7-7-2020

Parallel Implementation of Privacy Preserving Multi-Layer Neural Networks

Dipeshkumar Shaileshkumar Patel
University of Windsor

Parallel Implementation of Privacy Preserving Multi-Layer Neural Networks

By

Dipeshkumar Shaileshkumar Patel

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
at the University of Windsor

Windsor, Ontario, Canada

2020

Parallel Implementation of Privacy Preserving Multi-Layer Neural Networks

by

Dipeshkumar Shaileshkumar Patel

APPROVED BY:

B. Balasingam
Department of Electrical and Computer Engineering

D. Alhadidi
School of Computer Science

S. Samet, Advisor
School of Computer Science

May 7th, 2020

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

With recent technological advancements, the amount of personal user data that is being generated is immense. Due to the large volume of data, machine learning algorithms such as neural networks are serving as the backbone to derive patterns from this data quickly. This need for big data analytics comes at the cost of the privacy of user data. The second challenge that must be solved relates to the scalability of the machine learning algorithm. Neural networks are known to deteriorate as the volume of the data increases due to complex sum and sigmoid calculations. Therefore in this thesis, an attempt to parallelize the neural network while also maintaining the privacy of user data is made. This model would provide a viable option for big data analytics without sacrificing the privacy of individual users while also maintaining precision and the classification accuracy of the model. The implementation of the parallelized privacy preserving neural network will be based on the MapReduce computing model which provides advanced features such as fault tolerance, data replication, and load balancing.

DEDICATION

Dedicated to my late father ***Shaileshkumar Motibhai Patel****, my mother* ***Jayana Shaileshkumar Patel****, and my brother* ***Akshay*** *for their overwhelming enthusiasm, support, love, and for making everything possible.*

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to **Dr. Saeed Samet** for his continuous support and inputs through out my work. I could not have imagined a better mentor for my study. I would also like to thank **Dr. Dima Alhadidi** and **Dr. Balakumar Balasingam** for their inputs that helped me in refining and improving my work and helping me in guiding my work in proper direction and leading my thesis towards a gracious completion.

Also, I would like to thank my batch mates **Parth Shukla**, **Lokesh Gupta**, **Jayanth Kulkarni** and **Prit Patel**, for keeping me encouraged, without whom I could not have imagined my journey at the University. Special thanks to my best friends **Jay Patel**, **Neel Patel**, **Yash Patel**, and **Kushang Patel** for always being there.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Big Data and Privacy Requirements

As of 2013, 2.5 quintillion bytes of data are created daily. The volumes of data are vast; the generation speed of data is so fast that the data and information space has become global. The analysis of the databases that store this data can provide opportunities to solve major problems in our society like healthcare and others. A hefty portion of organizations decide not to incorporate the services of big data analytics due to the absence of standard security and privacy protection tools. So essentially, big data specifically refers to those data sets that are so large and complex that traditional data processing tools and techniques are not sufficiently applicable. The amount of data that is generated on the internet, social networking, Internet of Things (IoT) devices, and many other companies, is drastically increasing every day. Big Data is formally defined as "The information asset characterized by such a high volume, velocity, and variety to require specific technology and analytical methods for its transformation into value." [15]. Based on this definition, the three properties most suitable for the term are the 3V's, also known as, volume, velocity, and variety. Although these are the three main properties, others complement the features of big data such as veracity, validity, variability, and vagueness. When the term volume is related to big data, it refers to the large quantity of data at hand. Velocity is the measure of how fast the data is coming in. The flow of the information is known as the velocity vector. The diversity of big data, i.e., they may contain text, audio, image, video, etc. is signified by variety.

Privacy and security in terms of big data is an important issue. For complex applications the big data security model is not recommended as it will get disabled by default. However, without it, data can be breached easily. To understand the difference between privacy and security, we must understand the context of each term. Privacy refers to the privilege to have some control over how personal information is collected and used. Which provides users the capacity to stop information about themselves from becoming known to people other than those they give information to. One serious issue is the de-identification of personal information during transmission over the Internet. When we think about security we think about the practice of defending information and information assets using technology and processes from a wide array of security aspects such as: unauthorized access, modification, recording, destruction, disruption, and disclosure. So when comparing both security and privacy we understand that security is fundamental for protecting data, its not sufficient for addressing privacy. To ensure big data privacy, various tools and techniques have been developed in the past few years. The tools and techniques have been prepared per the big data life cycle, i.e., data generation, data storage, and data processing. In the initial phase of data generation, access restriction and data falsification techniques are used. The middle stage of data storage relies on encryption techniques. These encryption techniques rely on Identity-Based Encryption (IBE), Attribute-Based Encryption (ABE), and storage path encryption. In addition, to protect sensitive information, hybrid clouds are utilized to store the sensitive data on the cloud. The data processing phase relies on privacy preserving data publishing (PPDP) [15]. Privacy preserving data publishing consists of anonymization techniques such as generalization and suppression. These techniques can be further coupled to solve classification, clustering, and association rule mining problems. Classification and clustering techniques tend to split the input into different groups, while association rule mining techniques find relationships and trends within the input data. To handle the various measurements of big data in terms of variety, velocity, and volume, there is a need for efficient and effective frameworks to process this type of data that tends to come in at extremely high speeds for a plethora of sources. The scope of our implementation will rely on

big data privacy preservation in the phase of data processing. This type of model categorizes the system into a stream, batch, graph, and machine learning processing. Two stages are needed for privacy protection in the data processing phase. The initial stage will safeguard the data from disclosure due to the possibility of it containing sensitive information. The second stage will consist of extracting meaningful information from the data without breaking privacy.

1.2 Motivation Behind The Work

Deep learning based off of artificial neural networks has become the backbone for modeling, classifying and recognizing complex data such as images, speech, and text. The unparalleled accuracy of these methods has made them the foundation for AI-based products and services on the internet. Commercial companies that collect user data on a large scale have become the main beneficiaries of this upcoming technology. This is largely because the success of deep learning methods is directly proportional to the amount of data available for training. Companies such as Google, Facebook, and Apple take advantage of massive amounts of training data collected from their users and the vast computational power of GPU farms to deploy deep learning on a large scale. Massive data collection required for deep learning presents obvious privacy issues. Highly personal user data is kept by large organizations indefinitely. The users can neither delete it, nor restrict the way the organization chooses to use it. Images and voice recordings often contain accidentally captured items such as faces, computer screens, license plates, and the sound of other people talking. This data is kept and subject to subpoenas and warrants, as well as warrantless spying by national security and intelligence outfits. Many data owners, such as hospitals, are unable to share data due to privacy and confidentiality concerns and cannot benefit from large scale deep learning. Another issue that is not addressed in the literature is that there are a lack of suitable privacy-preserving machine learning for users that want to utilize them at a larger scale. Data privacy is also required by law in some cases, e.g., for medical or financial data. Additionally, the desire of customers and

clients for discretion and privacy is growing and has become a topic of public focus.

1.3 Problem Statement & Outlined Solution

The aim of this research is to attempt to parallelize the neural network while also maintaining the privacy of the users data. This approach would provide a viable option for big data analytics without sacrificing the privacy of individual users while also maintaining the precision and the classification accuracy of the model. The deployment of the parallelized privacy preserving neural network will be based on the MapReduce computing model which provides advanced features such as fault tolerance, data replication, and load balancing.

We tackle the problem with points stated below which we discuss comprehensively in subsequent chapters of thesis.

- Development of the neural networks will be done using TensorFlow-privacy. The networks will be optimized using a differentially private stochastic gradient algorithm which will provide strong privacy guarantees.
- We present three novel neural network architectures to tackle three different data intensive scenarios.
- The three scenarios tackle situations varying from: large volume of testing data being present, large volume of training data being present, and a vast number of neurons being present in the neural network.

1.4 Structure of Thesis

The subsequent chapters of the thesis are organized as follows. In **Chapter 2**, we present the readers with succinct knowledge which lays a technical foundation for the rest of the thesis. As for **Chapter 3**, we present the Literature Survey of related work done in the area. **Chapter 4** explains the implemented Methodology to solve the problem. **Chapter 5** includes the experiment results and analysis of the proposed

solution and comparison with other methods. And lastly, In **Chapter 6**, we conclude the thesis and address the work that can be possibly done in the future to refine the proposed solution.

CHAPTER 2

Overview of Related Technologies

2.1 Privacy Preserving Methods in Big Data

The methods described in this section have been used to provide privacy to a certain amount but their demerits have also led to the creation of more advance methods and applications.

2.1.1 Anonymization

Anonymization is one of the most traditional techniques for privacy preserving data mining, where generalization and suppression techniques are applied in order to protect an individuals privacy. In generalization, quasi-identifiers are replaced with less particular but semantically consistent values. Whereas, in suppression techniques some information is not released at all. However, re-identification is a weakness of anonymization. To alleviate this threat, the concepts of k-anonymity, l-diversity, and t-closeness were proposed to enhance traditional techniques for privacy preserving data mining. When anonymization is transferred to privacy preserving big data analytics, the risk of re-identification increases so we must ensure that the privacy preserving algorithms used are efficient. In the next few paragraphs we will explore the three main anonymization algorithms: k-anonymity, l-diversity, and t-closeness.

2.1.1.1 K-Anonymity

K-anonymity is a property of a data set, usually used in order to describe the dataset's level of anonymity. A dataset is k-anonymous if every combination of identity-

revealing characterisitics occurs in at least k different rows of the dataset[29]. K-anonymity is prone to two attacks namely homogeneity attack and the background knowledge attack. K-anonymity can be applied on patient data as seen in table 2.1.1.

S.No	Zip	Age	Disease
1	57677	29	Cardiac Problem
2	57602	22	Cardiac Problem
3	57678	27	Cardiac Problem
4	57905	43	Skin Allergy
5	57909	52	Cardiac Problem
6	57906	47	Cancer
7	57605	30	Cardiac Problem
8	57673	36	Cancer
9	57607	32	Cancer

Table 2.1.1: Data set before applying K anonymity [3]

K-anonymity algorithm is applied with k value as 3 to ensure 3 indistinguishable records when an attempt is made to identify a particular person's data. K-anonymity is applied on two attributes, Zip and age as seen in Table 2.1.1. The result after applying K anonymity can be seen in Table 2.1.2. The above technique has utilized generalization to achieve anonymization. If we were to figure out that John's age is 27 and he lives in 57677 zip codes then we can conclude that John has a cardiac problem even after the anonymization as shown in Table 2.1.2. This is called Homogeneity attack. For example, if John is 36 and it is known that John does not have cancer, then John definitely must have a Cardiac Problem. K-anonymity can be achieved by using either generalization or suppression [29]. K-anonymity can be optimized to prevent data loss by minimizing generalization. Identity disclosure is the major privacy threat which cannot be guaranteed by K-anonymity.

S.No	Zip	Age	Disease
1	576**	2*	Cardiac Problem
2	576**	2*	Cardiac Problem
3	576**	2*	Cardiac Problem
4	5790*	>40	Skin Allergy
5	5790*	>40	Cardiac Problem
6	5790*	>40	Cancer
7	576**	3*	Cardiac Problem
8	576**	3*	Cancer
9	576**	3*	Cancer

Table 2.1.2: Data set after applying K anonymity [3]

2.1.1.2 L-Diversity

To address the homogeneity attack that can break K-anonymity, L- diversity was introduced. As per L-diversity there must be L well represented values for the sensitive attribute (disease) in each equivalence class [23]. If the variety of the data is large, it becomes difficult to implement L-diversity. L-diversity has its own weakness in the form of the skewness attack [23]. When the overall data distribution is skewed into a few equivalence classes attribute disclosure cannot be ensured. L-diversity is also prone to the similarity attack[23]. From table 2.1.3 it can be seen that if it is known that John is 27 years old and lives in the 57677 zip, then John is definitely in the low income group because the salaries of all the people in the 576** zip is low compared to others in the table. This is known as the similarity attack.

Sno	Zip	Age	Salary	Disease
1	576**	2*	5k	Cardiac Problem
2	576**	2*	6k	Cardiac Problem
3	576**	2*	7k	Cardiac Problem
4	5790*	>40	20k	Skin Allergy
5	5790*	>40	22k	Cardiac Problem
6	5790*	>40	24k	Cancer

Table 2.1.3: Data set after applying L diversity [3]

2.1.1.3 T-Closeness

Another improvement to L diversity is the T-closeness measure. T- closeness is a measure where an equivalence class is considered to have 'T-closeness' if the distance between the distributions of sensitive attribute in the class is no more than a threshold and all equivalence classes have T- closeness [19]. T-closeness can be calculated on every attribute with respect to sensitive attribute. From table 2.1.4 it can be observed that if we know John is 27 years old, it is still difficult to estimate whether John has a cardiac problem or not and if he is under the low income group or not. T-closeness may ensure attribute disclosure but implementing it may not give a proper distribution of data all the time. Earth Movers Distance metric is used to measure the distance or closeness between two attributes[19]. The idea of EMD is to imagine both probability distributions as piles of dirt and calculate the minimum amount of work needed to reshape the first pile so that it has the same shape as the second [19]. The key attribute of EMD is that it takes distance into account.

S.No	Zip	Age	Salary	Disease
1	576**	2*	5k	Cardiac Problem
2	576**	2*	16k	Cancer
3	576**	2*	9k	Skin Allergy
4	5790*	>40	20k	Skin Allergy
5	5790*	>40	42k	Cardiac Problem
6	5790*	>40	8k	Flu

Table 2.1.4: Data set after applying T closeness [3]

2.1.2 Secure Multi-Party Computation

Secure Multi-party Computation (SMPC) is an important subset of cryptography. It has the potential to enable real data privacy. SMPC refers to the cryptographic protocols allowing the distributed computation of a function over inputs that are distributed without disclosing any information about the inputs [10]. Yao [31] introduced this idea in 1982 with the help of the Millionaire's problem, which allowed two parties to determine whose values was larger without disclosing anything about the individuals values. To get a better understanding, lets use an example: a secret number that is larger than 1000, a hundred users within a system, and a quorum of 10 users. Each user is given a unique number between 100 and 199. Since the minimum quorum requirement is 10 users, anytime a group of 10 or more people come together, their combined numbers, x, will reveal the secret ($x > 1000$) without revealing any persons individual number. Protocols based off of secure multi-party computations tend to be compared to ideal protocols that utilize a trusted third party. A trusted third party (TTP) is essentially an additional party that all participants fully trust. These comparisons to a secure multi-party computation protocol become trivial, because each party can then just send their input to the third party who then sends the results to each party. The goal of SMPC is to perform these computations and achieve the security of a TTP protocol without the TTP.

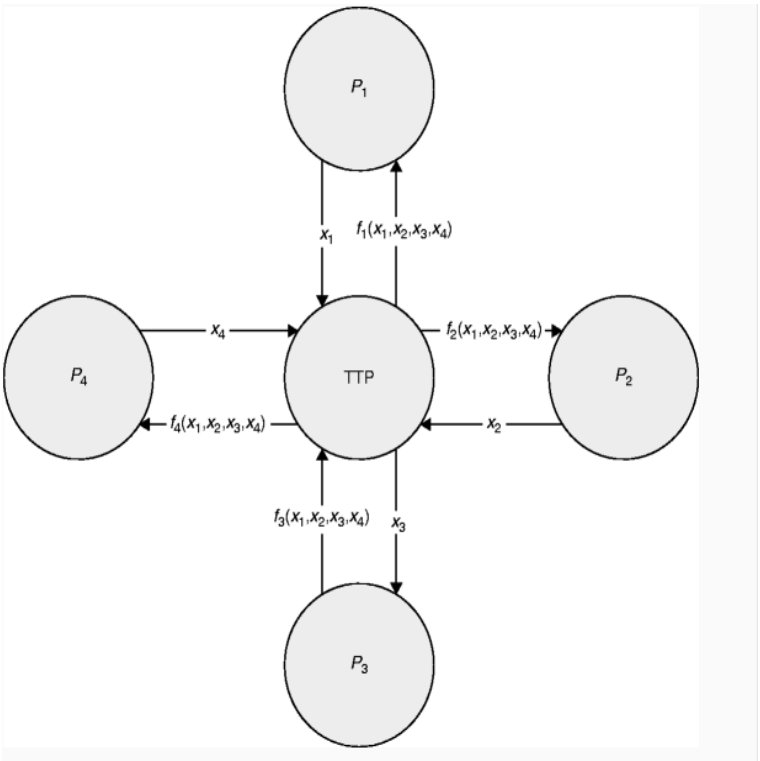


Fig. 2.1.1: Ideal Protocol with 4 Parties [10]

From figure 2.1.1 we observe that there are n parties, denoted as $P_1, .., P_n$ having inputs from $x_1, ..., x_n$. There are also a set of publicly known functions $f_1, ..., f_n$ each associated with the n inputs. The goal of SMC is that at the end of the protocol, each participant receives the output for the corresponding function. The security goal is that each participant should only be aware of their input and the output received from the function and nothing else. Also, if a set of participants collude with each other that nothing should be revealed other than what can be inferred.

2.1.3 Differential Privacy

Differential privacy is essentially a very rigorous mathematical definition of privacy. An algorithm is differentially private if when an algorithm performs some statistical analysis on a database, after looking at the output of that analysis, one cannot tell if any user's data was included in the original dataset or not. Simply put, differential privacy guarantees that the behaviour of an algorithm will hardly changes when

an individual joins or leaves the dataset [8]. Anything the algorithm might output on a database containing some individual's information is almost as likely to have come from a database without that individual's information [8]. This guarantee holds for any individual and any dataset. There are many other privacy measures that can be used to secure user information. Anonymization is the most commonly used technique. We have discussed the shortcomings of anonymization techniques. They tend to be susceptible to threats such as the homogeneity attack, background knowledge attack, linkage attack, differencing attack, etc. For example, Sweeney [29] showed that quasi-identifiers such as gender, date of birth, and zip code can be used to re-identify the majority of American people. Differential privacy provides mathematical guarantees against these types of attacks. Differential privacy does not define privacy under a binary notion of data being private or not. Differential privacy assess privacy as an accumulative risk. Every time a users data is processed it is at more risk to be exposed. The two parameters used to quantify privacy when working with differential privacy are (ϵ, δ). Where ϵ denotes a metric of privacy loss at a differential change in data, and δ bounds the probability of an arbitrary change in the model behaviour.

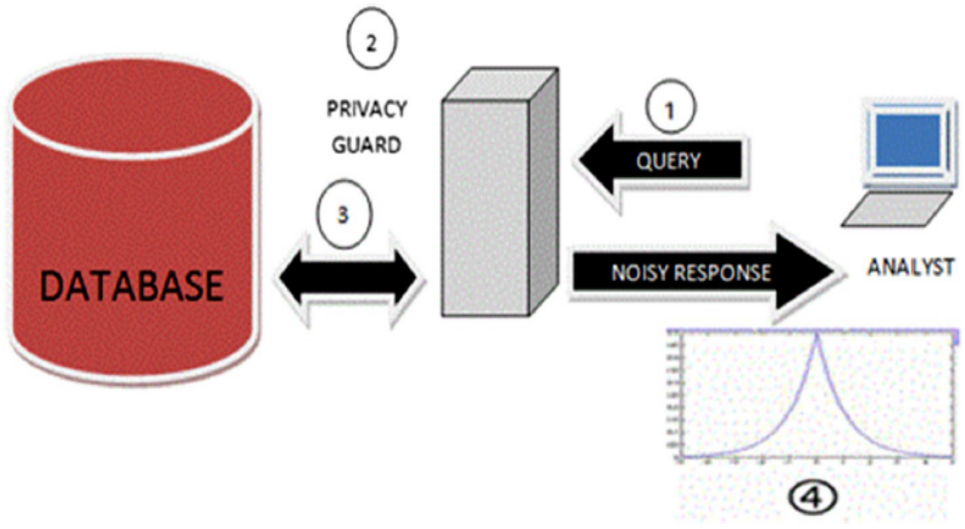


Fig. 2.1.2: Working of Differential Privacy [15]

In differential privacy, analysts are not allowed to interact with databases containing sensitive user information directly. A piece of software known as the privacy

guard is introduced between the analyst and the database to help protect privacy. Figure 2.1.2 can be further explained in 4 steps:

- *Step 1* The analyst can query the database through the privacy guard
- *Step 2* The privacy guard evaluates the query and other previous queries to assess the privacy risk.
- *Step 3* The privacy guard will then fetch the answer from the database
- *Step 4* Some noise will be added to the answer according to the privacy risk and finally forwarded to the analyst

To finally summarize differential privacy in the form of a definition presented in [8], a randomized function $\mathcal{K}$ gives ϵ - differential privacy if for all datasets D_1 and D_2 differing on at most one element, and for all S $\subseteq$ $Range(\mathcal{K})$,

$$Pr[\mathcal{K}(D_1) \in S] \geq exp(\in) \times Pr[\mathcal{K}(D_2) \in S]$$

2.2 The Perceptron

The perceptron is the simplest form of a neural network in the form of a linear classifier used for binary predictions. In order for a perceptron to work, the data must be linearly separably. It will not converge if the data is nonlinear [26]. A perceptron takes several binary inputs $x_1, x_2, ...$, and produces a single binary output.

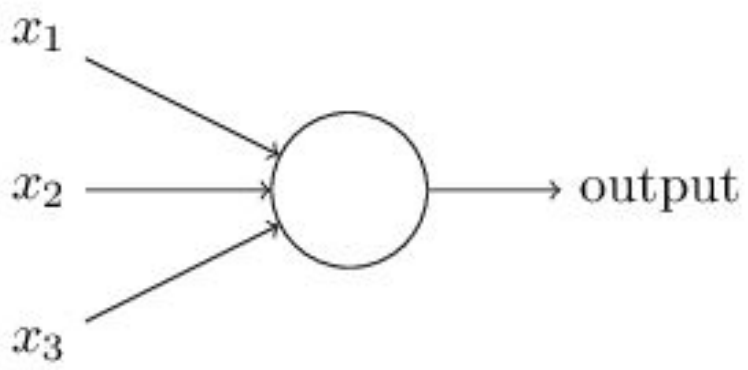


Fig. 2.2.1: The Perceptron

From figure 2.2.1, we can see that the perceptron takes in three inputs, x_1, x_2, x_3. It can have more or fewer inputs. To determine the output, the perceptron model is dependent on the summation of weights. Weights $w_1, w_2, ...,$ are real numbers that express the importance of the respective inputs to the output. Whether the output of the neuron is, 0 or 1, is determined if the weighted sum, $\Sigma_j w_j x_j$, is less than or greater than some threshold value θ [26]. In algebraic terms, the notation can be seen in equation 1.

$$output = \begin{cases} 0, & \text{if } \Sigma_j w_j x_j \leq \theta. \\ 1, & \text{if } \Sigma_j w_j x_j > \theta. \end{cases} \tag{1}$$

Equation 1 represents the basic mathematical model for the perceptron algorithm. Although it is easy to understand we can still simplify the equation by making two notation changes. The initial change will be to write $\Sigma_j w_j x_j$ as a dot product, $w \cdot x \equiv \Sigma_j w_j x_j$, where w and x represent a vector that consists of the weights and the inputs[26]. The second change is to move the threshold to the other side of the inequality and to then replace it by what's known as the perceptron's bias, $b \equiv -\theta$ [26]. Using the bias instead of the threshold, the rule can be rewritten:

$$output = \begin{cases} 0, & \text{if } w \cdot x + b \leq \theta. \\ 1, & \text{if } w \cdot x + b > \theta. \end{cases} \tag{2}$$

In effect, the bias is a measure that will measure how easy it is for the perceptron to output a 1. Or to put in other terms, it is a measure that measure how easy it is to get the perceptron to "fire". If a perceptron has a large bias value, the perceptron can easily output a 1, but if the bias is a large negative value, it becomes the difficult for the perceptron to fire a 1 [26]. Once this value is computed, it is then passed through an activation function. This function is non-linear and is called the activation function. The purpose of the activation function is to introduce non-linearity into the output of the neuron. This is an important aspect since most real world data is non-linear and we want neurons to learn these non-linear representations. Every activation function will take a single input and perform a certain fixed mathematical operation on it.

2.3 Multi-Layer Neural Networks

A Multi-Layer Neural Network contains one or more hidden layers apart from one input and output layer. While a single layer perceptron can learn only linear functions, multi-layer neural networks can also learn non-linear functions.

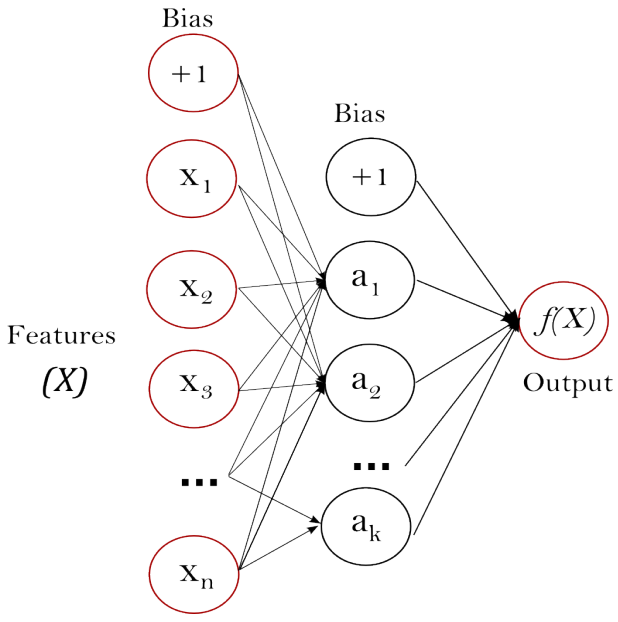


Fig. 2.3.1: Multi-layer Neural Network Architecture

figure 2.3.1 shows a multi-layer neural network. Here, the units are arranged into a set of layers, and each layer contains some number of identical units. Every unit in one layer is connected to every unit in the next layer. The term used to denote this is "fully connected". The first layer is the input layer, and its units take the values of the input features. The last layer is the output layer, and it has one unit for each value the network outputs. A single unit is denoted in figure 2.3.1, as this particular architecture would be used for a regression or binary classification task. All the layers between these are known as hidden layers. The units within each layer is known as input units, output units, and hidden units, respectively. The number of layers is expressed as the depth, and the number of units in each layer is defined as the width. The basic neural network model can be described as a series of functional transformations [4]. First we construct M linear combinations [4] of the input variables $x_1, ..., x_D$ in the form

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{3}$$

Where $j = 1, ..., M$, and the superscript (1) indicates that the respective parameters are in the first layer of the neural network[4]. The parameter w_{ji} denotes the weights and w_{j0} denotes the bias. The quantities a_j are known as the activations. Each of them will be transformed by utilizing a differentiable, non-linear activation function to give

$$z_j = h(a_j). \tag{4}$$

The nonlinear function $h(\cdot)$ are usually chosen to be the logistic sigmoid function or the 'tanh' function [4]. These values are once again linearly combined to give

$$a_k = \sum_{j=1}^{M} w_{kj}^{(1)} z_j + w_{k0}^{(2)} \tag{5}$$

where $k = 1, ..., K$, and K is the total number of outputs. These transformations are

related to the second layer of the neural network and $w_{k0}^{(2)}$ still corresponds to the bias values. After this the output units are finally transformed using an appropriate activation function to give a set of outputs y_k. Each output unit is transformed using a logistic sigmoid function or any other activation function that is suitable according to the distribution of the data [4]. We denote the logistic sigmoid function below in equation 6.

$$y_k = \sigma(a_k) \tag{6}$$

where,

$$\sigma(a) = \frac{1}{1 + exp(-a)} \tag{7}$$

In essence, we can combine all the different stages of the entire network that, the sigmoidal function would take the form of

$$y_k(x, w) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right) \tag{8}$$

In equation 8, the weights and bias variables have been combined into a vector w. So as you can see, the neural network is simply a nonlinear function from a set of input variables to a set of output variables controlled by a vector of modifiable parameters [4].

2.4 MapReduce

MapReduce is a programming framework that allows us to perform distributed and parallel processing on large data sets in a distributed environment [6]. The framework consists of two distinct tasks - Map and Reduce. In the Map job, the block of data is read and processed to produce key-value pairs as intermediate outputs. These key-value pairs are then used as inputs into the Reducer. The Reducer then aggregates these intermediate data tuples into a smaller set of key-value pairs or tuples to produce

the final output. The overall MapReduce process can be seen in figure 2.4.1.

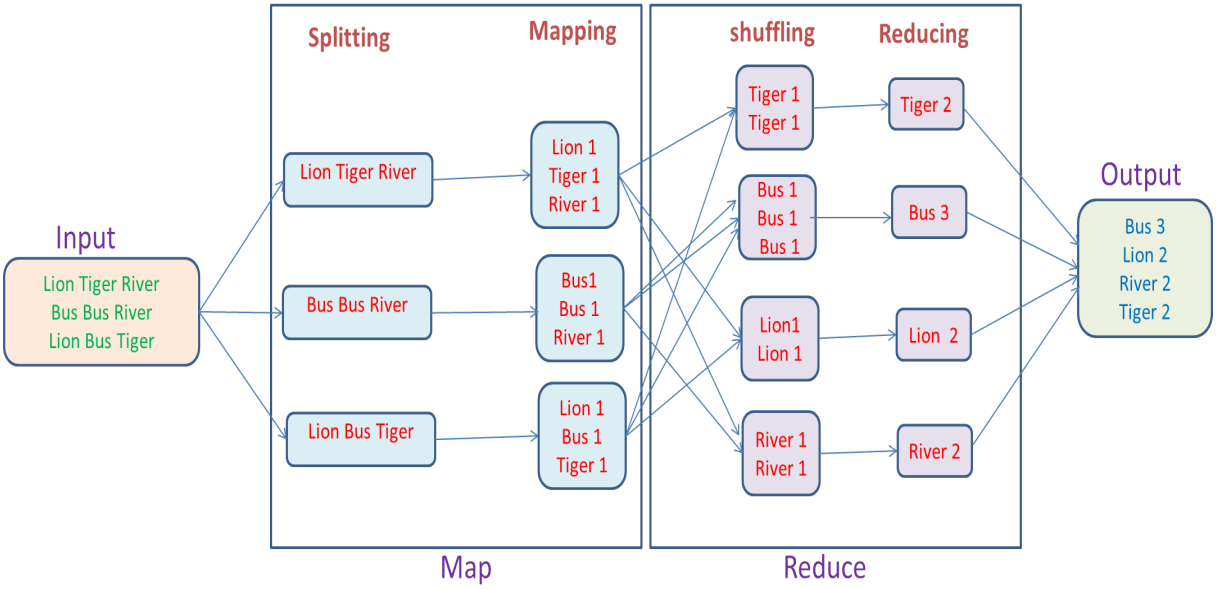


Fig. 2.4.1: MapReduce

To get a more in depth understanding, when we split the file into chunks, each one of them forms the input of a map task. Each map task is assigned to a computer node called worker, which applies the map function to each key-value pair of the corresponding input chunk[6]. The intermediate key-value pairs produced are stored in the local disks of the workers. In order to speed up the algorithm and balancing the load of workers, each intermediate key-value pair is stored in the local disk into buckets through a hash function h [6]. Each bucket is assigned to a different worker, which applies a reduce function specified by the user [6]. There are two important functions that are included in these three phases, the shuffle and sorting function. The shuffle function is the process of transferring data from the mappers to the reducers. In order to speed up the algorithm this function can start even before the map phase has finished. Shuffle is usually the most cost expensive round. The sorting function is the process of sorting the series of values with the same key. If zero reducers are specified, the MapReduce job will stop at the end of the map phase.

CHAPTER 3

Related Works

3.1 Privacy Preserving Neural Networks

The introduction of novel machine learning techniques has brought on a technological revolution. The ability to train a machine to make decisions as a human would brings a new perspective to what can be achieved. Machine learning benefits fields ranging from social engineering, image recognition, healthcare services, financial services, etc. The quality of the output of a machine learning algorithm is directly proportional to the size and quality of the dataset that its trained on. Since data collection is a scattered process, a lot of effort is required to collect them. Users tend to reluctantly submit their data to third-party collectors. To convince users that their data is secure and private, an approach to train the machine learning algorithms in a privacy preserving way must be utilized. For this fundamental techniques such as encryption, differential privacy, and other miscellaneous approaches can be used. When these techniques are utilized on the data and then used to train a neural network we can say that the neural network has satisfied a privacy preserving constraint. The scientific term that is used to identify neural networks trained on encrypted data is *CryptoNets*. Other ways to create privacy preserving neural networks include differential privacy, genetic algorithms, third party servers, etc. which we will discuss further in this chapter.

3.1.1 CryptoNets

ML Confidential [11] which was developed by Graepel et. al., is a convolutional neural network that works on a homomorphic encryption (HE) scheme. Instead of the nonlinear activation function they use a polynomial based approximation method. They assume a cloud service based scenario, and ensure privacy during the transfer period of the data from client and server. A public and private key is generated for each client in the key generation phase [11]. Client data will be encrypted using homomorphic encryption during the data transfer period to the server[11]. The training of the model will be done using the encrypted data on the cloud and then use the training model to classify the test dataset [11].

Another approach to creating cryptonets would be through using a low latency framework like Gazelle, proposed by [17], they combine HE with garbled circuits to create a privacy preserving Prediction as a Service (PaaS) environment. The authors utilize Single Instruction Multiple Data (SIMD) to improve the encryption speed of the homomorphic encryption process [17]. The goal of Gazelle is to allow the client to do the classification process without revealing the input to the server and also preserve the privacy of the model in the server[17]. To preserve the privacy of the convolutional neural network the authors hide the weight, stride size, and bias in the convolutional layers[17]. They go on to show that Gazelle completely outperforms the cryptonets [7] proposed by Gilad-Bachrach et al., and MiniONN [20] in terms of runtime.

The cryptonets [7] proposed by Gilad-Bachrach et al. infuses homomorphic encryption in to the convolutional neural network. They show that cloud services can apply encrypted predictions on encrypted training data, and then return the encrypted prediction to the client[7]. The client can then decrypt the prediction using their own private key[7]. This implementation would be very useful to hospitals in particular. The major disadvantage that this method presents is that with increase in the number of non-linear layers the performance is limited. The error rate tends to increase and the accuracy drops with deeper neural networks.

We will now further evaluate on MiniONN proposed by [20]. MiniONN is a framework that converts neural networks into oblivious neural networks. The conversion results in accuracy loss due to the transformations of nonlinear functions. Two oblivious transformations are provided for the piecewise linear activation function and for the smooth activation function [20]. A smooth functions can be changed into a continuous polynomial by parting the function into a few sections. At that point, for each part, polynomial estimation is utilized for the estimation, bringing about a piecewise linear function. MiniONN supports any activation function that has a monotonic range, piecewise polynomial, or can be approximated into polynomial functions[20].

SecureML, proposed by [24] is a method that leverages secure multi-party computation for privacy preserving deep learning. It utilizes Oblivious Transfer (OT), Yao's Garbled Circuits, and Secret Sharing [24]. They utilize linear and logistic regression in a deep learning environment. They propose an addition and multiplication algorithm for values that are secretly shared in linear regression. Stochastic Gradient Descent (SGD) is used to calculate the optimum value of regression [24]. The main weakness of this scheme is that its optimal for simple neural networks only. This results in a very low accuracy.

3.1.2 Miscellaneous Approaches

The authors in [2] propose three machine learning models that consist of the combination of Kernel Learning mode and Deep Neural Networks to lead to a novel Multi-Kernel Learning and Hybrid Learning model to help better preserve privacy in big data. The authors try to address the issue of privacy preserving machine learning via utility maximizing lossy compression of data. The authors proposed models try to give users access to only information necessary for the intended use, but nothing else [2]. For this the authors focus on kernel learning which is efficient in learning low-dimensional data and privacy – preserving utility cases, and the second candidate the authors focus on is deep learning. Deep learning will allow the effective extraction of utility information from numerous feature representations else [2]. The authors state that to the best of their knowledge they are first to propose this model

of Multi-Kernel Learning and Hybrid Learning. Neural Network Architecture Before kernel selection for the neural network is done, kernel-based compression is done as it has shown to be effective in utility- maximizing lossy compression. Kernel-based compression is done to obtain the dimensional projections from each kernel induced vector space. There are two reasons for this, if the class is normally distributed after mapping with the covariance then the projections are known to be optimal. Second, these observations will capture the maximum mutual information between the observations and the utility subspace spanned by the class centers [2]. Because different kernels provide different classification performances for a given task the authors perform a filtering process based on discriminant information metric. The discriminant information metric measures the mutual information between the mapping and the label. If a mapping has a low DI score it is removed. This will reduce the amount of information and regularize the learning space allowing the DNN to learn more effectively [2]. A direct effect would be seen by an increase in classification accuracy. Due to the reduction in information privacy preservation would also improve [2]. DI metric is directly proportional to predictive accuracy. A fully-connected feed-forward neural network with Rectified Linear Units (ReLU). Non-linear feature mappings will occur up until the narrow layer, which will then be project on to the subspace. The mappings are learnt by the neural network together with the projection [2]. In the hybrid model, the input layer of the DNN will consist of multi-kernel features. The rest will be a regular feed forward neural network. This separation between the features and the DNN gives us the separation between public and private spheres [2]. Now for the compressive hybrid model, the multi-kernel features will again be the input to the DNN. The hybrid model will include a narrow funneling layer which will ensure the separation between public and private spheres. The hybrid model provides the best privacy preservation among all the competing models [2].

Shokri et. al design, implement, and evaluate a system that allows numerous parties to learn simultaneously using an accurate neural network model without the sharing of their respective input datasets [27]. The authors parallelize and execute the stochastic gradient descent optimization algorithm asynchronously, while also allow-

ing each party to train their input dataset independently. This minimizes the sharing of the models' key parameters. The users' preserve the privacy of their respective data and still benefit from the models of other users. This will boost learning accuracy beyond what they would achieve if a single dataset was used [27]. The authors state that they are the first to implement this to the best of their knowledge. No previous work has addressed the problem of collaborative deep learning with multiple participants using distributed stochastic gradient descent [27]. The aim of deep learning is to extract complex features from high dimensional data and utilize those features to build a model to produce an output. Neural networks that are used in deep learning are usually multi-layered networks so more abstract features can be computed as nonlinear functions of lower level features. Usually, in a multi-layer neural network, each neuron will receive the output of the neurons from the previous layer along with a bias signal. The core protocol of the author's approach is a Selective Stochastic Gradient Descent protocol. During this protocol certain parameters contribute more to a neural networks objective function and so these parameters undergo much bigger updates during a particular iteration of training. In Selective SGD the user selects approximately half of the parameters to be updated at each iteration. These can be selected at random or by selecting those parameters whose values are farthest from the local optima (larger gradient) [27]. The distributed selective SGD assumes that two or more participants are training independently and at the same time. Participants will asynchronously share the gradients they individually computed with each other. Each participant has full control on what they can share [27]. The sharing of the gradients can be done directly, through a trusted central server, or through secure multi-party computations [27]. The overall architecture of the proposed system can be seen above. It is assumed that there are N participants and each has their own local private dataset for training. The existence of a parameter server is assumed in this proposed system [27]. Initialization of the parameters is done by each individual and these parameters can be uploaded to the parameter server using a parameter exchange protocol [27]. The parameters can be accessed by each participant through this server. Now each participant can begin training using the stochastic gradient

descent algorithm. The distributed selective stochastic gradient descent algorithm (DSSGD) is independently run by the participants during each iteration [27]. The steps of the DSSGD are shown below: Choose initial parameters and learning rate.

- Step 1: Download the parameters from the server and replace the corresponding local parameters.
- Step 2: Run Stochastic Gradient Descent on the local dataset and the local parameters should be updates according to step 1.
- Step 3: The gradient vector is computed, which is the vector of changes in all local parameters due to SGD.
- Step 4: Upload the gradient vector to the parameter server.

The authors in [12] approach the problem of privacy-preservation in neural networks from a different perspective.They couple incremental learning and genetic algorithms to achieve privacy preservation in neural networks. They go on to present that their method allows to obtain an accurate model based on information in distributed databases without any information sharing during the training process without degrading the classification accuracy. This approach allows to construct a global classifier based on the horizontal partitioning of the data. The main characteristic of the work done by [12] is that the distributed learning is done without exchanging any pattern between the different operations hence preserving privacy. Local models are built in accordance to the number of data partitions. Once each local classifiers are trained they send the results to a central entity. At this central entity, a genetic algorithm is used to build a more accurate final model taking advantage of the incremental learning capacity [12]. This global model is then sent to each entity for classification of new data. The authors evaluated their proposed approach using six classification problems: shuttle, letter, adult, nursery, waveform and mushroom [12]. The proposed model in [12] was compared with the Naive-Bayes, Tree-Augmented Naive-Bayes (TAN), C4.5, and with the artificial neural network. These were used a local classifiers on each node.

3.1.3 Trusted Hardware Components and Trusted Execution Environments

Software Guard Extensions, that are featured in Intel processors starting with Skylake, provide safe areas (enclaves) that protect code and data from every other software on the platform. This includes privileged software such as the operating system and hypervisor. Any code running in an enclave can operate on sensitive data without the fear of accidental exposure of the data on the platform. The privacy and integrity of the enclave is supported by the hardware. We can observe Chiron, proposed by the authors in[14] which is a system for privacy preserving machine learning as a service. Chiron conceals the training data from the service operator and also hides the training algorithm and model structure[14]. Although its immplemented on Intel SGX enclaves, it relies on other principles to achieve dual data privacy and model confidentiality[14]. Chiron will run the standard machine learning process in the enclave. A Ryoan sandbox is utilized to prevent the model from leaking training data to the service operator[14]. With the help of a parameter server, Chiron is able to achieve distributed training as model parameters can be shared within the enclave.

3.2 Approaches to Scale Neural Networks

In this section we will see the current work implemented in parallelization of neural networks through MapReduce. When we think about the traditional machine learning algorithms, we usually separate them according to their differences. The authors in [5] show that most of the traditional machine learning algorithms fit a Statistical Query Model (SQM) and they can be conveyed in a certain "summation form" which enables them to be easily parallelized on multicore componenents. A variety of learning algorithms that are covered in [5] include locally weighted linear regression, k-means, logisitic regression, naive Bayes, SVM, ICA, PCA, guassian dicriminant analysis, EM, and backpropagated Neural Networks. The neural networks will be our main focus as that is the core area we plan to expand our work on. For multicore systems,

concurrent applications benefit the most due to little communication between the cores [5]. According to [5] the Statistical Query Model permits a learning algorithm to access the learning problem only through a statistical query oracle. Suppose we have a function $f(x, y)$ over instances, the statistical query oracle returns an estimate of the expectation of $f(x, y)$, essentially and average over the training/test distribution. So any algorithm that calculation statistics or gradients can fit this model sinces these computations can be batched and expressed as a sum over data points. When the algorithm does calculations, these calculations can be divided over the multiple cores. A division of the data set into as many pieces as there are cores is done. Each core is given its share of the data to sum the equations over, and the results of the all the cores are then aggregated in the form of the algorithm called "summation form". The architecture that is proposed in [5] is largely inspired from the MapReduce architecture proposed in the original paper [6]. The data is then cached for subsequent MapReduce calls. Every algorithm will have its own instance, and each MapReduce task will be assigned to its engine [5]. Similar to the MapReduce engine proposed in [6], this engine will also have a master node which helps coordinates the mappers and the reducers. The masters job consists of assigning the split data to different mappers and then collects the intermediate data from the mappers. After this data is collected, the master node will then delegate the data to the reduce to process it and return the final results. The parallel implementation of the neural network model will focus on the backpropagation process [5]. The network structure defined allows each mapper to propagate its set of data through the neural network. For each training example, the error is backpropagated and a partial gradient is calculated for each of the weights in the network [5]. The reducer then sums all the partial gradients from each mapper and performs a batch gradient descent to update all the weights in the network [5]. The authors experimentation phase consisted of each algorithm running on the MapReduce framework and the other being a serial implementation without the framework. The experiments were performed on 8 machine learning data sets from the UCI Machine Learning repository and two others from research groups. In [22], Long and Gupta presented a scalable parallel artificial neural network using

MPI (Message Passing Interface) for parallelization.It is worth noting that MPI was designed for data intensive applications with high performance requirements. MPI provides little support in fault tolerance. If any fault happens, an MPI computation has to be started from the beginning. As a result, MPI is not suitable for big data applications, which would normally run for many hours during which some faults might happen. Sun et. al [28] implement a deep learning algorithm to train the input data, where a MapReduce programming model is made use of to parallelize the computation. They utilize a cloud computing hadoop cluster. The progress of their machine learning algorithm takes place in three steps [28]. First, the step of pre-training which makes use of deep learning technology to initialize weights, the step to fine-tune the weights, and the last step aiming at improving the precision. The motivation for adding a pre-training step is to counter act the inefficiency of back propagation when converting high-dimensional data into low-dimensional data. They utilize restricted Boltzmann machine for this. In the fine-tuning step, the algorithm will train the pre-trained weights using back-propagation to get a precise value for the weights. In the last step Adaboosting is utilized to refine the result of the data training. For testing their algorithm, the authors in [28] utilize the MNIST dataset and the AWS cloud computing platform with multiple EC2 instances. From the overall evaluations the authors result prove that MapReduce has an excellent speed-up performance and that high classification accuracy can also be achieved once the overfitting problem has been overcame.

CHAPTER 4

Methodology

To tackle the problem statement, we propose three architectures that when coupled with a differentially private algorithm to help preserve privacy and to deal with different types of data-intensive scenarios. The initial situation that we aim to address is in which there is a large volume of testing data to be classified. In this scenario, each mapper builds the same differentially private neural network classifier using the same set of training data and a portion of the testing data. The second data-intensive scenario focuses on the volume of the training data being very large. In this case, the training data is segmented into data chunks which are processed by mappers in parallel. Each mapper still builds the same differentially private neural network but uses only a portion of the training dataset to train. An ensemble technique known as bootstrapping will be used to maintain accuracy. The third scenario targets a situation in which the number of neurons in the neural network is significant. In this case, the third model parallelizes and distributes the neural network among the mappers in such a way that each mapper utilizes a portion of the neurons for training. The neural networks will be implemented using TensorFlow-private enforcing differential privacy, ensuring that the data will remain private. The infrastructure will be created with the help of AWS EMR instances. The Hadoop cluster will consist of 5 nodes, in which four will be Datanodes, and the remaining one will be a Namenode. The hardware setup that will be utilized can be seen in table 4.0.1.

Namenode	CPU: Core i7@3GHz Memory:8 GB SSD: 750 GB OS: Linux
Datanodes	CPU: Core i7@3.8GHz Memory: 32 GB SSD: 250 GB OS: Linux
Network bandwidth	1Gbps
Hadoop version	3.1.1

Table 4.0.1: Hadoop Specifications

4.1 Privacy-Preserving MapReduce Differentially Private Neural Network 1

As previously stated the initial architecture proposed will be dealing with a situation in which there is a large volume of testing data to be classified. Consider a testing dataset that is segmented across mappers $q_i = r_1, r_2, r_3, ..., r_in, q_i \in Q$, where

1. q_i denotes an instance;
2. Q denotes a dataset;
3. in denotes the length of q_i; it is also used to determine the number of inputs to the neural network;
4. the inputs are enclosed in the format of $\langle instance_k, target_k, type \rangle$;
5. $instance_k$ represents q_i, which is the input to the neural network;
6. $target_k$ represents the desired output in $instance_k$ is a training instance.

7. *type* consist of two values "train" or "test", which will be marked by $instance_k$, if the value is "test" the $target_k$ field will be empty.

The files that contain the instance are stored and saved into the Hadoop Distributed File System (HDFS) first. Each file will consist of the entire training data and a portion of the testing data. The number of files will determine the number of mappers to be utilized. In the initialization phase, each mapper will initialize a neural network. Hence, there will be n differentially private neural networks in the cluster. All the neural networks will follow the same structure and parameters. The mappers will read the data in the form of $\langle instance_k, target_k, type \rangle$ from the files and parses the data files. If the type field consists of the "train" value then that instance will be the input in the neural network. The output of each of the layers is computed within the neural network until the output layer generates the final output indicating the end of the feed forward process. The neural networks on each mapper will then begin the back propagation process. The weights and biases for each neuron will be updated. This process is repeated until all the instances that are labelled as "train" are processed and the error is satisfied.

Each mapper will now begin classifying all instances labelled as "test" by running the feed forward process. Since each mapper only classifies a portion of the test set, efficiency has increased. After that every mapper outputs an intermediate key value pair in the form of $\langle instance_k, o_{jm} \rangle$, where $instance_k$ is the key and o_{jm} is the output of the m^{th} mapper. The reducer then collects and merges the outputs the results into the Hadoop Distributed File System. The architecture can be seen in figure 4.1.1.

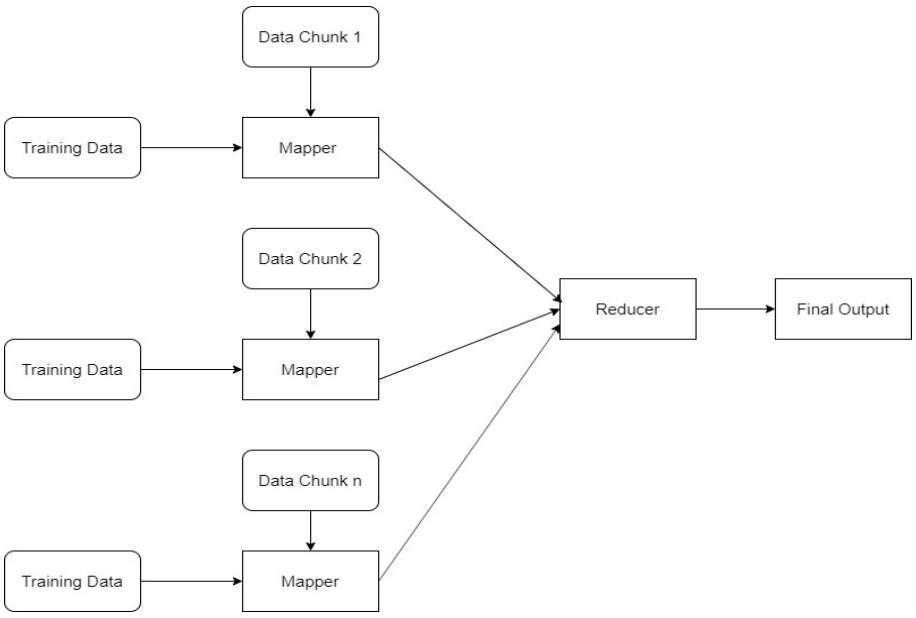


Fig. 4.1.1: Privacy-preserving Model 1 [21]

4.2 Privacy-Preserving MapReduce Differentially Private Neural Network 2

The second scenario focuses on when the training data to be processed is significantly large. In this situation, the training data is divided into chunks. Each of these data chunks is fed as an input to the mappers. The criteria to determine the number of mappers is dependent on the number of data chunks generated. So, each neural network on the mappers will produce an independent classifier based on the training parameters. The neural networks are trained using only a portion of the training data to help reduce the computation overhead. A major issue is that the classification accuracy of each neural network will degrade as it is trained on only a portion of the training data. To solve this an ensemble technique such as bootstrapping is utilized to convert numerous weak learners into a number of strong learners. To give a brief overview, bootstrapping is a statistical re-sampling technique used to estimate statistics on a population by sampling a dataset with replacement. In the initial step of this algorithm balanced bootstrapping is performed to generate a number of

subsets which will then be saved as individual files in the Hadoop Distributed File System (HDFS). When the algorithm starts each mapper constructs one differentially private neural network where the weights and biases are initialized in a range from -1 to 1. The first record is then input into the neural network from the input file. The mapper first parses the data and finds the instance type. If the instance type is "train" the data is fed into the classifier. Each neuron in the different layers will then compute their outputs and generate the final output indicating the end of the feed-forward process. The back-propagation process will then begin and the weights and bias values will be computed and updated accordingly. The training process will continue until all the training instances are run through. Now the all the instances of type "test" will be fed into the classifier for classification. The mappers will produce the final classification results in the format of $\langle instance_k, o_{jm} \rangle$. In the last step, the reducer will collect all the intermediate outputs. The outputs with the same key will then be merged together. The reducer will then perform the majority voting operation and output the final results into the Hadoop Distributed File System. The architecture for the second algorithm can be seen in figure 4.1.2

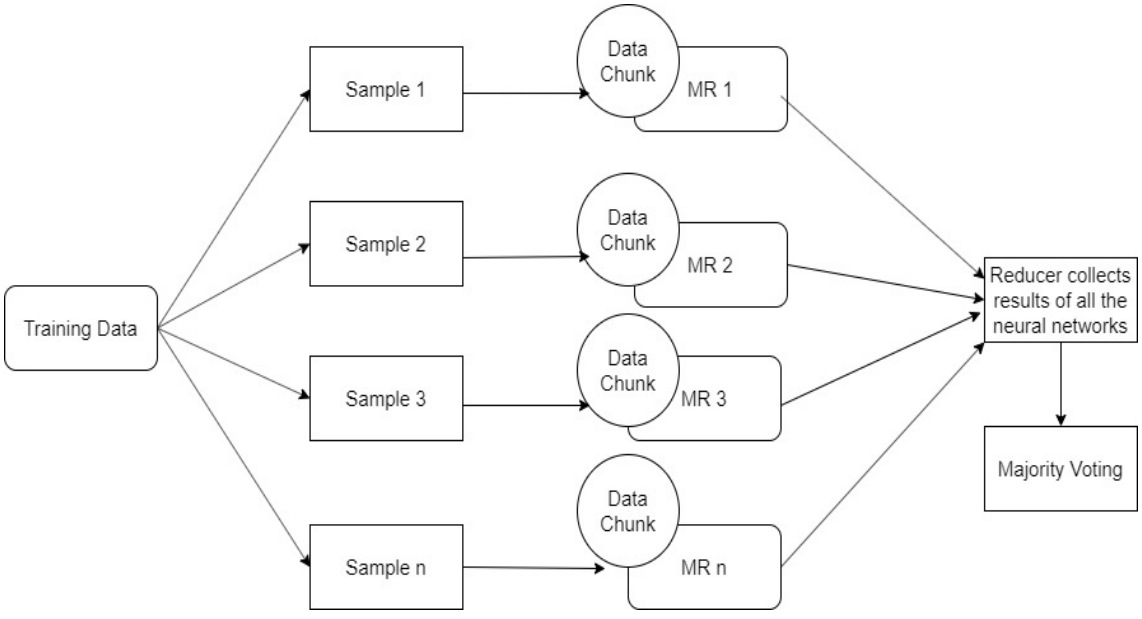


Fig. 4.2.1: Privacy-preserving Model 2[21]

4.3 Privacy-Preserving MapReduce Differentially Private Neural Network 3

The third privacy-preserving algorithm aims to accommodate a significantly large neural network on one MapReduce cluster. Hence, each mapper will be responsible for holding many neurons. Many iterations take place in the algorithm containing n layers. There will be $n-1$ MapReduce tasks to implement the algorithm. The feed-forward process in this algorithm will run for $n-1$ iterations, while the back-propagation process takes place in the last iteration. A data format in the form of $\langle reducer_m, instance_n, w_{ij}, \theta_j, target_n, (w_{ij}^2, \theta_j^2, ..., w_{ij}^{n-1}, \theta_j^{n-1})\rangle$ is used to guarantee that data passes from mapper to reducer in an efficient flow. Here $reducer_m$ represents the m^{th} reducer and $instance_n$ represents the n^{th} training or testing instance from the dataset. The weights and biases are represented by W_{ij} and θ_j. Unlike our previous two models, in this model the parameters are not initialized at startup, they are saved in one file on the Hadoop Distributed File System. When the initial computation begins, the mappers will compute the neurons output and publish in the format of $\langle reducer_m, x_j, (w_{ij}^2, \theta_j^2, ..., w_{ij}^{n-1}, \theta_j^{n-1}), target_n\rangle$, where x_j is the output of the neuron. The first parameter will guarantee that the m^{th} reducer will collect the output, maintaining the structure of the network. In this initial phase the weights and biases will be set to values between -1 to 1. This reducer then produces m outputs. The $reducer_{m'}$ output indicates to the $m^{'th}$ mapper to start processing the output files. These output files are the input files for the next layer in the network. This process continues where the mappers compute the neurons outputs and adjusting the values of the weights and biases accordingly. The process continues until the end of the feed forward process. The last round of back-propagation starts by the mappers processing $\langle reducer_{m'}, x_j, (w_{ij}^{n-1}, \theta_j^{n-1}), target_n\rangle$, computing the output of neurons, and publishing the results in the form of $\langle x_j, target_n\rangle$. One reducer will then collect the outputs in the form of $\langle x_{j1}, x_{j2}, x_{j3}, ..., x_{jk}, target_n\rangle$. The reducer will then begin the back-propagation process and compute new weights and biases for each layer. The previous input files containing the weights, biases, and outputs

are retrieved and updated with the new values. The new values are updated in the $\langle reducer_m, instance_n, w_{ij}, \theta_j, target_n, (w_{ij}^2, \theta_j^2, ..., w_{ij}^{n-1}, \theta_j^{n-1})\rangle$ format. The second instance will be read by the reducer in the way of $\langle instance_{n+1}, target_{n+1}\rangle$ since $instance_{n+1}$ and $target_{n+1}$ represent the updated values. The training process will continue until all instances are processed and the error has been satisfied. During classification, only the feed-forward process will be run and the reducer will publish in the same format as the previous two models. The architecture for this model can be seen in figure 4.3.1.

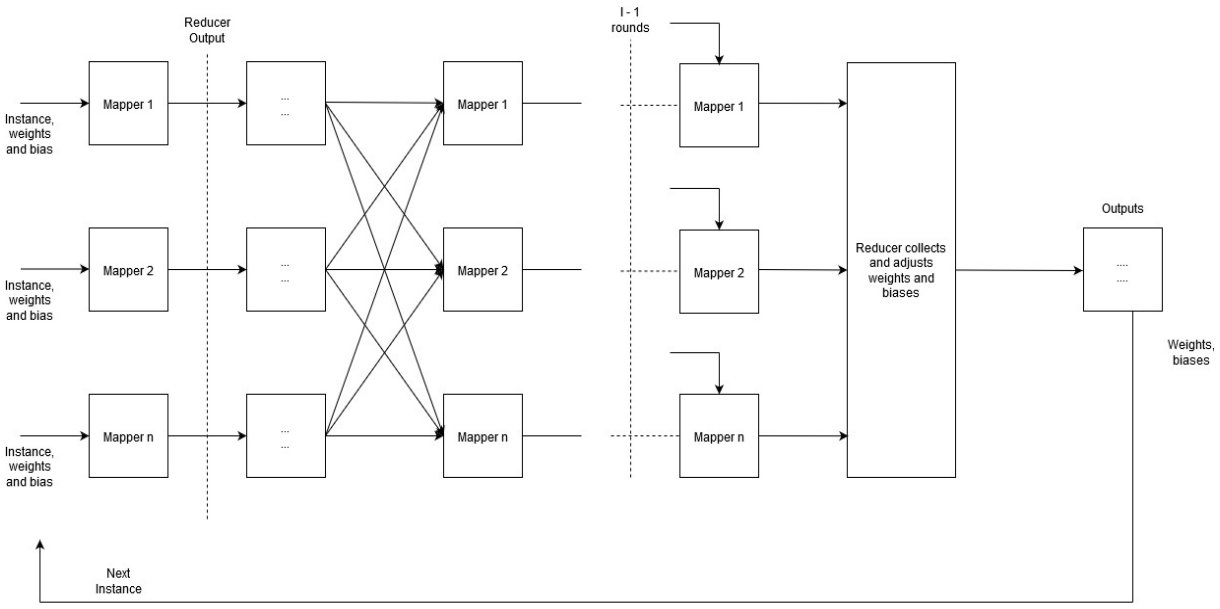


Fig. 4.3.1: Privacy-preserving Model 3 [21]

4.4 Differentially Private Stochastic Gradient Descent

Before we discuss the working of the differentially private stochastic gradient descent, lets look at the working of the traditional stochastic gradient descent. Stochastic gradient descent is an iterative process. During each iteration a random batch of data is sampled from the training set. The error is then calculated between the model prediction and the training label. The error which is also known as the loss is

then differentiate with respect to the model parameters. These derivatives allow us to see which model parameters to tweak in order to reduce the error. This will help the model come closer to predicting the correct label. Iteratively computing the gradients and applying them to update model parameters is known as *descent*. To ensure that traditional stochastic gradient descent is differentially private two modifications are needed. Foremost, the sensitivity of each gradient must be bounded. In simpler words, there is a need to limit the amount of influence an individual training point can have on a gradient computation. To solve this, the gradient is clipped on each training point that it is computed on. This way it allows to bound how much each training point can influence the model parameters. Second, we must randomize the algorithms behaviour to make it statistically impossible to know whether or not any data point was included in the training set by comparing the updates SGD utilizes when it operates with or without the particular point in the training set. This is achieved by sampling random noise and adding it to the clipped gradient. The algorithm, proposed by [1] can be seen in figure 4.4.1.

Algorithm 1 Differentially private SGD (Outline)

Input: Examples $\{x_1, \ldots, x_N\}$, loss function $\mathcal{L}(\theta) = \frac{1}{N}\sum_i \mathcal{L}(\theta, x_i)$. Parameters: learning rate η_t, noise scale σ, group size L, gradient norm bound C.

Initialize θ_0 randomly

for $t \in [T]$ **do**

Take a random sample L_t with sampling probability L/N

Compute gradient

For each $i \in L_t$, compute $\mathbf{g}_t(x_i) \leftarrow \nabla_{\theta_t}\mathcal{L}(\theta_t, x_i)$

Clip gradient

$\bar{\mathbf{g}}_t(x_i) \leftarrow \mathbf{g}_t(x_i) / \max\left(1, \frac{\|\mathbf{g}_t(x_i)\|_2}{C}\right)$

Add noise

$\tilde{\mathbf{g}}_t \leftarrow \frac{1}{L}\left(\sum_i \bar{\mathbf{g}}_t(x_i) + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I})\right)$

Descent

$\theta_{t+1} \leftarrow \theta_t - \eta_t \tilde{\mathbf{g}}_t$

Output θ_T and compute the overall privacy cost (ε, δ) using a privacy accounting method.

Fig. 4.4.1: Differentially Private Stochastic Gradient Descent [1]

4.5 Neural Network Architecture

The architecture of the neural network that is used in our algorithm can be seen in table 4.5.1.

Layers	Filter	Pool Size	Units	Kernel Size	Strides	Padding	Activation Function
Convolutional	16	-	-	8	2	same	RELU
Max Pooling	-	2	-	-	1	-	-
Convolutional	32	-	-	4	2	valid	RELU
Max Pooling	-	2	-	-	1	-	-
Flatten Layer	-	-	-	-	-	-	-
Dense Layer	-	-	32	-	-	-	RELU
Dense Layer	-	-	10	-	-	-	SOFTMAX

Table 4.5.1: Neural Network Architecture

4.6 Datasets

The two datasets that have been utilized are the MNIST and a synthetic dataset that was generated. MNIST is a collection of images that represent handwritten digits. The MNIST dataset consists of a total of 70,000 images. Each image is 28 pixels in width and 28 pixels in height, for a total of 784 pixels. The images are divided in 10 different classes representing numbers from 0 to 9. The synthetic dataset that we have created consists of 8 million values. The dataset consists of 12 features and there are 3 different classes.

CHAPTER 5

Experimental Results & Analysis

Based on the methodology discussed we develop the system and test it various scenarios and derive the analysis of the results. The implementation of the three privacy preserving algorithms was done using Hadoop, an open source implementation framework of the MapReduce computing model. The Hadoop cluster was built on amazon web services using Elastic MapReduce instances.

5.1 Scalability Analysis

Several experiments were carried out to measure the scalability of the algorithms using the MNIST and synthetic dataset. Figure 5.1.1 is showing the computational efficiency of the first privacy-preserving algorithm. From the graph, we can observe the computational efficiency of a stand-alone neural network and the first proposed privacy-preserving algorithm. Initially, our privacy-preserving algorithm behaves similarly to the independent neural network. As the volume of the data increases, it becomes evident that our privacy-preserving algorithm outperforms the independent neural network. This is mainly due to the distribution of the testing data on the mappers and the data nodes. Now figure 5.1.2 which shows the computational efficiency of our second privacy-preserving algorithm. From the figure, we can see that even from the start, the second proposed algorithm slightly beats the stand-alone

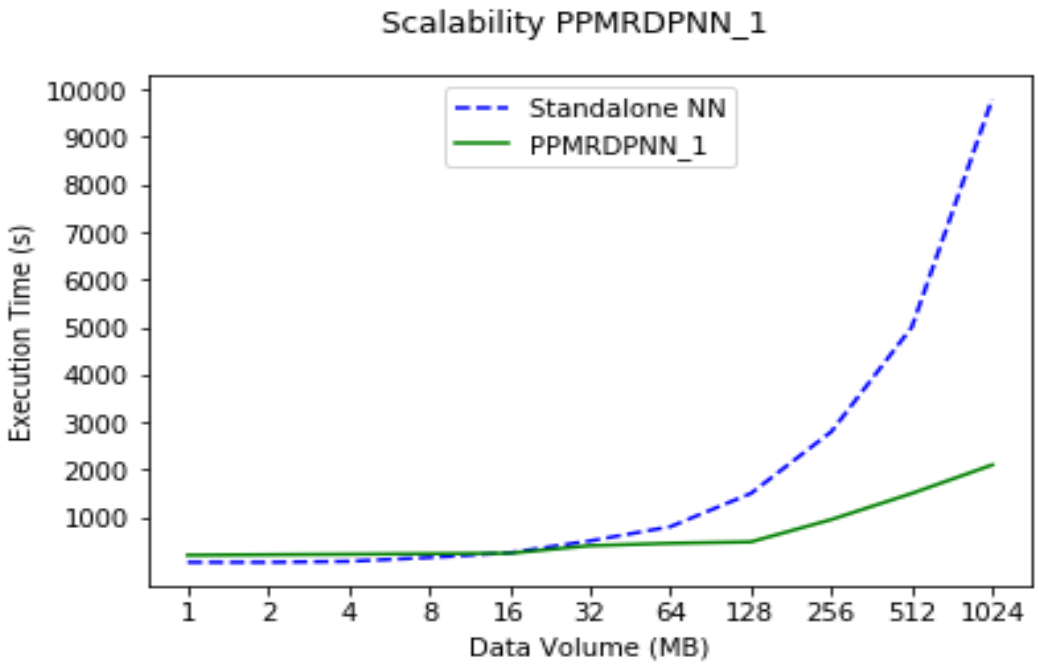


Fig. 5.1.1: Scalability for PPMRDPNN 1

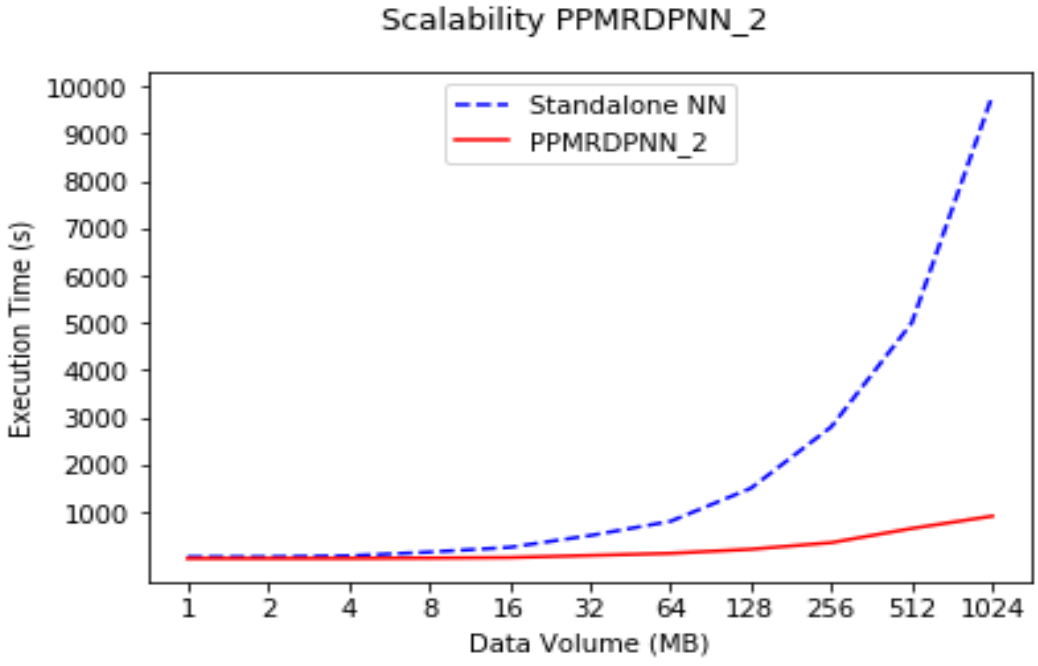


Fig. 5.1.2: Scalability for PPMRDPNN 2

neural network and moves on to ultimately outperform the independent neural network and PPMRDPNN_1. Similar to PPMRDPNN_1 algorithm, PPMRDPNN_2 also

scales well with growth in the data.

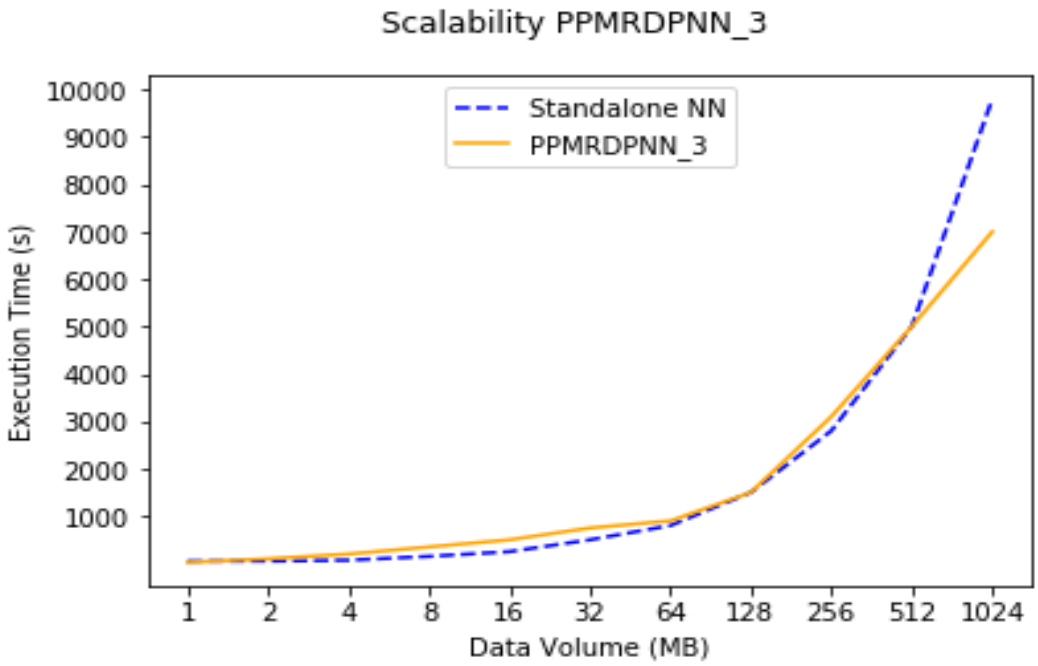


Fig. 5.1.3: Scalability for PPMRDPNN 3

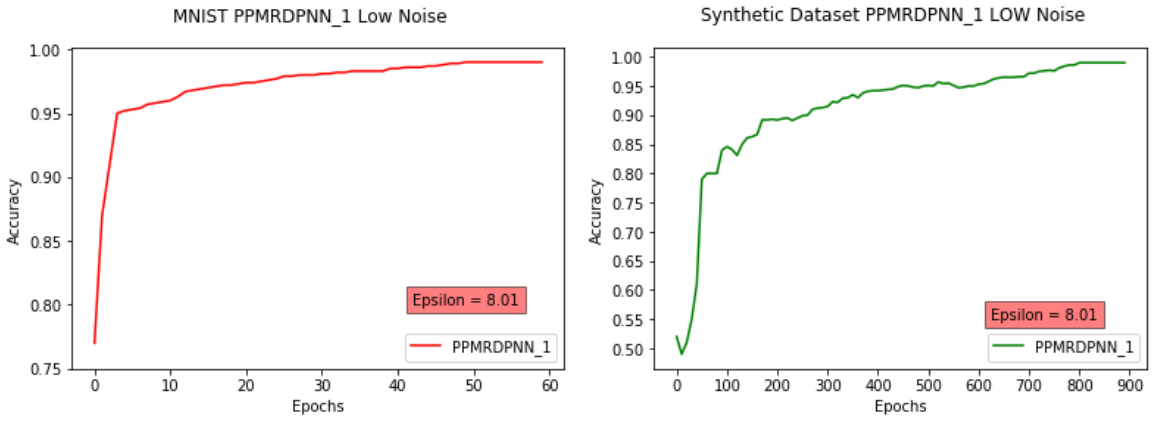
Figure 5.1.3 shows the computation overhead incurred by the third privacy-preserving algorithm. We can observe that the algorithm scales similarly to the independent neural network, although it's still faster. The third privacy-preserving algorithm incurs higher overhead than our previous two proposed algorithms PPMRDPNN_1 and PPMRDPNN_2. This is because the map and reduce tasks are performed in one scheduled job in our last two algorithms while our third privacy-preserving algorithm performs many other duties causing the mappers and reducers to start and stop multiple times continuously. Although PPMRDPNN_3 does not scale as well as PPMRDPNN_1 and PPMRDPNN_2, it still performs better than the independent neural network as the dataset grows larger. From the analysis of the three graphs presented we can see that in terms of scalability, PPMRDPNN_1 and PPMRDPNN_2 would be the most efficient out of the proposed privacy-preserving algorithms. In the next section we will discuss the performance analysis of all three privacy-preserving algorithms.

5.2 Performance Analysis

The performance of all three privacy-preserving algorithms is measured as a function of the privacy budget, ϵ. The algorithms are tested using three different levels of the privacy budget. Each dataset was tested on the model under conditions where the noise multiplier was changing to gauge a better understanding of the tradeoff between accuracy and privacy. The increase in value of the noise multiplier is directly proportional to the increase in security of the data. The levels range from low noise, medium noise, and high noise. When the privacy budget ϵ is small, it signifies higher privacy guarantees but lower accuracy. When the privacy budget ϵ is higher, it means lower privacy guarantees but higher accuracy. The closer the value of ϵ is to zero indicates a higher level of security. In this section, we observe the performance analysis and find an optimal tradeoff between the privacy budget ϵ and accuracy.

5.2.1 PPMRDPNN_1 Analysis

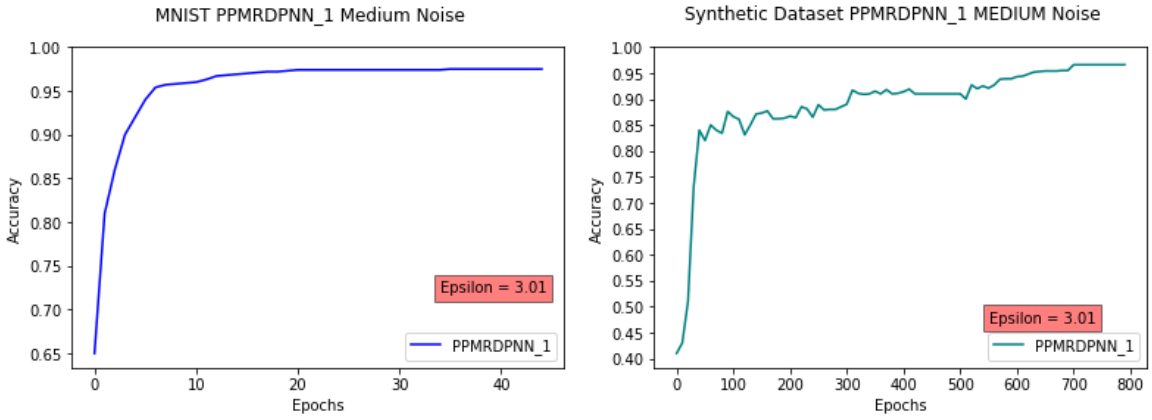
In this section, we will observe the performance for the first privacy preserving algorithm, PPMRDPNN_1. The privacy budget, ϵ is set at 8.01 to indicate low noise injection. So, from Figure 5.2.1 (a), it can be observed that as the algorithm continues to train on the MNIST dataset, the accuracy reaches to 99%. Now this is possible since the noise injected into the data is fairly low.



(a) MNIST (b) Synthetic Dataset

Fig. 5.2.1: PPMRDPNN_1 Low Noise

The model is trained using 32 mappers for 60 epochs on the MNIST dataset and for 900 epochs on the synthetic dataset. Each of the 32 mappers gets an entire portion of the training set and 1 of 32 splits of the test data. Each mapper would classify 1,250 instances for the MNIST dataset and 165,000 instances for the synthetic dataset. By observing Figure 5.2.1(b), we can see that the training process is rougher than the training process on the MNIST dataset. This is due to a few reasons, such as the quality of the dataset. The MNIST dataset is much more refined than our synthetic dataset. Also, we must consider that each mapper is training on more values using synthetic dataset, which would result in more fluctuations. Regardless, PPMRDPNN_1 still achieves the same accuracy of 99% since there is low noise injection.

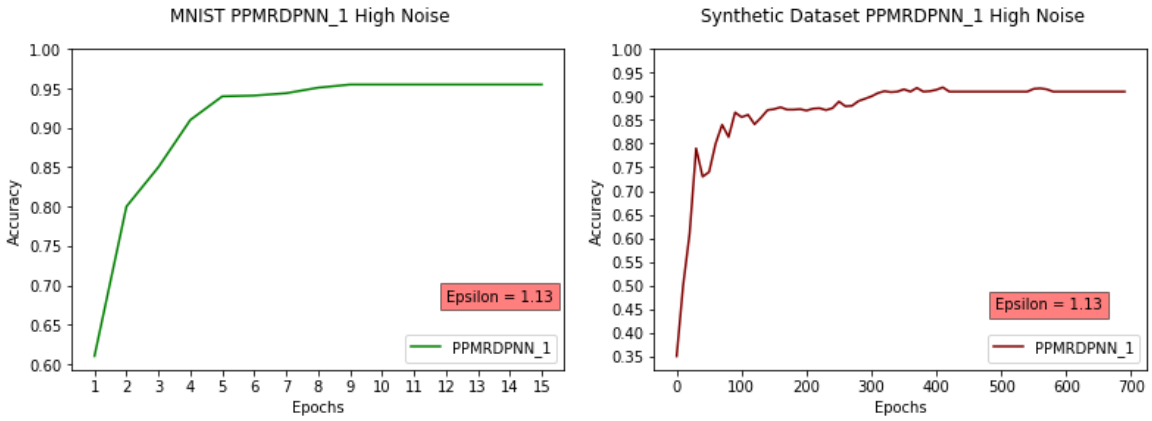


(a) MNIST (b) Synthetic Dataset

Fig. 5.2.2: PPMRDPNN_1 Medium Noise

From figure 5.2.2, we can observe the performance of PPMRDPNN_1 when a medium amount of noise is injected into the data. The privacy budget ϵ is equal to 3.01 in this case. The same measures as previous are used to evaluate the model when medium noise is injected. We can observe that with the increase in noise our model achieved 97.5% on the MNIST dataset and 96.6% on the synthetic dataset. It can also be seen that the curve for training is smoother for the MNIST dataset than our synthetic dataset due to reasons we have already discussed. It can also be seen to achieve a lower privacy budget our model is trained for less epochs on both datasets. This time we train for 45 epochs on the MNIST dataset and 800 epochs on the synthetic dataset

allowing the model fewer epochs to query the noisy data when training resulting in a lower privacy budget. Although there is a minor drop in accuracy, the increase in security allows the model to be an positive choice for privacy preserving deep learning.



(a) MNIST (b) Synthetic Dataset

Fig. 5.2.3: PPMRDPNN_1 High Noise

The graphs in figure 5.2.3 show the performance of PPMRDPNN_1 under the conditions when high noise is injected into the system. The privacy budget ϵ is equal to 1.13 making it as close to industry standard as possible. We train the neural network for even fewer epochs this time. We train the model for 15 epochs on the MNIST dataset and for 700 epochs on the synthetic dataset. There is a significant drop in accuracy in comparison to when $\epsilon = 8.01$ and $\epsilon = 3.01$. When we inject high noise we obtain an accuracy of 95.5% for the MNIST dataset and 91% for the synthetic dataset. Although the accuracy for the MNIST dataset is not bad, the accuracy drop for the synthetic dataset indicates that for larger datasets the accuracy could be below expectations. Looking at the overall results regarding the scalability of the algorithm and the privacy - accuracy tradeoff, PPMRDPNN_1 has proven to be an optimistic model for large scale privacy preserving deep learning. Table 5.2.1 shows the overall results of the model for the MNIST dataset can be seen along with other parameters used to train the model. Table 5.2.2 shows the overall results and parameters for the synthetic dataset.

Learning Rate	Noise Multiplier	Number of Microbatches	Epochs	Privacy ϵ	Accuracy
0.25	1.3	256	15	1.13	95.5%
0.25	1.0	256	45	3.01	97.5%
0.25	0.7	256	60	8.01	99.0%

Table 5.2.1: PPMRDPNN_1 MNIST Overall Results

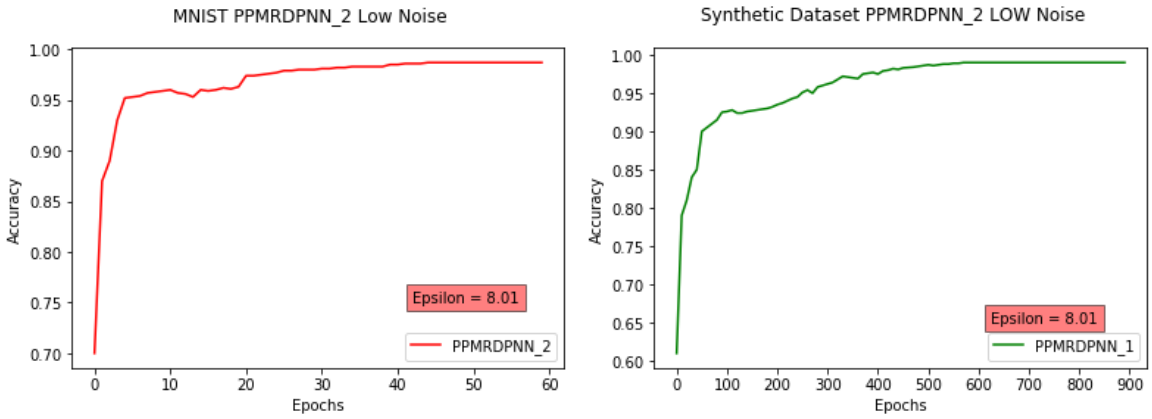
Learning Rate	Noise Multiplier	Number of Microbatches	Epochs	Privacy ϵ	Accuracy
0.25	7	33,360	700	1.13	91%
0.25	6.6	33,360	800	3.01	96.6%
0.25	5.9	33,360	900	8.01	99.0%

Table 5.2.2: PPMRDPNN_1 Synthetic Data Overall Results

After analyzing both tables we can come to the conclusion that an optimal tradeoff between privacy and accuracy can be achieved while distributing the computation of the neural network.

5.2.2 PPMRDPNN_2 Analysis

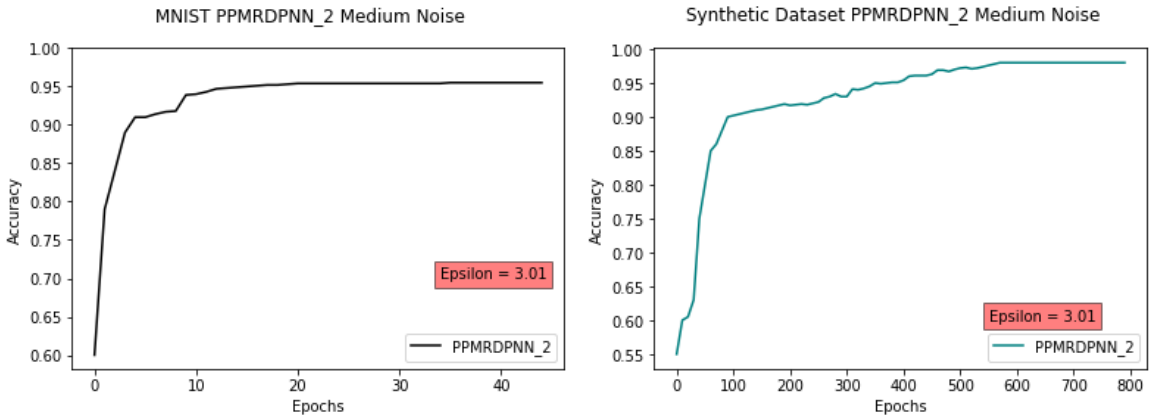
To evaluate PPMRDPNN_2, we use the same two datasets, where the mappers are trained by subsets of the training data and produced classification results of the testing instances based on bootstrapping and majority voting. Figure 5.2.4 presents the accuracy results of the PPMRDPNN_2 on the two datasets when low noise is injected. It shows that with an increase in the number of training epochs in each sub neural network, the accuracy based on majority voting continues to increase. In the scenario of low noise injection, PPMRDPNN_2 achieves an accuracy of 98.7% on the MNIST data set when trained for 60 epochs. An accuracy of 99% is produced on the synthetic dataset when trained for 900 epochs. When we compare the accuracy curve of both datasets to the PPMRDPNN_1 low noise results, we see that the curve is smoother for the PPMRDPNN_2 algorithm. This indicates that the PPMRDPNN_2 model shows higher stability. The privacy budget for low noise injection remains the



(a) MNIST (b) Synthetic Dataset

Fig. 5.2.4: PPMRDPNN_2 Low Noise

same at $\epsilon = 8.01$. This time 18 mappers are used with each mapper classifying 1100 instances for the MNIST dataset and 147,000 instances for the synthetic dataset. We can observe the performance results for PPMRDPNN_2 with medium noise injection in figure 5.2.5.

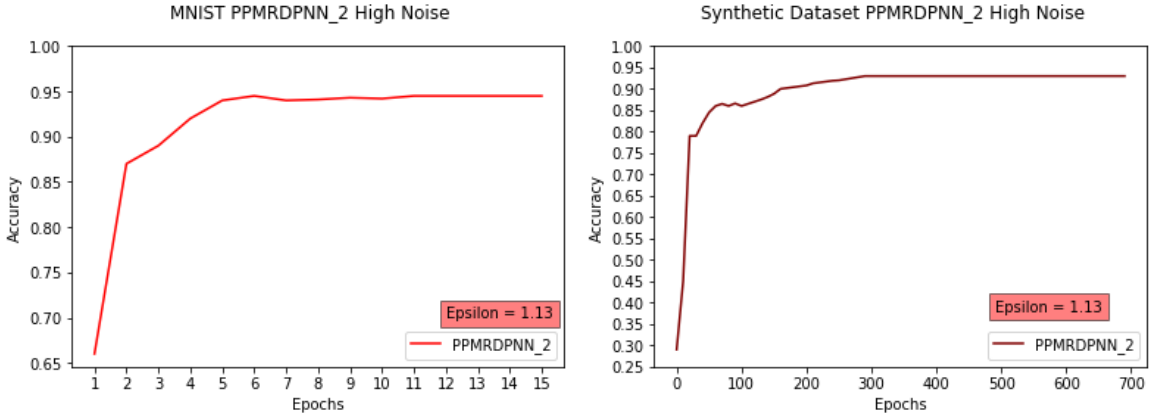


(a) MNIST (b) Synthetic Dataset

Fig. 5.2.5: PPMRDPNN_2 Medium Noise

The PPMRDPNN_2 model is able to achieve an accuracy of 95.5% on the MNIST dataset when the privacy budget $\epsilon = 3.01$ and an accuracy of 98% on the synthetic dataset. To achieve an accuracy this high with medium level noise injection proves that this model is better than our previous PPMRDPNN_1 algorithm. The smooth learning curve further proves our hypothesis of PPMRDPNN_2 being more stable

than PPMRDPNN_1. The model starts off slowly with a low accuracy for both datasets and then proceeds to increase drastically with increase in training. Hence, the bootstrapping algorithm and majority voting maintain the classification accuracy even though the mappers are training on segmented datasets. Figure 5.2.6 presents



(a) MNIST (b) Synthetic Dataset

Fig. 5.2.6: PPMRDPNN_2 High Noise

the accuracy under high noise injection. The accuracy drop is minimal for the MNIST dataset under high noise injection as an accuracy of 94.5% is still achieved. Although, we see a significant drop in accuracy from 98% to 93% for the synthetics dataset. This is due to the dataset being significantly larger than the MNIST dataset meaning at higher noise injection it becomes to difficult for the model to extract meaningful patterns. The privacy budget $\epsilon = 1.13$ shows that a high amount of noise was introduced into the system and our algorithm still produces optimistic results. Evaluating Table 5.2.3 and 5.2.4 we can see that with medium noise injection an optimal tradeoff can be achieved and our second privacy preserving algorithm is a viable solution for large scale privacy preserving deep learning.

Learning Rate	Noise Multiplier	Number of Microbatches	Epochs	Privacy ϵ	Accuracy
0.25	1.3	256	15	1.13	94.5%
0.25	1.0	256	45	3.01	95.5%
0.25	0.7	256	60	8.01	98.7%

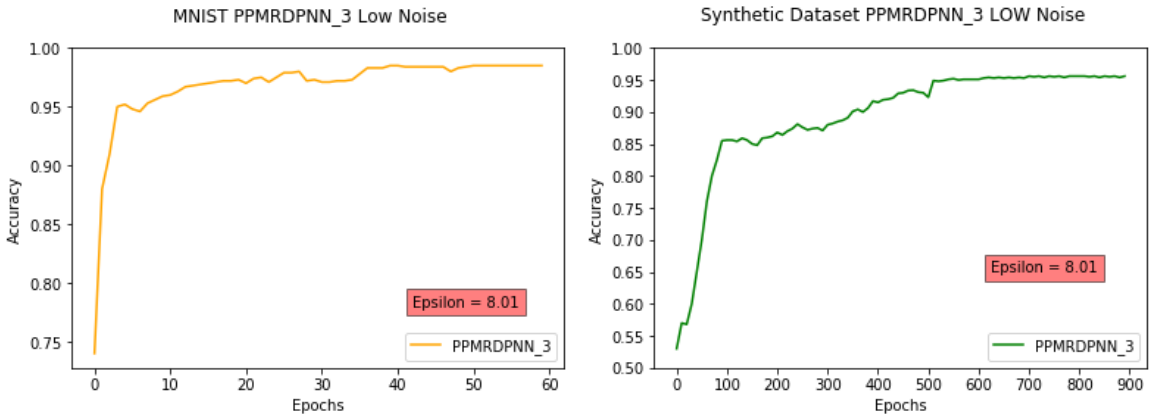
Table 5.2.3: PPMRDPNN_2 MNIST Overall Results

Learning Rate	Noise Multiplier	Number of Microbatches	Epochs	Privacy ϵ	Accuracy
0.25	7	33,360	700	1.13	93.0%
0.25	6.6	33,360	800	3.01	98.0%
0.25	5.9	33,360	900	8.01	99.0%

Table 5.2.4: PPMRDPNN_2 Synthetic Data Overall Results

5.2.3 PPMRDPNN_3 Analysis

PPMRDPNN_3 executes an entirely parallel and circulated neural network utilizing Hadoop to manage a complex network with many neurons. Figure 5.2.7 presents the results of our third proposed method with low noise injection. From observing the

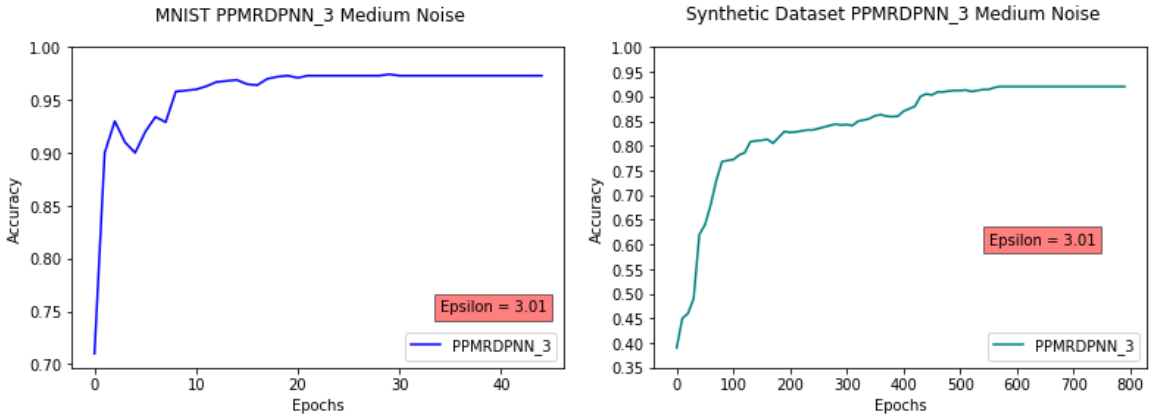


(a) MNIST (b) Synthetic Dataset

Fig. 5.2.7: PPMRDPNN_3 Low Noise

figure we can see that the PPMRDPNN_3 trains smoothly on the MNIST dataset but fluctuations can be observed in the training of the synthetic dataset. We utilize 32 mappers in the implementation of this algorithm. In comparison with our previous two algorithms, PPMRDPNN_3 produces a lower accuracy when there is low noise injection. An accuracy of 98.5% and 95.6% is achieved for the MNIST and synthetic dataset. As seen in the other algorithms, the accuracy continues to increase as it is trained on more instances. The privacy budget ϵ is the same for all low noise injection scenarios at 8.01. The accuracy obtained is respectable but its not comparable to the

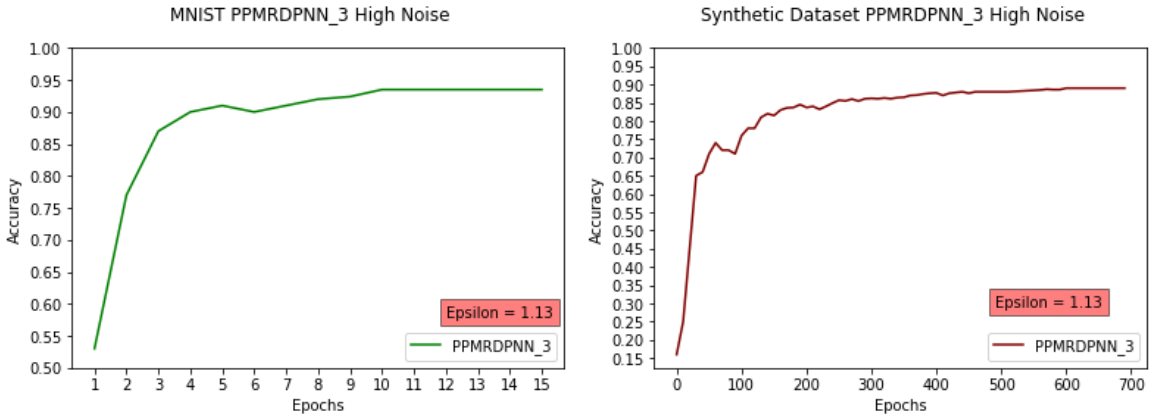
accuracy achieved by PPMRDPNN_1 and PPMRDPNN_2.



(a) MNIST (b) Synthetic Dataset

Fig. 5.2.8: PPMRDPNN_3 Medium Noise

When we observe the results obtain by PPMRDPNN_3 in the case of medium noise injection, we can notice the fluctuation in the accuracy curve for both datasets. We see an accuracy of 97.3% and 92% achieved for the MNIST and synthetic datasets. The accuracy achieved for the MNIST dataset is respectable with the privacy budget being 3.01. Regardless, the accuracy obtained for the synthetic dataset is fairly low in comparison with our previous model under the condition of medium noise injection. Based on the fluctuations seen in the accuracy curves we can assess that this model is not as stable as our previous two models.



(a) MNIST (b) Synthetic Dataset

Fig. 5.2.9: PPMRDPNN_3 High Noise

This is justified since PPMRDPNN_3 has to perform several other tasks discussed in chapter 4 unlike PPMRDPNN_1 and PPMRDPNN_2. Figure 5.2.9 shows the results for PPMRDPNN_3 under high noise conditions. Under the high noise conditions the accuracy drops significantly for this model. An accuracy of 93.5% and 89% is achieved for the MNIST and synthetic dataset. In comparison to all the other tests performed, PPMRDPNN_3 performs poorly under these conditions. When we couple these results with the scalability analysis of PPMRDPNN_3 we see that in comparison to the other two models, this model is outperformed. Although it does not perform as well as the other two models, the results are still respectable under the conditions through which it was implemented and the additional tasks that it performs. The model performs well under the persepective of feasibility testing. Table 5.2.5 and 5.2.6 show the overall results and other model parameters.

Learning Rate	Noise Multiplier	Number of Microbatches	Epochs	Privacy ϵ	Accuracy
0.25	1.3	256	15	1.13	93.5%
0.25	1.0	256	45	3.01	97.3%
0.25	0.7	256	60	8.01	98.5%

Table 5.2.5: PPMRDPNN_3 MNIST Overall Results

Learning Rate	Noise Multiplier	Number of Microbatches	Epochs	Privacy ϵ	Accuracy
0.25	7	33,360	700	1.13	89.0%
0.25	6.6	33,360	800	3.01	92.0%
0.25	5.9	33,360	900	8.01	95.6%

Table 5.2.6: PPMRDPNN_3 Synthetic Data Overall Results

5.3 Comparative Analysis

To get a better understanding of our results, we compare them with several states of the art privacy-preserving neural networks. The results in table 5.3.1 represent the results obtained by other researchers, and table 5.3.2 represents the results obtained by the execution of our privacy-preserving models. When we compare our results

Model	Dataset	Neural Network Depth	Accuracy
Shokri et.al. [27]	MNIST	11 layers	99.17%
MiniONN [20]	MNIST	9 layers	97.6%
Gilad-Bachrach et. al. [7]	MNIST	9 Layers	99.0%
SecureML [24]	MNIST	5 Layer	93.1 %

Table 5.3.1: Results Obtained by Peer Researchers

Model	Privacy Budget (ϵ)	Neural Network Depth	Accuracy		Scalability
			MNIST	Synthetic	
PPMRDPNN_1	1.13	7 Layers	95.5%	91.0%	Good
	3.01		97.5%	96.6%	
	8.01		99.0%	99.0%	
PPMRDPNN_2	1.13	7 Layers	94.5%	93.0%	Excellent
	3.01		95.5%	98.0%	
	8.01		98.7%	99.0%	
PPMRDPNN_3	1.13	7 Layers	93.5%	89.0%	Poor
	3.01		97.3%	92.0%	
	8.01		98.5%	95.6%	

Table 5.3.2: Results Obtained by Proposed Methods

with the results obtained by other researchers, we see that there is not a significant difference in the accuracy obtained. The privacy models implemented by [20], [24], [7] are all secure cryptonets that we have discussed in chapter 3. Now, these

secure cryptonets use encryption techniques to protect user data during the training phase. When these encryption techniques are coupled with the complex sigmoid calculations of the neural network, the scalability of the neural networks collapses. When we compare the accuracy of these approaches, we see that our models match the performance of the state of the art methods under the conditions of low noise injection. Our neural networks can achieve similar accuracies using fewer layers of processing. We also present the scalabilities of all of our models and prove that an optimal privacy-accuracy tradeoff can be achieved for large scale privacy-preserving deep learning. For example, if we look at table 5.3.2 we can see that PPMRDPNN_2 with medium and high noise injection still achieves a respectable accuracy with minor discrepancies when compared to the models proposed by [20], [24], [7], and [27]. An additional benefit of PPMRDPNN_2 is the excellent scalability of the model for large datasets. PPMRDPNN_1 also provides excellent results under high and medium noise injection with good scalability. PPMRDPNN_3 achieves respectable accuracy scores for the MNIST dataset under high and medium noise injection in comparison to the peer models. The poor scalability aspect does not make it a viable model for large scale datasets. When we compare our models with the performance presented by [27], we see that Shokri et al. achieves higher accuracy. Still, the scalability of their model is not tested against large datasets. Additionally, their model is still susceptible to reconstruction attacks since they rely on a third party parameter server. In conclusion, our PPMRDPNN_1 and PPMRDPNN_2 model prove to be a viable option for large scale privacy-preserving deep learning as they provide optimal scalability coupled with an efficient privacy and accuracy tradeoff.

CHAPTER 6

Conclusion & Future Work

6.1 Conclusion

In this thesis, we address the scalability and privacy issues of neural networks for large scale privacy-preserving deep learning with the help of the MapReduce framework. We implement the neural network architecture with the help of TensorFlow-privacy and optimize the system using a differentially private stochastic gradient descent algorithm to ensure strong privacy guarantees. In the initial chapters, we discuss the various needs for privacy-preserving deep learning and different privacy-preserving algorithms. We also discuss the utility of the neural network and also discuss methods used in literature to scale and preserve the privacy using the neural network model. From this, we pick up a new problem of implementing multiple neural networks in parallel using various data distribution techniques that utilizes an existing differentially private algorithm to enable a way to secure user data while extracting the full benefits of deep learning. We propose three novel models to achieve privacy-preserving deep learning, and we evaluate and analyze the models thoroughly.

In the later sections, we do an in-depth analysis of the scalability aspects of all three models. We test all three models under three levels of noise injections to provide strong privacy guarantees. We then do a performance analysis of our model by examining the effects by tweaking core parameters. We benchmark the performance of our models with existing approaches. Finally, although the architecture of our peers work is different, we can say that the performance of our models can match the performance of their work in aspects such as accuracy.

6.2 Future Work

When implementing our models, we restrict it to the MapReduce framework. In the future, using Apache Spark would be beneficial as it provides in-memory processing to speed up the execution process. Additionally, we use TensorFlow-privacy and differential privacy to implement and train our neural networks. The models could be trained on a secure and trusted hardware environment such as Intel SGX to provide additional privacy guarantees. Furthermore, our models are most viable for multiple data owners. There is a future scope to implement these models and make them feasible for a broader audience.

REFERENCES

[1] Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., and Zhang, L. (2016). Deep Learning with Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 308–318, Vienna, Austria. Association for Computing Machinery.

[2] Al, M., Chanyaswad, T., and Kung, S. Y. (2018). Multi-Kernel, Deep Neural Network and Hybrid Models for Privacy Preserving Machine Learning. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2891–2895.

[3] Aldeen, Y. A. A. S., Salleh, M., and Razzaque, M. A. (2015). A comprehensive review on privacy preserving data mining. *SpringerPlus*, 4(1):694.

[4] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.

[5] Chu, C.-T., Kim, S. K., Lin, Y.-A., Yu, Y., Bradski, G., Ng, A. Y., and Olukotun, K. (2006). Map-reduce for Machine Learning on Multicore. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, NIPS'06, pages 281–288, Cambridge, MA, USA. MIT Press. event-place: Canada.

[6] Dean, J. and Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. In *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, pages 137–150, San Francisco, CA.

[7] Dowlin, N., Gilad-Bachrach, R., Laine, K., Lauter, K., Naehrig, M., and Wernsing,

J. (2016). CryptoNets: applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 201–210, New York, NY, USA. JMLR.org.

[8] Dwork, C. and Roth, A. (2014). The Algorithmic Foundations of Differential Privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407.

[9] Efron, B. (1979). Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, 7(1):1–26. Publisher: Institute of Mathematical Statistics.

[10] Frikken, K. B. (2011). Secure Multiparty Computation (SMC). In van Tilborg, H. C. A. and Jajodia, S., editors, *Encyclopedia of Cryptography and Security*, pages 1121–1123. Springer US, Boston, MA.

[11] Graepel, T., Lauter, K., and Naehrig, M. (2013). ML Confidential: Machine Learning on Encrypted Data. In Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Mattern, F., Mitchell, J. C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M. Y., Weikum, G., Kwon, T., Lee, M.-K., and Kwon, D., editors, *Information Security and Cryptology – ICISC 2012*, volume 7839, pages 1–21. Springer Berlin Heidelberg, Berlin, Heidelberg. Series Title: Lecture Notes in Computer Science.

[12] Guijarro-Berdiñas, B., Martínez-Rego, D., and Fernández-Lorenzo, S. (2009). Privacy-Preserving Distributed Learning Based on Genetic Algorithms and Artificial Neural Networks. In Omatu, S., Rocha, M. P., Bravo, J., Fernández, F., Corchado, E., Bustillo, A., and Corchado, J. M., editors, *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, Lecture Notes in Computer Science, pages 195–202. Springer Berlin Heidelberg.

[13] Hesamifard, E., Takabi, H., Ghasemi, M., and Wright, R. N. (2018). Privacy-preserving Machine Learning as a Service. *Proceedings on Privacy Enhancing Technologies*, 2018(3):123–142.

[14] Hunt, T., Song, C., Shokri, R., Shmatikov, V., and Witchel, E. (2018). Chiron: Privacy-preserving Machine Learning as a Service. *arXiv:1803.05961 [cs]*. arXiv: 1803.05961.

[15] Jain, P., Gyanchandani, M., and Khare, N. (2016). Big data privacy: a technological perspective and review. *Journal of Big Data*, 3(1):25.

[16] Ji, Z., Lipton, Z. C., and Elkan, C. (2014). Differential Privacy and Machine Learning: a Survey and Review. *arXiv:1412.7584 [cs]*. arXiv: 1412.7584.

[17] Juvekar, C., Vaikuntanathan, V., and Chandrakasan, A. (2018). Gazelle: A Low Latency Framework for Secure Neural Network Inference. *arXiv:1801.05507 [cs]*. arXiv: 1801.05507.

[18] Li, G., Su, X., and Wang, Y. (2019). A Privacy Protection Method for Learning Artificial Neural Network on Vertically Distributed Data. In Deng, K., Yu, Z., Patnaik, S., and Wang, J., editors, *Recent Developments in Mechatronics and Intelligent Robotics*, Advances in Intelligent Systems and Computing, pages 1159–1167. Springer International Publishing.

[19] Li, N., Li, T., and Venkatasubramanian, S. (2007). t-Closeness: Privacy Beyond k-Anonymity and l-Diversity. *2007 IEEE 23rd International Conference on Data Engineering*, pages 106–115.

[20] Liu, J., Juuti, M., Lu, Y., and Asokan, N. (2017). Oblivious Neural Network Predictions via MiniONN Transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security - CCS '17*, pages 619–631, Dallas, Texas, USA. ACM Press.

[21] Liu, Y., Yang, J., Huang, Y., Xu, L., Li, S., and Qi, M. (2015). MapReduce Based Parallel Neural Networks in Enabling Large Scale Machine Learning.

[22] Long, L. N. and Gupta, A. (2008). Scalable massively parallel artificial neural networks. *Journal of Aerospace Computing, Information and Communication*, 5(1):3–15.

[23] Machanavajjhala, A., Kifer, D., Gehrke, J., and Venkitasubramaniam, M. (2007). L-diversity: Privacy Beyond K-anonymity. *ACM Trans. Knowl. Discov. Data*, 1(1).

[24] Mohassel, P. and Zhang, Y. (2017). SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–38, San Jose, CA, USA. IEEE.

[25] Pathak, M. A. and Raj, B. (2013). Privacy-Preserving Speaker Verification and Identification Using Gaussian Mixture Models. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(2):397–406.

[26] Rajaraman, A. and Ullman, J. D. (2011). *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA.

[27] Shokri, R. and Shmatikov, V. (2015). Privacy-Preserving Deep Learning. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1310–1321, New York, NY, USA. ACM. event-place: Denver, Colorado, USA.

[28] Sun, K., Wei, X., Jia, G., Wang, R., and Li, R. (2015). Large-scale Artificial Neural Network: MapReduce-based Deep Learning.

[29] Sweeney, L. (2002). K-anonymity: A Model for Protecting Privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570.

[30] Tanuwidjaja, H. C., Choi, R., and Kim, K. (2019). A Survey on Deep Learning Techniques for Privacy-Preserving. In Chen, X., Huang, X., and Zhang, J., editors, *Machine Learning for Cyber Security*, Lecture Notes in Computer Science, pages 29–46, Cham. Springer International Publishing.

[31] Yao, A. C. (1982). Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164, USA. IEEE Computer Society.

VITA AUCTORIS

NAME:	Dipeshkumar Shaileshkumar Patel
PLACE OF BIRTH:	Lusaka, Zambia
YEAR OF BIRTH:	1996
EDUCATION:	Higher Secondary Diploma – Science Stream, Knowledge High School, Nadiad, Gujarat, India, 2014
	Charotar University of Science and Technology (CHARUSAT), B.Tech in IT Engineering, Anand, Gujarat, India, 2018
	University of Windsor, M.Sc in Computer Science, Windsor, Ontario, Canada, 2020