof. Matthew Blaschko and Prof. Frederik Vercauteren, who proposed this interesting topic and offered it to me. Your critical guidance kept my project always on the right track. I appreciated my assessors, Dr. Wouter Castryck and Dusan Grujicic, for reading and evaluating this thesis.

I would like to express my heartfelt gratitude to my daily supervisors, Jiayi Kang, Junyi Zhu, and Robin Geelen. Your continuous support, help, and guidance have encouraged me to overcome many difficulties during the process. I enjoyed a lot in every regular meeting throughout the whole academic year.

Finally, I want to thank my family and friends for their company and love. This thesis is dedicated to my dearest grandmother and the most important friend in my life, Wangdong Guo.

Yingshuo Xi

Contents

Abstract

Homomorphic Encryption (HE) enables computations on encrypted data, offering secure data processing when combined with Machine Learning (ML). In the context of HE, Processing with non-linear activation functions is a challenge. Existing CNN/MLP studies are categorized by their solutions to this challenge. Moreover, this thesis derives a fully general expression for polynomial activation functions and proposes an adjustable form for such functions within neural networks. A two-hidden-layer neural network using the proposed fourth or fifth-order functions achieved 96.3% accuracy on the unencrypted MNIST dataset. However, increasing polynomial orders yielded few accuracy improvements but higher computational costs.

List of Figures and Tables

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

ML	Machine Learning
MLP	Multi-Layer Perceptron
NN	Neural Network
ANN	Artificial Neural Network
CNN	Convolutional Neural NetworK
DNN	Deep Neural Network
ReLU	Rectified Linear Unit
LReLU	Leaky Rectified Linear Unit
PReLU	Parametric Rectified Linear Unit
MOR	Multi-output Regression
MSE	Mean Squared Error
MSD	mean squared deviation
BCE	Binary Cross-Entropy

HE	Homomorphic Encryption
FHE	Fully Homomorphic Encryption
BGV	Brakerski-Gentry-Vaikuntanatha
BFV	Brakerski-Fan-Vercauteren
GSW	Gentry-Sahai-Waters
FHEW	Fast Fully Homomorphic Encryption over the Weyl-Heisenberg group
TFHE	Torus Fully Homomorphic Encryption
CKKS	Cheon-Kim-Kim-Song
LWE	Learning With Errors
RLWE	Ring Learning With Errors
LUT	Lookup Table
NoA	Number of Additions
NoM	Number of Multiplications
DoM	Depth of Multiplications

Symbols

$x_i, a_i^{(0)}$	i^{th} input of NN
y_i	real/targeted value of i^{th} output
$\widehat{y_i}$	predicted value of i^{th} output
$\sigma(x)$	Activation function
$w_{ij}^{(n)}$	Weight on i^{th} input in j^{th} perceptron of layer n
$b^{(n)}$	Bias of n^{th} layer
$\tilde{a}_i^{(n)}$	Intermediate parameter within i^{th} perceptron of layer n
$a_i^{(n)}$	the output of the i^{th} perceptron in the n^{th} layer ($i > 0$)
N_P, N_Q	Sub-network mappings with parameter sets P and Q
$F_B^{(n)}$	j^{th}-order polynomial mapping with coefficient set
m	Number of samples
o	Number of perceptrons in the output layer
$\mathbb{Z}$	Integers

Chapter 1

Introduction

1.1 Motivation

As per the findings presented by DLA Piper, an international law firm that has diligently monitored the reported instances of data breaches within the EU, the occurrence of data breaches has exhibited a continuous pattern of growth for four "consecutive years", starting from the implementation of the General Data Protection Regulation (GDPR) on 25th May 2018. Notably, this trend only decreased in the preceding year (2022). According to their 2023 report, a total of 109,000 personal data breaches were officially notified to regulatory authorities in 2022. [50]

As the proverb says, "Better safe than sorry." In the current environment, the protection of information has become particularly difficult. The client's crisis of trust in the data processor has begun to intensify. Therefore, there could be potential clients willing to cost extra for data analysis for those highly sensitive data (such as hospital information and company confidential data) to ensure the extreme security of the information. Machine Learning (ML) combined with Homomorphic Encryption (HE) makes this possible. This technology enables data processors to manipulate data without accessing the data's actual content.

While HE offers potential, it is currently limited in its support for only a subset of operations utilized in contemporary ML. Consequently, the quest for alternative approaches to substitute for unsupported operations has become a primary driver for much research in this field. This thesis focuses on the research concerning the replacement of non-linear activation functions in neural networks.

1.2 Background Overview

Encryption is a mathematical tool to protect information during the process of propagation. HE further expands the encryption function and allows the numerical information to be directly processed (by addition and multiplication operations) without decryption. It provides the possibility to pursue privacy-preserving processing of ML on encrypted data. However, common activation functions in ML, like the Sigmoid function, are not representable as low-degree polynomials, whose evaluations

are infeasible for HE schemes such as BGV [7], BFV [16], and CKKS [11]. The exploration of the corresponding alternatives is the main focus of this thesis. All the detailed background related to HE and ML will be discussed in Chapter 2 and Chapter 3.

1.3 Thesis Topics and Objectives

The mathematical properties of HE dictate that it yields identical results for supported operations on encrypted and non-encrypted data. Given the scope of the workload of this thesis, the experiments conducted are exclusively focused on non-encrypted data. The evaluation of encrypted data aimed at achieving consistent results will be addressed in future research endeavors. This thesis aims to identify a HE-friendly activation function form and develop the corresponding training methodologies for the fully connected neural networks, i.e., MLPs.

1.4 Chapters Overview

First, Chapters 2 and 3 provide background introductions to HE and ML. Then, Chapter 4 introduces how the current Artificial Neural Networks (ANNs) based on HE solve the issue of non-linear activation functions. Furthermore, one of the core chapters, Chapter 5, encompasses deriving a fully general expression for activation functions. Moreover, Chapter 6 presents the proposed form of the activation function, along with the selected datasets, network architecture, and testing objectives. In addition, results are presented in Chapter 7, while the corresponding discussion is written in Chapter 8. Finally, the conclusion of the whole thesis is given in the last chapter, Chapter 9.

1.5 Main contributions

This thesis makes three primary contributions. Firstly, this thesis provides a new perspective in the literature review (Chapter 4) by categorizing the existing research according to the varied approaches to non-linear activation processing on HE. Secondly, this thesis extensively investigates the general forms of polynomial activation function from second to fifth order in Chapter 5. This exploration confirms the rationale behind choosing activation functions in earlier works and inspires the idea for new activation functions. Thirdly, this thesis introduces a new form of activation function with adjustable coefficients, followed by the corresponding testing experiments in Chapter Methodologies (Chapter 6).

Chapter 2

From Cryptography to Fully Homomorphic Encryption

2.1 Cryptography Overview

In Greek, the word *Cryptography* means 'hidden writing'. Generally, cryptography is the study of exploring and evaluating strategies that protect confidential private messages from the public or third parties [46]. It covers several protocols, algorithms, and techniques to enable consistency between senders and receivers and postpone illegal access to sensitive data. There are five vital aspects of cryptography [21]:

1. **Privacy/confidentiality**: Ensuring only the designated receiver can get access to the information;
2. **Authentication**: Identity verification;
3. **Integrity**: Guaranteeing the received message is not altered from the original during propagation;
4. **Non-repudiation**: Method to prove the authenticity of the sender;
5. **Key exchange**: Strategies to share the crypto keys between sender and receiver.

In the context of cryptography, the basic process of spreading data can be divided into encryption and decryption. From the unencrypted data, referred to as plaintext (P), the sender performs encryption to obtain the ciphertext (C) using a key $k1$. The receiver will decrypt the ciphertext back to plaintext after receiving the message using key $k2$. These two processes can be expressed by:

$$\begin{aligned} C &= Enc_{k1}(P) \\ P &= Dec_{k2}(C) \end{aligned} \tag{2.1}$$

where P is plaintext, C is ciphertext, Enc is encryption, Dec is decryption, and $k1$, $k2$ are crypto keys.

2.1.1 Cryptosystem Types

There are three main types of algorithms in cryptographic systems (also known as cryptosystems) depending on the formation of the keys [3, 21]:

1. **Secret key cryptography (symmetrical encryption)**: *key1* and *key2* in eq. (2.1) are the same. However, this might bring the potential risk since *Key Exchange* (the fifth aspect of cryptography) might be attacked while passing the key from the sender to the receiver.

2. **Public key cryptography (asymmetrical encryption)** *key 1* and *key 2* are different. Messages encrypted by *key 1* can only be decrypted by its paired *key 2*. In this case, *key 1* is used as a public key to encrypt plaintext or verify a digital signature, while *key 2* is considered a private key to decrypt ciphertext or create a digital signature. Here, digital signatures verify the *Authentication*, *Integrity*, and *Non-repudiation* of digital messages or documents.

3. **Hash Functions**: The encryption process of hash functions does not need a key, and the generated ciphertext is not recoverable. It converts the message of any length into a corresponding certain fixed-length sequence of numbers or letters, known as a hash value or hash code, to ensure *Integrity*.

2.1.2 Security and Key Length

There are two main types of attacks on a ciphertext: brute force and cryptanalysis [44]. A brute force attack means the attacker generates all the potential keys and tests them on the ciphertext to acquire comprehensible results. In contrast, a cryptanalysis attack refers to understanding the mathematical mechanism of a specific encryption pattern.

Key length, also known as key size, is the size of the cryptographic key. The number of the potential key option is proportional to the key length. Modern computers usually launch brute force attacks [21]. As the key length increases, the difficulty of cracking the ciphertext rises, leading to better security.

2.2 Homomorphic Encryption (HE)

2.2.1 HE Overview and Motivation

Homomorphic encryption (HE) is a unique form of asymmetrical cryptography, which allows the processing of the ciphertexts without getting access to the plaintexts.

Some highly sensitive data with large sizes, like medical information or imaging, simultaneously require strong computational processes and high confidentiality. Sometimes, the confidentiality of these data is the most important of all. Thus, the accountability of corresponding regulations like GDPR and the pre-signed agreement provide legal protection. However, data leakage caused by human factors cannot be completely avoided. For instance, predictive analytics in the healthcare industry might be challenging to implement through a third-party service provider regarding

privacy issues. HE gives the probability that third-party data processors can only provide the computation service without learning any information from the encrypted data. It can enormously reduce the privacy concern and highly increase security from the source. [40]

2.2.2 HE Types

There are three common types of HE [10]:

1. **Partially HE (Somewhat HE)**: Only support either addition or multiplication, which is the weakest notion of HE;
2. **Leveled fully HE**: Support arbitrary operation but with predetermined depth;
3. **Fully HE (FHE)**: Support arbitrary computation, which is the strongest notion of HE.

2.2.3 HE Parameters and Security

Similar to the key length in common cryptography, more parameters are required to be selected in HE. These parameters are related to the security level of encryption, the plaintext type, and computation operations. Two important HE parameters are [10]:

1. ***n dimension***, similar to key length, larger n means better security;
2. ***q ciphertext modulus***, larger q means lower security but leaves more evaluation space.

Generally, ciphertexts in HE contain a noise component that is rapidly growing while applying more operations. The ciphertext can be correctly decrypted only if the noise is smaller than the ciphertext modules. In other words, large q allows more operations on the ciphertexts. However, it is opposite to the security level.

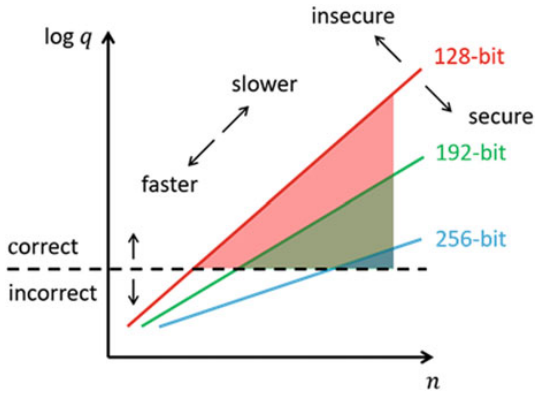


FIGURE 2.1: Relationship Among HE Parameters[10] (n: ciphertext dimension, q: ciphertext modulus, color: key length)

Figure 2.1 represents the security level related to the common HE parameters. The same as normal encryption, a larger key length means better security. Increasing n or decreasing q can also lead to a better security. Nevertheless, as mentioned before, decreasing q means less operation can be performed due to noise accumulation.

2.3 General Background and Some FHE Schemes

2.3.1 FHE Development Overview

In 2009, the first feasible FHE scheme was proposed by Craig Gentry using lattice-based cryptography[23], which is considered as the beginning of the first-generation FHE. Two years later, the proposal of the Brakerski-Gentry-Vaikuntanathan (BGV) scheme by Brakerski Zvika at al. [7] was a crucial step in the history of FHE, opening the era of second-generation FHE. The following similar Brakerski-Fan-Vercauteren (BFV) scheme [16] is also considered in this generation. Later, In 2013, a new Gentry-Sahai-Waters (GSW) [24] scheme was proposed. This scheme and its variants, Fast Fully Homomorphic Encryption over the Weyl-Heisenberg group (FHEW)[15] and Torus Fully Homomorphic Encryption(TFHE)[12], are considered as the third-generation FHE. Later in 2016, the fourth-generation FHE Cheon-Kim-Kim-Song (CKKS) [11] scheme was published, which supports the fixed-point arithmetic operations. Its efficient rescaling operation after multiplication is functionally similar to the bootstrapping of second-generation FHE.

2.3.2 Basic Notations [33]

The ring $\mathbb{Z}/t\mathbb{Z}$ is denoted as $\mathbb{Z}_t$ for an integer t. Any integer $a \in \mathbb{Z}$ is mapped into its representative $\mathbb{Z}_t$ through notation $[\cdot]_t$.

X denotes an indeterminate of polynomials. $R = \mathbb{Z}[X]/(f(X))$ represents a polynomial ring where $f(X) \in \mathbb{Z}[X]$, and $f(X)$ means the ideal generated by the element $f(X)$.

2.3.3 Learning With Errors (LWE) Problem

Learning With Errors (LWE)[45] is a core mathematical problem in cryptography related to encryption algorithms. The information is encrypted in a set of formulas with errors [36].

Dimension parameter n and modulus parameter q are positive integers. m denotes the number of samples. χ is a probability distribution over rational integers. Let $\mathbb{Z}_q$ be the ring $\mathbb{Z}/q\mathbb{Z}$, and $\mathbb{Z}_q^n/\mathbb{Z}_q^m$ be length-n/length-m vectors over $\mathbb{Z}_q$.

The definition of LWE assumption is as follows:

Definition 2.3.1 (LWE Assumption[1])**.** *LWE problem indicates the following two distributions are computationally indistinguishable.*

$$\textit{Distribution 1} = (A, A\boldsymbol{s} + \boldsymbol{e})$$

where $A \in (\mathbb{Z}_q)^{m \times n}$ *is a uniformly random matrix with size* $m \times n$*, uniformly random vector* $\boldsymbol{s}$ *is from the vector space* $\mathbb{Z}_q^n$*, vector* $\boldsymbol{e}$ *is chosen from the distribution* χ*.*

$$\textit{Distribution 2} = (A, \boldsymbol{c})$$

where $A \in (\mathbb{Z}_q)^{m \times n}$ *is still the uniformly random matrix with size* $m \times n$*, and* $\boldsymbol{c}$ *is a random vector from* $\mathbb{Z}_q^m$*.*

Furthermore, the LWE problem is defined by

Definition 2.3.2 (LWE Problem[20, 37])**.** *LWE problem tries to find* $\boldsymbol{s} \in Z_q^n$ *and* $A \in (Z_q)^{m \times n}$ *such that*

$$A\boldsymbol{s} + \boldsymbol{e} = \boldsymbol{b} \mod q$$

where $\boldsymbol{e} \in Z_q^m$ *is chosen coordinate-wise from the distribution* χ*.*

2.3.4 Ring Learning With Errors (RLWE) Problem

Ring Learning With Errors (RLWE), or learning with errors over rings, is the LWE problems $(\mathbb{Z}_q)^{n+1}$ over finite fields $(R_q)^2$ where R is defined in previous Section 2.3.2 and R_q is the ring of R modulo q. Similar to LWE, the RLWE problem is defined as

Definition 2.3.3 (RLWE Assumption[37])**.** *RLWE problem is trying to find* $\boldsymbol{s} \in R_q$ *that*

$$(a, b = s \cdot a + e) \in R_q \times R_q$$

where a *is chosen uniformly at random in* R_q*, and* $e \in R_q$ *is sampled from distribution* χ*.*

RLWE involves polynomial rings and introduces a noise-like term, making it difficult to distinguish between polynomial values. This inherent difficulty forms the basis for the security of lattice-based encryption schemes. The FHE schemes introduced in the following sections leverage the mathematical problem of RLWE to achieve their security.

2.3.5 Some FHE schemes

Since FHE serves as the motivation for this thesis rather than its primary focus, the introduction of FHE schemes will not go into their detailed mathematical principles here.

BGV and BFV [22]

BGV and BFV are very similar FHE schemes which have the same interface. They both encrypt and operate on plaintexts $m \in R_p$ for positive integers, referred to as plaintext modulus, where R_p is denoted as the quotient ring of R modulo $p(\geq 2)$. In BFV, A typical choice of R is $\mathbb{Z}[X]/(X^n + 1)$, where n is a power of 2.

The main difference between BGV and BFV is the plaintext encoding that plaintext is encoded in the "least significant bits" of the ciphertext by the BGV

scheme, whereas in "most significant bits" by BFV. Both schemes can be considered "dual", differing slightly in how they implement homomorphic multiplication.

Furthermore, BGV and BFV support the same homomorphic operations:

- Addition: The process takes two inputs, either two ciphertexts or one ciphertext and one plaintext, producing an output ciphertext that encrypts the sum of the corresponding plaintexts. This operation is relatively cheap with little noise growth.
- Multiplication: The inputs are two ciphertexts, and the output is the ciphertext of the encrypted product to their plaintexts. The process is by tensoring the inputs, obtaining a tuple of three elements, which can be reduced back to the original standard format of two elements by key switching.

Cheon-Kim-Kim-Song (CKKS)[11]

The high-level overview of CKKS can be found in Fig. 2.2. It includes extra encoding and decoding processes between the message and plaintext. CKKS uses polynomials for a good trade-off between security and efficiency. It must be noted that CKKS allows HE operations on complex-valued vectors (real values as well).

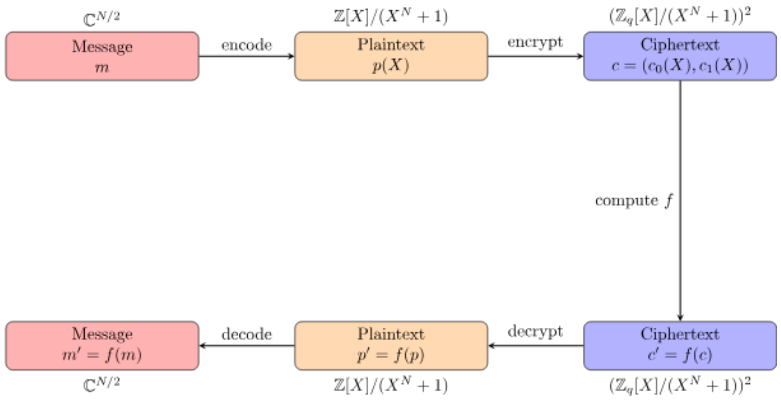


Figure 2.2: High-Level View of CKKS [29]

Relinearization in CKKS can convert the larger ciphertexts back to the original predefined size with the same decryption circuit. Moreover, another technique, Rescaling, can reduce the noise after performing homomorphic operations by dividing the ciphertext coefficients by a rescaling factor. However, rescaling need to be performed carefully because it can result in a loss of precision. In contrast, few rescaling may have excessive noise accumulation, leading to incorrect computation.

2.3.6 Conclusion

By satisfying the prerequisites of FHE (addition and multiplication operations), the mathematical properties of the FHE schemes ought to make ciphertext and plaintext operation results consistent. Therefore, considering the workload, the subsequent

phases of this thesis will focus on conducting experiments on plaintext only, while the corresponding experiments concerning ciphertexts will be mentioned in future work. Moreover, the reason for ciphertext noise growth primarily comes from multiplication operations. Consequently, even though the method is not very rigorous, counting the number of multiplication and addition operations conducted on plaintexts can serve as reasonable indicators for measuring the algorithmic and network complexity in ciphertexts.

Chapter 3

Artificial neural networks

Artificial neural networks (ANNs) are essential parts of machine learning algorithms developed in cognitive and computer science in the 1980s. An ANN is a computational model inspired by the behavior of biological neurons, generating output values from the given inputs [2]. Considering the vast background information of ANN, only the information related to this thesis' work, multilayer perceptrons (MLPs), will be introduced here.

3.1 From Biological Neurons to ANN

Fig. 3.1 represents the simplest axon-synapse-dendrite structure of biological neurons. The signal generated from the dendrites is passed through the soma and sent to axon terminals. Neurons do not work alone; they cooperate to form a network to show cognitive function. However, the cognitive functions of neurons in the human brain are super complicated, which is still not clearly figured out today. however, the basic characteristics of neuron networks can still be summarized by six points: Learning and adaptation, Generalisation, Massive parallelism, Robustness, Associative storage of information, and Spatiotemporal information processing [47].

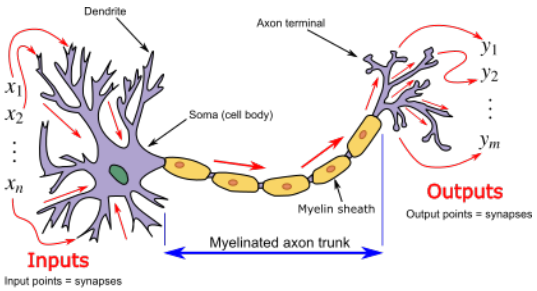


FIGURE 3.1: Biological Neurons [5]

In 1943, As a concept of perceptron, the McCulloch-Pitts neuron was proposed by McCulloch and Pitts [38]. It is a single-layer mathematical model shown in Fig. 3.2 There are different parameter weights (denoted with w) assigned to corresponding

inputs (denoted with x). After summing up the products, the result passes through the threshold function to get the final result.

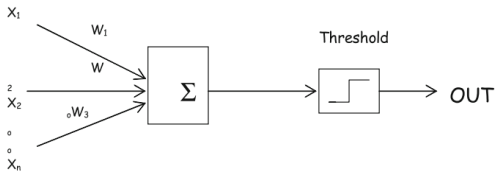


Figure 3.2: Original Perceptron (McCulloch-Pitts neuron) [43]

Although perceptron learning ability has been proven to solve all the tasks as long as it can represent [52], and was given several convincing models [49, 53], the function of perceptron was very limited to its simple one-layer structure. In 1969, Minsky proved the highly restricted representability of simple perceptron and even questioned the existence of a multilayer neural network learning method[39]. Ironically, just before his book was published, the concept of multilayer perceptrons (MLPs) [53] was already invented. From then on, ANNs developed and improved rapidly, such as backpropagation, associative memories, self-organizing networks, etc.

3.2 Modern Artificial Neuron

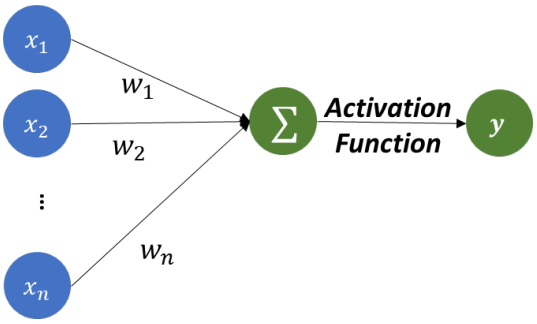


Figure 3.3: Artificial Neuron Structure

The modern artificial neuron shown in Fig. 3.3 is very similar to the McCulloch-Pitts neuron (Fig. 3.2) with the change in threshold function. This structure can also be written as

$$\tilde{y} = \sum_{i=1}^{n} (x_i w_i), \quad \hat{y} = \sigma(\tilde{y}) \tag{3.1}$$

where x_i is the i^{th} input, w_i is the weight to the corresponding x_i, σ is the activation function which will be discussed in Sec. 3.3, $\tilde{y}$ is the intermediate parameter of the linear combination result, and $\hat{y}$ is the predicted output. It can be also written in matrix form as Eq. 3.2

$$y = \sigma\left(W^T X\right) \tag{3.2}$$

where W and X are column vectors of the same length.

3.3 Activation Function

An activation function determines the activated status of a specific neuron. It simply projects a particular combination of inputs to its output, indicating the significance of its contribution to the final prediction.[51] The commonly used activation functions can be divided into three different categories [17]:

1. **Ridge functions**: multivariate functions applied on the linear combinations of the input variables;
2. **Radial functions**: a series of radial basis functions (RBFs) that are used in RBF networks, which are efficient as universal function approximators;
3. **Fold functions**: fold function is typically used in the pooling layers of convolutional neural networks (CNNs) or the network output layers of multi-class classification. They usually are not in analytical expression but perform an aggregation over the inputs like mean, min, max, etc.

Considering this thesis only focuses on fully connected neural networks, only some ridges activation functions will be discussed below. For backpropagation (Sec. 3.4.4), the derivation of activation functions is involved and will be introduced here.

3.3.1 Linear function

A linear function is the most straightforward activation function of all. Considering the combination of inputs and corresponding weights in a perceptron is already linear, the linear activation function serves the same purpose, which is mathematically useless. However, some libraries require the user to always specify an activation function. The option linear function is used as default, meaning not to apply any activation function. The function and its derivatives are:

$$\begin{aligned} \sigma(x) &= x \\ \frac{d\sigma(x)}{dx} &= 1 \end{aligned} \tag{3.3}$$

3.3.2 Sigmoid(Logistic) function

The sigmoid function indicates a set of mathematical functions with an "S"-shaped curve, referred to as s sigmoid curve [19]. However, in NN, the Sigmoid activation function specifically means Eq. 3.3.2

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

The sigmoid function is plotted in Fig. 3.4. Sigmoid function is fully bounded and differentiable across all the real input x. Moreover, it has a non-negative derivative at each point [27]. In addition, the derivative of a sigmoid function can be expressed

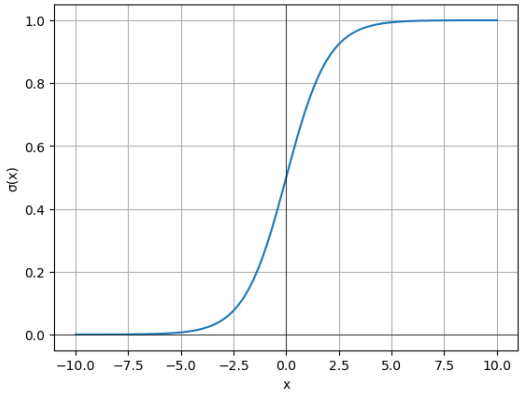


Figure 3.4: Sigmoid Function

in terms of the original function, which can further accelerate the computation in NN:

$$\begin{aligned} \frac{d\sigma(x)}{dx} &= \frac{1}{1+e^{-x}} - \frac{1}{(1+e^{-x})^2} \\ &= \sigma(x)(1-\sigma(x)) \end{aligned} \tag{3.4}$$

3.3.3 Rectified Linear Unit (ReLU)

ReLU function is the most typical piecewise activation function, which is one of the most successful attempts by its fast computation. The function and its derivatives are:

$$\sigma(x) = \begin{cases} x & \text{if } x > 0, \\ 0 & otherwise. \end{cases}$$

$$\frac{d\sigma(x)}{dx} = \begin{cases} 1 & \text{if } x > 0, \\ 0 & otherwise. \end{cases}$$

The derivative at point 0 does not exist, which can be chosen by common values like 0, 0.5, or 1 depending on different implementations. Because the property of ReLU is not fully differentiable, the softplus function (also called SmoothReLU, an approximate ReLU function) is proposed:

$$\sigma(x) = ln(1+e^x)$$

$$\begin{aligned} \frac{d\sigma(x)}{dx} &= \frac{1}{1+e^{-x}} \\ &= \sigma_{\text{sigmoid}}(x) \end{aligned} \tag{3.5}$$

Moreover, some other functions like Leaky Rectified Linear Unit (LReLU) and Parametric Rectified Linear Unit (PReLU) are further derived from ReLU, which are very similar in shape but with different purposes of usage, which are all plotted in Fig. 3.5.

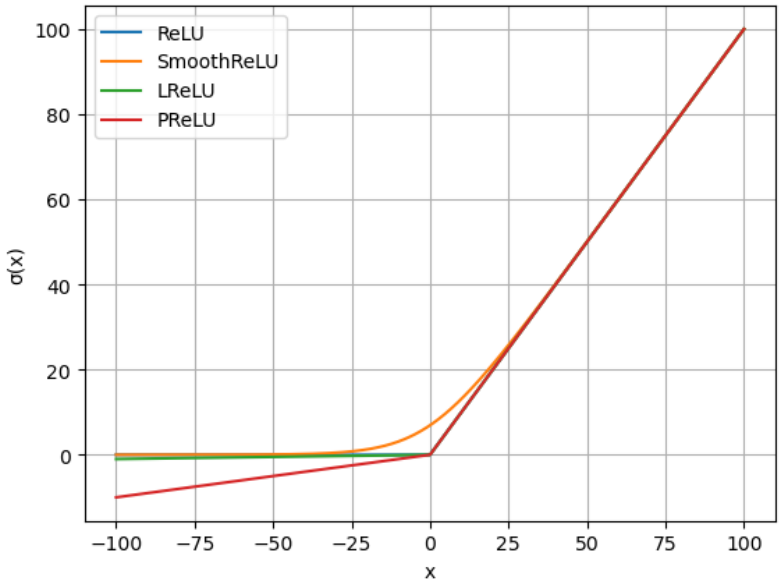


FIGURE 3.5: Plot of ReLU, SmoothReLU, LReLU, and PReLU

3.4 Multi-Layer Perceptrons and Neuron Networks

Multi-layer perceptrons (MLPs) are a subclass of ANNs typically a fully connected feedforward network. An MLP consists of three parts of layers: an input layer, one or multiple hidden layers, and an output layer, shown in Fig. 3.6. Except for the input layer, every layer contains multiple perceptron working in parallel. The inputs of every perceptrons are all the outputs from the front layer (or also with bias depending on the settings). In other words, the inputs of the perceptrons of the same layer are the same. The lines connected between two perceptrons across two layers are the weight applied on the corresponding inputs, which are all unique and adjustable. Some notations here are:

- x_i or $a_i^{(0)}$: the i^{th} input $(i > 0)$
- $b^{(n)}$: the bias term of the n^{th} layer
- $a_i^{(n)}$: the output of the i^{th} perceptron in the n^{th} layer $(i > 0)$
- $w_{ij}^{(n)}$: the weight applied on i^{th} input in j^{th} perceptron of n^{th} layer.

It needs to be noticed that because of the specificity of the input layer, it is not considered a standard layer in this paper (or written as layer 0). As a result, layer 1 of this paper will mean the first hidden layer.

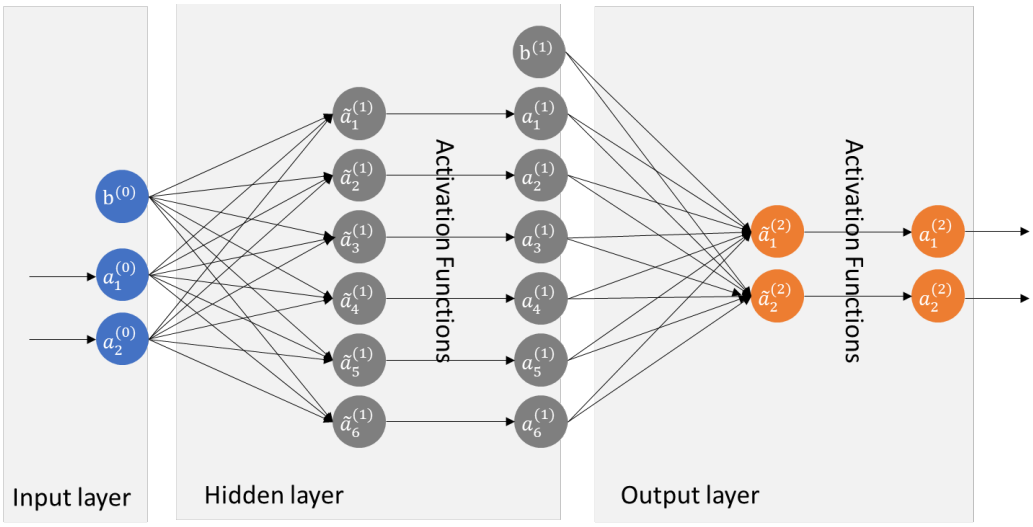


Figure 3.6: The structure of MLP with one hidden layer

The training of MLPs are divided into two parts: feedforward prediction and backpropagation, which will be detailed later in this section.

3.4.1 Classification v.s. Regression

Tasks can be easily divided into two types based on their aims: classification or regression. In machine learning, plenty of algorithms are designed for these two types of problems. Only the parts related to the topic of this thesis (NNs) will be introduced here.

Regression

In regression tasks, the outputs are one or multiple continuous real numbers. Regression algorithms are trying to find the best mapping between the input domain and output domain.

For multiple-output tasks in MLP, multiple single-output regressions customize different networks for different outputs with certain flexibility and scalability but require large computational power. In contrast, it is also possible to do multi-output regression (MOR) in one network by setting more than one perceptron in the output layers. This can be very computationally efficient and further discover the relationship between inputs and outputs. However, MOR requires a relatively large network, which is easier for overfitting.

Classification

In classification tasks, the output variables are discrete numbers that indicate different classes, not real values. There are no requirements for the input data, which can be discrete, continuous, categorical, or a combinations of these three. Typically for categorical input data, every category needs to be set as a corresponding input whose value is 0 (not belonging to this category) or 1 (belongng to this category).

Binary classification is the most common problem for separating samples into two groups. Moreover, similar to regression, there are two options for the problems of multiple classes: using multiple binary classifiers or one multi-class classifier. In MLP, the way to do classification is different from regression. Manually defining the output intervals as classes using a regression model is unreasonable since the output values refer to the classes instead of the real number. Commonly, the number of outputs is set equal to the number of classes, where 1 indicates true while 0 means not (this can also be reversed since the number is the symbol of class, not the real value), which is referred to as one-hot encoding. The predicted class is the serial number of the largest output.

3.4.2 Loss function

The loss function, also known as the cost function or the error function, is a customized mathematical objective function trying to give the best way to represent the "cost" of the given problem. The optimization problem attempts to minimize its loss function to acquire the optimal result. Depending on the application of the network, different loss functions are designed for regression or classifications. The most commonly used loss functions [30] are:

- **Classification**: 1) Binary Cross-Entropy Loss, 2) Hinge Loss.
- **Regression**: 1) Mean Square Error, 2) Mean Absolute Error, 3) Huber Loss, 4) Log-Cosh Loss, 5) Quantile Loss.

Binary cross-entropy loss and mean square error are essential in this topic, introduced in detail below. Similar to the activation function in Sec. 3.3, the derivative of the loss functions is also needed in backpropagation (Sec. 3.4.4), which will also be given here. Some notations here are:

- m : the number of samples
- o: the number of output perceptrons
- $\widehat{y_i}$: the predicted value of the i^{th} output
- y_i : the real value of the i^{th} output

To simplify the expression later, $\widehat{y_i}$ now is defined as a function of any independent variables x with parameter w, written as

$$\widehat{y_i} = f_w(x)$$

Mean Squared Error (MSE)

Mean squared error (MSE), or mean squared deviation (MSD), is the average of the squares of the errors, whose squaring operation is vital to decrease the complexity of negative signs making the function differentiable across all real numbers.

The MSE loss function of multiple outputs is calculated as the average loss of every output

$$L = \frac{1}{o}\sum_{i=1}^{o}\left(\widehat{y}_i - y_i\right)^2$$

Furthermore, the derivative of loss respected to the parameter w is

$$\frac{dL}{dw} = \frac{1}{o}\sum_{i=1}^{o}\left(\frac{\partial L}{\partial \widehat{y}_i}\frac{d\widehat{y}_i}{dw}\right) = \frac{2}{o}\sum_{i=1}^{o}\left(\left(\widehat{y}_i - y_i\right)\frac{d\widehat{y}_i}{dw}\right) \tag{3.6}$$

In addition, given a dataset with m samples, the MSE loss of the whole dataset is written as

$$L = \frac{1}{mo}\sum_{1}^{m}\sum_{i=1}^{o}\left(\widehat{y}_i - y_i\right)^2 \quad \text{and} \quad \frac{dL}{dw} = \frac{2}{mo}\sum_{1}^{m}\sum_{i=1}^{o}\left(\left(\widehat{y}_i - y_i\right)\frac{d\widehat{y}_i}{dw}\right)$$

To simplify the expression, the subscripts related to the samples are omitted.

Binary Cross-Entropy (BCE)

For a given set, the discrete cross-entropy of the distribution q relative to a distribution p [18] is defined as:

$$H(P,Q) = -\sum_{x\in\chi} p\left(x\right)\log q\left(x\right)$$

In the case of binary cross-entropy, $\chi = \{y = 1, y = 0\}$, $p(x)$ is the probability distribution of the true value (y), while $q(x)$ is the probability distribution of predicted value $(\widehat{y})$, which are listed in Table 3.1.

$x \in \chi$	$x : y = 1$	$x : y = 0$
$p(x\|y = 1)$	1	0
$p(x\|y = 0)$	0	1
$q(x)$	$\widehat{y}$	$1 - \widehat{y}$

Table 3.1: Probability Distribution for $p(x)$ and $q(x)$ in Binary Cross-Entropy

After substituting the variables in Eq. 3.4.2 by values in Table 3.1. the equation is converted into

$$H(\widehat{y}, y) = -\left(y\log\widehat{y} + (1-y)\log\left(1-\widehat{y}\right)\right)$$

As a result, the BCE, referred to as log loss in machine learning, is defined as

$$L = -\frac{1}{o}\sum_{i=1}^{o}\left(y_i\log\widehat{y}_i + (1-\widehat{y}_i)\log\left(1-\widehat{y}_i\right)\right)$$

The derivative of loss respected to variable x can be expressed by

$$\frac{dL}{dw} = \sum_{i=1}^{o} \left(\frac{\partial L}{\partial \widehat{y}_i} \frac{d\widehat{y}_i}{dw} \right) = -\frac{2}{o} \sum_{i=1}^{o} \left(\left(\frac{y_i}{\widehat{y}_i} - \frac{1-y_i}{1-\widehat{y}_i} \right) \frac{\widehat{y}_i}{dw} \right) \tag{3.7}$$

Similar to MSE, given a dataset with m samples, the BCE loss of the whole dataset can be expressed by

$$\begin{aligned} L &= -\frac{1}{mo} \sum_{1}^{m} \sum_{i=1}^{o} \left(y_i \log \widehat{y}_i + (1-\widehat{y}_i) \log \left(1-\widehat{y}_i\right) \right) \\ \text{and} \quad \frac{dL}{dw} &= -\frac{2}{mo} \sum_{1}^{m} \sum_{i=1}^{o} \left(\left(\frac{y_i}{\widehat{y}_i} - \frac{1-y_i}{1-\widehat{y}_i} \right) \frac{\widehat{y}_i}{dw} \right) \end{aligned} \tag{3.8}$$

To simplify the expression, the subscripts related to the samples are omitted.

It needs to be noticed here that BCE is typically utilized together with the Sigmoid activation function whose output range is $(0, 1)$. However, for other activation functions, the outputs might be unbounded ($\pm\infty$), leading to infinite loss or undefined $\log(\text{neg})$ in Eq. 3.4.2. The problem is solved by clipping to a predefined small replacement (e.g. -100) if it happens. Moreover, the output values might continuously cross 0 or 1, which may cause the denominator in Eq. 3.7 becoming zeros. To solve this problem, a small value ϵ is introduced to replace $\widehat{y}_i$ and $(1-\widehat{y}_i)$ by $\max(\widehat{y}_i, \epsilon)$ and $\max((1-\widehat{y}_i), \epsilon)$ to avoid the situation of infinite value. These special cases are all considered in advanced DNN libraries like TensorFlow or PyTorch. [42]

However, given the computational expense of such operations within FHE is very costly, the typical classification loss MSE will replace BCE in this thesis.

3.4.3 Feedforward

Recalling the modern artificial neuron expression (Eq. 3.1) and the structure of MLP (Fig. 3.6), the forward training progress calculates the outputs of every layer in series from the input layer until the output layer. As a result, Eq. 3.1 can be written in a more general form between the i-length layer$^{(n-1)}$ and the j-length layer$^{(n)}$, both with bias:

$$\tilde{a}_j^{(n)} = \sum_{i=0}^{n} \left(a_{ij}^{(n-1)} w_{ij}^{(n)} \right) \quad a_j^{(n)} = \sigma^{(n)} \left(\tilde{a}_j^{(n)} \right) \quad (n > 1) \tag{3.9}$$

Specifically, the perceptrons in the same layer work in parallel. Thus, Eq. 3.9 can be further derived in matrix form:

$$\tilde{A}^{(n)} = \left(A_{bias}^{(n-1)} \right)^T W^{(n)}, \quad A^{(n)} = \sigma^{(n)} \left(\tilde{A}^{(n)} \right), \quad \text{where} \quad A_{bias}^{(n-1)} = \begin{bmatrix} b^{(n)} \\ A^{(n-1)} \end{bmatrix} \tag{3.10}$$

Here, $A^{(n)}$ is the column vector of all the outputs of layer$^{(n)}$, $W^{(n)}$ is the weight matrix between layer$^{(n)}$ and layer$^{(n-1)}$, and $b^{(n)}$ is the bias term of layer$^{(n)}$. To make it clearer, these parameters can be further expressed as:

$$A^{(n)} = \begin{bmatrix} a_1^{(n)} \\ a_2^{(n)} \\ \vdots \\ a_j^{(n)} \end{bmatrix} \quad A_{bias}^{(n-1)} = \begin{bmatrix} b^{(n)} \\ a_1^{(n)} \\ a_2^{(n)} \\ \vdots \\ a_i^{(n)} \end{bmatrix} \quad W^{(n)} = \begin{bmatrix} w_{01}^{(n)} & w_{02}^{(n)} & \cdots & w_{0j}^{(n)} \\ w_{11}^{(n)} & w_{12}^{(n)} & \cdots & w_{1j}^{(n)} \\ w_{21}^{(n)} & w_{22}^{(n)} & \cdots & w_{2j}^{(n)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{i1}^{(n)} & w_{i2}^{(n)} & \cdots & w_{ij}^{(n)} \end{bmatrix} \tag{3.11}$$

3.4.4 Backpropagation

Backpropagation is a famous application of the Leibniz chain rule [34]. It calculates the gradients (derivatives) of the selected loss function respected to every adjustable weight parameter. One iteration backprpagation contains three different steps:

1. The process of feedforward is applied to acquire the output $\widehat{y}$.
2. The gradients of every weight are calculated.
3. The weights subtract the corresponding gradients times an adjustable predefined step size, also called learning rate. in the hyperparameter of optimization algorithms.

Computing the gradients of the loss function respected to the parameters is the first step of backpropagation. First and foremost, depending on the selection of loss function, the derivatives of loss function respected to different outputs $dL/d\widehat{y}_i$ are calculated, which can also be written as $dL/da^{(\text{last layer})}$ from Fig. 3.6. The derivative of MSE and BCE are already computed in Eq. 3.6 and Eq. 3.7.

For the n^{th} layer internal, the derivative of outputs respected to the intermediate parameter $\tilde{a}^{(n)}$ is

$$\frac{da_j^{(n)}}{d\tilde{a_j}^{(n)}} = \frac{d\sigma^{(n)}\left(\tilde{a_j}^{(n)}\right)}{d\tilde{a_j}^{(n)}} \tag{3.12}$$

Here, the result depends on the choice of activation function, which is computed in Eq. 3.3 , Eq. 3.4 , and Eq. 3.5 for Linear, Sigmoid, and ReLU, respectively.

Then, the derivative of $\tilde{a_j}^{(n)}$ (intermediate parameters) respected to $w_{ij}^{(n)}$ (targeted weight) is computed as:

$$\frac{\partial \tilde{a}_j^{(n)}}{\partial w_{ij}^{(n)}} = a_{ij}^{(n-1)} \tag{3.13}$$

Moreover, the relationship between the adjacent layers should also be established based on the summation of partial derivatives. The derivative of $\tilde{a_j}^{(n)}$ (intermediate parameters in layer$^{(n)}$) respected to $a_i^{(n-1)}$ (outputs in layer$^{(n-1)}$) is:

$$\frac{\partial \tilde{a}_j^{(n)}}{\partial a_i^{(n-1)}} = w_{ij}^{(n)} \tag{3.14}$$

According to the Leibniz chain rule, the gradients of weights respected to the loss function from the last layer can be expressed as

$$\begin{aligned}
\frac{L}{dw_{ij}^{(n)}} &= \sum_{1}^{o} \frac{\partial L}{\partial a^{(n)}} \frac{da^{(n)}}{dw} = \sum_{1}^{o} \frac{\partial L}{\partial a^{(n)}} \frac{da_j^{(n)}}{d\tilde{a_j}^{(n)}} \frac{\partial \tilde{a}_j^{(n)}}{\partial w_{ij}^{(n)}} \\
\frac{L}{dw_{ij}^{(n-1)}} &= \sum_{1}^{o} \frac{\partial L}{\partial a^{(n)}} \sum_{1}^{o^{(n-1)}} \frac{da^{(n)}}{d\tilde{a}^{(n)}} \frac{\partial \tilde{a}^{(n)}}{\partial a^{(n-1)}} \frac{da_j^{(n-1)}}{d\tilde{a_j}^{(n-1)}} \frac{\partial \tilde{a}_j^{(n-1)}}{\partial w_{ij}^{(n-1)}} \\
&\vdots \\
\frac{L}{dw_{ij}^{(n-k)}} &= \sum_{1}^{o} \frac{\partial L}{\partial a^{(n)}} \sum_{1}^{o^{(n-1)}} \frac{da^{(n)}}{d\tilde{a}^{(n)}} \frac{\partial \tilde{a}^{(n)}}{\partial a^{(n-1)}} \sum_{1}^{o^{(n-2)}} \frac{da^{(n-1)}}{d\tilde{a}^{(n-1)}} \frac{\partial \tilde{a}^{(n-1)}}{\partial a^{(n-2)}} \cdots \\
&\quad \sum_{1}^{o^{(n-k)}} \frac{da^{(n-1)}}{d\tilde{a}^{(n-k+1)}} \frac{\partial \tilde{a}^{(n-k+1)}}{\partial a^{(n-k)}} \frac{da_j^{(n-k)}}{d\tilde{a_j}^{(n-k)}} \frac{\partial \tilde{a}_j^{(n-k)}}{\partial w_{ij}^{(n-k)}}
\end{aligned} \tag{3.15}$$

It can be further observed that a lot of the intermediate results (marked blue in Eq. eq:backpropagation) can be shared across the backpropagation process, which strongly accelerates the computing process.

3.4.5 Conclusion

The characteristics of backpropagation in MLPs provide the capability to compute gradients of the loss function respected to any parameters within the network. This implies that by introducing additional parameters into the activation function, the activation function can be adjustable through backpropagation. This idea lays the foundation for the subsequent chapters' exploration of coefficients adjustable polynomial activation functions.

Chapter 4

Literature Review: CNN and MLP on FHE

This chapter is mainly for the literature review of the neural network activation functions in the context of FHE. Considering the general FHE schemes are friendly to addition and multiplication, evaluating non-polynomial activation functions such as Sigmoid is usually costly. As pointed out in the paper [37], the current shortcomings of NNs using FHE are high computational complexity, low efficiencies, and inadequacy of deployment in real-world scenarios. It further suggests possible research directions are algorithm improvement and hardware acceleration. Approximation of activation functions using polynomials is one of the main parts in the former direction.

So far, the NN activation functions of FHE can be summarized into four methods: fixed low-degree polynomials like x^2, polynomial approximation of existing activation functions, lookup tables (LUTs), and non-polynomial supported compiler, which will be detailedly illustrated below. In addition, the famous datasets for the NN on FHE are all images dataset, like MNIST, CIFAR-10, and cancer images. Commonly, that the networks are chosen as convolutional neural networks (CNNs) instead of ANNs to obtain better accuracy and faster speed. In the existing research, the aims are either improving the accuracy of NN on FHE inference or reducing the time required for FHE training.

4.1 Low-Degree Polynomials

The first CNN on FHE (YASHE scheme) was implemented by Gilad-Bachrach et al. [25]. A square function (x^2) serves as the activation function. The accuracy on the MNIST dataset was achieved by 99%, but the output results were unstable which relied highly on the inputs.

Jiang et al. [32] proposed a novel matrix encoding method and an efficient evaluation strategy to speed up the computation process. Via the activation of square-function in CNN on FHE (CKKS scheme), the accuracy on the MNIST dataset was 98.1%.

Ahmad et al. [4] accelerated the CNNs running performance on encrypted data with GPUs. Using square activation function for CNN on FHE (BFV scheme). The accuracy on MNIST was 99% and 77.55% for CIFAR-10.

4.2 Polynomial Approximation

Some methods like, Tayler expansion, can convert the targeted activation function to finite order polynomials, while other approximation methods can only be performed within the selected interval like, least-squares, and the intermediate data out of the preset interval will lose control. In general, higher-degree polynomials and larger approximation intervals (if required) lead to better performance but higher computational costs.

Hesamifard et al. [28]. utilized Chebyshev polynomials and Taylor expansions for activation function approximation in the proposed CNN on FHE (BGV scheme), including ReLU, Sigmoid, and Tangent Hyperbolic functions. ReLU function with a Taylor expansion approximation performed better than Sigmoid or Tangent Hyperbolic. The accuracy on the MNIST dataset could reach to 99.25%.

Least-squared approximation of the ReLU function was proposed by Chabanne et al. [8] to deeper CNN on FHE (BGV scheme). The accuracy on the MNIST dataset achieved 99.3%, by average pooling and batch normalization technique.

Similarly, Ishiyama et al. [31] utilized least-squared approximation but on the Swish function and got 99.22% classification accuracy of the MNIST dataset and 80.48% on the CIFAR-10 dataset by CNN on FHE.

Yagyu et al. [54] performed pre-trained optimization on the coefficients of the polynomial approximation of the Mish function by CNN on FHE. The accuracy on MNIST was 99.29%, and CIFAR-10 was 66.55%.

Edward et al. [13] derived an approximation for ReLU/Swish functions to achieve maximally-sparse encodings and to minimize the approximation error. The results were 99.1% on the MNIST dataset and 75.99% on the CIFAR-10 dataset.

In the 2017 iDASH secure genome analysis competition, Chen et al. [9] used minimax polynomial approximation on the Sigmoid function to build a one-layer logistic regression network on FHE (BFV scheme). The accuracy of the cyphertext result is almost identical to the plaintext result by the Sigomid function.

4.3 Non-polynomial Supported Compiler

The nGraph-HE2 compiler by Boemer at al. [14] based on Intel nGraph [6] support non-polynomial activation functions. However, their idea was to bypass the non-linearity processing problem and shift it from the server to the client. This two-party approach combined the plaintext interaction, which required the clients to decrypt the output, compute the non-linearity in plaintext, and return the encrypted results to the compiler.

4.4 Lookup tables

Karthik et al. [41] proposed a new method by packing the data elements of the activation function within the ciphertext in lookup tables (LUTs) to avoid the non-linearity calculation from nGraph-HE2. Unlike other works using CNN, they trained a fully-connected neural network (i.e., MLP) on FHE (BGV scheme) with two hidden layers and obtained 96% accuracy on the MNIST dataset.

However, Lou et al. [35] pointed out that Karthik's approach was strongly limited by its long training latency, that the BGV Sigmoid LUTs took around 98% of the training time. As a result, Lou et al. proposed a technique called Glyph to speed up the training process. Glyph is a logic-operation-friendly TFHE-based method with short latency, which is used to implement the activation function such as ReLU and softmax. By switching between TFHE and BGV/BFV, Glyph reduced 69% - 99% latency over Karthik's method and obtained 98.6% accuracy on MNIST by CNN.

4.5 Conclusion

Except for the work of Karthik, all the other works utilized CNNs to acquire good accuracy on the MNIST dataset. Besides NN inference accuracy, the problem of large time consumption in FHE training is much harder to solve. To speed up the training process, Lou's solution implemented the activation function under LUTs friendly FHE scheme. In contrast, this thesis is trying to investigate whether the activation function of MLP itself is adjustable within any given FHE scheme and can reach to a comparable accuracy to other existing methods. As a result, Karthik's work in MLP will be treated as a target in Chapter 6.

Chapter 5

Fully General Expression

This chapter explores the simplest form of fully general expression of polynomial activation function in MLPs. It is worth evaluating this part since the fully general expression of polynomial activation functions does not narrow the searching space but can potentially reduce the computations. Meanwhile, reducing the number of addition and multiplication operations in FHE leads to faster training and inference time.

5.1 Definition

The concept of fully general expression refers to the expression that covers all possible cases or allows for maximum flexibility and generality. Here, fully general expression is defined as

Definition 5.1.1 (Polynomial Fully General Expression)**.** *For the network N*

$$x \overset{N_P}{\mapsto} \overset{F_B^{(n)}}{\mapsto} \overset{N_Q}{\mapsto} f(x) \tag{5.1}$$

where N_P, N_Q are sub-network mappings with parameter sets P, Q, respectively. $F_B^{(n)}$ is a n-degree polynomial most general mapping with variable coefficient set $B = \{\beta_0, \beta_1, \ldots, \beta_{n-1}\}$, expressed as

$$F_B^{(n)}(x) = x^n + \beta_{n-1}x^{n-1} + \cdots + \beta_2 x^2 + \beta_1 x + \beta_0$$

For a proposed coefficient set $\widehat{B} = \{\widehat{\beta_0}, \widehat{\beta_1}, \ldots, \widehat{\beta_{n-1}}\}$ where $\widehat{\beta}$ elements can be fixed to a real number, if for any P, Q, and B in the old mapping in Eq. 5.1, there always exist $\widehat{P}$ and $\widehat{Q}$, with the same network structures of N_P and N_Q, for $\widehat{B}$ such that the new mapping

$$x \overset{N1_{\widehat{P}}}{\mapsto} \overset{F_{\widehat{B}}^{(n)}}{\mapsto} \overset{N2_{\widehat{Q}}}{\mapsto} f(x)$$

obtains the same results as the old one mapping, written as:

$$N_Q(F_B^{(n)}(N_P(x))) = N_{\widehat{Q}}(F_{\widehat{B}}^{(n)}(N_{\widehat{P}}(x))) \tag{5.2}$$

Then, $F^{(n)}_{\widehat{B}}$ is called the fully general expression of polynomial $F^{(n)}_{B}$ in network N.

For example, if a network N holds

$$N_P : x \mapsto p_1 x + p_0 \quad \text{and} \quad N_Q : x \mapsto x$$

Proposed $\widehat{B} = \{\widehat{\beta_0} = 0\}$, i.e. $x \mapsto x$, is the fully general expression of $F^{(1)}_{B}$ in network for the first order polynomial. Because for any B in most first-order general $F^{(1)}_{B}(x) = x + \beta_0$, there always exist a solution of $\widehat{P}$ that holds

$$p_1 x + p_0 + \beta_0 = \widehat{p_1} x + \widehat{p_0} \tag{5.3}$$

where

$$\begin{cases} \widehat{p_1} = p_1 \\ \widehat{p_0} = p_0 + \beta_0 \end{cases}$$

However, polynomial functions (mappings) like $x \mapsto (x + 1)$ or $x \mapsto (x - 0.5)$ are also the fully general expressions of $F^{(1)}_{\beta}$ in network N. In real cases, there might be infinite fully general expressions within a given network. As a result, the simplest form of the fully general expression needs to be defined.

Definition 5.1.2 (Simplest Form of Polynomial Fully General Expression)**.** *The simplest form of polynomial fully general expression is defined as the form with the minimal number of non-zero polynomial coefficients in set $\widehat{B}$ of n-degree function $F^{(n)}_{\widehat{B}}$. Moreover, it is stipulated that a lower degree term with a non-zero coefficient is simpler than a higher degree term with a non-zero coefficient.‘*

According to this definition, or example, in second-order polynomials, $x \mapsto (x^2+x)$ is simpler than $x \mapsto (x^2 + x + 1)$ for less non-zero polynomial coefficients, while $x \mapsto (x^2 + 1)$ is simpler than $x \mapsto (x^2 + x)$ for a lower degree term. As a result, the simplest form of polynomial fully general expression of the previous example is $x \mapsto x$ or written as $F^{(1)}_{\widehat{B}}(x) = x$ for the given network N.

5.2 Abstracting the MLP Structure

Inspired by the example before, to prove the given function is a fully general expression, it is not necessary to list all the possible situations but to find a solution for the equation like Equation 5.3 (or Equation 5.2 in Definition 5.1.1) within the given network. Suppose the system of equations is not solvable. In that case, some of the possible parameter combination options are excluded, and the searching space is narrowed down. This means the proposed function is not a fully general activation function, and vice versa.

This idea requires that MLP be abstracted into the standard form written in Eq. 5.1. Recalling the MLP structure Fig. 3.6 in Chapter 3, a sub-network structure focusing on only one hidden-layer activation function is shown in Fig. 5.1.

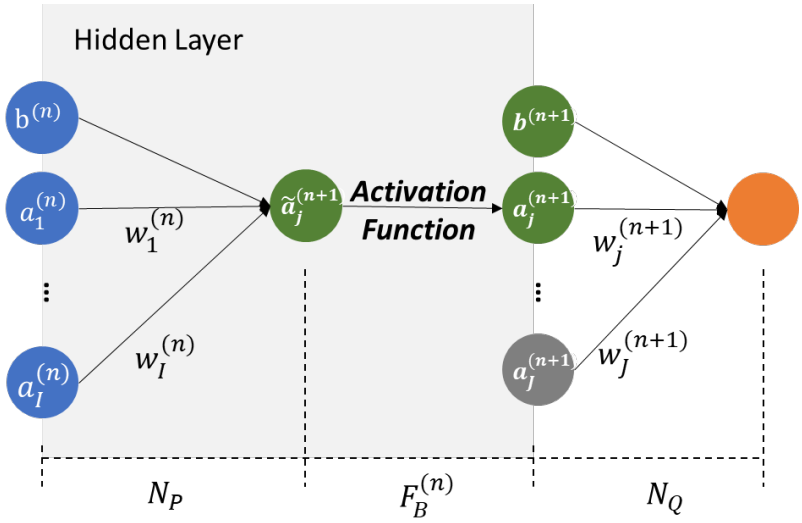


FIGURE 5.1: Sub-network Structure of Activation Function in Single Neuron of Hidden Layer

Here, the network assumes layer$^{(n)}$ contains I neurons and layer$^{(n+1)}$ contains J neurons. The intermediate variable $\tilde{a}_j^{(n+1)}$ of the j^{th} neuron ($j \in \{1 \dots J\}$) in layer$^{(n+1)}$ is

$$\tilde{a}_j^{(n+1)} = \sum_{i=1}^{I} \left(a_i^{(n)} w_i^{(n)} \right) + b^{(n)}$$

Since $w_i^{(n)} (i > 0)$ and $b^{(n)}$ are all adjustable parameters, N_P in Equation 5.1 of this case can be derived by

$$\tilde{a}_j^{(n+1)} = w_1^{(n)} \sum_{i=1}^{I} \left(a_i^{(n)} \frac{w_i^{(n)}}{w_1^{(n)}} \right) + b^{(n)}$$

$$N_P : x \mapsto p_1 x + p_0 \quad (p_1 \neq 0, p_0 \neq 0)$$

where

$$x = \sum_{i=1}^{I} \left(a_i^{(n)} \frac{w_i^{(n)}}{w_1^{(n)}} \right), \quad p_1 = w_1^{(n)}, \quad p_0 = b^{(n)}$$

Similarly, due to the existence of bias in the next layer, the mapping of N_Q is

$$N_Q : x \mapsto q_1 x + q_0 \quad (q_1 \neq 0, q_0 \neq 0)$$

It needs to be noticed that the output layer is different in N_Q mapping because there is no corresponding weight and bias of the next layer for the neuron. Thus the N_Q mapping of the output layer is

$$N_Q : x \mapsto x \tag{5.4}$$

, while N_P remains the same.

5.3 Fully General Expression of Polynomial Activation Function for Different Orders

5.3.1 Second Order

First, the fully general expression of 2nd-order polynomial activation functions in hidden layers are discussed here. The most general expression of 2nd-order polynomials is

$$F_B^{(2)}(x) = x^2 + \beta_1 x + \beta_0$$

Substitute Eq. 5.3.1 into Eq. 5.2 in Definition 5.1.1, trying to find at least one solution of $\widehat{P}$ and $\widehat{Q}$ for the proposed form $F_{\widehat{B}}^{(2)}$ that fulfill

$$N2_Q(F_B^{(2)}(N1_P(x))) = N2_{\widehat{Q}}(F_{\widehat{B}}^{(2)}(N1_{\widehat{P}}(x)))$$

Starting from the most possible simplest form $F_{\widehat{B}}^{(2)} = x^2$, the aim is to discover whether there exist sets $\widehat{P}$ and $\widehat{Q}$ make the statement Eq. 5.5 below always holds.

$$\begin{aligned} & q_1\left(\left(p_1x + p_0\right)^2 + \beta_1\left(p_1x + p_0\right) + \beta_0\right) + q_0 \\ \stackrel{?}{=}\ & \widehat{q_1}\left(\widehat{p_1}x + \widehat{p_0}\right)^2 + \widehat{q_0} \end{aligned} \tag{5.5}$$

Eq. 5.5 can be further simplified to

$$\begin{aligned} & \left(q_1\beta_2p_1^2\right)x^2 + \left(2q_1\beta_2p_1p_0 + q_1\beta_1p_1\right)x + \left(q_1\beta_2p_0^2 + q_1\beta_1p_0 + q_1\beta_0 + q_0\right) \\ \stackrel{?}{=}\ & \widehat{q_1}\widehat{p_1}^2x^2 + 2\widehat{q_1}\widehat{p_1}\widehat{p_0}x + \left(\widehat{q_1}\widehat{p_0}^2 + \widehat{q_0}\right) \end{aligned}$$

Then, the problem is transferred to check whether there is a solution that the following system of equations always holds.

$$\begin{cases} \widehat{q_1}\widehat{p_1}^2 & \stackrel{?}{=} & q_1p_1^2 \\ 2\widehat{q_1}\widehat{p_1}\widehat{p_0} & \stackrel{?}{=} & 2q_1p_1p_0 + q_1\beta_1p_1 \\ \widehat{q_1}\widehat{p_0}^2 + \widehat{q_0} & \stackrel{?}{=} & q_1p_0^2 + q_1\beta_1p_0 + q_1\beta_0 + q_0 \end{cases} \tag{5.6}$$

However, unlike a linear system of equations, it is generally hard to find the solution to a nonlinear system of equations, which can widely vary depending on different variable combinations. Therefore, the MATLAB's Symbolic Math Toolbox[48] is used here. This toolbox allows users to set symbolic parameters, and the program can solve equations, perform algebraic simplifications, etc. The *solve* function in *Equations and Systems Solver* of MATLAB tries to find a general solution or particular solution depending on the complexity of the system of equations. By setting *ReturnConditions* = *false*, the function tries to give particular solutions to the system of equations. In contrast, *ReturnConditions* = *true* indicates that the function returns the complete (general) solution of the system of equations with

variables but costs much time. Selecting *ReturnConditions* = *true*, the general results from MATLAB to the system of functions 5.6 are

$$\begin{aligned}
\widehat{p_1} &= \frac{p_1\sqrt{q_1 z}}{z} \\
\widehat{p_0} &= \frac{\sqrt{q_1 z}\,(\beta_1 + 2p_0)}{2z} \\
\widehat{q_1} &= z \\
\widehat{q_0} &= q_0 + \beta_0 q_1 - \frac{1}{4}\beta_1^2 q_1
\end{aligned}$$

where z is a variable. Although the solution contains squared roots, which gives the natural constraint $q_1 z \geq 0$, this constraint can always be fulfilled by adjusting the value of variable z. It means that $F^{(2)}_{\widehat{B}} = x^2$ is a fully general expression of the second order polynomial in the hidden layer sub-network of MLP, which is also the simplest form.

In the output layer, the mapping of N_Q is $x \mapsto x$ (Eq. 5.4). By following the same steps as the hidden layer for $F^{(2)}_{\widehat{B}} = x^2$, there is no solution for the system of equations. As a result, $F^{(2)}_{\widehat{B}} = x^2 + \beta_0$ is proposed, the current simplest form according to the definition (Definition 5.1.2). The Eq. 5.3.1 becomes

$$(p_1 x + p_0)^2 + \beta_1 (p_1 x + p_0) + \beta_0 \quad \stackrel{?}{=} \quad (\widehat{p_1} x + \widehat{p_0})^2 + \widehat{\beta_0}$$

Setting *ReturnConditions* = *true*, the general solutions from MATLAB is

$$\begin{aligned}
\widehat{p_1} &= p1 \\
\widehat{p_0} &= \frac{1}{2}\beta_1 + p0 \\
\widehat{\beta_0} &= -\frac{1}{4}\beta_1^2 + \beta_0
\end{aligned}$$

As a result, $F^{(2)}_{\widehat{B}} = x^2 + \beta_0$ is a fully general expression under the given situation. However, setting an adjustable variable in constant terms of the output layer is not feasible through gradient descent. Thus, another solution needs to be sought. The remaining option, $F^{(2)}_{\widehat{B}} = x^2 + \beta_1 x$ for the output layer activation function, needs to be tested. Setting*ReturnConditions* = *true*, the general solutions to the equation

$$(p_1 x + p_0)^2 + \beta_1 (p_1 x + p_0) + \beta_0 \quad \stackrel{?}{=} \quad (\widehat{p_1} x + \widehat{p_0})^2 + \widehat{\beta_1} (\widehat{p_1} x + \widehat{p_0})$$

are

$$\begin{aligned}
\widehat{p_1} &= p1 \\
\widehat{p_0} &= \frac{1}{2}\beta_1 + p0 \\
\widehat{\beta_1} &= \frac{-\beta_1^2 + 4\beta_0}{2\beta_1 + 4p_0}
\end{aligned}$$

which verifies that $F^{(2)}_{\widehat{B}} = x^2 + \beta_1 x$ is the fully general expression of second-order polynomial activation functions in output layer. It is important to note that the symbol $\widehat{\beta_1}$ in the equations is an adjustable parameter that must be tuned during training.

5.3.2 Third and Higher Orders

Following the same procedure, the fully general expression of third-order polynomial activation functions is firstly discussed here. $F^{(3)}_{\widehat{B}} = x^3$ is tested by the same method as the second order, which does not have a solution and is thus not the fully general expression of the hidden layer network. Then, the current most possible simplest form needs to be tested. It needs to be noted here that adding the constant parameter like $F^{(3)}_{\widehat{B}} = x^3 + \beta_0$ or in higher order cases will never work in hidden layers of the given network because the constant parameter $\widehat{\beta_0}$ will always merge in to $\widehat{q_0}$ in calculation expressed as $\widehat{q_1}\widehat{\beta_0} + \widehat{q_0}$ which is precisely functionally the same as $\widehat{q_0}$. As a result, the next alternative function is $F^{(3)}_{\widehat{B}} = x^3 + \beta_1 x$. The derivation steps are the same as before.

MATLAB cannot give a general solution (*ReturnConditions* = *true*) of third-order polynomial within half an hour. Therefore, it is artificially stipulated here that if a particular solution (*ReturnConditions* = *false*) exists, then the function is considered fully general. This derivation is not very rigorous, because the root operation will bring natural restrictions which can also narrow down the search space. Generally, as the order increases, the program search space of MATLAB will expand rapidly to find general solutions. Waiting endlessly is not a wise option. This aspect of the issue will be recorded in the limitations which can be explored in future research.

Moreover, the particular solution provided by MATLAB is exceptionally complex. Whether the solution exists is the core instead of the solution itself. Thus, the results will not be recorded in detail. It shows that $F^{(3)}_{\widehat{B}} = x^3 + \beta_1 x$ has a particular solution and is now considered the simplest form of third-order fully general expression under the given MLP network.

Similar to the second order, the third order fully general expression of activation function in output layer $F^{(3)}_{\widehat{B}} = x^3 + \beta_2 x^2 + \beta_1 x$.

For higher orders, the processes are the same, which will not be repeatedly discussed here . All the results are listed in Table 5.1

Functions	Hidden Layer	Output Layer
$F^{(2)}_{\widehat{B}}$	x^2	$x^2 + \beta_1 x$
$F^{(3)}_{\widehat{B}}$	$x^3 + \beta_1 x$	$x^3 + \beta_2 x^2 + \beta_1 x$
$F^{(4)}_{\widehat{B}}$	$x^4 + \beta_2 x^2 + \beta_1 x$	$x^4 + \beta_3 x^3 + \beta_2 x^2 + \beta_1 x$
$F^{(5)}_{\widehat{B}}$	$x^5 + \beta_3 x^3 + \beta_2 x^2 + \beta_1 x$	$x^5 + \beta_4 x^4 + \beta_3 x^3 \beta_2 x^2 + \beta_1 x$

TABLE 5.1: Fully General Expression of Higher Order Polynomial Activation Functions in Given MLP Network with Bias

5.4 Conclusion

This chapter on polynomial activation function fully general expression gives some mathematical support to previous works and opens up new ideas.

In Section 4.1 *Low Degree Polynomials* of the literature review, the choice of square-function as activation function is not only because of the simplicity but it is also proved in this chapter that square function maintains all the possible search space for second-order polynomials in hidden layers. Although all three papers utilized CNNs, the fully connected classifier at the end of the network can be considered as an MLP structure. In other words, all the properties in this chapter related to fully general expression can be used in the hidden/output layers of the classifier in CNNs.

Moreover, introducing the polynomial coefficient parameters (set $\widehat{B}$) has brought new perspectives to the research. In addition to adjusting the weight parameters, backpropagation can involve tuning polynomial coefficient parameters. This idea will be used later in Chapter 6.

In addition, a limitation arises with the absence of general solutions when proving the fully general expression of the third order and higher. It is possible to introduce additional natural constraints from the root operations, consequently reducing the search space size. This issue is one of the possible future work.

Chapter 6

Methodologies

6.1 Datasets and Preprocessing

6.1.1 Make Moons Dataset

The Make Moons dataset is generated using the function *make_moons* from the scikit-learn (sklearn) library for binary classification tasks. The dataset consists of two interleaving half circles, as illustrated in Fig. 6.1. Each sample in the dataset contains two input features and a corresponding output label (0 or 1). The parameter *noise* controls the standard deviation of the Gaussian noise applied to the samples.

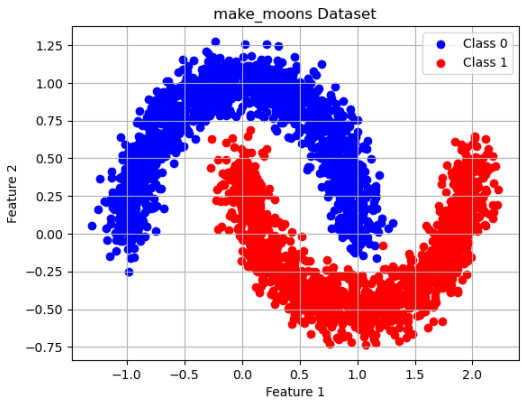


FIGURE 6.1: Generated Make Moons Dataset (*n_samples*=3000, *noise*=0.1, *random_state*=42)

The dataset is split into a training set (80%) and a test set (20%). To ensure consistency, both sets are projected onto the scale (-1, 1) using the *MinMaxScaler* from the sklearn library.

The class distribution of the Make Moons dataset is visualized in Fig. 6.2, and the corresponding detailed data is presented in Table 6.1.

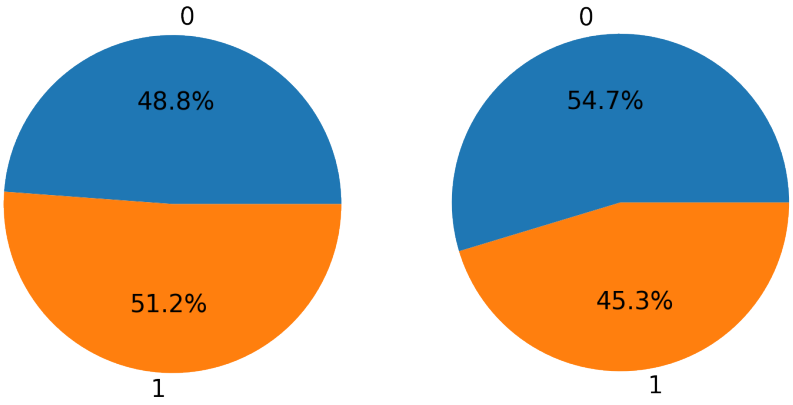


Figure 6.2: Class Distribution of Make Moons dataset. Left: Train set, Right: Test Set. Numbers outside the pie chart: name of classes, Percentage inside the pie chart: corresponding proportion. The detailed sample numbers are listed in Table 6.1.

	Classes	
Sets	0	1
Train	1172	1228
	48.8%	51.2%
Test	1172	272
	54.7%	45.3%

Table 6.1: Classes Distribution of Make Moons dataset (2 classes)

6.1.2 MNIST Dataset

The MNIST database (Modified National Institute of Standards and Technology database) is a widely-used dataset for machine learning tests. The dataset used in this study is fetched from TensorFlow comprising a train set of 60,000 samples and a test set of 10,000 samples. Each sample consists of a 28×28 matrix, representing handwritten digits from 0 to 9 (ten classes).

The first ten samples from the train set are visualized in Fig 6.3. The class distribution of the MNIST dataset is depicted in Fig. 6.4, and detailed data can be found in Table 6.2.

As mentioned earlier, Karthik's work [41] is considered a target of this thesis. The same preprocessing steps are applied here. Both the train and test sets are normalized by subtracting the average of the train set, dividing by the standard deviation of the train set, and converting the data into d_0 dimensions.

FIGURE 6.3: First Ten samples from the MNIST dataset

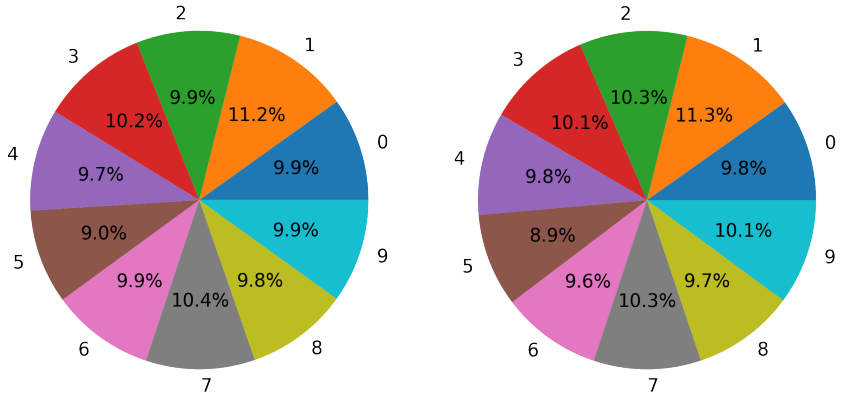


FIGURE 6.4: Class Distribution of the MNIST dataset. Left: train set, Right: test Set. Numbers outside the pie chart: name of classes, Percentage inside the pie chart: corresponding proportion. The detailed sample numbers are listed in Table 6.2.

	Classes									
Sets	0	1	2	3	4	5	6	7	8	9
Train	5923	6742	5958	6131	5842	5421	5918	6265	5851	5949
	9.9%	11.2%	9.9%	10.2%	9.7%	9.0%	9.9%	10.4%	9.8%	9.9%
Test	980	1135	1032	1010	982	892	958	1028	974	1009
	9.8%	11.3%	10.3%	10.1%	9.8%	8.9%	9.6%	10.3%	9.7%	10.1%

TABLE 6.2: Classes Distribution of the MNIST dataset (10 classes)

6.2 Neural Network Settings

The configuration of the neural networks entails two main aspects. Firstly, evaluate the performance of a simple architecture comprising one hidden layer of polynomial activation functions. Secondly, investigate the results of multi-layer networks incorporating polynomial activation functions in various layers. The selection of network

structures is guided by the research objectives and the complexity of the dataset.

6.2.1 Structures

Given the relatively simple Make Moons dataset, a network with one hidden layer is deemed appropriate. To ensure sufficient intermediate variables, the number of hidden layer neurons is set to three times the number of inputs (i.e., six). Employing one-hot encoding, the network architecture is expressed as $2 \times 6 \times 2$. Two sub-tests are conducted to further evaluate the model: one with the activation function added solely to the hidden layer and the other with the activation function applied to both the hidden and output layers.

For the more extensive MNIST dataset, to maintain comparability, the structure employed in the paper [41] is adopted. This architecture consists of two hidden layers, with 128 and 32 neurons. Considering the ten distinct classes in MNIST, the network structure is represented as $784 \times 128 \times 32 \times 10$. As previously stated, this part of the test aims to demonstrate the feasibility of utilizing polynomial activation functions in a multi-layer network. Consequently, activation functions are applied to both the hidden and output layers.

6.2.2 Initialization

The initialization of the Make Moons dataset employs the Gaussian distribution of Xavier/Glorot Initialization, as described in Theorem 1.

Theorem 1 (Xavier/Glorot Initialization[26])**.** *Biases are initialized to 0, while the weights w_{ij} at each layer are initialized according to either of the following schemes:*

$$w_{ij} \sim N\left(0.0, \frac{2}{n_{in} + n_{out}}\right) \quad \textbf{\textit{or}} \quad w_{ij} \sim U\left(\sqrt{\frac{6}{n_{in} + n_{out}}}, -\sqrt{\frac{6}{n_{in} + n_{out}}}\right)$$

where $N(\mu, \sigma^2)$ represents the Gaussian distribution with mean μ and variance σ^2, U denotes the normal distribution, n_{in} is the size of the previous layer, and n_{out} is the size of the current layer.

Furthermore, for the MNIST network, all weights are initialized using the Gaussian distribution $N(0.0, 0.01)$ with a mean of 0.0 and a standard deviation of 0.1, and biases are initialized to 0.0, which follows the same initialization strategy as mentioned in the paper [41].

6.3 Activation Functions

The Sigmoid function is very commonly used in neural networks for its differentiability and smooth transition. Moreover, the output range is $(0, 1)$, which is very suitable for binary classification problems. As a result, it is considered as a standard baseline for comparison.

An important aspect to consider is that the polynomial activation functions are unbounded, which means that networks with more than two layers using polynomial activation functions are susceptible to overflow. Overflow can occur due to two possible main reasons. Firstly, initialization plays a role, as high-power polynomial terms can amplify the results of neuron summation, especially in multi-layer networks. Secondly, high-order coefficients suffer from the overflow problem because they can often receive large gradients due to the high-order term in the polynomial function. Thus, a smaller learning rate can alleviate this issue. Nevertheless, setting the learning rate too low can result in slow convergence. It is a trade-off between stability and speed. Therefore, one solution could be assigning different learning rates to different power coefficients. The higher the power coefficient, the lower the learning rate should be set to prevent overflow.

This thesis proposed new forms of polynomial activation functions, which are listed in Table 6.3. The initial value of β_1 is 1, and β_{2-5} (if exists) are initialized to 0. These forms possess an advantageous property that ensures the coefficient parameters change step-by-step from lower-order terms to higher-order terms (β_{n+1} constantly changes later than β_n). Because the gradient of loss function respected to β_{n+1} contains a term $\prod_{i=1}^{n} \beta_i$, which is not zeros if and only if all the parameters $\beta_i (i = 1 \dots n)$ are not zero. This form provides extra constraint to higher-order coefficient parameter gradients, which ensures that higher-order coefficient parameters have a much slower rate of change. For the first-order (linear) activation function, adding β_1 keeps the form consistent without influencing the results much because it is still a linear operation.

The simplified, fully general expressions of third and fifth-order polynomial activation functions are also shown in Table 6.3 which is indicated by asterisk *. Some terms and their corresponding parameters are removed. Consequently, the learning rate should be smaller for the highest-order β parameter.

Functions	Activation Functions
Linear	x
1	$\beta_1 x$
2	$\beta_2\beta_1 x^2 + \beta_1 x$
3	$\beta_3\beta_2\beta_1 x^3 + \beta_2\beta_1 x^2 + \beta_1 x$
4	$\beta_4\beta_3\beta_2\beta_1 x^4 + \beta_3\beta_2\beta_1 x^3 + \beta_1\beta_2 x^2 + \beta_1 x$
5	$\beta_5\beta_4\beta_3\beta_2\beta_1 x^5 + \beta_4\beta_3\beta_2\beta_1 x^4 + \beta_3\beta_2\beta_1 x^3 + \beta_2\beta_1 x^2 + \beta_1 x$
3*	$\beta_1\beta_3 x^3 + \beta_1 x$
5*	$\beta_1\beta_2\beta_3\beta_5 x^5 + \beta_1\beta_2\beta_3 x^3 + \beta_1\beta_2 x^2 + \beta_1 x$

TABLE 6.3: New Proposed Forms of Activation Function, where * implies fully general expression

6.4 Training Strategy

Two possible optional strategies are considered for training: updating weights and polynomial coefficients alternately or updating them simultaneously. The former requires nearly double the computing time of the latter but offers the potential for improved stability. These two methods will be tested separately on the Make Moons dataset. Due to time constraints, for the relatively large dataset MNIST, only the results for simultaneous training are reported in this thesis.

Moreover, fully-batch gradient descent is employed with a batch size equal to the sample size for the Make Moons dataset, and the training runs for 500 epochs (one round of alternate training is also considered as one epoch here). In contrast, the MNIST dataset uses mini-batch gradient descent with a batch size of 60 to expedite the convergence process, following the approach taken in [41]. Additionally, the number of epochs is set to 200.

Among the hyperparameters, the most critical are the learning rates for weights and the different polynomial coefficient parameters. Under the premise of maintaining the stability of the network, appropriate relatively large learning rates should be chosen to speed up the process. Moreover, the learning rate for the same set of experiments should remain consistent.

6.5 Performance Analysis

The Make Moons and MNIST datasets are highly balanced datasets, as evident from Fig. 6.2 and Fig. 6.4. Therefore, the test accuracy is enough to provide a reasonable indicator of the experimental results' performance. As previously mentioned, the polynomial activation functions are unbounded making initialization crucial to the results. Statistical results of different initializations will be presented in Chapter 7. To obtain more precise results, different weight initializations of the experiments on the same dataset will be consistent by setting the program seed.

The performance analysis of ML should focus on two main aspects. Firstly, presenting the final test accuracies and their corresponding standard deviations. Secondly, providing diagrams, if needed, with error bars for train losses, test losses, and test accuracies, illustrating the experiments' stability, convergence speed, and performance. Because the initializations are consistent, independent samples t-test will be used while comparing the methods, namely the necessity of adding polynomial activation function to output layers and the choice of alternate/continuous coefficients update.

Furthermore, it is also vital to consider the computational complexity introduced by the utilization of the proposed polynomial activation functions in the context of FHE, which is related to noise growth. Due to time constraints, the discussion primarily focuses on network inference (feedforward), while the aspects of backpropagation will be reserved for future work. The analysis encompasses two categories of metrics: the number of operations and the computational depth. To elaborate further, the number of additions (NoA), the number of multiplications (NoM), and the depth of

multiplications (DoM) associated with the weights and different activation functions should be discussed. Here DoM is defined as

Definition 6.5.1 (Depth of Multiplications)**.** *Given that the network only employs addition and multiplication operations, it can be fully expanded into a summation of many multiplication terms. The depth of multiplications(DoM) refers to the number of multiplication operations within the term with the highest number of consecutive multiplication operations.*

For instance, DoM for fomula $a * b + c * d$ is considered as 1, while $a * b * c + d$ has a depth-2 multiplications, even though they both hold NoA = 1 and NoM = 2.

The weights and activation functions parts of the MLP network can be considered concatenated, implying that NoA, NoM, and DoM can be computed separately for each.

6.6 Conclusion

The methodologies can be summarized into several tasks based on different datasets. The network and training settings are listed below.

Tasks related to Make Moons dataset: The network structure is $2 \times 6 \times 2$ initialized by Gaussian Xavier (20 groups). The training uses full-batch gradient descent for 500 epochs.

1. Task of different orders: apply functions of Sigmoid, Linear, and 1-5 in Table 6.3 on both hidden and output layers with simultaneous updates on weights and polynomial coefficient parameters;

2. Task of fully general expression: the same settings as task 1, but with 3^* and 5^* functions;

3. Task of polynomial positions: the same settings as task 1, but function $1 - 5$ are only applied on hidden layer;

4. Task of training strategies: the same settings as task 1, but applied alternate updates on weights and polynomial coefficient parameters separately.

Tasks related to MNIST dataset: The network structure is $784 \times 128 \times 32 \times 10$ initialized by Gaussian $N(0.0, 0.01)$. The training uses mini-batch gradient descent with batch size 60 for 200 epochs.

1. Task of different orders: apply functions of Sigmoid, Linear, and 1-5 in Table 6.3 on both hidden and output layers with simultaneous updates on weights and polynomial coefficient parameters;

2. Task of polynomial positions: the same settings as task 1, but function $1 - 5$ are only applied to hidden layers.

Chapter 7

Results

The results of tasks listed in Sec. 6.6 are presented in this chapter. The experiments are implemented using the PyTorch library with GPU acceleration. All the function names in this chapter are corresponded to Table 6.3, which will not be repeatedly illustrated in every presented result below.

7.1 Tasks related to the Make Moons dataset

To ensure comparability among tasks, the picked learning rates are shown in Table 7.1. It should be noted that the Sigmoid function (the only bounded function) is fundamentally different from the polynomial functions, necessitating a significantly larger learning rate for the weights.

		Coefficient parameters				
Functions	Weights	β_1	β_2	β_3	β_4	β_5
Sigmoid	5	-	-	-	-	-
Linear	0.01	-	-	-	-	-
1	0.01	0.1	-	-	-	-
2	0.01	0.1	0.1	-	-	-
3	0.01	0.1	0.1	0.5	-	-
3*	0.01	0.1	-	0.01/0,05	-	-
4	0.01	0.1	0.1	0.5	0.5	-
5	0.01	0.1	0.1	0.5	0.5	0.5
5*	0.01	0.1	0.1	0.5	-	0.25

TABLE 7.1: Parameter-Specific Learning Rates for Different-order Activation Functions on Make Moons Dataset

The following results are generated after 500 Epochs over 20 consistent initializations

7.1.1 Results of different orders

The network structure includes activation functions in both hidden and output layers, and the parameter-specific learning rates are from Table 7.1.

The results focus on the performance of different-order polynomial activation functions compared to Sigmoid and Linear functions. The corresponding test losses and test accuracies are plotted in Fig. 7.1 and Fig. 7.2, respectively, where $1 - \sigma$ error bars are marked in shadow.

Detailed accuracy statistical results after 500 epochs are recorded in Appendix Table A.1.

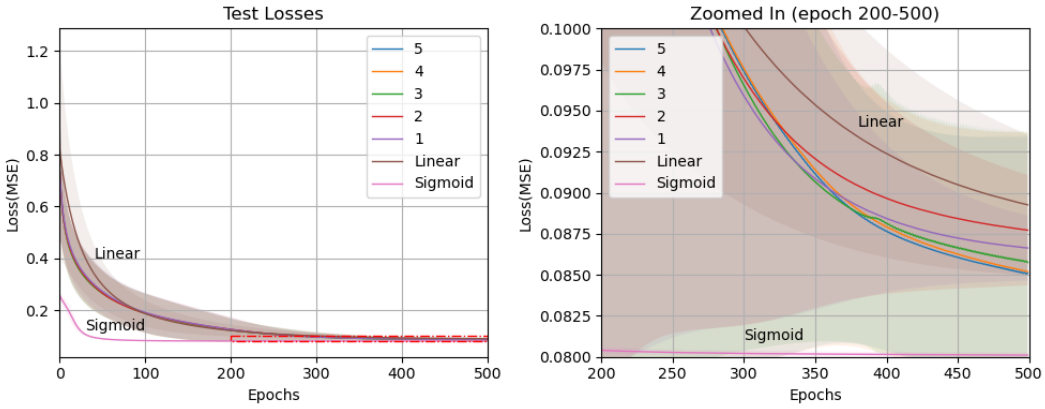


Figure 7.1: Left: Test Losses of Sigmoid, Linear and Different-order Polynomial Activation Functions, Right: Zoomed Red Rectangular area

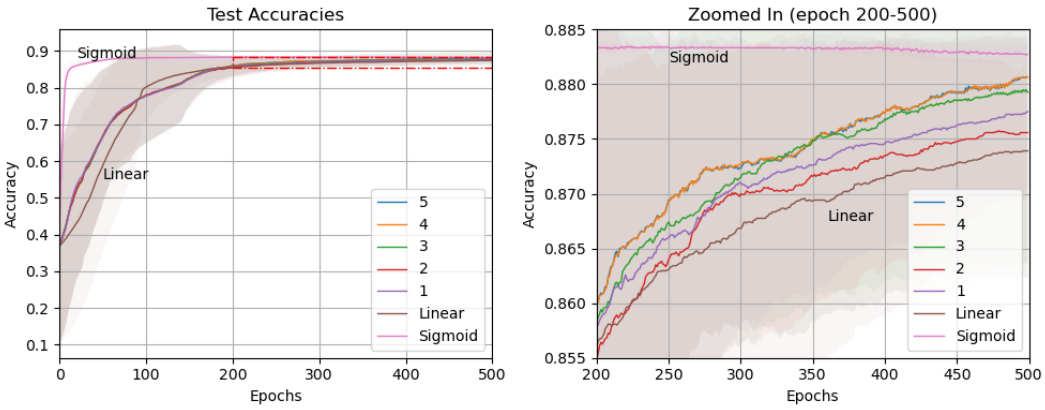


Figure 7.2: Left: Test Accuracies of Sigmoid, Linear and Different-order Polynomial Activation Functions, Right: Zoomed Red Rectangular area

7.1.2 Results of Third and Fifth-Order Fully General Expressions

Similar to Sec. 7.1.1, the third and fifth-order activation functions of fully general expressions are plotted in Fig. 7.3 and Fig. 7.4.

Detailed statistical results after 500 epochs are recorded in Appendix Fig. A.1.

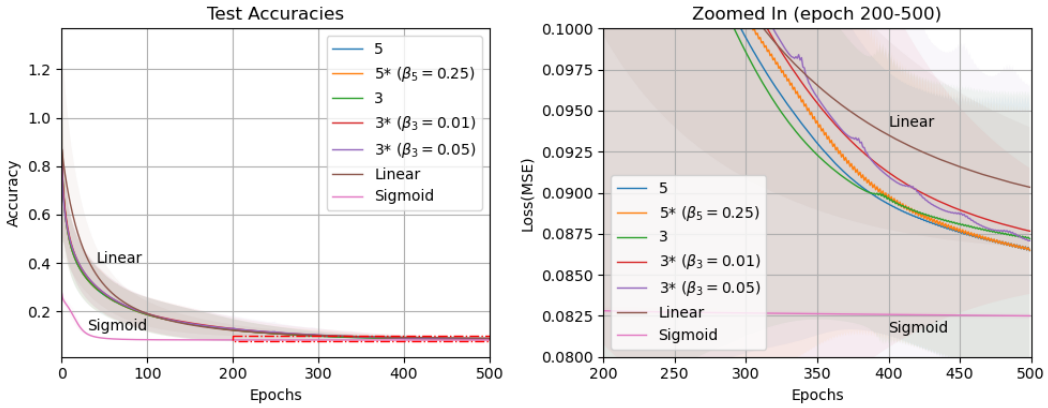


FIGURE 7.3: Left: Test Losses of Simplified 3rd and 5th Polynomial Activation Functions, Right: Zoomed Red Rectangular area

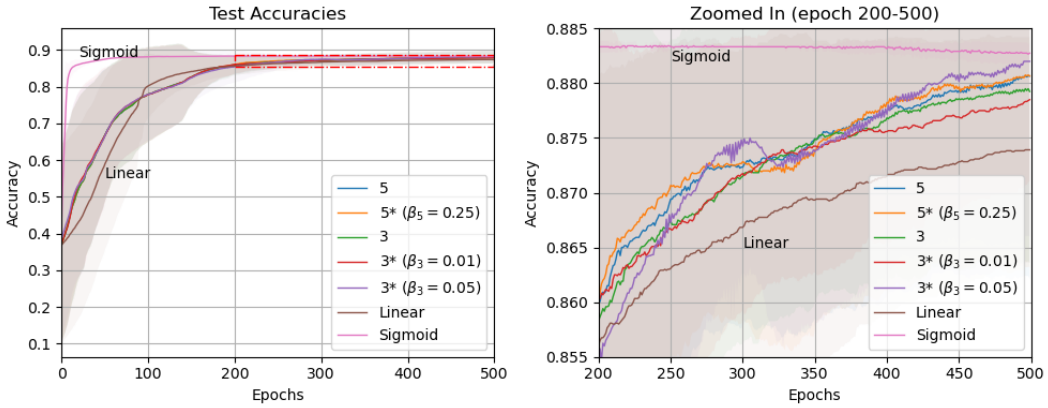


FIGURE 7.4: Left: Test Accuracies of Simplified 3rd and 5th Polynomial Activation Functions, Right: Zoomed Red Rectangular area

7.1.3 Results of Polynomial Positions

The objective is to examine whether adding polynomial activation functions to the output layer can enhance the network's performance. The test accuracy differences compared to Sec. 7.1.1 for different orders are plotted in Fig. 7.5 where error bars are colored in shadow. Moreover, The p values of independent samples t-test are plotted in Fig. 7.6, with a significance level of 0.05.

Detailed comparison of accuracy statistical results after 500 epochs are recorded in Appendix Table A.2.

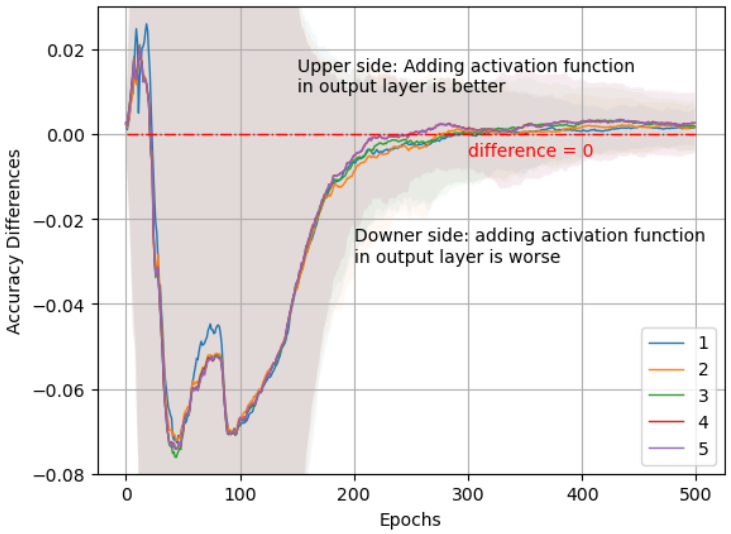


Figure 7.5: Test Accuracies Differences and the Error Bars (in shadow) for Different Positions of Activation Functions

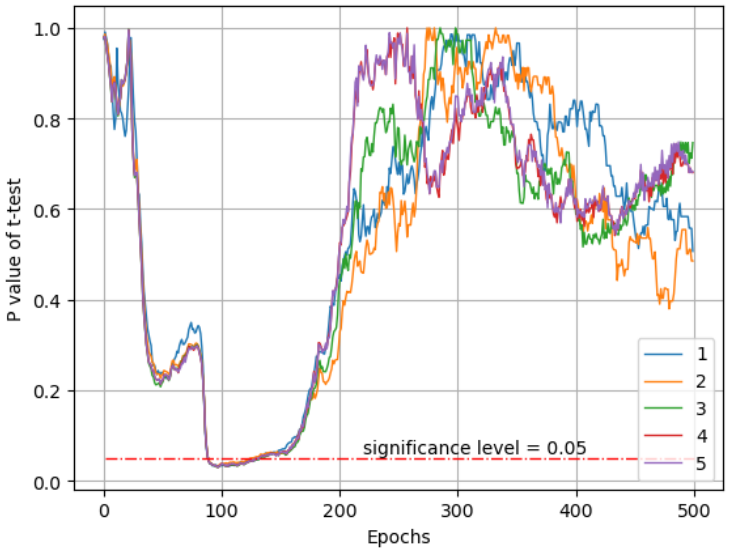
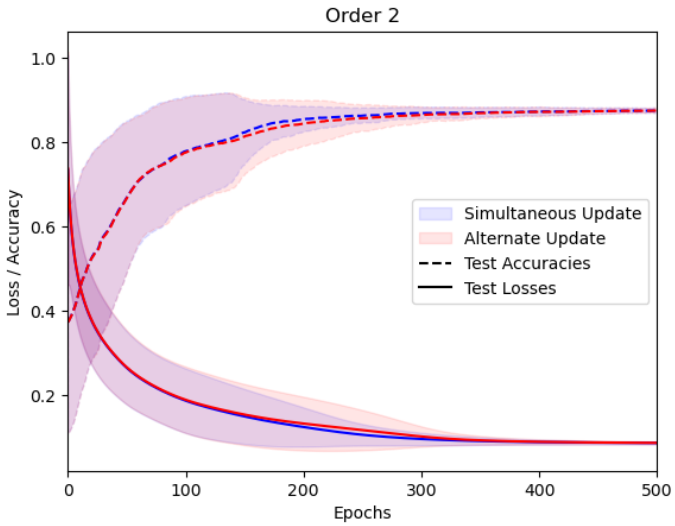


Figure 7.6: P-Values of Independent Samples T-Test of Test Accuracies Differences for Different Positions of Activation Functions

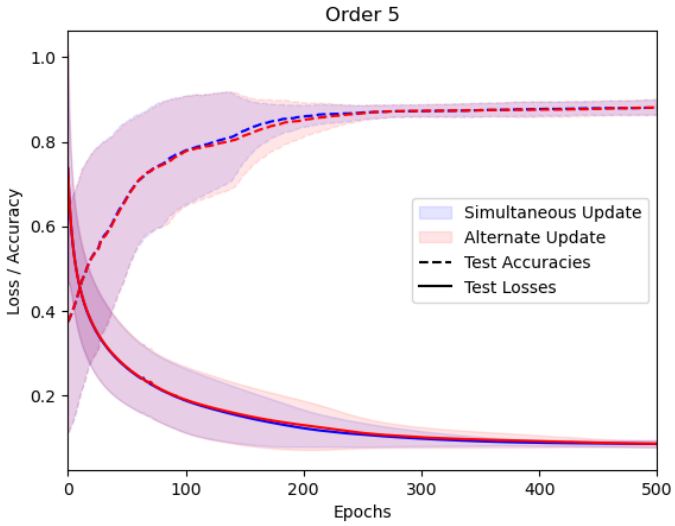
7.1.4 Results of training strategies

This part focuses on investigating the impact of the alternate training strategy compared to simultaneously updating weights and polynomial coefficient parameters.

The test accuracies and test losses comparison between Simultaneous and alternate update strategies are plotted in pairs in Fig. 7.7 of selected second and fifth orders.



(A) Second Order



(B) Fifth Order

FIGURE 7.7: Comparison between Simultaneous and alternate update strategies of Second and Fifth Orders.

7.2 Tasks related to the MNIST dataset

The tasks aim to explore the performance outcomes and scalability of the proposed polynomial activation function on a larger dataset (MNIST) and a more complex network structure (NN with two hidden layers). Similar to the Make Moons dataset, the learning rates of activation function coefficient parameters and weights are fixed as Table 7.2 shows.

		Coefficient parameters				
Functions	Weights	β_1	β_2	β_3	β_4	β_5
Linear	0.02	-	-	-	-	-
Sigmoid	5	-	-	-	-	-
1	0.02	0.02	-	-	-	-
2	0.02	0.02	0.01	-	-	-
3	0.02	0.02	0.01	0.01	-	-
4	0.02	0.02	0.01	0.01	0.5	-
5	0.02	0.02	0.01	0.01	0.01	0.01

Table 7.2: Parameter-Specific Learning Rates for Different-order Activation Functions on MNIST Dataset

The following results are generated after 200 Epochs over 5 consistent initializations

7.2.1 Results of Different Orders

The network contains activation functions in every hidden and output layer. Only the test accuracies are plotted here in Fig. 7.8, , where 1σ error bars are marked in shadow. Detailed statistical results after 200 epochs are recorded in Appendix Table A.3.

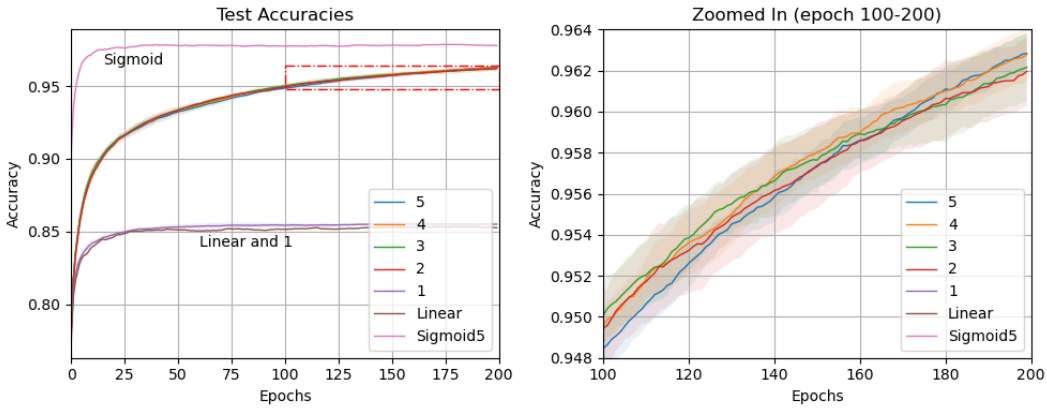


Figure 7.8: Left: Test Losses of Sigmoid, Linear, and Different-order Polynomial Activation Functions, Right: Zoomed Red Rectangular area

7.2.2 Results of Polynomial Positions

The objective is the same as the Make Moons dataset. The test accuracy differences compared to Sec. 7.2 for different orders plotted in Fig. 7.9 where error bars are colored in shadow. Moreover, The p values of independent samples t-test are plotted in Fig. 7.10, with a significance level of 0.05.

Detailed comparison of accuracy statistical results after 200 epochs are recorded in Appendix Table A.4.

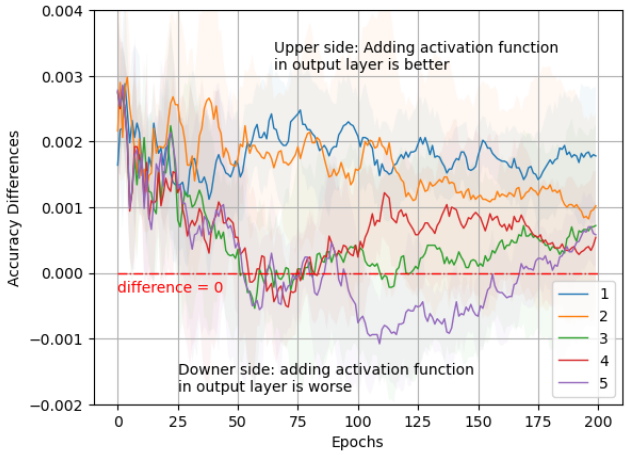


FIGURE 7.9: Test Accuracies Differences and the Error Bars (in shadow) for Different Positions of Activation Functions

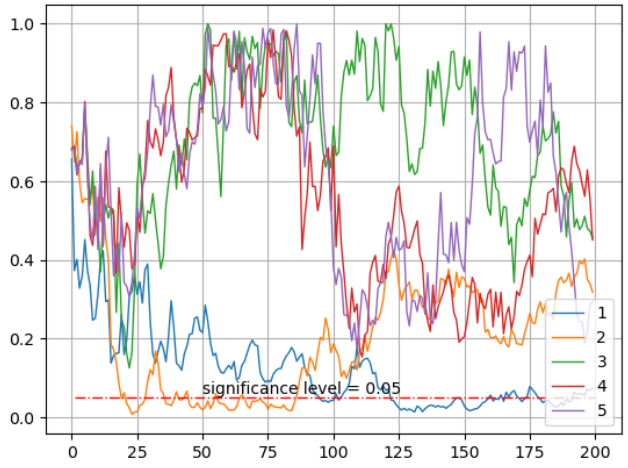


FIGURE 7.10: P-Values of Independent Samples T-Test of Test Accuracies Differences for Different Positions of Activation Functions

Chapter 8

Discussion

8.1 Make Moons Discussion

8.1.1 Discussion on Results of Different Orders

Firstly, a comparison is conducted for polynomial activation functions of orders one through five, utilizing the proposed form, with specific reference to the Sigmoid and Linear functions. Fig. 7.1 illustrates the averaged outcomes of 20 distinct initialization runs across 500 epochs. From the graph, it can be observed that the test loss convergence rate for the proposed polynomial activation function is generally comparatively slower than that of the sigmoid activation function. This is mainly because the learning rates are uniformly set to smaller values to ensure the stability of high-order activation function training. In fact, lower-order polynomial activation functions exhibit the capacity to hold larger learning rates, thereby speeding up the convergence of the loss function.Under current equivalent learning rate conditions, higher-order polynomials yield smaller loss values than lower-order counterparts except for order 2. The second-order polynomial activation function in the Make Moons dataset under the given structure even performs worse than case the linear adjustable first-order case. The reason remains unclear. The suboptimal mapping is cased by the non-monotinicity of the second-order polynomial function, or the shortcomings of the second-order polynomial to fit the given Make Moons Dataset. In addition, the fourth and fifth-order results hold very similar performances. Consequently, it can be inferred that further increasing the polynomial order of the proposed polynomial structures in the current dataset is not necessary.

Fig. 7.2 presents the test accuracies corresponding to the test loss (Fig. 7.1). The figure aligns with the analysis of the test loss results above, where the Sigmoid function exhibits the most favorable performance, while the Linear demonstrates the least favorable outcome. Order 1 demonstrates higher accuracy compared to order 2, and the difference between order 4 and order 5 is indistinguishable.

8.1.2 Discussion on Results of Third and Fifth Order Fully General Expressions

The second part tests whether the proposed form of the third- and fifth-order polynomial fully general expression in Table 6.3 derived in Chapter 5 could obtain similar results. Figure 7.3 provides a comparison of loss values results in over 20 different consistent initialization. For the third-order equation, which lacks a quadratic term in its fully general expression,selecting the highest-order β_3 learning rate becomes vital. As shown in Figure 7.3, an excessively small learning rate on order 3^* ($\beta_3 = 0.01$) slow down the convergence speed, resulting in disadvantaged performance compared to order 3. In contrast, a larger learning rate on order 3^* ($\beta_3 = 0.05$) yields a lower loss but contains oscillations in the loss reduction process. This phenomenon might be due to the large β_3 learning rate, which mismatches the overall network's learning rate, leading to network instability. Order 3^* ($\beta_5 = 0.25$) also exhibits some oscillations, indicating that a larger value might be selected for β_5 learning rate. However, the oscillating performance is not obvious due to its small coefficients β_5 (mostly below 10^{-7}).

These scenarios find further correspondence in Figure 7.4. Although the final accuracy of order $3^* \cdot (\beta_3 = 0.05)$ configuration is the highest among all the polynomial cases, its growth trajectory is markedly unstable, indicating large β_3 learning rate is not recommended. Similarly, order 5^* configuration also exhibits instability. Combining the analysis from Figure 7.3, it becomes clear that the simplified fully general expression offers a degree of substitution for the most general form, thereby simplifying computations to some extent. However, within the polynomial structure proposed by this thesis, the selection of learning rates has emerged as a primary concern, as inappropriate choices can yield counterproductive outcomes. Whether to utilize the simplified fully general expression is a trade-off between computational cost and training stability.

8.1.3 Discussion on Results of Polynomial Positions

Fig. 7.5 illustrates that favorable outcomes of adding activation functions in the output layer are only evident when the epoch is over 300. However, the insights derived from Figure Fig. 7.6 reveal that this observed improvement lacks statistical significance, with the value p of the t-test much greater than 0.05. Conversely, when the epoch ranges between 50 and 100, applying polynomial activation functions solely to the hidden layer can perform even better than involving both hidden and output layers. In this context, the p-value derived from the t-test may descend below the threshold of 0.05, which holds statistically better performance. Nevertheless, an examination of Fig. 7.2 reveals that, during this interval of epochs, the accuracy has not yet converged. In summation, for simple datasets like Make Moons, adding polynomial activation functions in the output layer of an MLP under one hidden layer network structure lacks statistical significance. Conversely, it can further bring higher computational demands and increase computational depth.

8.1.4 Discussion on Results of Different Training Strategies

The need for a t-test is even unnecessary, as Fig. 7.7 already vividly demonstrates that alternately updating weights and polynomial coefficient parameters is entirely unwarranted. Such an approach merely attenuates the standard deviation during the progression of lower-order equations yet fails to yield any enhancement in accuracy. Conversely, adopting the strategy of alternate updates transforms a single-round update into a dual-round process, doubling the computational burden across all operations, which is an unfavorable selection.

8.1.5 Discussion on FHE Computational Complexity

Furthermore, some analysis of the computational complexity from the perspective of FHE needs to be conducted. According to Sec. 6.5, the complexity is evaluated number of additions (NoA), number of multiplications (NoM), and depth of mutiplications(DoM), which will be firstly calculated in one feedforward process of on single activation function. It needs to be noticed here that NoA and NoM are not directly counted since similar terms can be combined to simplify the process, which will also be written in Table 8.1.

Functions	Simplified Expression	NoA	NoM	DoM
Linear	x	0	0	0
1	$\beta_1 x$	0	1	1
2	$(\beta_2 x + 1)\,\beta_1 x$	1	3	3
3	$((\beta_3 x + 1)\,\beta_2 x + 1)\,\beta_1 x$	2	5	5
4	$(((\beta_4 x + 1)\,\beta_3 x + 1)\,\beta_2 x + 1)\,\beta_1 x$	3	7	7
5	$((((\beta_5 x + 1)\,\beta_4 x + 1)\,\beta_3 x + 1)\,\beta_2 x + 1)\,\beta_1 x$	4	9	9
3^*	$(\beta_3 x^2 + 1)\,\beta_1 x$	1	4	4
5^*	$(((\beta_5 x^2 + 1)\,\beta_3 x + 1)\,\beta_2 x + 1)\,\beta_1 x$	3	8	8

TABLE 8.1: NoA, NoM, and DoM Required by One Activation Function in One Feedforward Process.

For the current network structure of $2 \times 6 \times 2$, $6 + 2 = 8$ activation functions exist. The number of operations in Table 8.1 needs to be multiplied by the number of activation functions.

Beside the operation in activation functions, the calculation of weights depends on the neuron quantities in the preceding and succeeding layers. Suppose the n^{th} layer comprises $N^{(n)}$ neurons, and the $(n+1)^{th}$ layer encompasses $N^{(n+1)}$ neurons. In that case, each neuron in the $(n+1)^{th}$ layer requires $N^{(n)}$ additions (bias included) and $N^{(n)}$ multiplications prior to the activation function computation. Therefore, the weight-related calculations between these two layers encompass $N^{(n)}N^{(n+1)}$ additions and $N^{(n)}N^{(n+1)}$ multiplications. For the present network configuration of $2 \times 6 \times 2$, the weight-related NoA and NoM are both $2 * 6 + 6 * 2 = 24$. DoM related to weights

is always 1 per layer (the product of layer inputs and the corresponding weights) no matter how many neurons between two layers exist.

For small networks, more than half of the computations are utilized in calculating polynomial activation functions if higher-order ones are chosen. However, the impact of this cost on larger networks is not much in proportion, which will be discussed later in Section 8.2. Given the current state of the MLP network, opting for third-order polynomial activation functions on the hidden layer only proves to be a favorable choice.

8.2 MNIST Discussion

For MNIST, the investigation was extended to a larger architecture with a two-hidden-layer MLP network and MNIST dataset.

8.2.1 Discussion on Results of Different Order

In the appendix, it is evident that for the network architecture of $784 \times 128 \times 32 \times 10$, the accuracy achieved using the Sigmoid activation function is 97.8%, aligning precisely with the accuracy reported in the paper [41], where the method utilized LUTs. This consistency in accuracy is proved that utilizing of LUTs does not entail any compromise on accuracy within the context of FHE. The result of accuracy achieved by this thesis on MNIST is 96.3%, starting from the polynomial of order 4 after 200 epochs.

Despite the considerably reduced error bars of this larger network than the preceding section's analysis of the Make Moon dataset, the distinctive advantage of employing higher-order polynomial activation functions, as formulated by this paper, still remains unremarkable in Fig. 7.8. Upon close observation, it becomes apparent that at the 100-epoch, the accuracy corresponding to polynomial functions of order 4 and 5 trails behind that of second and third-order counterparts. However, after 200 epochs, the former outperforms the latter, thus signifying the potential good performance of higher-order equations after more epochs, but still facing the problem of slow convergence.

8.2.2 Discussion on Results of Polynomial Positions

Fig. 7.8 and 7.10 demonstrate that, under the given network structure, the inclusion of activation functions in the output layer generates a statistically significant improvements in accuracy for lower-order polynomials (order 1 or 2). However, adding an activation function in the output layer becomes unnecessary when the selected polynomial function reaches or exceeds the third order.

8.2.3 Discussion on FHE Computational Complexity

Total NoA, NoM, and DoM in MNIST are listed in Table 8.2.

Functions	NoA	NoM	DoM
Linear	0	0	0
1	0	170	3
2	170	510	9
3	340	850	15
4	510	1190	21
5	680	1530	27
Weights	4768	4768	3

TABLE 8.2: Total NoA, NoM, and DoM in MNIST network

For this relatively larger network, opting for higher-order polynomials as activation functions does not significantly increase in the number of additions (NoA) and multiplications (NoM). The computational load is predominantly attributed to the weights. However, the depth of multiplications (DoM) is jointly influenced by the number of layers and the polynomial selection. Synthesizing the insights derived from this section's analysis, it is advisable to utilize lower-order polynomials and incorporate them in the output layer for networks characterized by a greater number of layers. Conversely, a smart choice for networks with fewer layers would involve selecting higher-order polynomials and abstaining from adding an activation function in the output layer.

Furthermore, it is vital to consider the properties of the given FHE scheme. This entails analyzing the computational time associated with multiplication operations and the accumulation of noise after each operation. Subsequently, an informed decision can be made regarding selecting an appropriate architectural configuration.

8.3 Conclusion

This chapter aims to explore a training process for fully-connected neural networks utilizing polynomial activation functions through the results and analysis. For single-hidden-layer networks, the choice of third-order or higher polynomials, given a sufficient number of training epochs, yields negligible outcome differences. Moreover, using fully general expressions for polynomials can speed up the computations. Remarkably, including activation functions in the output layer for polynomials of order three and beyond is deemed unnecessary.

From the perspective of FHE cost, selecting lower-order polynomials for networks characterized by a greater number of layers effectively decreases the depth of multiplication, thereby minimizing computational load. In contrast, networks featuring a higher input dimension and fewer layers can explore the adoption of higher-order polynomials to attain enhanced performance.

In this thesis, employing a fully-connected neural network comprising two hidden layers, combined with fourth or fifth-order polynomial activation functions, can achieve an accuracy of 96.3% on the MNIST dataset. Four additional perspectives

of future work can contribute to a more comprehensive exploration.

Chapter 9

Conclusion

9.1 Summary

In the context of information protection, sensitive data analysis has posed a challenge to trust between clients and servers. Different Fully Homomorphic Encryption (FHE) schemes present potential solutions for operating on encrypted data, which can be further combined with Machine Learning (ML). However, the implementation of non-linear activation functions within neural networks becomes problematic due to limitations in performing only addition and multiplication operations.

Consequently, this thesis primarily investigates the feasibility of utilizing polynomial activation functions on neural networks in an FHE-friendly environment. The thesis begins by introducing cryptography and machine learning (ML) backgrounds, providing brief explanations of FHE and Multi-Layer Perceptrons (MLP). Subsequently, Chapter 4 categorizes the existing research of CNN/MLP on FHE based on their solutions for non-linear activation functions. Moreover, Chapter 5 establishes the general expression for polynomial activation functions. Incorporating the insights derived from this expression, Chapter 6 introduces the form, training approach, and evaluation method for the proposed adjustable polynomial activation functions.

In the experimental phase, the paper achieved 96.3% accuracy to the MNIST dataset by applying the fourth or fifth-order proposed activation function on the fully-connected neural network with two hidden layers. The results indicate that increasing the order within the proposed polynomial form does not substantially improve accuracy. A trade-off exists between potential higher accuracy and overall network instability, coupled with increased computational load. Additionally, the potential cost for FHE was also analyzed.

9.2 Limitations and Possible Future Works

There are the four obvious limitations of this thesis, along with the corresponding track for potential future work:

The first limitation, as mentioned in Sec. 5.3.2, contains insufficient proofs for fully general expressions of third-order and higher-order polynomials. In future work,

the acceleration of network training and prediction can be achieved by identifying and simplifying polynomial forms. Alternatively, despite of fully general expressions, exploring strategies to find superior-performing sub-searching spaces is also feasible, thereby investigating certain specialized forms of polynomials.

The second limitation concerns the analysis in Sec. 6.5 about FHE computational cost, which lacks a foundation in the context of backpropagation. While feedforward operations correspond to the analysis of the cost required for inference using a pre-trained model, incorporating both feedforward and backpropagation processes corresponds to the cost of training the neural network. This study falls short in addressing the part of network training time/efficiency, necessitating further investigation and supplementation.

The third limitation lies in the fact that all experiments in this study were conducted in the plaintext domain, relying entirely on the mathematical properties of FHE to infer the same results in the ciphertext domain. However, further research is essential to comprehensively explore the adaptation on FHE, particularly in selecting the most suitable scheme to accelerate the entire training and prediction process.

The last limitation belongs to the context introduced in Chapter 4, where mainstream FHE applications often utilize CNNs rather than MLPs. Extending the testing of the proposed adjustable polynomial activation functions to CNN architectures would contribute to a more comprehensive and robust set of conclusions.

Appendices

Appendix A

Supplementary information for Chapter Results

In this appendix, the supplementary information for Chapter 7 will be recorded.

A.1 Tasks related to Make Moons dataset

Functions	Train Losses	Test Losses	Test Accuracies
Linear	0.090 ± 0.004	0.089 ± 0.004	0.874 ± 0.010
Sigmoid	0.083 ± 0.000	0.080 ± 0.000	0.883 ± 0.001
1	0.088 ± 0.001	0.087 ± 0.002	0.878 ± 0.007
2	0.089 ± 0.004	0.088 ± 0.003	0.876 ± 0.006
3	0.087 ± 0.009	0.086 ± 0.008	0.879 ± 0.016
3* ($\beta_3 = 0.01$)	0.088 ± 0.004	0.086 ± 0.004	0.879 ± 0.007
3* ($\beta_3 = 0.05$)	0.087 ± 0.009	0.085 ± 0.008	0.882 ± 0.016
4	0.087 ± 0.009	0.085 ± 0.008	0.881 ± 0.018
5	0.086 ± 0.009	0.085 ± 0.008	0.881 ± 0.019
5* ($\beta_3 = 0.25$)	0.086 ± 0.008	0.085 ± 0.008	0.881 ± 0.017

TABLE A.1: Statistical Train Losses, Test Losses, and Test Accuracies Results Over 20 Different Initializations after 500 epochs.

	Test Accuracies	
Functions	Only Hidden Layer	Hidden & Output Layers
1	0.876 ± 0.008	0.878 ± 0.007
2	0.874 ± 0.008	0.876 ± 0.006
3	0.878 ± 0.017	0.879 ± 0.016
4	0.878 ± 0.020	0.881 ± 0.018
5	0.878 ± 0.020	0.881 ± 0.019

Table A.2: Comparison After 500 Epoch of Statistical Test Accuracies Results for Different Polynomial Positions

A.2 Tasks related to MNIST dataset

Functions	Train Losses	Test Losses	Test Accuracies
Linear	0.034 ± 0.000	0.039 ± 0.000	0.853 ± 0.000
Sigmoid	0.000 ± 0.000	0.004 ± 0.000	0.978 ± 0.000
1	0.034 ± 0.000	0.039 ± 0.000	0.855 ± 0.001
2	0.009 ± 0.001	0.012 ± 0.000	0.962 ± 0.002
3	0.006 ± 0.001	0.009 ± 0.000	0.962 ± 0.002
4	0.005 ± 0.000	0.009 ± 0.000	0.963 ± 0.001
5	0.004 ± 0.001	0.008 ± 0.000	0.963 ± 0.001

Table A.3: Statistical Train Losses, Test Losses, and Test Accuracies Results Over 20 Different Initializations after 200 epochs.

	Test Accuracies	
Functions	Only Hidden Layer	Hidden & Output Layers
1	0.853 ± 0.001	0.855 ± 0.001
2	0.961 ± 0.001	0.962 ± 0.002
3	0.961 ± 0.001	0.962 ± 0.002
4	0.962 ± 0.000	0.963 ± 0.001
5	0.962 ± 0.000	0.963 ± 0.001

Table A.4: Comparison After 200 Epoch of Statistical Test Accuracies Results for Different Polynomial Positions

Bibliography

[1] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018.

[2] A. N. N. an overview | ScienceDirect Topics. Activation function. https://www.sciencedirect.com/topics/neuroscience/artificial-neural-network.

[3] K. C. an overview | ScienceDirect Topics. Activation function. https://www.sciencedirect.com/topics/computer-science/key-cryptography.

[4] A. A. Badawi, J. Chao, J. Lin, C. F. Mun, J. J. Sim, B. H. M. Tan, X. Nan, K. M. M. Aung, and V. R. Chandrasekhar. Towards the alexnet moment for homomorphic encryption: Hcnn, thefirst homomorphic cnn on encrypted data with gpus, 2020.

[5] V. Bindushree, R. Sameen, V. Vasudevan, T. Shrihari, D. Devaraju, and N. Mathew. Artificial intelligence: In modern dentistry. *Journal of dental research and review*, 7(1):27–31, 2020.

[6] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski. ngraph-he2: A high-throughput framework for neural network inference on encrypted data. *CoRR*, abs/1908.04172, 2019.

[7] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory*, 6(3):1–36, 2014.

[8] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff. Privacy-preserving classification on deep neural network. *IACR Cryptol. ePrint Arch.*, 2017:35, 2017.

[9] H. Chen, R. Gilad-Bachrach, K. Han, Z. Huang, A. Jalali, K. Laine, and K. Lauter. Logistic regression over encrypted data from fully homomorphic encryption. *BMC Medical Genomics*, 11(S4), 2018.

[10] J. H. Cheon, A. Costache, R. C. Moreno, W. Dai, N. Gama, M. Georgieva, S. Halevi, M. Kim, S. Kim, K. Laine, Y. Polyakov, and Y. Song. *Introduction to Homomorphic Encryption and Schemes*, pages 3–28. Springer International Publishing, Cham, 2021.

[11] J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. *Advances in Cryptology - ASIACRYPT 2017*, pages 409–437, 2017.

[12] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene. Tfhe: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2019.

[13] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei. Faster cryptonets: Leveraging sparsity for real-world encrypted inference, 2018.

[14] S. Cyphers, A. K. Bansal, A. Bhiwandiwalla, J. Bobba, M. Brookhart, A. Chakraborty, W. Constable, C. Convey, L. Cook, O. Kanawi, R. Kimball, J. Knight, N. Korovaiko, V. K. Vijay, Y. Lao, C. R. Lishka, J. Menon, J. Myers, S. A. Narayana, A. Procter, and T. J. Webb. Intel ngraph: An intermediate representation, compiler, and executor for deep learning. *CoRR*, abs/1801.08058, 2018.

[15] L. Ducas and D. Micciancio. Fhew: Bootstrapping homomorphic encryption in less than a second. *Advances in Cryptology – EUROCRYPT 2015*, pages 617–640, 2015.

[16] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012. https://eprint.iacr.org/2012/144.

[17] W. Foundation. Activation function. https://en.wikipedia.org/wiki/Activation_function, Jul 2023.

[18] W. Foundation. Cross entropy. https://en.wikipedia.org/wiki/Cross_entropy, Jul 2023.

[19] W. Foundation. Sigmoid function. https://en.wikipedia.org/wiki/Sigmoid_function, May 2023.

[20] O. FoundationnAI. Learning with errors. https://en.wikipedia.org/wiki/Learning_with_errors, Jun 2023.

[21] P. Gary C. Kessler, 2023.

[22] R. Geelen and F. Vercauteren. Bootstrapping for bgv and bfv revisited. *Journal of Cryptology*, 36(2), 2023.

[23] C. Gentry. Fully homomorphic encryption using ideal lattices. *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 2009.

[24] C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. *Advances in Cryptology - CRYPTO 2013*, pages 75–92, 2013.

[25] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 201–210, New York, New York, USA, 20–22 Jun 2016. PMLR.

[26] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *Journal of machine learning research*, 9:249–256, 2010.

[27] J. Han and C. Moraga. The influence of the sigmoid function parameters on the speed of backpropagation learning, Jan 1995.

[28] E. Hesamifard, H. Takabi, and M. Ghasemi. Deep neural networks classification over encrypted data. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, CODASPY '19, pages 97–108, New York, NY, USA, 2019. Association for Computing Machinery.

[29] D. Huynh. Ckks explained: Part 1, vanilla encoding and decoding, Mar 2021.

[30] B. In. Common loss functions. https://builtin.com/machine-learning/common-loss-functions.

[31] T. Ishiyama, T. Suzuki, and H. Yamana. Highly accurate CNN inference using approximate activation functions over homomorphic encryption. *CoRR*, abs/2009.03727, 2020.

[32] X. Jiang, M. Kim, K. Lauter, and Y. Song. Secure outsourced matrix computation and application to neural networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 1209–1222, New York, NY, USA, 2018. Association for Computing Machinery.

[33] J. Kang. Efficient homomorphic encryption for fixed point arithmetic, 2021.

[34] Leibniz. The early mathematical manuscripts of leibniz. *The Mathematical Gazette*, 10(152):286–287, 1921.

[35] Q. Lou, B. Feng, G. Charles Fox, and L. Jiang. Glyph: Fast and accurately training deep neural networks on encrypted data. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9193–9202. Curran Associates, Inc., 2020.

[36] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM*, 60(6):1–35, 2013.

[37] C. Marcolla, V. Sucasas, M. Manzano, R. Bassoli, F. H. Fitzek, and N. Aaraj. Survey on fully homomorphic encryption, theory, and applications. *Proceedings of the IEEE*, 110(10):1572–1609, 2022.

[38] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[39] M. Minsky. *Perceptrons : an introduction to computational geometry*. The MIT Press, Cambridge, Massachusetts, [2017 edition]. edition, 2017.

[40] K. Munjal and R. Bhatia. A systematic review of homomorphic encryption and its contributions in healthcare industry. *Complex & amp; amp; Intelligent Systems*, 2022.

[41] K. Nandakumar, N. Ratha, S. Pankanti, and S. Halevi. Towards deep neural network training on encrypted data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019.

[42] OpenAI. Gpt-3.5: Language model by openai. https://openai.com, 2021.

[43] W. Pitts and W. S. McCulloch. How we know universals the perception of auditory and visual forms. *The Bulletin of Mathematical Biophysics*, 9(3):127–147, 1947.

[44] Rafael. What is key length in cryptography and why is important?, Oct 2022.

[45] O. Regev. On lattiacmces, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6):1–40, 2009.

[46] R. L. Rivest. Cryptography. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, pages 717–755. Elsevier and MIT Press, 1990.

[47] S. Shanmuganathan. Artificial neural network modelling: An introduction. *Artificial Neural Network Modelling*, pages 1–14, 2016.

[48] I. The MathWorks. *Symbolic Math Toolbox*. Natick, Massachusetts, United State, 2023.

[49] S. university. *Reliable, trainable networks for computing and control*. Technical reports 1554-2. Stanford university, Stanford (Calif.), 1962.

[50] Website.

[51] Website. What is a neural network activation function. https://www.v7labs.com/blog/neural-networks-activation-functions#what-is-a-neural-network-activation-function.

[52] B. Widrow. Generalization and information storage in network of adaline 'neurons'. 1962.

[53] B. Widrow and M. E. Hoff. *Associative Storage and Retrieval of Digital Information in Networks of Adaptive "Neurons"*, pages 160–160. Springer US, Boston, MA, 1962.

[54] K. Yagyu, R. Takeuchi, M. Nishigaki, and T. Ohki. Improving classification accuracy by optimizing activation function for convolutional neural network on homomorphic encryption. In L. Barolli, editor, *Advances on Broad-Band Wireless Computing, Communication and Applications*, pages 102–113, Cham, 2023. Springer International Publishing.