PRIVACY-PRESERVING DISTRIBUTED DEEP LEARNING

A Thesis by

Aseem Prashar

Bachelor of Engineering, Birla Institute of Technology and Science, 2013

Submitted to the Department of Electrical Engineering and Computer Science
and the faculty of the Graduate School of
Wichita State University
in partial fulfillment of
the requirements for the degree of
Master of Science

May 2020

SECURE PRIVACY PRESERVING DEEP LEARNING AGAINST GAN ATTACKS

The following faculty members have examined the final copy of this thesis for form and content, and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science with a major in Computer Science.

Sergio Salinas, Committee Chair

Akmal Mirsadikov, Committee Member

Remi Chou, Committee Member

Ajita Rattani, Committee Member

DEDICATION

I dedicate this thesis to my family, friends, and colleagues.

Somewhere, something incredible is waiting to be known.

ACKNOWLEDGEMENTS

I would like to thank my adviser, Dr. Sergio A. Salinas Monroy, for his thoughtful input, guidance and support in all stages of this project. I also extend my gratitude to members of my committee, Dr. Ajita Rattani, Dr. Remi Chou and Dr. Akmal Mirsadikov for their valuable time and consideration.

ABSTRACT

Deep neural networks are becoming popular in a variety of fields due to their ability to learn from large-scale data sets. Recently, researchers have proposed distributed learning architectures that allow multiple users to share their data to train deep learning models. Unfortunately, privacy and confidentiality concerns limit the application of this approach, preventing certain organizations such as medical institutions to fully benefit from distributed deep learning. To overcome this challenge, researches have proposed algorithms that only share neural network parameters. This approach allows users to keep their private datasets secret while still having access to the improved deep neural networks trained with the data from all participants. However, existing distributed learning approaches are vulnerable to attacks where a malicious user can use the the shared neural network parameters to recreate the private data from other users.

We propose a distributed deep learning algorithm that allows a user to improve its deep-learning model while preserving its privacy from such attacks. Specifically, our approach focuses on protecting the privacy of a single user by limiting the number of times other users can download and upload parameters from the main deep neural network. By doing so, our approach limits ability of the attackers to recreate private data samples from the reference user while maintaining a highly accurate deep neural network.

Our approach is flexible and can be adapted to work with any deep neural network architectures. We conduct extensive experiments to verify the proposed approach. We observe that the trained neural network can achieve an accuracy of 95.18%, while protecting the privacy of the reference user by preventing it from sharing both its private data and deep neural network parameters with the server or other users.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

GAN	Generative Adversarial Network
SMC	Secure Multi-party Computation
MLP	Multilayer Perceptron
GD	Gradient Descent
DNN	Deep Neural Network
SGD	Stochastic Gradient Descent
PS	Parameter Server

LIST OF SYMBOLS

N	Number of participants
M	Mini batch size used for stochastic gradient descent
θ_u^i	Fraction of parameters selected for upload for ith user
θ_d^i	Fraction of parameters selected for download for ith user
W_k	Weight matrix for layer K in neural network
w_i	Flattened vector of all parameters in neural network for ith user
Δw_i	Vector of changes in all local parameters due to gradient descent
$w^{(global)}$	Flattened parameter vector for server
E	Error function computed over expected value of the objective function
α	Learning rate of the stochastic gradient descent algorithm
S_i	Set of indices selected from flattened vector of parameters for ith user
u_0	Reference user

CHAPTER I

INTRODUCTION

In the past few decades, deep learning has generated a lot of interest in the research and academic community due to its ability to automatically classify large amounts of data. This has led to breakthroughs in many fields ranging from autonomous driving and natural language processing to genetic research [1, 2, 3, 4]. This revolutionary technology is especially fruitful for large corporations that need to automatically process very large data sets to provide deep learning inference services to their users. For example, data collection at Facebook enabled them to create DeepText, a text understanding engine that is able to extract meaningful context from text [5].

Although it is possible to collect large-scale data to train deep learning models in some application domains, e.g., online social networks, there are other fields where such centralized data collection is currently infeasible due to privacy concerns. In particular, users' data can contain instances of private information that needs to be kept a secret from third-parties such as the companies that centrally collect data to provide deep learning services. Such data can include intellectual property (IP), medical data protected by Health Insurance and Portability and Accountability Act (HIPA), and student records protected by the Family Educational Rights and Privacy Act (FERPA) [6, 7, 8].

Instead of sharing private data with a third-party, users could locally train their own deep learning models using their own data. However, since a single user only has access to a small data set compared to the data set that could be centrally collected by the third-party, its locally trained deep learning model have low accuracy, which significantly degrades its ability to provide correct inferences. Moreover, locally trained models can suffer from over generalization which can prevent them from being used in practical scenarios. For instance, a hospital trained a deep neural network to recognize patients with pneumonia based on their chest X-ray images using its own records. Although this neural network was successful in classifying records from patients that were treated in this hospital, its performance significantly dropped when tested with images from other patient treated at a different hospital [9]. This was due to the fact the trained deep neural network was basing its inferences on the particular imaging machine used at the hospital rather than on the the content of the image itself.

The above outlined issues beg for a deep learning solution that can achieve high accuracy while preserving privacy. Researchers have made some headway in this direction by leveraging distributed learning. In this approach [10], a set of users locally train a deep learning model only using their local data. To overcome the challenge of over generalization and small data sets, they send certain parameters of their model to a central server for aggregation. Then, the users can download the aggregated parameters to improve their local models. This approach achieves high accuracy. Although users can protect their private data by only sharing the model parameters, recent studies have shown that it is possible for malicious users to recreate samples of other users by using a generative adversarial network (GAN) [11]. This attack can be successful even when the victim users obfuscated their private data before performing local training.

In this paper, we design, implement, and analyze a distributed deep learning framework that enables a user to benefit from highly accurate distributed learning while preserving the privacy of its local data. Specifically, instead of attempting to protect the privacy of all users as in [10], we focus on protecting the privacy of a single user, called the reference user, by preventing it from uploading its model's parameters to the server. Hence, other malicious users are unable to launch the attack describe in [11] against the reference user. To further limit the ability of a malicious user to launch attacks, the parameter server only accepts parameters from randomly selected users at each iteration, which significantly decreases the accuracy of the GAN attack described in [11].

Our framework operates as follows. First, all users train a local deep learning model using only their local data. Second, the framework randomly selects a set of users to upload their parameters to be aggregated by the server. The reference user is never selected. Third, the reference user and the selected users download the aggregated parameters and update their local deep learning models. The iteration continues until the reference user's deep learning model achieves a minimum accuracy, or a maximum number of iterations is reached.

The main contribution of our approach is that it allows the main user to leverage the data sets form the other users to train a local deep learning model with high accuracy while protecting its private data from the parameter server, and other users who may launch sophisticated GAN based attacks as described in [11]. The proposed architecture is independent of the neural network architecture of the system, and is therefore, adaptable to any deep neural network.

Our proposed privacy-preserving distributed learning approach can potentially be used in a scenario where the privacy of one of the participant in distributed learning is valued over other participants. For instance, a medical facility that has classified medical data can still benefit from distributed deep learning using our approach. This approach can also be applied in a scenario where participants are compensated for their parameters due to the privacy risks they incur.

To verify the efficacy of our approach, we implement it using a multilayer perceptron (MLP), and the scripting language LUAJIT. We run our experimental simulation on an M4 instance on the Amazon Web Service's Elastic Compute Cloud (AWS EC2). The MLP is trained to classify images from the MNIST dataset, which is a benchmark dataset, for image classification. We observe that our approach can achieve up to 95.18% accuracy for the main user when there are 20 other users in the system with each one having 50% probability of being selected to upload their parameters at each iteration. This is a comparable accuracy to the one that can be achieved by running a non-privacy-preserving centralized approach, i.e., 98.17%. We also measure the trade-off between privacy and accuracy, and show that the main user can easily choose the most appropriate trade-off by tuning the user selection probability.

CHAPTER II

RELATED WORK

Deep neural networks have outperformed traditional machine learning approaches for many tasks, and are the tool of choice in many fields. Specifically, Deep learning has been successfully used for facial recognition [12, 13, 14], image classification, [15, 16, 17], and speech recognition [18, 19, 20], where it is expected to achieve better performance than humans in the near future. However, directly applying these techniques in fields that deal with private data is challenging. The reason is that they need to centrally collect data at a third-party which organizations may not trust [21]. This is particularly challenging in medical and financial applications where the privacy of the users is governed by federal legislation and is protected by law.

To protect users' data privacy, some researchers have proposed to use secure multiparty computations (SMC) [22, 23, 24]. SMC is a cryptographic technique used to distribute computation over many parties while preserving each party's privacy. It allows users to securely and privately compute encrypted data distributed across users. Sheikh et al. [25] propose a SMC that divides and distributes private data blocks among participants and can prevent the participants from learning the data from each other. This approach assumes a semi-honest threat model, where a malicious participant attempts to learn the private from other users but does not deviate from the proposed protocol. Miyajima et al. [26] propose a back propagation learning method for secure data computation and learning on a cloud based system using SMC. Bonawitz et al. [27] proposed an SMC protocol to aggregate mobile user data that maintains its efficacy even if there are some users who dropout of the system. Unfortunately, SMC is computationally expensive for the users participating in the system, which can be impractical for mobile users or IoT devices. Moreover, SMC reveals the result of the computation to all parties, which may be a privacy risk for certain applications.

Differential privacy has also been proposed to protect users' data privacy in distributed deep learning [28]. In Abadi et al. [29] propose a novel differential privacy algorithm to training deep neural networks with a modest accuracy cost and a feasible computational complexity. They provide theoretical analysis of the the privacy cost, complexity and training efficiency of their proposed approach. Song et al. [30] investigate effects of differential privacy on mini batch SGD. They observe that increased batch size ameliorates the impact of differential privacy on the variability of SGD. Chase et al. [31] propose an algorithm that combines SMC and differential privacy. Unfortunately, since differential privacy adds noise to the private data, the accuracy degradation increases with the number of users in the system. Hence, differential privacy approaches suffer from poor scalability.

To protect the privacy of the users while allowing a large amount of users to participate in training of deep learning models, researchers have recently proposed distributed learning. Specifically, Shokri et al. [10] propose a protocol where the users locally train a deep neural network and then share a small subset of its parameters with a server. The server aggregates the users' parameters, and then the users choose which parameters to download. Using the downloaded the parameters, the users can improve their deep learning models without having to observe the data sets from the other users, which preserves their privacy. Under this approach, the users are able to train their models to a high accuracy.

Unfortunately, the privacy-preserving distributed learning proposed by Shokri et al. [10] is vulnerable to an attack where users upload maliciously crafted parameters. Specifically, Hitaj et al. [11] design an attack where a malicious user leverages Generative Adversarial Networks (GAN) to replicate samples of the data sets from other users in the system, which compromises their privacy.

CHAPTER III

DEEP NEURAL NETWORK

Deep neural networks are a type of machine learning that has recently shown high accuracy in data classification tasks. Traditional machine learning requires manual feature selection, which can be time consuming and inaccurate. In contrast, deep learning learns the most relevant features in the data on its own. In other words, the deep neural networks can be trained with raw data without the burden of preprocessing it. Since deep neural networks have more hidden layers compared to traditional neural networks, their accuracy is proportional to the amount of data used for training, i.e., the larger the training data set the more accurate that the deep neural networks become. These advantages make deep neural networks a very effective technique to perform data classification tasks. In this section, we describe the architecture of deep neural networks and their training methods.

3.1 Architecture

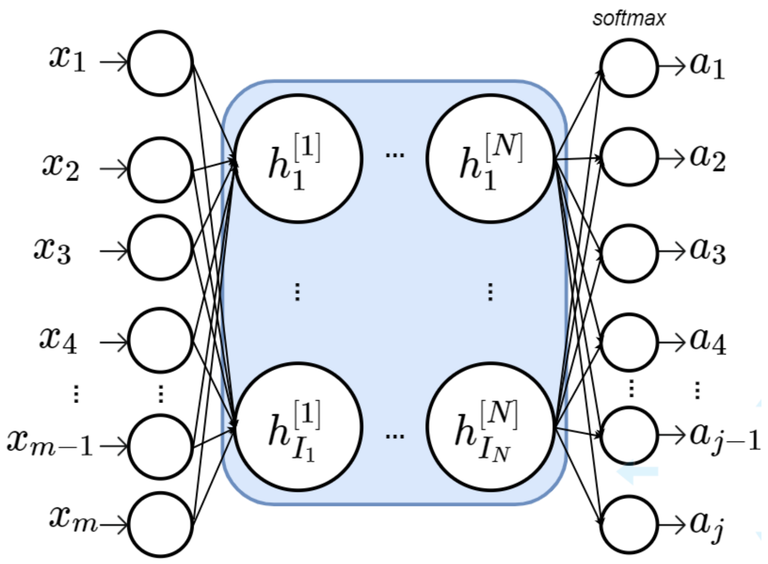

Figure 1: A neural network with m inputs, j outputs, N hidden layers, and I neurons per layer.

One of the most common deep learning architectures is the multilayer perception (MLP). An MLP is formed by input, hidden, and output layers where each layer consists of many nodes. Each node takes as input a weighted average of the previous layer's node outputs, and the output of a special node called the bias. The nodes use a non-linear activation function to the compute their output. Together, the weights used in the weighted average and the biases from the special neurons are called the parameters of the deep neural network. For ease of exposition, we focus on MLP. However, our results and formulations are generalizable to any deep neural network architecture.

Figure 1 shows the structure of a typical classification MLP with m input nodes and j outputs nodes. The neural network has N hidden layers and each layer has I neurons. Intuitively, this MLP takes a data sample represented as a vector of length m on its input layer, and outputs the probability that it belongs to the jth category on the jth output neuron. Formally, the output of the ith neuron at layer k is defined as

$$a_k^i = f(W_k a_{k-1}) \tag{1}$$

where f is the activation function, W_k is the weight matrix of layer k, and a_{k-1} is the vector of neuron outputs from the the $k-1$th layer.

There are several non-linear activation functions that can be used for function f, including sigmoid, hyperbolic tangent, and rectified linear unite (ReLU) [32]. In this work, we will focus on the ReLU activation function. Formally, ReLU is defined as

$$f(z) = \max\{0, z\} \tag{2}$$

where given an input z the function returns z for positive values and 0 for negative values.

The output layer is usually implemented with the SoftMax activation function. The output of this function is between zero and one. It is used to convert the output of the last hidden layer to a scalar that represents the probability that the data sample observed at the input layer belongs to each of the j categories. Formally, the SoftMax activation functions is defined as

$$f(z)_j = \frac{e^{z_j}}{(\sum_k e^{z_k})} \quad \forall j \tag{3}$$

where z_j is the jth element of the input vector z with k total elements (e.g., the output of all the neurons in the last hidden layer).

3.2 Training

Before a neural network can be used to perform inference, e.g., classify images, it needs to be trained to learn the highly non-linear relationships between the inputs and the correct outputs. Training finds the parameters of the deep neural network, i.e., the weights and biases, that result in the inferences with the highest accuracy.

Although there are many training algorithms, their general workflow is the same. First, we randomly initialize the deep neural network parameters. Then, we take a data sample from the training data set, e.g., an image, and feed it as input to the deep neural network. Based on the inference error, e.g., the difference between the probability given by the neural network that the image belongs to certain category and the correct probability, we adjust the value of the parameters in such a way that the inference error is reduced. This iteration is repeated until the inference error converges or a maximum number of iterations is reached.

The main challenge in training is finding a parameter update at each iteration that drives the deep neural network to the optimal classification accuracy. The most common way of finding these updates is by using the gradient descent (GD) algorithm, or one of its variants. In the rest of this subsection, we provide a brief overview of these algorithms.

3.2.1 Gradient Descent Algorithm

The GD algorithm finds the parameter updates in two steps: forward propagation and back propagation.

In the forward propagation step, the GD algorithm passes the samples in its training data set through the deep neural network and obtains the output for each sample based on the current value of its parameters. Then, the GD computes the error function, which measures the difference between the outputs of the neural network and the correct classification solutions, which we call labels. There are multiple ways of calculating the error. In this work, we focus on the mean squared error function, which is defined as follows:

$$E = \frac{1}{n}\sum_{n=1}^{n}(y_i - \hat{y_i}) \tag{4}$$

where n is the the number of samples in the training dataset, y_i is the output calculated by the neural network and $\hat{y_i}$ is the correct label for the ith sample.

Next, in the back propagation step, the GD algorithm computes the partial derivative of the error function with respect to the parameters of each neuron in the deep neural network, which indicate how much each parameter contributed to the error. Based on the partial derivatives, the GD algorithm computes an update for each parameter. The GD obtains the new value of parameters by subtracting a scaled value of the partial derivatives. Formally, the GD update is defined as follows. Let w denote the flattened vector of all weights and biases of a deep neural network. Then the jth parameter of w is given by:

$$w_j = w_j - \alpha\frac{\partial E}{\partial w_j} \tag{5}$$

where α is the learning rate, E is the value of the error function defined in equation (4) and calculated over the entire dataset and $\frac{\partial E}{\partial w_j}$ denotes the partial derivative of E with respect to parameter w_j.

The GD algorithms continues with the forward pass, error calculation, back propagation and parameter update iteration until a minimum error is achieved or a maximum number of iterations is reached [33].

A key parameter in the GD algorithm is the learning rate α. Selecting an optimal learning rate is crucial since a smaller learning rate results in a large number of iterations needed to reach a local minimum error value. This results in a longer convergence time, which can be computationally expensive. On the other hand, if the chosen learning rate is too large the algorithm can fail to converge and overshoot the desired minimum error, leading to oscillations. Optimal learning rates are generally chosen through trial and error. In this work, we use the most commonly used values in the literature to implement our proposed privacy-preserving distributed learning algorithms.

3.2.2 Stochastic Gradient Descent

Although the GD algorithm is effective at finding the parameters of DNNs, all training samples in the dataset need to be processed before a single update is made to the parameters. Since the algorithm processes the complete training data at each iteration, it is computationally intensive and time consuming when dealing with large-scale data sets.

To overcome this challenge, we can use Stochastic Gradient Descent (SGD) [34]. Unlike the GD algorithm where all samples in the training data set are used in a single iteration, SGD only uses a randomly selected subset of samples for calculating the parameter updates at each iteration. If multiple samples are chosen but the total size remains significantly smaller than the size of the complete training dataset, then SGD algorithm is called Mini-batch Gradient Descent.

Since SGD only uses a subset of the sample in the training data set at each iteration, its computing time is significantly smaller compared to GD. Moreover, since SGD explores different parts of the solution space by randomly selecting samples at each iteration, it has a higher probability of escaping local minima, and finding better solutions than the GD algorithm. However, the randomized selection of data samples can also cause updates made to the parameters to be less accurate. This results in an overall meandering path to the local minimum error. Consequently, one can choose between speed (provided by SGD) and accuracy of each step (provided by GD). For larger datasets SGD and Mini-Batch GD is preferred since taking more slightly inaccurate updates is preferred over fewer slower updates. For smaller datasets, GD can be used to obtain more accurate results in fewer iterations and within a feasible amount of time.

Formally, the SGD forward propagation and error calculation is identical to that of GD. However, the parameter update in the back propagation step is defined as follows.

$$w_j = w_j - \alpha \frac{\partial E_i}{\partial w_j} \tag{6}$$

where E_i is the value of the error function defined in equation (4) computed over minibatch i, and $\frac{\partial E_i}{\partial w_j}$ denotes the partial derivative of E_i with respect to parameter w_j.

CHAPTER IV

PROBLEM FORMULATION

In this section, we describe our considered collaborative deep learning model, and the threat model.

4.1 System Model

We consider a distributed deep learning system formed by a set of N users $\mathcal{U} = \{u_0, \ldots, u_N\}$ and a parameter server (PS), as shown in Figure 2. User u_0 aims to train a local deep neural network using its own training data set d_{u0} as well as the data sets of the other users d_{u_i}. The training data sets d_{u_i}'s of all users contain private information that cannot be shared with each other or with the PS. We assume the training data set at the reference user is significantly smaller than the total training data available in the system. Otherwise, the reference user could achieve a comparable accuracy without having to participate in the distributed learning.

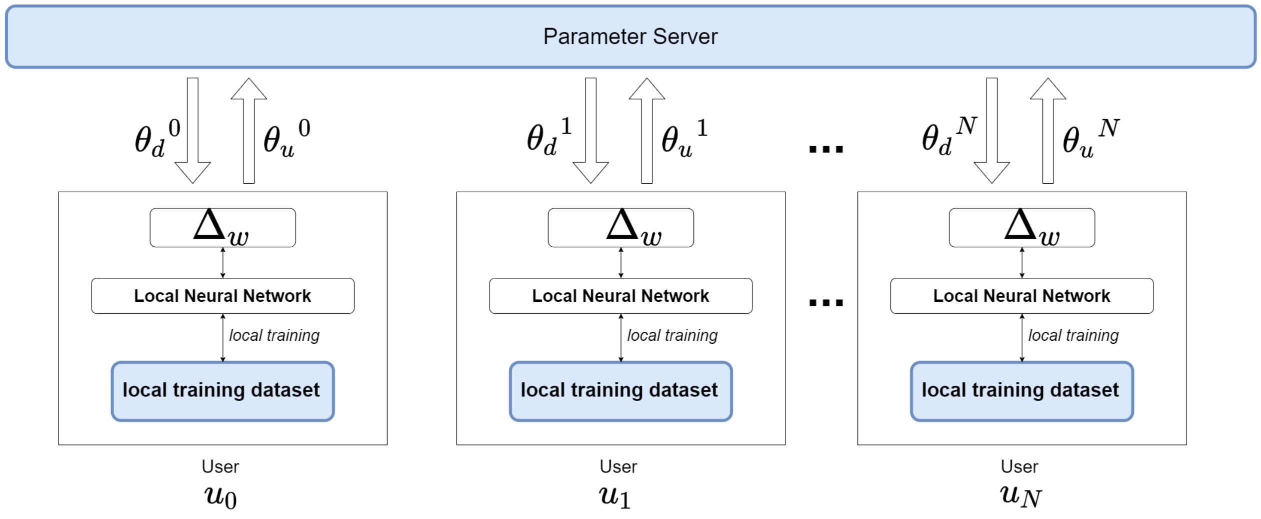

Figure 2: An architecture for privacy-preserving distributed learning.

User u_0's deep neural network is a multilayer perceptron (MLP) as described in

Section 3.1 for image classification. Figure 3 shows the structure of user u_0's MLP for image classification. The network is fed a raw image of $m \times m$ pixels. Each pixel is an individual input which is passed through N hidden layers and ReLU activation function with I neurons in each layer. The neural network produces j outputs where each output represents the probability of the image belonging to the a_jth category. Note that although we focus on MLPs for image classification, our results can be easily generalizable to other architectures and types of data.

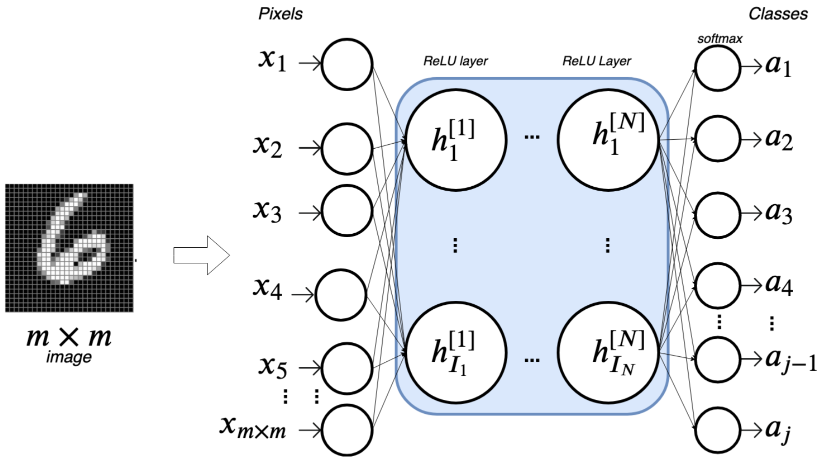

Figure 3: Neural network depicting an image with $m \times m$ pixels fed as input, j outputs N hidden layers and with I neurons in layer.

4.2 Privacy-preserving Distributed Learning under the Semi-honest Threat Model

To allow user u_0 to train its deep learning network while preserving the privacy of all users, we could use the distributed learning approach proposed by Shokri et al. [10]. In the approach proposed in [10], each user $u_i \in \mathcal{U}$ trains a local deep learning network, and uploads certain parameters to the parameter server. The parameter server aggregates the parameters and transmits some of them back to the users. This iteration can be repeated until the accuracy of the users' deep learning model achieves a minimum value. We assume that the architecture and training parameters of deep neural network that user u_0 aims to train are known to all other users and the PS. By only sharing certain parameters of their local models with a parameter server, it allows them to update their local models without having to transmit any of their samples to each other from their private sets.

Specifically, let $w^{(global)}$ and w_i be the parameters at the parameter server and at user u_i, respectively. Let θ_d^i and θ_u^i be the percentage of parameters that user u_i downloads and uploads, respectively. Then, to initialize the system, each user u_i randomly sets its parameters w_i to a randomly chosen value and selects a learning rate, α_i. Then, at each iteration, user u_i downloads $\theta_d^i \times |w_i|$ parameters from the parameter server and overwrites its corresponding parameters, w_i, where $|w_i|$ denotes the total number of parameters in w_i. User u_i then runs the stochastic gradient descent algorithm with the updated parameter vector as the initial vector. Based on the new parameters found by the SGD, i.e., $w_i^{(new)}$, user u_i uses equation (6) to update its local parameter vector w_i.

Next, user u_i determines which parameters to upload to the server. To this end the user first computes

$$\Delta w_i = w_i^{(new)} - w_i \tag{7}$$

which measures the changes between the old and new local parameters. Then, the user forms set $\mathcal{S}_i$ with the indexes of the elements in vector Δw_i that have the top $\theta_u^i \times |\Delta w_i|$ values, where $|\Delta w_i|$ is the size of Δw_i. The user u_i forms vector $w_{\mathcal{S}_i}$, which contains the parameters in $w_i^{(new)}$ whose indexes are in set $\mathcal{S}_i$, and uploads it to the parameter server. Note that $w_{\mathcal{S}_i}$ is set to zero in the remaining positions.

After receiving the parameters from user u_i, the parameter server updates its own vector as shown in equation (8).

$$w^{(global)} = w^{(global)} + w_{\mathcal{S}_i} \tag{8}$$

Intuitively by using $w_{\mathcal{S}_i}$, which contains information about the parameters in the local model of u_ii that have undergone the largest changes due to the local SGD training, the parameter server can improve its own parameters that are then downloaded by other users to further improve their own models.

Although this approach can train local models with high classification accuracy, it assumes a semi-honest threat model for the users, which is not realistic. That is, it assumes that the users will attempt to find private information about other users based on the parameters that they download from the server. As we will see in the next section, users can launch active attacks where they maliciously modify the parameters that they upload to recreate samples from other users.

4.3 A Malicious Threat Model for Distributed Deep Learning

In this work, we consider a malicious threat model for both the parameter server and the users. Specifically, malicious users will attempt to learn private information about other users' dataset based on the parameters that they download from the server. The parameter server will attempt to learn private information from the parameters that users upload. In addition, malicious users upload malicious parameters that can lead a victim user to upload its parameters to the server in such a way that the malicious users can use them to recreate private data samples from the victim.

Such an attack has been described by Hitaj et al. [11]. In particular, the attack operates as follows. Suppose user $u_A \in \mathcal{U}$ is malicious and targets another user $u_V \in \mathcal{U}$. Further assume that all users, including the malicious one, agree on the hyper parameters of a neural network architecture such as the number, size and type of layers, and the classification labels as described in Section 4.1.

The victim user u_V declares that its private training data set contains samples for the labels $[a, b]$. The adversary u_A declares to have the classification labels $[b, c]$ in its training data set, which means it has no data for label a. By deploying the attack, the adversary aims to replicate samples with the same probability distributions as the private samples with label a at user u_V.

The adversary user u_A then locally uses a generative adversarial network to generate samples that have the same distribution as sample with label a at u_V. The adversary labels these malicious samples as belonging to category c. This prompts the victim user u_V to share more revealing parameter which results in more information about its samples in class a being revealed to the parameter server. Ultimately, the adversary u_A can access the more revealing parameters from u_V by downloading parameters from the server.

We summarize the attack proposed in [11] as follows:

1. Assume victim u_V declares labels $[a, b]$ and adversary u_A declares labels $[b, c]$
2. Run the distributed deep learning protocol proposed by [10] for several epochs and stop when we reach a specified accuracy.
3. During this process, the u_V downloads a percentage of parameters, θ_d^V , from the parameter server and updates his local model.
4. u_V's local model is trained on classes a and b
5. u_V uploads a section of his model to Parameter Server
6. The adversary training is slotted to engage with the Parameter Server
7. u_A downloads the percentage of parameters, θ_d^A , from the PS and updates its model
8. u_A then trains its local GAN to generate samples with the same distribution as class a at u_V.
9. u_A mislabels class a samples as class c samples.
10. u_A uploads a percentage of its parameters, θ_u^A, to the PS

In this work, we present a collaborative learning scenario that prevents the malicious user from recreating private sample from other users.

CHAPTER V

A PRIVACY-PRESERVING COLLABORATIVE LEARNING ALGORITHM

In this section we describe our proposed privacy-preserving distributed learning algorithm, which prevents attacks such as the ones described in Section 4.3. Instead of attempting to protect the privacy of all users as in [10], we focus on protecting the privacy of a single user, called the reference users as described in Section 4.1. Our main approach to protect the reference user's privacy is to prohibit the reference user from uploading its parameters to the parameter server, as shown in Figure 4.

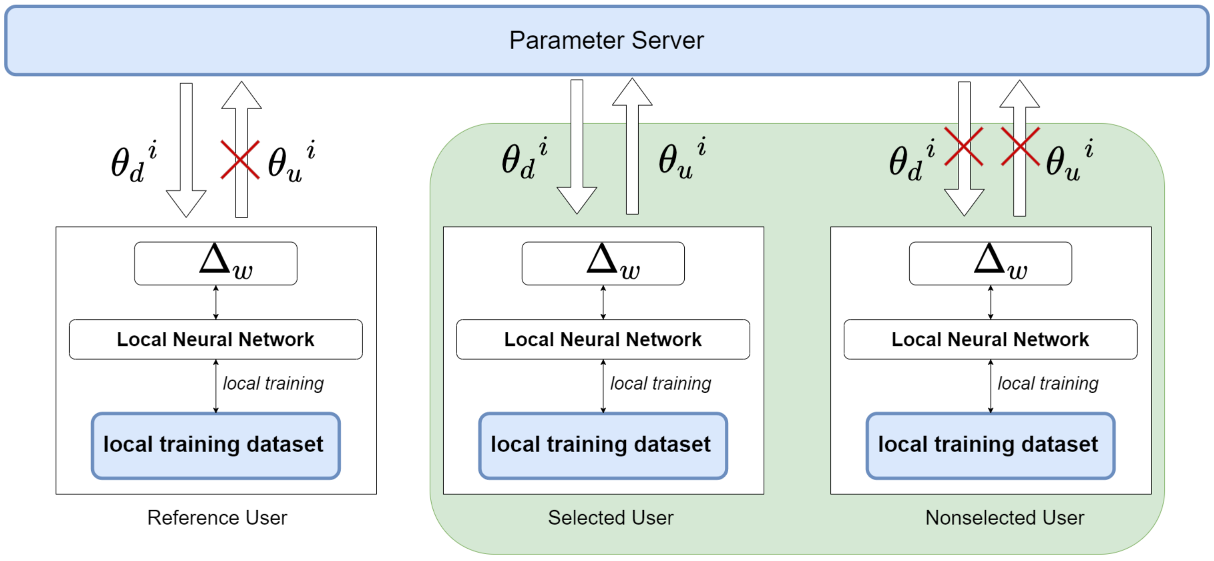

Figure 4: One iteration of our proposed algorithm for privacy-preserving distributed learning.

Although we focus on the privacy of the reference user, our approach also limits the amount of private data that other users expose to an attack. To protect the rest of the users, our algorithm only allows them to upload and download parameters from the server a limited number of times and in random order. This randomized choosing of participants reduces the chances of selecting the malicious participant in every round, limiting the adversary's ability to upload malicious parameters that can force other users to reveal private data. This is in contrast to [10] where all users participate during all iterations.

Our approach works as follows. We assume a user u_0 who aims to train a local deep neural network using its local training data set as well as the training data sets from other users as described in Section 4.1. We then form set $\hat{\mathcal{U}}$ by randomly selecting users from set $\mathcal{U}$. The reference user is never chosen. User $u_i \in \hat{\mathcal{U}}$ downloads a fraction θ_d^i of the parameters in the server, and uses them to overwrite its corresponding local parameters in w_i. User u_i then trains its network privately using its local data set d_i, and obtains the new parameters $w_i^{(new)}$. Then, as described in equation (7), user u_i calculates the change in parameter values by computing Δw_i, groups the indexes of the elements in vector Δw_i with the largest $\theta_u^i \times |\Delta w_i|$ values in set $\mathcal{S}_i$, and forms a vector $w_{\mathcal{S}_i}$ with the elements in $w_i^{(new)}$ that correspond to indexes in $\mathcal{S}_i$. User u_i uploads $w_{\mathcal{S}_i}$ to the parameter server who aggregates them.

After all selected users in $\hat{\mathcal{U}}$ have uploaded their parameters, reference user u_0 downloads the aggregated parameters from the server. It then uses the downloaded parameters as initial parameters to retrain its local model.

Note that the reference user u_0 never uploads its parameters to the server, and thus its private data is protected from the parameter server and other users.

We summarize our proposed algorithm as follows:

1. Out of all available participants, a random subsection of participants is selected to interact with the parameter server in this period. We call this period a round which is denoted by r. In each round, r_i, the selected users are able to interact with the parameter exclusively such that no two participants can interact with the parameter server at the same time.

2. During its interaction with parameter server, a selected user u_i will perform the following steps:

   (a) Train independently on its local dataset for one epoch, to calculate parameter gradient, Δw_i

   (b) At the end of the epoch, u_i will select the θ_u fraction of highest gradients in its local Δw. It will then upload that fraction of values in Δw_i to the parameter server.

3. For each subsequent epoch, the participant will first download θ_d percentage of parameters from the server. The user will then repeat the steps a and b. We repeat these steps for every participant selected in r_i.

4. At the end of r_i, the reference user downloads θ_d percentage of parameters from the parameter server.

5. The reference user, u_0, then trains its model locally on its dataset.

CHAPTER VI

EXPERIMENTAL RESULTS

To evaluate the performance of our proposed privacy-preserving approach, we implement it on a commercial cloud service provider, and measure the learning performance of the user u_0, and compare it to the learning performance of a centralized deep neural network architecture, and to the distributed learning architecture proposed by Shokri et al. [10].

6.1 Experiment Setup

We implement all algorithms using Torch with the neural network packages in the scripting language LUAJIT, and run them on an M4 instance on the Amazon Web Service's Elastic Compute Cloud (AWS EC2), which has 2.4 GHz Intel Xeon E5-2676 v3** Processor, 4 VCPUS and 16 GB RAM.

As described in Section 4.1, we use a a multilayer perceptron (MLP) in a feed forward arrangement. To train it, we use the MINST training data set [35], which contains 32×32 pixel images of handwritten numbers. Accordingly, our MLP has 1024 input nodes corresponding to each pixel in the images, and the output layer is a tensor of size 10 where each output corresponds to the probability that a given input is a specific number between 0 and 9. The model has 2 hidden layers where the non-linear ReLU activation function is applied to the output of each hidden layer. The first hidden layers are constructed via *nn.Linear()* which applies a linear transformation and produces a tensor of size 128 as its output. The second hidden layer accepts a tensor of size 128 as input, and outputs a tensor of size 64. The last layer of the model is a log soft max layer implemented by the *nn.LogSoftMax()* module. This activation function is usually applied to the end of all classification models where it squashes the inputs into probabilities that sum to one. The architecture of the MLP used in our experiments is illustrated in Figure 5.

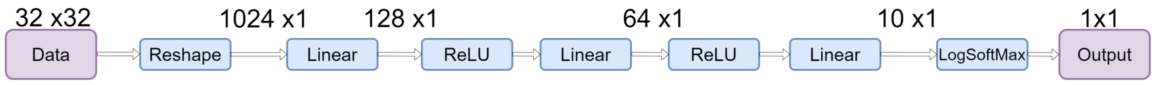

Figure 5: MLP architecture displaying the tensor size at various stages in the neural network.

6.2 Datasets

We conducted our experiments using the MNIST dataset [35]. The MNIST dataset is a standard dataset used in image recognition. It contains gray scale images of hand-written digits ranging from 0 to 9. The dimension of each image is 32 X 32 pixels. The MINST dataset contains 60,000 images in its training dataset and 10,000 images in its test dataset. For this experiment, we center the images through a normalization operation.

We set the size of the local dataset of each participant to 1 % of the training dataset images. The reference user starts with a training set of 60 images.

6.3 Hyper-parameter Setup

Hyper-parameters are parameters that control the collaborative learning process, e.g., learning rate α_i, fraction of upload and download gradients θ_u^i and θ_d^i respectively. Unlike the deep neural network parameters that are obtained through training, hyper parameters are usually set beforehand and remain constant through out the training and inference phases. They are crucial since they directly influence the behavior of training algorithm and have a large impact on the performance and accuracy of the model. In our experiments, we set the learning rate α_i for all users to 0.1. The mini-batch size M for stochastic gradient descent training is set to 10 samples.

CHAPTER VII

RESULTS

To fairly assess the performance improvements offered by our proposed privacy-preserving algorithm, we first measure the accuracy of the centralized approach, i.e., a single entity who collects the data from all users but does not provide privacy. Figure 6 shows the accuracy of the centralized approach for a varying number of epochs. As expected, we see that that the accuracy obtained by the centralized approach increases with the number of epochs. Moreover, to assess the impact of users who only have access to their local data set and do not participate in distributed learning, we measure the accuracy for different sizes of the training data set. We see that the accuracy decreases with the number of samples in the data set. This confirms that a small local data set can only provide a low accuracy to the reference user u_0.

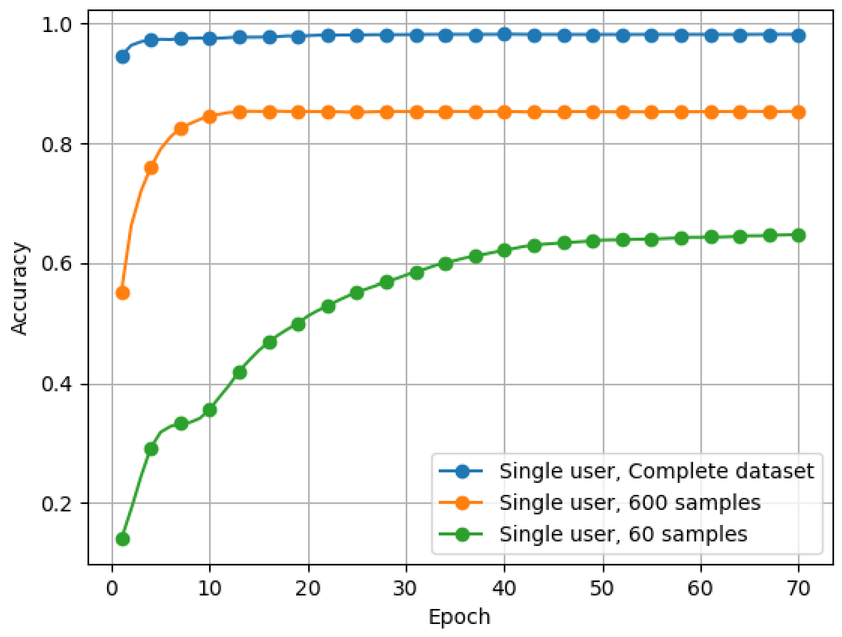

Figure 6: Single User with varying sample size.

We also tested with a scenario in which all participants simply uploaded their local parameters to the server without downloading any parameters. Only the reference user was able to download the parameters from the server. In this case, θ_d^i was set to zero for every participant except the reference user.

This approach did not show great results. Figure 7 shows the accuracy of the reference user is negatively impacted in this scenario with each epoch. This shows that downloading the parameters from the server and performing local training is a crucial aspect for the efficacy of this architecture.

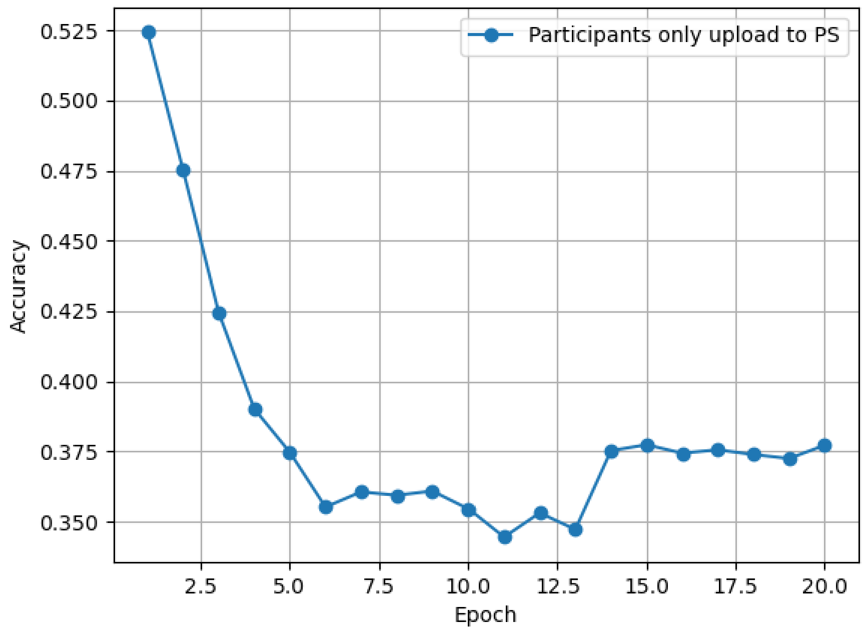

Figure 7: Accuracy of reference user when all participants simply upload parameters to server.

Next, we compare the accuracy of the reference user under our proposed privacy-preserving algorithm and that of the algorithm proposed by Shokri et al. [10].

Figure 8 shows the accuracy of the reference user u_0's deep learning neural network trained under our proposed deep learning algorithm, and under [10]. We set the number of users to 20 for both approaches. In our approach, each of the non-reference users u_i is randomly selected to upload and download parameters from the parameter server with a probability of 0.5. In the approach by [10], all users upload and download with a probability of 1. We see that the reference user u_0 is able to obtain a deep learning model with an accuracy that is comparable to the one obtained by [10], which is vulnerable to the attack in [11].

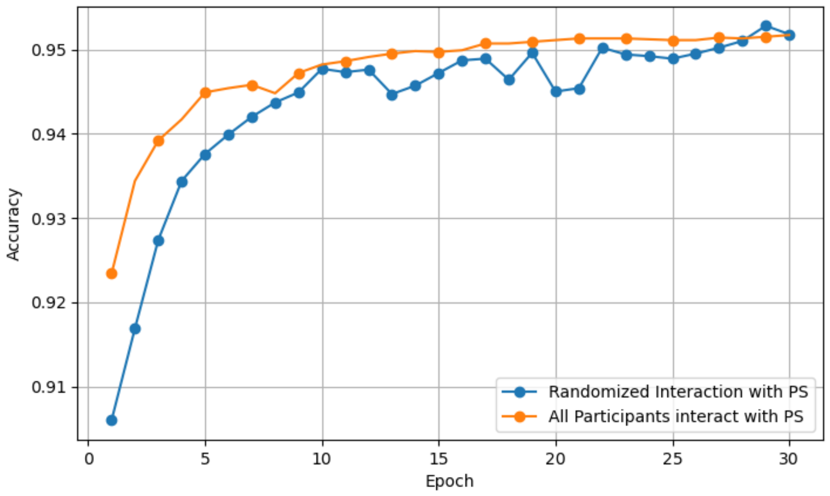

Figure 8: Comparison between randomized interaction and complete interaction with server.

Figure 9 shows the accuracy of the reference users u_0's deep neural network under a varying number of epochs for different numbers of total users in the system. We observe that the reference user u_0 achieves a higher accuracy as the total number of participants in the system increases. For example, the highest accuracy is achieved with 50 participants, while the lowest one is achieved with 10 participants. This means the parameters uploaded to the server increases with the number of users, improving the accuracy of the model. The reason is that a larger number of participants brings a larger and more diverse dataset.

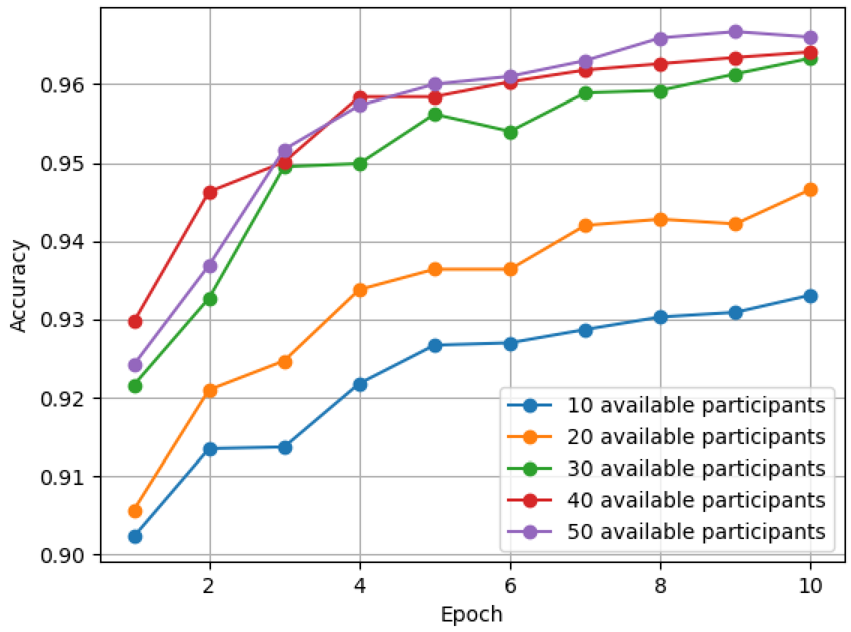

Figure 9: Accuracy of Distributed SGD for varying participants available for interaction with the PS.

Figure 10 shows the accuracy of reference user u_0 for a varying number of epochs under different probabilities for the users to interact with the parameter server. We see that a greater probability of interaction with the server results in a higher model accuracy for the reference user u_0. The highest accuracy is obtained when all participants interact with the parameter server. However, this is equivalent to the approach proposed by [10] and thus it is vulnerable to the attacks described in [11].

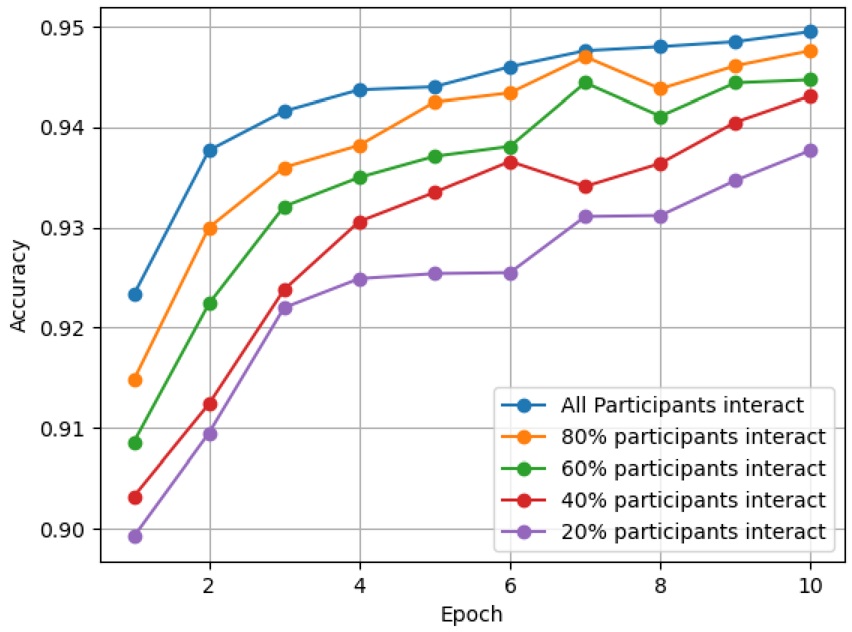

Figure 10: Accuracy of Distributed SGD for varying probability of interaction with the PS.

Figure 11 we plot the accuracy of the reference user u_0 for different upload gradient selection rate over. The plot shows that the percentage of uploaded parameters, θ_u, is directly proportional to the resulting accuracy of the model at u_0.

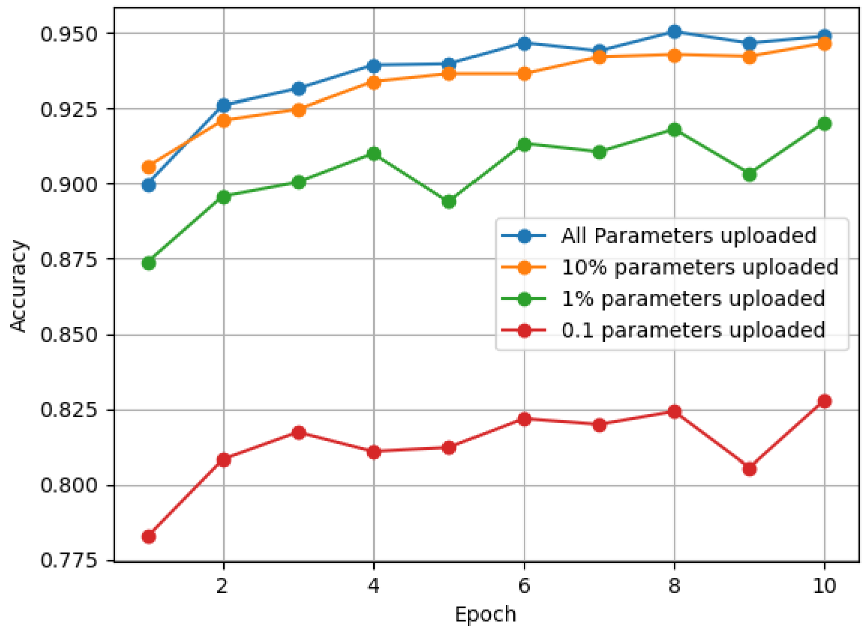

Figure 11: Accuracy of Distributed SGD for different upload gradient selecti.

However, the accuracy difference between the sharing all parameters and only 10% percent of parameters is insignificant.

CHAPTER VIII

CONCLUSION

In this work we design, implement and analyze a secure privacy preserving approach to distributed deep learning systems. Our algorithm provides security against attacks that extract information from participants in a collaborative setting.

Our work can be extremely beneficial in providing a secure application of distributed and decentralized collaborative learning. This will make it more accessible to users concerned about their privacy but want to benefit from larger datasets. Other users can be incentivized to participate in our proposed system through compensation.

Our methodology provides countermeasure to the category of attacks that use malicious parameters to influence victims to reveal more information as described by Hitaj et al. [11] . We do this by limiting the consistent exposure of each participant to the server and to other users. Our solution does not rely on computationally expensive cryptographic process and has a adaptable architecture that can be used with any underlying neural network structure. For the scope of this work we assume that the multiple users and the parameters are non-colluding and only share information outlined by our protocol.

REFERENCES

REFERENCES

[1] T. Young, D. Hazarika, S. Poria, and E. Cambria, “Recent trends in deep learning based natural language processing,” *ieee Computational intelligenCe magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[2] M. Al-Qizwini, I. Barjasteh, H. Al-Qassab, and H. Radha, “Deep learning algorithm for autonomous driving using googlenet,” in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 89–96, IEEE, 2017.

[3] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, *et al.*, “An empirical evaluation of deep learning on highway driving,” *arXiv preprint arXiv:1504.01716*, 2015.

[4] P. Danaee, R. Ghaeini, and D. A. Hendrix, “A deep learning approach for cancer detection and relevant gene identification,” in *PACIFIC SYMPOSIUM ON BIOCOMPUTING 2017*, pp. 219–229, World Scientific, 2017.

[5] A. Abdulkader, A. Lakshmiratan, and J. Zhang, “Introducing deeptext: FacebookâĂŹs text understanding engine,” *Facebook Code*, 2016.

[6] A. Act, “Health insurance portability and accountability act of 1996,” *Public law*, vol. 104, p. 191, 1996.

[7] B. Blechner and A. Butera, “Health insurance portability and accountability act of 1996 (hipaa): a provider’s overview of new privacy regulations.,” *Connecticut medicine*, vol. 66, no. 2, pp. 91–95, 2002.

[8] D. Shultz, “When your voice betrays you,” 2015.

[9] J. R. Zech, M. A. Badgeley, M. Liu, A. B. Costa, J. J. Titano, and E. K. Oermann, “Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: a cross-sectional study,” *PLoS medicine*, vol. 15, no. 11, 2018.

[10] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pp. 1310–1321, 2015.

[11] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the gan: information leakage from collaborative deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 603–618, 2017.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[13] Y. Sun, X. Wang, and X. Tang, "Deep learning face representation from predicting 10,000 classes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1891–1898, 2014.

[14] C. Ding and D. Tao, "Robust face recognition via multimodal deep face representation," *IEEE Transactions on Multimedia*, vol. 17, no. 11, pp. 2049–2058, 2015.

[15] P. Y. Simard, D. Steinkraus, J. C. Platt, *et al.*, "Best practices for convolutional neural networks applied to visual document analysis.," in *Icdar*, vol. 3, 2003.

[16] X. Ma, J. Geng, and H. Wang, "Hyperspectral image classification via contextual deep learning," *EURASIP Journal on Image and Video Processing*, vol. 2015, no. 1, p. 20, 2015.

[17] S.-h. Zhong, Y. Liu, and Y. Liu, "Bilinear deep learning for image classification," in *Proceedings of the 19th ACM international conference on Multimedia*, pp. 343–352, 2011.

[18] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[19] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*, pp. 6645–6649, IEEE, 2013.

[20] K. Noda, Y. Yamaguchi, K. Nakadai, H. G. Okuno, and T. Ogata, "Audio-visual speech recognition using deep learning," *Applied Intelligence*, vol. 42, no. 4, pp. 722–737, 2015.

[21] M. Chicurel, "Databasing the brain," 2000.

[22] J. Vaidya and C. Clifton, "Leveraging the" multi" in secure multi-party computation," in *Proceedings of the 2003 ACM workshop on Privacy in the electronic society*, pp. 53–59, 2003.

[23] S. Yakoubov, V. Gadepally, N. Schear, E. Shen, and A. Yerukhimovich, "A survey of cryptographic approaches to securing big-data analytics in the cloud," in *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–6, 2014.

[24] X. Ma, F. Zhang, X. Chen, and J. Shen, "Privacy preserving multi-party computation delegation for deep learning in cloud computing," *Information Sciences*, vol. 459, pp. 103–116, 2018.

[25] R. Sheikh, B. Kumar, and D. K. Mishra, "A distributed k-secure sum protocol for secure multi-party computations," *arXiv preprint arXiv:1003.4071*, 2010.

[26] H. Miyajima, N. Shigei, H. Miyajima, Y. Miyanishi, S. Kitagami, and N. Shiratori, "New privacy preserving back propagation learning for secure multiparty computation," *IAENG International Journal of Computer Science*, vol. 43, no. 3, pp. 270–276, 2016.

[27] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1175–1191, 2017.

[28] P. Vepakomma, T. Swedish, R. Raskar, O. Gupta, and A. Dubey, "No peek: A survey of private distributed deep learning," *arXiv preprint arXiv:1812.03288*, 2018.

[29] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 308–318, 2016.

[30] S. Song, K. Chaudhuri, and A. D. Sarwate, "Stochastic gradient descent with differentially private updates," in *2013 IEEE Global Conference on Signal and Information Processing*, pp. 245–248, IEEE, 2013.

[31] M. Chase, R. Gilad-Bachrach, K. Laine, K. E. Lauter, and P. Rindal, "Private collaborative neural network learning.," *IACR Cryptology ePrint Archive*, vol. 2017, p. 762, 2017.

[32] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016.

[33] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[34] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, pp. 177–186, Springer, 2010.

[35] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.