

Secure Machine Learning

Bhavish Raj Gopal

Roll No: MS16049

*A dissertation submitted for the partial fulfilment
of BS-MS dual degree in Science*

Under the guidance of

Dr. Satyajit Jena



April 2021

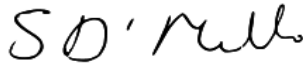
**Indian Institute of Science Education and Research Mohali
Sector - 81, SAS Nagar, Mohali 140306, Punjab, India**

Certificate of Examination


This is to certify that the dissertation titled “**Secure Machine Learning**” submitted by **Bhavish Raj Gopal** (Reg. No. MS16049) for the partial fulfillment of BS-MS dual degree programme of the Institute, has been examined by the thesis committee duly appointed by the Institute. The committee finds the work done by the candidate satisfactory and recommends that the report be accepted.



Dr. Neeraja Sahasrabudhe



Dr. Shane D'Mello



Dr. Satyajit Jena

(Supervisor)

Dated: May 24, 2021

Declaration

The work presented in this dissertation has been carried out by me under the guidance of Dr. Satyajit Jena at the Indian Institute of Science Education and Research Mohali.

This work has not been submitted in part or in full for a degree, a diploma, or a fellowship to any other university or institute. Whenever contributions of others are involved, every effort is made to indicate this clearly, with due acknowledgment of collaborative research and discussions. This thesis is a bonafide record of original work done by me and all sources listed within have been detailed in the bibliography.

Bhavish Raj Gopal

Bhavish Raj Gopal
(Candidate)

Dated: May 24, 2021

In my capacity as the supervisor of the candidate's project work, I certify that the above statements by the candidate are true to the best of my knowledge.



Dr. Satyajit Jena
(Supervisor)

Acknowledgement

First and foremost, I would like to thank my thesis supervisor Dr. Satyajit Jena, without whose help and supervision, this thesis would have never been possible. The discussions that I had with him has enhanced my capabilities as a researcher.

I would like to extend my thanks to Dr. Somitra Sanadhya from IIT Ropar for his invaluable supervision, support and tutelage during the course of my thesis. I would also like to thank my thesis committee members Dr. Neeraja and Dr. Shane for their encouragement, insightful comments and questions.

To my dearest friends: Utkarsh, Antriksh, Arpit, Tejaswer and Saswat for their support and all the sweet memories of IISER life. I would also like to thank members of EHEP lab – Rohit, Nishat, Ruthik, Asrith, Aman, Bidisha, Shubangi, Sourav and Yogesh for a cherished time spent together in the lab, and in social settings. My appreciation also goes out to my family and friends for their encouragement and support all through my studies.

And lastly, I am grateful to INSPIRE for monetary support, the IISER Mohali library for all the resources that helped me in the last 5 years, and MNIST public database for without these my project would not have been possible.

Bhavish Raj Gopal

MS16049

IISER Mohali.

List of Figures

3.1	Recurrent Neural Network Cell	14
4.1	CrypTFlow overview	22
4.2	implementing Secure Training in CrypTFlow	22
5.1	Number of weak learners vs Accuracy	29

List of Tables

3.1	Table to test captions and labels	20
3.2	Table to test captions and labels	20
4.1	comparison of communication complexity of various protocols; $\log p = 8$.	24
4.2	Table to test captions and labels	24
5.1	Comparison of training accuracy	30
5.2	Comparison of test accuracy	30
5.3	Comparison of Training and Inference time	30
D.1	Network A architecture	39
D.2	Network D: LeNet architecture	39

List of Algorithms

1	Matrix Multiplication, $\prod_{MatMul}(\{P_0, P_1\}, P_2)$ [WGC18]	16
2	Compute MSB, $\prod_{MSB}(\{P_0, P_1\}, P_2)$	18
3	Compute DTanH, $\prod_{DTanH}(\{P_0, P_1\}, P_2)$	19

List of Acronyms

AI - Artificial Intelligence

CNN - Convolution Neural Networks

DP - Differential Privacy

FL - Federated Learning

HE - Homomorphic Encryption

ML - Machine Learning

MLaaS - Machine Learning as a Service

MPC - Multi Party Computations

NN - Neural Networks

ReLU - Rectified Linear Unit

RNN - Recurrent Neural Network

TH - Trusted Hardware

Contents

Acknowledgement	i
List of Figures	iii
List of tables	v
List of algorithms	vii
List of Acronyms	ix
Abstract	xv
1 Introduction	1
1.1 Vision	2
1.2 Privacy Enhancing Technologies	2
1.2.1 Privacy Advocates	3
1.3 Multiparty Computations	3
1.4 Target Applications of MPC	4
1.5 Summary of Contributions	5
2 Background and Related Works	7
2.1 Mathematical Concepts	7
2.1.1 Groups, Rings and Fields	7
2.1.2 Communication Complexity	8
2.2 Multi-Party Computation	8
2.2.1 Adversarial Models	9
2.2.2 Arithmetic Secret Sharing	10
2.3 Machine Learning	10

2.3.1	Neural Networks	11
2.3.2	Datasets	11
3	Secure Training and Inference for Recurrent Neural Networks	13
3.1	SecureNN Architecture	13
3.2	Recurrent Neural Networks	14
3.3	Protocol Construction	16
3.3.1	Compute Matrix Multiplication	16
3.3.2	Private Compare	17
3.3.3	Compute MSB	17
3.3.4	Compute DTanH	18
3.3.5	Division	19
3.3.6	End-to-End Protocols	19
3.4	Results	19
3.4.1	Theoretical Evaluations	19
3.4.2	Experimental Evaluations	20
3.5	Summary	20
4	Secure Training in CrypTFlow	21
4.1	CrypTFlow	21
4.1.1	CrypTFlow Architecture	21
4.2	Secure Training on CrypTFlow	22
4.2.1	Compiling ML models to HLIL	23
4.2.2	Float to Fixed Conversion	23
4.2.3	Compiling to MPC protocols	23
4.3	Results	24
4.3.1	Theoretical Evaluations	24
4.3.2	Experimental Evaluations	24
4.4	Summary	25
4.5	Limitations of SecureNN and CrypTFlow	25
5	Secure Ensemble Machine Learning	27
5.1	Ensemble learning	27
5.1.1	Bagging Technique	28

5.2	Sampling The Training Data	28
5.2.1	Experimental Analysis	29
5.3	Results	30
5.3.1	Experimental Evaluations	30
5.4	Summary	30
6	Conclusion	31
6.1	Future Work	32
A	Chapter 3: Supplementary Material	33
A.1	Fixed-Point Arithmetic	33
B	Chapter 4: Supplementary Material	35
B.1	Pseudo Random Functions(PRFs)	35
C	Chapter 5: Supplementary Material	37
C.1	Number of weak learners vs Accuracy	37
D	Network Architecture	39
	Bibliography	40

Abstract

In the last decade, there has been an increase in technologies involving applications of Machine Learning. For instance, Hospitals use Machine Learning tools to predict a disease; Navigation systems predict traffic flow using machine Learning. In the heart of all this technology is sensitive user data, which has led to several privacy concerns. The development of privacy-enhancing technologies enabled systems to collect and perform computations on data while preserving privacy.

We can use several cryptographic tools to develop privacy-enhancing technologies. Multi-party computation(MPC) is one such cryptographic tool where non-colluding parties perform joint computation over data. Privacy is preserved by no party having any information about the data being computed on. In our work, we focus on implementing Multi-Party Computation(MPC) techniques in Machine Learning setting. More specifically, we focus on improving SecureNN, a three-party secure computation framework for Neural Networks(NN) training, and inference.

The SecureNN framework is state-of-the-art; however, it is mainly limited to Convolutional Neural Networks(CNN). In our work, we extend the SecureNN framework to other neural networks such as RNNs, GRU, and LSTMs. We also work on making SecureNN user-friendly by integrating it with TensorFlow. For this, we make significant improvements to the CrypTFlow, a framework for secure inference in TensorFlow. We implement secure training in CrypTFlow by implementing Secure Training functionalities from SecureNN. We also explore ML algorithms that are computationally less expensive and enable parallel computations to reduce the overheads of SecureNN.

Chapter 1

Introduction

The development of Machine Learning(ML) has reshaped technology in the last decade. Today, ML is transforming various sectors like Healthcare, Technology, Social Networks, and E-Commerce. ML has a wide range of applications like image recognition, speech recognition, traffic prediction, self-driving cars, virtual assistant, etc. In the heart of any ML application is data. ML algorithms require a massive amount of data to provide better services. Thus an essential part of developing an ML algorithm is dedicated to data collection.

The data often comes from recording the target users' behavior. The collection of user information is easier than ever before due to its prevalence in the digital world. This raises a privacy concern due to the lack of transparency in how the data is used. Moreover, the user is rarely made aware of the data collection or given control over this data. Medical records, email logs, and location history are some of the commonly collected data. The development of virtual assistants such as Siri, Google Assistant, and Cortana, who are listening to the user all the time, has given rise to more privacy concerns. Finally, the development of IoT services has made user data more accessible without regard for privacy. While ML technologies focus more and more on data collection and improvement on algorithms, they rarely address growing privacy issues. The increasing privacy concern can affect data collection and hinder the development of ML technologies.

In this thesis, we focus on private computations - where the data used in the computation is kept confidential, to build privacy-preserving ML models. This will enable us to use sensitive data for ML while ensuring appropriate privacy for the data. While several tools can achieve private computation, the use of these tools in ML models suffers from significant

communication and computational overheads. Thus, reducing these overheads is essential to build privacy-preserving ML models.

1.1 Vision

I envision a future where technologies are built with a privacy-first approach. The current digital infrastructure requires users to submit their private data entirely in return for services. A privacy-conscious digital infrastructure would enable users to access ML services such as medical diagnosis or video surveillance without surrendering their sensitive information. It would also allow multiple entities to collaborate without having to reveal their private information to one another.

Developing a privacy-conscious world requires advances in multiple paradigms. We need to design efficient cryptographic protocols and privacy-friendly hardware and implement strict policies that enforce the privacy protection. The current private computation protocols are inefficient. For example, the state-of-the-art multiparty private computation protocol is 100 times slower than the plaintext protocol [WTB⁺20]. Thus it is imperative to improve the existing private computation protocol to lay the foundations of a privacy-conscious world.

To this end, This thesis focuses on improving the existing state-of-the-art private computation models and making them more accessible for real-life applications. For example, in chapter 4 we will see how the privacy-preserving cryptographic tools can be made accessible to ML developers.

1.2 Privacy Enhancing Technologies

Privacy Enhancing Technologies enables systems for data collection while preserving privacy. There are several cryptographic tools that can be used to build privacy-enhancing technologies. These tools are called Privacy Advocates. Each tool has different underlying assumptions and privacy guarantees. So it is essential to understand them before building a privacy focused system.

1.2.1 Privacy Advocates

Privacy advocates are mathematical techniques/tools that help in designing privacy systems. Multiparty computation(MPC), Federated Learning(FL), Trusted Hardware(TH), Differential Privacy(DP), and Homomorphic Encryption(HE) are a few commonly used techniques. Each tool has a set of assumptions and provides privacy guarantees based on the assumptions.

For instance, MPC assumes a no collusion assumption along with standard cryptographic assumptions to provide privacy guarantees. DP assumes a centralized entity with data and provides privacy in the notion that - the queries performed to the database are independent of the individual data in the dataset. Similarly, TH has a trust assumption on the part of the system or a piece of hardware and builds privacy systems upon it.

Each tool has its advantages and disadvantages. For example, MPC provides stronger security but suffers from significant communication overheads while FL compromises privacy for faster computation. In this thesis, we focus on MPC and try to reduce the communication overheads without compromising the end-to-end privacy of the systems built on MPC.

1.3 Multiparty Computations

MultiParty computation allows a set of parties with private inputs to compute a joint function without revealing their private input to one another or anyone else. One of the earliest examples of MPC is Yao's Millionaires problem introduced by Andrew Yao in 1982 [Yao86]. Consider two millionaires who wish to know who is richer without revealing any information about their wealth to each other. Normally they would find a trusted third party to disclose their information and do the computation. Yao, in 1982, laid the foundations of MPC that can solve the millionaires problem without the need for a trusted third party. Soon, Yao introduced the first set of MPC protocols called the Garbled circuit [Yao82, Yao86]. This protocol was applicable for any number of parties and any arbitrary function. However, the protocol suffered from significant overheads due to the lack of computation power and inefficient algorithms.

Since then, MPC protocols have become more efficient due to the improvements in computation, protocol designs, communication infrastructure, and hardware support. Today,

the state-of-the-art MPC protocol FALCON [WTB⁺20] can solve Yao’s Millionaires problem in $1.28\mu s$, which is just 1 order of magnitude slower than plaintext comparison. The progress of MPC has enabled us to scale it for larger and more complex problems.

This thesis focuses on designing and improving MPC protocols specifically for Neural networks(NN). Instead of generic protocols, targeting one class of algorithms help us build more efficient protocols.

1.4 Target Applications of MPC

Neural Networks(NN) is a class of machine learning algorithms widely used for Healthcare, Image classification, Natural Language Processing, etc. The accuracy of a NN model directly depends on the size of training data. So, multiple contributors may need to pool their data to train a NN model efficiently. But they may not be able to do so due to the data’s sensitive nature or compliance requirements. MPC can provide an elegant solution in this scenario. Moreover, in cases where a service provider has a trained model and provides inference on sensitive user data, the service provided and the client can execute an MPC protocol to run a secure inference. MPC guarantees that neither the service provider nor the client will be able to learn any information about each other. Below we describe a real-life example where the methods developed in this thesis can be used.

Healthcare Prediction: Suppose a service provider provides a healthcare prediction service by monitoring users’ daily activity. The user data collected contains sensitive information like sleep patterns, daily routine, etc. The service provider has to pool the user data to train an efficient ML model. But the sensitive nature of the data here hinders the development of the ML service. Using MPC protocols developed in this thesis, we can provide an elegant solution to this problem. The users and the client can run a secure joint training protocol. First, each user sends ”secret shares” of their data to private servers using an arithmetic secret sharing scheme (ref. section 2.2.2). The servers collectively run a MPC protocol to train a NN over the joint data. The protocol ensures that the parties’ private data is not revealed to any other party or the service provider. The trained model can now be used for inference by any party without having to reveal their inputs.

An extended application of the above model is the following: A group of N hospitals wishes to jointly train a model on sensitive patient data. They can train the NN model using the

protocol described above and set up an MLaaS to help predict rare diseases. The service can be set so that the input and output are revealed only to the patient. Further, it can also be set such that the NN model’s information is not revealed to the user.

1.5 Summary of Contributions

This thesis aims to improve the efficiency of MPC-based private ML. It proposes new techniques that reduce the communication overheads significantly for adoption. Apart from building efficient protocols, it also explores private learning from an ML perspective, adopting computationally less expensive models and parallel learning models to make real-life applications possible. Specifically, this thesis makes the following contributions:

1. **Secure training and inference on Recurrent neural networks:** A novel MPC protocol for private learning on recurrent neural networks. The approach builds upon the SecureNN [WGC18] framework by improving the underlying protocols and developing protocols specific for RNNs.
2. **Secure Training on CrypTFlow [KRC⁺20]:** A framework to securely train NNs built on TensorFlow. This framework makes private ML more accessible to ML developers. It also introduces efficient protocols that reduce communication overheads by 16%.
3. **Secure Ensemble Learning:** A framework for securely training ensemble NNs. This framework reduces neural networks’ training time by more than 50% by training multiple computationally less expensive models in parallel and combining them.

Chapter 2

Background and Related Works

In this chapter, we introduce some concepts that help in a deeper understanding of this thesis work. We will also introduce the notations followed in this thesis along with the required primitives. We start by going through some basic mathematical concepts like group theory, and complexity. Later we introduce Multi-Party Computations and describe the 3-Party Computation(3PC) used in chapters 3, 4, and 5. Finally, we also describe the Neural Networks used for evaluation.

2.1 Mathematical Concepts

2.1.1 Groups, Rings and Fields

Definition 2.1.1 *A Set G equipped with the binary operation $*$ is called a Group if it satisfies the following axioms*

- **Closed under the binary operation $*$:** For all $a, b \in G, a * b \in G$
- **Associative:** For all $a, b, c \in G, (a * b) * c = a * (b * c)$
- **Identity:** There exist an unique element in G , denoted by e , such that for all $a \in G, (a * e) = e * a = a$
- **Inverse:** For all $a \in G$, There exists $a^{-1} \in G$ such that $a * a^{-1} = a^{-1} * a = e$

The group G is Abelian if all the elements commute under the operation $*$, i.e For all $a, b \in G, a * b = b * a$

Definition 2.1.2 *A Set R equipped with two binary operation $(+, \times)$ is called a Ring if it satisfies the following*

- **$(R, +)$ is an abelian group**
For all $a, b, c \in G, (a \times b) \times c = a \times (b \times c)$

- **Distributive:** For all $a, b, c \in G$,

$$(a + b) \times c = (a \times c) + (b \times c)$$

$$a \times (b + c) = (a \times b) + (a \times c)$$

Definition 2.1.3 A Set F equipped with two binary operation $(+, \times)$ is called a Field if it satisfies the following

- $(R, +)$ is an abelian group
- $(R - \{0\}, \times)$ is an abelian group

Fields are fundamental to cryptography due to the existence of multiplicative inverse. The set of integers modulo n , denoted by \mathbb{Z}_n , is a Ring. \mathbb{Z}_p is a field when p is a prime number. We use the Ring \mathbb{Z}_n as the number space for Fixed Point Representations (ref Appendix A).

2.1.2 Communication Complexity

Communication complexity refers to the amount of communication required to solve a particular problem. Suppose Alice and Bob have their respective l -bit inputs x and y and wish to compute a function $f(x, y)$. Then, the communication time refers to the number of communications required between Alice and Bob to compute $f(x, y)$. Formally,

Definition 2.1.4 Let $X = Y = \{0, 1\}^n$, $Z = \{0, 1\}$ and $f : X \times Y \rightarrow Z$ be a function. Let Alice hold a input $x \in X$ and bob hold a input $y \in Y$. Alice and Bob can communicate by transmitting one bit at a and wish to compute $f(x, y)$. Then the worst case communication complexity is given by $D(f)$.

$D(f)$ = minimum number of communication required in the worst case scenario.

The function $D(f)$ is a function of the input size. We use worst case communication complexity to measure the efficiency of the protocols designed in this thesis.

2.2 Multi-Party Computation

In MPC, a set of parties jointly compute a function without revealing their private inputs. At the end of the protocol, all parties have the same output, and no subset of parties colluding with each other should be able to force an incorrect output(Correctness). It is important to understand MPC does not deal with the amount of information revealed by the function output as the parties involved are already aware of that risk. Instead, it ensures that the

parties involved learn nothing more about the inputs other than what is revealed by the output(Privacy).

The MPC protocols take place in rounds which consist of a computation phase and a communication phase. In the computation phase, each party performs as much computation as possible locally. In the communication phase, each party communicates as much data as possible before moving to the next round. The computations done locally are private and accessible only to the party performing them. The party can then choose to transmit the necessary output to other parties during the communication phase. The protocol's efficiency depends on the number of communication rounds between the parties in computing a function.

The Adversarial models describe the conditions under which MPC protocols are secure. The type of Adversarial model may modify the properties(Privacy and Correctness) of MPC protocols. For instance, a particular model may ensure correctness for collusion between any proper subset of parties in one adversarial model. In contrast, in another adversarial model, the property may hold only for a threshold of parties colluding. The following section discusses the different adversarial models considered in this thesis.

2.2.1 Adversarial Models

The adversarial model is used to quantify the security properties of the MPC protocol. Several parameters can define the power of the adversary like the computing power, threat model, threshold of corruption, etc. We will introduce the adversarial model used in this thesis.

Semi-honest adversary: A semi-honest adversary follows the protocol strictly while trying to infer as much information as possible. For instance, If the protocol describes to chose a random number and send it to another party, the adversary follows through with it honestly and tries to infer any information from that. Such an adversary is similar to real-life situations where parties that wish to train a model jointly will not deter from the protocol to steal information.

The adversary is computationally bounded, meaning it is assumed to run in probabilistic polynomial time. The security relies on the assumed hardness of some problem(like factoring a large number).

2.2.2 Arithmetic Secret Sharing

Secret sharing is a scheme where an ℓ -bit secret is distributed between a set of parties such that each party has a share of the secret. The distribution is such that the individual party's share does not reveal any information about the secret and the secret can be reconstructed only when a sufficient number of parties combine their share. The secret sharing scheme was invented by Adi Shamier [Sha79] and George Blakely [Bla79] in 1979. Below we explain the secret sharing scheme used in this thesis.

2-out-of-2 secret sharing - This is the scheme used to share a secret s between 2-parties. Given a secret s in a ring \mathbb{Z}_n , the shares s_1, s_2 are constructed such that $s_1 + s_2 = s \pmod{n}$.

3-out-of-3 secret sharing - This is the scheme used to share a secret s between 3-parties. Given a secret s in a ring \mathbb{Z}_n , the shares s_1, s_2, s_3 are constructed such that $s_1 + s_2 + s_3 = s \pmod{n}$.

2-out-of-3 secret sharing - This is the scheme used to share a secret s between three parties (3PC) such that the secret can be reconstructed from the shares of any two parties. Given a secret s in a ring \mathbb{Z}_n , the shares s_1, s_2, s_3 are constructed such that $s_1 + s_2 + s_3 = s \pmod{n}$. Now distributing $(s_1, s_2), (s_2, s_3), (s_3, s_1)$ between the three parties will form a 2-out-of-3 secret sharing scheme.

We will use $\langle s \rangle^L$ to denote secret sharing scheme in \mathbb{Z}_L and the shares of the secret will be denoted by $\langle s \rangle_0^L, \langle s \rangle_1^L$, and $\langle s \rangle_2^L$.

2.3 Machine Learning

Machine Learning is a subset of AI, where systems learn from data and identify patterns and inferences. Such algorithms are widely used for health risk predictions, image classification and natural language processing. In this thesis we focus specially on a class of ML algorithms called supervised learning algorithms. In supervised learning the algorithm is fed enormous amount of data along with the appropriate inference of each data point. The algorithm iterates over the data and modifies its parameter and converges for a set of values. This step is called training. The trained model is then evaluated on a test data set that is independent of the training dataset.

2.3.1 Neural Networks

We consider two NN architectures to evaluate our protocols in this thesis:

- **Network A:** is a simple 3-layered fully connected Neural Network with 118K parameters. (Ref: D.1)
- **Network B:** It is the LeNet network proposed in [LBBH98]. The network contains 2 convolution layers followed by 2 fully connected layers and has 431K parameters. (Ref: D.2)

Depending on the circumstances we evaluate the network with either ReLU or TanH activation Function.

2.3.2 Datasets

All our networks are trained and tested with MNIST [LC10] dataset that is used for image recognition. The MNIST dataset is a collection of images of hand written digits. The dataset has 60,000 images in training set and 10,000 images in testing set. Each image is of 28x28 pixel resolution along with a label between 0 to 9.

Chapter 3

Secure Training and Inference for Recurrent Neural Networks

Neural Network is a powerful machine learning tool used for real-life applications. Large amounts of training data are required to achieve high accuracy of a Neural Network. In real life, this requires multiple parties to combine their data, and it violates their privacy. This chapter focuses on building novel Three-Party secure training and inference protocol for recurrent Neural Networks(RNN) using the SecureNN framework.

SecureNN is a framework for secure training and inference on Neural Networks[WGC18]. However, SecureNN is mostly limited to Convolution Neural Network as it provides secure computation protocol only for ReLU activation function. This work provides a secure three-party computation protocol for tanH activation function based on the techniques used in SecureNN.

We start by building secure MPC protocols for the building blocks of RNN, such as matrix multiplication, TanH activation, and batch normalization. We then put them together to create an end-to-end secure computation protocol for RNNs.

3.1 SecureNN Architecture

Traditional Protocols use Beavers Triples or Homomorphic encryption for arithmetic computations and Yao's Garbled Circuits for boolean computations such as ReLU, Maxpool, and their derivatives. Additional Share Convert protocols are required to move from arithmetic encoding to boolean encoding. The communication cost for boolean computation

is expensive due to the use of Yao’s Garbled circuit. SecureNN introduces new protocols for boolean computations that reduce the communication cost and eliminates interconversion gates. We use similar techniques to compute TanH and its derivatives and extend the SecureNN framework for RNNs.

SecureNN is a three-party system, and P_0 , P_1 , and P_2 denote the three servers. P_0 and P_1 are the parties that wish to execute the secure computation protocol. P_2 provides relevant randomness and assists in the computation during the two-party protocol while not learning any sensitive information. There is an invariant 2-Out-of-2 secret sharing scheme. This means that the input and the output of any protocol between P_0 and P_1 is a secret shared between them. The Adversarial model used is a computationally bound semi-honest adversary, Which can also be extended to a malicious adversary[WGC18].

3.2 Recurrent Neural Networks

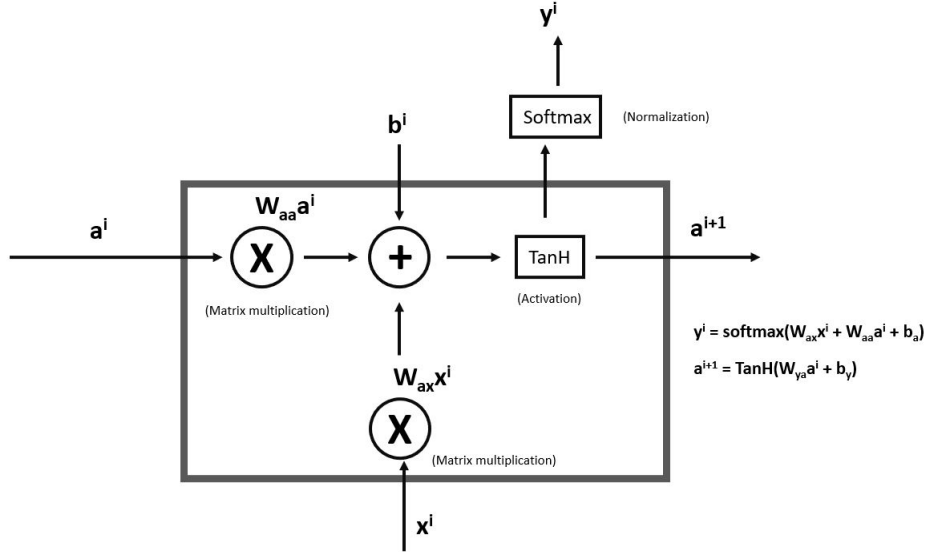


Figure 3.1: Recurrent Neural Network Cell

A simple RNN cell consists of three building blocks.

- Matrix Multiplication
- Non Linear activation Functions and derivatives
- Normalization (Softmax)

In this section, we describe our ideas for computing each of the building blocks.

Matrix Multiplication: The protocol for matrix multiplication uses Beavers triplets algorithm from [Bea91] generated using pseudo random functions. The protocol is highly

efficient and well defined.

Nonlinear Functions: We extend the techniques mentioned in SecureNN for computing ReLU to other activation functions like TanH. To do so, we first apply a piece-wise linear approximation for TanH function. The TanH activation function and its derivative is given by

$$\text{TanH}(x) = \frac{(e^x - 1)}{(e^x + 1)} \quad D\text{TanH}(x) = 1 - f(x)^2$$

The simplest approximation for TanH is given by the function

$$\text{TanH}(x) \approx f(x) = \begin{cases} -1 & x < -1 \\ x & -1 \leq x \leq 1 \\ 1 & x > 1 \end{cases}$$

$$D\text{TanH}(x) \approx f'(x) = \begin{cases} 0 & x < -1 \\ 1 & -1 \leq x \leq 1 \\ 0 & x > 1 \end{cases}$$

Thus $f'(x)$ can be represented as

$$f'(x) = (x - 1 < 0) \oplus (x + 1 < 1)$$

We used fixed-point representation with a precision of 13 to represent the numbers. The values of the numbers lie in the ring $\mathbb{Z}_{2^{64}}$. The first half elements of the ring represent positive numbers and the second half represents the negative numbers. Thus for any x in this number space, the $\text{MSB}(x) = 0$ if $x > 0$ and $\text{MSB}(x) = 1$ if $x < 0$. Therefore, our function $f'(x)$ can be represented as

$$f'(x) = \text{MSB}(x)(x - 1) \oplus \text{MSB}(x)(x + 1), \quad x \in \mathbb{Z}_{2^{64}}$$

The MSB computation can be converted to an LSB computation when the shares of the private input are over an odd ring. Precisely, $\text{MSB}(a) = \text{LSB}(2a)$ [WGC18]. So, we convert the shares of x in $\mathbb{Z}_{2^{64}}$ to shares in the ring $\mathbb{Z}_{2^{64}-1}$. Thus,

$$f'(x) = \text{LSB}(x)(2(x - 1)) \oplus \text{LSB}(x)(2(x + 1)), \quad x \in \mathbb{Z}_{2^{64}-1}$$

Normalization: For a given set of values $\{x_1, x_2, \dots, x_n\}$, the normalized values are given by $\{\frac{x_i}{\sum_{j=1}^n x_j}\}$. We use secure division protocol from [WGC18] for normalization of our outputs. We perform bit wise long division to compute the normalized value.

3.3 Protocol Construction

In the next section, we present the construction of the protocols required and explain the intuition behind them. Some of the protocols used are well established and thus we do not concern ourself with the security proofs. For the protocols that are developed in this thesis we provide the security proofs in the Appendix B.

3.3.1 Compute Matrix Multiplication

Algorithm 1: describes the protocol for matrix multiplication used in [WGC18]. At the start of the protocol P_0 and P_1 hold shares of $X, Y \in \mathbb{Z}_L^{m \times v}$. P_0 and P_1 receive shares of $Z = X \cdot Y$ at the end of the protocol.

Algorithm 1: Matrix Multiplication, $\prod_{MatMul}(\{P_0, P_1\}, P_2)$ [WGC18]

Input: P_0, P_1 hold $(\langle X \rangle_0^L, \langle Y \rangle_0^L)$ and $(\langle X \rangle_1^L, \langle Y \rangle_1^L)$, respectively

Output: P_0, P_1 get $\langle XY \rangle_0^L$ and $\langle XY \rangle_1^L$

- 1 P_2 picks random matrices $A, B \in \mathbb{Z}_L^{m \times v}$ and generates for $j \in \{0, 1\}$, $\langle A \rangle_j^L, \langle B \rangle_j^L, \langle C \rangle_j^L$ and sends to P_j , where $C = A \cdot B$
 - 2 For $j \in \{0, 1\}$, P_j computes $\langle E \rangle_j^L = \langle X \rangle_j^L - \langle A \rangle_j^L$ and $\langle F \rangle_j^L = \langle Y \rangle_j^L - \langle B \rangle_j^L$
 - 3 P_0 & P_1 reconstruct E & F by exchanging shares.
 - 4 For $j \in \{0, 1\}$, P_j outputs $-jE \cdot F + \langle X \rangle_j^L \cdot F + E \cdot \langle Y \rangle_j^L + \langle C \rangle_j^L + U_j$
-

Intuition: The algorithm is straightforward. We claim that that the outputs computed by P_0 and P_1 in step(4) is their respective share of $X \cdot Y$. P_0 computes,

$$\langle Z \rangle_0^L = \langle X \rangle_0^L \cdot F + E \cdot \langle Y \rangle_0^L + \langle C \rangle_0^L + U_0$$

and P_2 computes

$$\langle Z \rangle_1^L = -E \cdot F + \langle X \rangle_1^L \cdot F + E \cdot \langle Y \rangle_1^L + \langle C \rangle_1^L + U_1$$

. Adding $\langle Z \rangle_0^L$ and $\langle Z \rangle_1^L$, we get

$$\langle Z \rangle_0^L + \langle Z \rangle_1^L = C + X(Y - B) + Y(X - A) - (X - A)(Y - B) = X \cdot Y$$

3.3.2 Private Compare

We invoke \prod_{PC} , The private compare protocol from [WGC18] for comparison. P_0 and P_1 hold shares of two l -bit integers x and r at the start of the protocol. They also hold share of a bit α . P_2 learns $\alpha' = \alpha \oplus (x > r)$ at the end of the protocol, where $(x > r)$ is 1 if true. Note that P_2 never learns the value of $(x > r)$ as it is masked by a random bit α . However, P_2 and share the shares of α' to P_0 and P_1 and they can compute the shares of $(x > r)$. The algorithm itself is used as a black box and is omitted from discussion here.

3.3.3 Compute MSB

Algorithm 2: describes the protocol for Computing MSB used in [WGC18].

We first define a wrap function as follows,

$$wrap(x, y, L) = \begin{cases} 0 & x + y \leq L \\ 1 & x + y \geq L \end{cases}$$

The wrap function denotes whether $x + y$ overflows L . Thus, we can relate

Notice that for $x \in \mathbb{Z}_{2^{64}}$

$$MSB(x) = LSB(2x) = wrap(x, x, 2^{64})$$

With this we compute MSB using Algorithm 2. The parties P_0 and P_1 have the shares of x at the start of the protocol and end up with the shares of $MSB(x)$ at the end of the protocol.

Intuition: Notice that for $a_0, a_1 \in \mathbb{Z}_L$

$$a_0 + a_1 \pmod{L} = a_0 + a_1 - \gamma L, \text{ where, } \gamma = wrap(a_0, a_1, L)$$

we make use of this relation to calculate $\theta = wrap(2x_0, 2x_1, L)$. P_0 and P_1 start with

Algorithm 2: Compute MSB, $\prod_{MSB}(\{P_0, P_1\}, P_2)$

Input: P_0, P_1 hold $(\langle x \rangle_0)$ and $(\langle x \rangle_1)$, respectively

Output: P_0, P_1 compute $\langle MSB(x) \rangle_0$ and $\langle MSB(x) \rangle_1$

Common Randomness: P_0 and P_1 hold a random bit η'' , a random $r \in \mathbb{Z}_L$,
 $\alpha = \text{wrap}(\langle r \rangle_0, \langle r \rangle_1, L)$ and shares of zero over L denoted
by u_0 and u_1

- 1 P_0 and P_1 compute $(\langle a \rangle_0) = 2(\langle x \rangle_0)$ and $(\langle a \rangle_1) = 2(\langle x \rangle_1)$
 - 2 For $j \in \{0, 1\}$, P_j executes step 3-4.
 - 3 $\langle a' \rangle_j = \langle a \rangle_j + \langle r \rangle_j$
 - 4 Send $\langle a' \rangle_j$ to P_2
 - 5 P_2 computes $y = \langle a' \rangle_0 + \langle a' \rangle_1$ and $\delta = \text{wrap}(\langle a' \rangle_0, \langle a' \rangle_1, L)$
 - 6 P_2 generates shares $\{\langle y[i] \rangle_j\}_i$, and $\langle \delta \rangle_j$ sends to P_j
 - 7 P_0, P_1, P_2 call $\prod_{PC}(\{P_0, P_1\}, P_2)$ with $P_j, j \in \{0, 1\}$, having inputs
 $(\{\langle y[i] \rangle_j\}_{i \in [l]}, r - 1, \eta'')$ and P_2 learns η'
 - 8 P_2 generates $\langle \eta' \rangle_j$ and sends to P_j for $j \in \{0, 1\}$.
 - 9 For $j \in \{0, 1\}$, P_j executes Steps 10–11.
 - 10 $\langle \eta \rangle_j = \langle \eta' \rangle_j + (1 - j)\eta'' - 2\eta\langle \eta' \rangle_j$
 - 11 outputs: $\langle \theta \rangle_j = \langle \beta \rangle_j + (1 - j)(-\alpha - 1) + \langle \delta \rangle_j + \langle \eta \rangle_j$
-

common randomness α and r such that $\alpha = \text{wrap}(r_0, r_1, L)$. P_0 and P_1 compute $a_0 + r_0$ and $a_1 + r_1$ respectively and send it to P_2 . P_2 then computes $\delta = \text{wrap}(a'_0, a'_1, L)$ and $y = a'_0 + a'_1$ and sends shares of δ and bit wise shares of y to P_0 and P_1 . Now, P_2 assists P_0 and P_1 in realising the value of $\eta'' = (x > r - 1)$. Finally θ can be computed from the equation,

$$\theta = \beta_0 + \beta_1 - \alpha + \delta + \eta - 1$$

P_0 and P_1 contain either the shares or the whole of the components in the above equation. So, they can compute shares of θ .

3.3.4 Compute DTanH

The protocol for computing DTanH involves computing $MSB(x-1)$ and $MSB(x+1)$. We use Algorithm 2 to compute MSB.

Intuition: The protocol is straightforward. The parties first compute shares of $2(x - 1)$ and $2(x + 1)$. They then invoke \prod_{MSB} from Section 3.3.3 to get the output.

Algorithm 3: Compute DTanH, $\prod DTanH(\{P_0, P_1\}, P_2)$

Input: P_0, P_1 have $\langle x \rangle_0^L$ and $\langle x \rangle_1^L$, respectively

Output: P_0, P_1 compute $\langle MSB(x-1) \oplus MSB(x+1) \rangle_0^L$ and $\langle MSB(x-1) \oplus MSB(x+1) \rangle_1^L$

- 1 For $j \in \{0, 1\}$, parties P_j computes $\langle a \rangle_j^L = 2(\langle x \rangle_j^L - j)$ and $\langle b \rangle_j^L = 2(\langle x \rangle_j^L + j)$
 - 2 P_0, P_1, P_2 run $\prod_{MSB}(\{P_0, P_1\}, P_2)$ with $P_j, j \in \{0, 1\}$ for inputs $\langle a \rangle_j^{L-1}, \langle b \rangle_j^{L-1}$ and P_j learns $\langle \alpha \rangle_j^L = \langle MSB(x-1) \rangle_j^L$ and $\langle \beta \rangle_j^L = \langle MSB(x+1) \rangle_j^L$, resp.
 - 3 Party P_j outputs $\langle \alpha \rangle_j^L \oplus \langle \beta \rangle_j^L$
-

3.3.5 Division

We invoke \prod_{DIV} , The private division protocol from [WGC18] for division. At the start of the protocol, the parties have shares of x and y , and at the end, they receive shares of x/y . The protocol is used as a black box and is omitted from discussion here.

3.3.6 End-to-End Protocols

The above protocols can be easily put together for secure training and inference in RNNs. For a single cell of RNN, we first invoke Matrix multiplication protocol, then compute and store DTanH and then calculate the value of TanH. Finally, we normalize using division protocol. For back-propagation we use the value of DTanH store and make calls for Matrix multiplication protocol. The Protocols can be put together easily because of the 2-out-of-2 invariant arithmetic sharing property of all the protocols.

3.4 Results

3.4.1 Theoretical Evaluations

Table 3.1 shows the communication rounds and communication complexity for various building blocks involved. These evaluations are for ℓ -bit input and p denotes the field size. $\text{MatMul}(m, n, v)$ denotes matrix multiplication between matrices of dimension $m \times n$ and $n \times v$.

Protocol	Rounds	Communications
MatMul(m,n,v)	2	$2(2mn + 2nv + mv)\ell$
PrivateCompare	4	$2\ell \log p$
Compute MSB	5	$4\ell \log p + 6\ell$
DTanH	8	$8\ell \log p + 12\ell$

Table 3.1: Table to test captions and labels

3.4.2 Experimental Evaluations

We perform training and inference over MNIST data for 2 Networks with a learning rate of 2^{-5} . We perform an average of 10 iterations. We measured the time for 10 forward-backward passes in each iteration and used it to extrapolate the numbers for 15 epochs. (7000 iterations).

Network	Training Time(hrs)	Inference Time(hrs)
Network A	1.41	0.09
Network B	4.86	0.21

Table 3.2: Table to test captions and labels

3.5 Summary

We extend the SecureNN framework for a new class of NNs. We build on the protocols of [WGC18] and build slightly efficient protocol for computing Non-linear Activation functions. Using the improved protocol we compute TanH activation function. Using the methods we are able to achieve over 98% accuracy on MNIST datasets.

Chapter 4

Secure Training in CrypTFlow

In this chapter, we focus on making SecureNN accessible to ML developers. The main disadvantage of SecureNN is Fixed Point arithmetic and the use of static language. Many ML developers use Python or R for developing ML models and are not familiar with the cryptographic backend. We make SecureNN accessible by integrating it with TensorFlow using the CrypTFlow Framework. CrypTFlow is a framework for secure inference in TensorFlow. It is based on SecureNN and converts any TensorFlow inference code into a Secure Multi-Party Computation protocol. We improve CrypTFlow to compile models for secure training that will allow parties to jointly train models while preserving privacy. The improved model makes it more accessible for ML developers. Moreover, this extends the framework for online learning models.

4.1 CrypTFlow

CrypTFlow consists of three components Athos an end to end compiler that compiles TensorFlow inference code to secure MPC protocols, Porthos which is a semi-honest 3 party computation protocol based on SecureNN and Aramis which provides malicious security to any semihonest secure MPC protocol.

4.1.1 CrypTFlow Architecture

CrypTFlow converts any TensorFlow inference code to a Secure MPC protocol. Figure 1 shows the crypTFlow architecture. First, the metadata and graph dump is created while training the model. The graph dump stores information about the tensors involved and

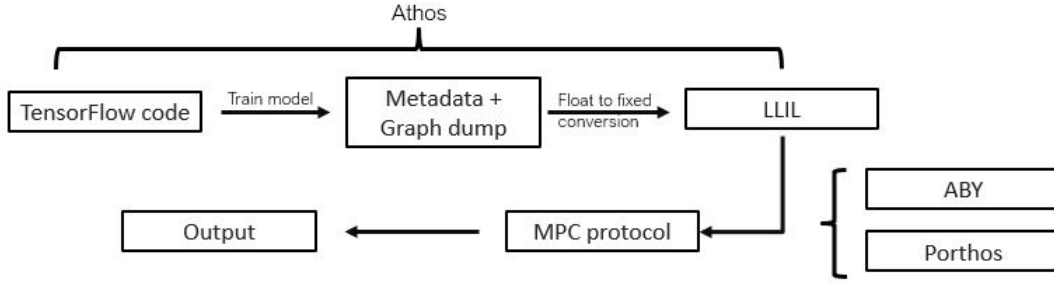


Figure 4.1: CryptFlow overview

their dimensions, which is required to compile the model to a High-Level Intermediate Language(HLIL) language. The model is then converted from floating-point to fixed-point arithmetic and compiled to a C-like language called Low-Level Intermediate Language(LLIL). The LLIL calls in functions from Porthos to compile the model into an MPC protocol. The MPC model is then used to perform secure inference.

4.2 Secure Training on CryptFlow

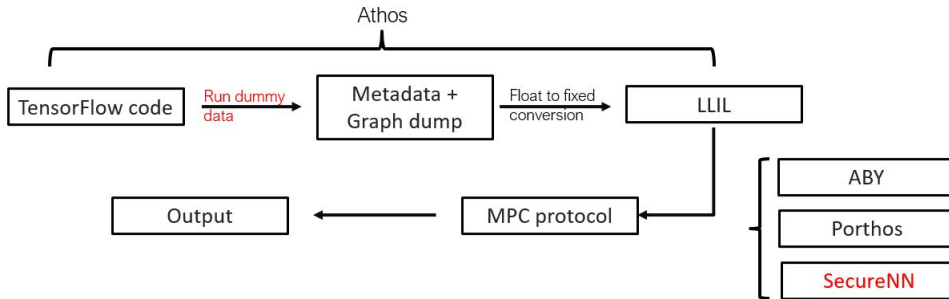


Figure 4.2: implementing Secure Training in CryptFlow

The CryptFlow framework allows for secure inference on the trained model. We focus on implementing secure training functionality on the CryptFlow framework to enable parties to train a model jointly. This improves the scope for adoption by ML developers. We introduce two changes to the CryptFlow framework. First, we enable CryptFlow to compile models and generate Metadata and Graph dump without training the model. Next, we import the secure training functionalities from SecureNN. This allows us to compile the models to Secure MPC protocol for joint training.

4.2.1 Compiling ML models to HLIL

The Athos frontend Compiles ML models into HLIL. The HLIL is statically typed with explicit tensor dimensions, while TensorFlow is dynamic and does not have tensor dimensions. We run the TensorFlow code on one dummy input and generate the metadata. The Athos Frontend can then compile the metadata into HLIL. The generation of dummy data and compiling of the model can be performed by one of the parties and does not affect privacy. We now explain in detail how to generate and store the metadata.

We use the TensorFlow graph transform tool [TFI] to generate a graph dump: the graph dump stores all the necessary operations involved and the corresponding tensor dimensions. We compile the graph dump to a High-Level Intermediate Language(HLIL) that supports tensor manipulations. The HLIL uses floating point arithmetic hence we perform float-to-fixed conversion before compiling.

4.2.2 Float to Fixed Conversion

This section explains how we move from floating-point arithmetic in HLIL to fixed-point arithmetic and then compile the models to Low-Level Intermediate Language(LLIL). MPC protocols are more efficient with fixed point integers. The translation from float to fixed is parametrized by a scale parameter s . The parameter s denotes the precision of the fixed-point numbers and is set to 13. We define a function $\rho \rightarrow \mathbb{Z}_{2^{64}}$ such that

$$\rho(r) = \lfloor r \cdot 2^{13} \rfloor$$

For Matrices of real values, the function ρ is applied element-wise to each element of the matrix. After applying the required conversions for the tensors the model is compiled to LLIL which is a cryptoaware C-like language. LLIL supports. The LLIL has integer valued tensors and can make calls to Porthos, ABY and SecureNN. The LLIL program is then compiled to C++ as an MPC protocol.

4.2.3 Compiling to MPC protocols

We implement Secure Training functionalities from SecureNN to compile the LLIL program into MPC protocol for secure training. We make one crucial change to the MSB

computation protocol following the ideas from [KRC⁺20]. The MSB computation protocol in Chapter 3 requires P_2 to send fresh shares of a value to parties P_0 and P_1 . It is observed that these shares can be generated as a output of Pseudo Random Function(PRF) B.1 shared between P_2 and P_0 . Say, the PRF outputs a value r which is shared between P_2 and P_1 . then the shares of a value y will be y and $y - r$. Then P_2 has to communicate the share only to P_0 and this reduces the communication cost by half. This reduces the overall communication cost of ComputeMSB by 25%.

We also make calls to Porthos from [KRC⁺20] to implement efficient protocols for Conv2D, Maxpool etc. Finally, we compile the program to C++ as an MPC protocol for Secure Training.

4.3 Results

4.3.1 Theoretical Evaluations

Table 4.1 shows the comparison between various protocols of Porthos [KRC⁺20] and the improved SecureNN. These evaluations are for ℓ -bit input and p denotes the field size.

Protocol	Communications(Porthos)	Communications(Improved SecureNN)
ComputeMSB	$4\ell \log p + 13\ell$	$3\ell \log p + 4\ell$
ReLU	$6\ell \log p + 19\ell$	$6\ell \log p + 9\ell$
DTanH	*	$6\ell \log p + 8\ell$

Table 4.1: comparison of communication complexity of various protocols; $\log p = 8$

4.3.2 Experimental Evaluations

We perform training and inference over MNIST data for 2 Networks with a learning rate of 2^{-5} .

Network	Training Time(hrs)	Inference Time(hrs)
Network A	1.26	0.09
Network B	4.06	0.21

Table 4.2: Table to test captions and labels

4.4 Summary

We implement secure training in CrypTFlow frameworks that allows multiple parties to jointly train the model. This makes Secure Machine learning accessible for ML developers. We improve the efficiency of ComputeMSB and ReLU protocol of CrypTFlow. The improved protocol reduces the overall communications by 16%.

4.5 Limitations of SecureNN and CrypTFlow

MPC protocols developed in this thesis have less communication overheads compared to prior works. However, they are still inefficient and not relevant for real-life applications. To make Secure Machine Learning available for real-life applications, we need to focus not only on improving the efficiency of MPC protocol but also explore ML techniques that are computationally less expensive. Some recent works focus on altering the training and inference algorithm to reduce overheads. In the next chapter, we explore ML techniques that are computationally less expensive and enable parallel computations. We develop MPC protocols for these techniques to improve the communication overheads and take one step further in making secure machine learning available for real-life applications.

Chapter 5

Secure Ensemble Machine Learning

Ensemble Learning [VM02] is a machine learning technique where multiple weak learners are strategically combined to form a strong learner. Training a weak learner is computationally less expensive. Certain Ensemble learning models can train the weak learners parallelly to reduce the training time [VSA⁺10]. In this chapter, we focus on developing MPC protocols for such ensemble models. In particular, we focus on Bootstrap Aggregating or Bagging ensemble model, which allows for parallel computation. Bagging has two additional steps compared to standard ML models - Splitting the training data and combining the weak learners. We introduce MPC protocols for these two steps in SecureNN [WGC18] to build an end-to-end secure computation protocol for ensemble learning.

5.1 Ensemble learning

Ensemble Learning is a powerful tool usually used to improve the performance of a model. It involves training weak learners and combining them to build a better model. But in this thesis, we use ensemble model for two main reasons.

- Training weak learners is computationally less expensive
- Parallel training of weak models reduces the total training time.

Some commonly used ensemble techniques are Bagging, Boosting, and Stacking [Bü12]. This thesis focuses only on bagging as the weak learners are independent of each other and can be trained in parallel [Ode19].

5.1.1 Bagging Technique

Bagging refers to Bootstrap Aggregating [Bü12]. Suppose we have training dataset T of size N , bagging generates m new subsets of training datasets (T_i) of size $n(\leq N)$ by sampling T uniformly with replacements. The subsets are called "bootstrap" samples. Sampling with replacement ensures that the m bootstrap samples are independent of each other. Now, m weak learners are trained using the m bootstrap samples and are aggregated by majority voting to form an ensemble learning model.

Below we describe our ideas of MPC protocols for sampling the data set and majority voting.

Sampling the datasets: P_0 and P_1 have shares of the training data $T = (X_1, X_2, X_3, \dots, X_N)$ of size N and wish to train m weak learners using a bootstrap sample of size $n(\leq N)$. P_0 chooses n samples from T in random with replacements and sends the index of the chosen samples to P_1 . The process is repeated m times to obtain bootstrap samples for m weak learners.

The bootstrap samples are used to train the weak models in parallel using the techniques developed earlier in this thesis (ref. Chapter 3).

Majority voting: The parties P_0 and P_1 exchange the shares of output and compute the majority voting separately. This works because of the semi honest nature of the adversary. In case of a malicious adversary we use PRFs shared between P_0 and P_1 to generate the bootstrap samples.

5.2 Sampling The Training Data

We train our model on the MNIST dataset. The total size of the dataset is 70,000, with 60,000 used for training and 10,000 used for validation. We generate a bootstrap sample of size 10,000 from the training dataset to train Network A and Network B.

The number of bootstrap samples required is the same as the number of weak learners we want to train. In this section, we analyze how the number of bootstrap samplings/weak learners affect the accuracy of the ensemble model.

we use "train until saturation" approach to determine the number of bootstrap samples required. We train 3 weak learners at a time and keep training until we achieve high accuracy.

5.2.1 Experimental Analysis

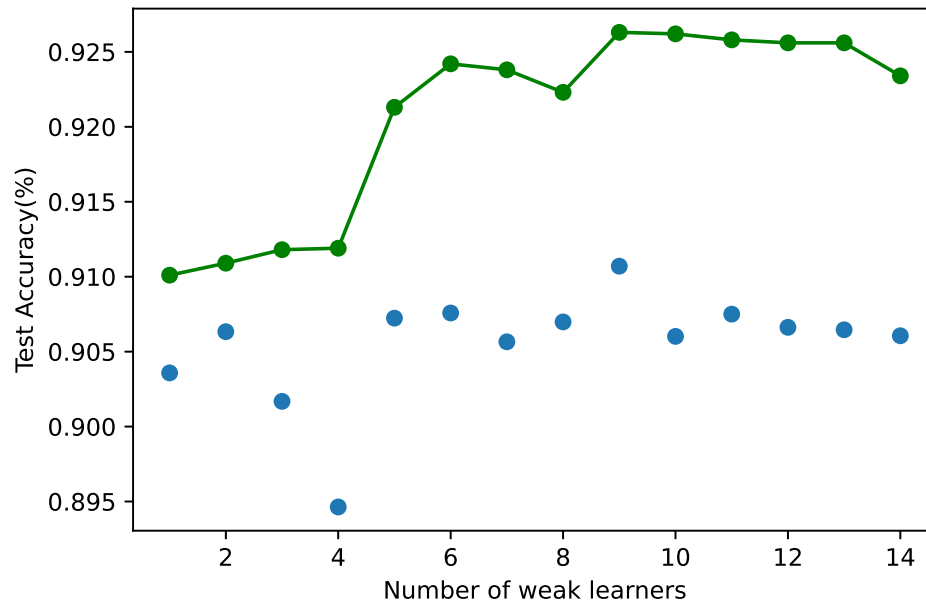


Figure 5.1: Number of weak learners vs Accuracy

Fig 5.1 shows the plot of the number of weak learners versus accuracy. We choose a bootstrap sample of size 10,000 to train Network A. Each weak model has a learning rate of 2^{-5} and is trained for only 1 epoch. It is well known that bagging reduces bias [FH07], so we do not worry about underfitting our weak learners.

Notice that the bagging model reaches its maximum accuracy for 7 weak learners, which is the expected number of samples required to sample the entire training data (ref: section 5.2.1).

5.3 Results

5.3.1 Experimental Evaluations

Training Time vs. Accuracy

Network	Number of weak learners	Percentage of Training data used	Training Accuracy		
			Weak Learner	Ensemble	Standard Model
Network A	7	72.12	63.7%	90.02%	99.7%
Network A	9	76.12%	70.05%	97.68%	99.8%

Table 5.1: Comparison of training accuracy

Network	Test Accuracy		
	Weak Learner	Ensemble	Standard Model
Network A	34.5%	91.58%	96%
Network B	70.05%	94.02%	98.42%

Table 5.2: Comparison of test accuracy

Training and Inference Time

Network	Training time (Hrs)		Inference time (s)	
	Ensemble model	Standard model	Ensemble model	Standard model
Network A	0.35	1.41	0.41	0.09
Network B	1.91	4.86	0.45	0.21

Table 5.3: Comparison of Training and Inference time

Table 5.3 shows the comparison of training and inference time between the standard model and the ensemble model. There is atleast 50% reduction in the training time of secure ensemble model. However, The inference time of the ensemble model is slightly higher as we run the inference on multiple model and take a majority vote.

5.4 Summary

We implement Secure Ensemble Learning in SecureNN framework that allows us to train models in parallel. Using this we are able to reduce the training time by atleast 50% without compromising much on the overall accuracy of the model.

Chapter 6

Conclusion

Driven by reducing the communication overhead of privacy-preserving machine learning, we develop efficient protocols that improve the performance of critical building blocks of such applications. To this end, we develop a secure neural network training and inference framework for Recurrent Neural Networks(RNN). For this, We adopt a modified and more efficient variant of techniques used in SecureNN [WGC18]. The methods used rely on arithmetic computations hence are more efficient when compared to prior works. The adoption rate of secure machine learning protocols is low. We address this by integrating secure training protocols with the popular ML library TensorFlow. For this, we modify the CrypTFlow framework to enable secure training.

The adoption rate of secure machine learning protocols for real-life applications is low due to the significantly high communication overheads. We propose Secure Ensemble Learning, a technique that reduces the communication overheads by training computationally less expensive ML models. Our implementation demonstrates that the Ensemble Learning model outperforms the Standard Model in MPC setting without significant accuracy loss.

Collectively, these techniques developed here provide a new foundation for the design of privacy-preserving algorithms. The bidirectional approach of improving secure computation protocols and modifying ML algorithms for better performance significantly reduces the overheads, thereby reducing the gap between secure computation and plaintext computation

6.1 Future Work

From here on, The vision of this thesis can be more broadly stated as

”Develop efficient techniques to adopt privacy-preserving machine learning for Real-life applications.”

I envision a privacy-conscious world that develops technologies and services in a manner that preserves user privacy. There is still a lot of work to be done on key research areas to accomplish this broad vision, a few of which I describe below:

Extending the ML base: Moving forward, I would like to extend the SecureNN framework for other NN classes like unsupervised learning, reinforcement learning, etc.

Adversarial Models: While the techniques developed in this work are for semi-honest adversarial models, they can be extended to malicious adversaries using techniques mentioned in [WTB⁺20]. However, it is an open line of research further to extend the model for dishonest majority adversarial model.

Appendix A

Chapter 3: Supplementary Material

A.1 Fixed-Point Arithmetic

The NN algorithms are generally encoded in floating point arithmetic. But MPC protocols work with fixed-point arithmetic. Below, we describe how the floating point numbers are encoded into integers. We use `uint64_t` datatype in C++ to map numbers to the integer ring $\mathbb{Z}_{2^{64}}$. we use a precision of 13 bits which means the first 13 digits from the left in the binary representation corresponds to decimals. Since we are using unsigned bit the 1 bit(MSB) from the left is used to denote sign of the number (1 if negative, 0 otherwise). By this an integer 2^{20} will correspond to float $8(2^{16} - 2^{13})$ and an integer 2^{64} will correspond to -1. Such encoding system is popular with MPC schemes.

Addition of two numbers is straightforward in fixed point. For multiplication, we multiply the decimals and truncate the last digits. Such truncation can also be performed with arithmetic secret shares[MZ17].

Appendix B

Chapter 4: Supplementary Material

B.1 Pseudo Random Functions(PRFs)

A pseudorandom function is an efficiently computable function that is indistinguishable from a random oracle (a function that generates completely random output).

Formal Definition: A function $f(K, x) : \{0, 1\}^s \leftarrow \{0, 1\}^s$ is a PRF if

- f can be computed in polynomial time
- if K is random then f can't be distinguished from a random function in polynomial time.

if two parties share the seed K then the output generated by the PRF is also shared between the parties. We make use of this property to reduce the communication rounds in Chapter 4.2.3.

Appendix C

Chapter 5: Supplementary Material

C.1 Number of weak learners vs Accuracy

Number of weak learners	Test accuracy		Percentage of data used
	Weak learner	Ensemble model	
1	90.97	90.88	16.6667
2	91.46	90.74	30.5283
3	91.08	92.07	42.0433
4	91.5	91.94	51.86
5	90.84	92.45	59.775
6	91.45	92.28	66.6117
7	91.44	92.39	72.12
8	90.77	92.17	76.8383
9	91.69	92.43	80.615
10	91.31	92.47	83.8
11	91.46	92.52	86.6983
12	91.66	92.28	88.7317
13	91.88	92.54	90.5383
14	91.46	92.59	92.1867
15	91.28	92.69	93.3433
16	90.66	92.48	94.5667
17	90.42	92.23	95.4867
18	91.04	92.49	96.2
19	91.48	92.44	96.77

Appendix D

Network Architecture

Layer	Description	Input Size	Output
Dense	fully-connected layer	28 x 28	128
ReLU/ TanH activation	Apply activation on each input	128	128
Dense	fully-connected layer	128	128
ReLU/ TanH activation	Apply activation on each input	128	128
Dense	fully-connected layer	128	10
ReLU/ TanH activation	Apply activation on each input	10	10

Table D.1: Network A architecture

Layer	Description	Input Size	Output
Conv2D	Window size 5x5 Stride(1,1), Padding(0,0), output channels 20	28 x 28 x 1	24x24x20
ReLU	Apply activation on each input	24x24x20	24x24x20
MaxPool	Window size 2x2, Stride(2,2)	24x24x20	12x12x20
Conv2D	Window size 5x5, Stride(1,1), Padding(0,0), Output Channels 50	12x12x20	8x8x50
ReLU	Apply activation on each input	8x8x50	8x8x50
MaxPool	Window suze 2x2, Srtride(2,2)	8x8x50	800
Dense	Fully Connected Layer	800	500
Relu	Apply activation on each input	500	500
Dense	Fully Connected Layer	500	10
ReLU/TanH	Apply activation on each input	10	10

Table D.2: Network D: LeNet architecture

Bibliography

- [Bea91] Donald Beaver, *Efficient multiparty protocols using circuit randomization*, Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings, Lecture Notes in Computer Science, vol. 576, Springer, 1991, pp. 420–432.
- [Bla79] G. R. Blakley, *Safeguarding cryptographic keys*, International Workshop on Managing Requirements Knowledge (MARK), 1979.
- [Bü12] Peter Bühlmann, *Bagging, boosting and ensemble methods*, Handbook of Computational Statistics (2012).
- [FH07] Jerome H. Friedman and Peter Hall, *On bagging and nonlinear estimation*, Journal of Statistical Planning and Inference **137** (2007), no. 3, 669–683, Special Issue on Nonparametric Statistics and Related Topics: In honor of M.L. Puri.
- [KRC⁺20] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharm, *Cryptflow : Secure tensorflow inference*, IEEE Symposium on Security and Privacy (2020), 336–353.
- [LBBH98] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE **86** (1998), no. 11, 2278–2324.
- [LC10] Yann LeCun and Corinna Cortes, *MNIST handwritten digit database*.

- [MZ17] Payman Mohassel and Yupeng Zhang, *Secureml: A system for scalable privacy-preserving machine learning*, 2017 IEEE Symposium on Security and Privacy (SP), 2017, pp. 19–38.
- [Ode19] Rising Odegua, *An empirical study of ensemble techniques (bagging, boosting and stacking)*, 03 2019.
- [Sha79] Adi Shamir, *How to share a secret*, Commun. ACM **22** (1979), no. 11, 612–613.
- [TFI] *Tensorflow graph transform tool*, https://github.com/tensorflow/tensorflow/tree/master/tensorflow/tools/graph_transforms.
- [VM02] Giorgio Valentini and Francesco Masulli, *Ensembles of learning machines*, vol. 2486, 05 2002, pp. 3–22.
- [VSA⁺10] Carlos Valle, Francisco Saravia, Héctor Allende, Raul Monge, and César Fernández, *Parallel approach for ensemble learning with locally coupled neural networks*, Neural Processing Letters **32** (2010), 277–291.
- [WGC18] Sameer Wagh, Divya Gupta, and Nishanth Chandran, *Securenn: Efficient and private neural network training*, Cryptology ePrint Archive, Report 2018/442, 2018, <https://eprint.iacr.org/2018/442>.
- [WTB⁺20] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin, *Falcon: Honest-majority maliciously secure framework for private deep learning*, 2020.
- [Yao82] A. C. Yao, *Protocols for secure computations*, 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982), 1982, pp. 160–164.
- [Yao86] ———, *How to generate and exchange secrets*, 27th Annual Symposium on Foundations of Computer Science (sfcs 1986), 1986, pp. 162–167.