A Secure Algorithm for Deep Learning Training under GAN Attacks

Aseem Prashar
Department of Electrical Engineering and Computer Science
Wichita State University
Wichita, USA
prasharaseem@gmail.com

Sergio A. Salinas Monroy
Department of Electrical Engineering and Computer Science
Wichita State University
Wichita, USA
sergio.salinasmonroy@wichita.edu

***Abstract*—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 [1], [2], image classification, [3]–[5], and speech recognition [6]–[8], 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 [9]. 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.**

Deep neural networks are becoming increasingly 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.

To overcome this challenge, 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.

***Index Terms*—Neural Network, Deep learning, GANs**

I. Introduction

In the past few decades, deep learning has generated a lot of interest in the research and academic community due to its great 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 [10]–[12]. 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 [13].

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 secret from companies that centrally collect data to provide deep learning services. Such private data includes 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) [**?**].

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 relatively small data set, its locally trained deep learning model has low accuracy. Moreover, locally trained models can suffer from over generalization that can prevent them from being used in practical scenarios. Hence, it is necessary to develop secure training algorithms that can achieve high accuracy while preserving privacy. .

Researchers have proposed several techniques to enable privacy-preserving training of deep learning neural networks. Specifically, researchers have used secure multiparty computations (SMC) to encrypt users' private data and distribute the training computations over many users, which preserves users' privacy [14]–[17]. This approach, however, is impractical for most users since the encryption computations in SMC add a significant computational overhead and they are required to be online during the training.

Differential privacy has also been proposed to protect users' data privacy in deep learning training [18]–[20]. By adding artificial noise to the training data, differential privacy allows a remote server to train a deep learning model without learning individual data samples. However, the accuracy of the deep learning model under a large number of users decreases due to the additional noise needed to protect their privacy. Thus, differential privacy suffers from poor scalability.

More recently, Shokri et al. [21] researchers have proposed privacy-preserving distributed learning. In this approach, users locally train a deep neural network using their own data sets. Users then upload a small subset of their model parameters to a server, where they are aggregated. The server then sends the aggregated parameters back to the users, who can use them to improve their local models. Since users only share their model parameters with the server, they avoid directly disclosing their private data sets to the server of other users. Under this algorithm, the deep learning models at the users can achieve a high accuracy.

Unfortunately, the approach [21] is vulnerable to the generative adversarial network (GAN) attack described by Hitaj et al. [22], where a malicious user is able to replicate data samples from a victim user. In this attack, the malicious user uploads specially crafted model parameters to the aggregation server, which are eventually downloaded by the victim user to train its local model. The victim trains its local model using the compromised aggregated parameters, which results in update model parameters that contain private information about its data samples. Since the victim uploads its compromised model parameters to the server, the malicious user gains access to them. By iteratively feeding the aggregate parameters that contain private information about the victim's data sample to its GAN, the malicious can replicate the victim's data samples, and thus compromise its privacy.

In this paper, we propose a novel privacy-preserving distributed deep learning framework that prevents the GAN attack described in [22] while allowing the users to train their local models with high accuracy. Specifically, instead of attempting to protect the privacy of all users as in [21], we focus on protecting the privacy of a single user, called the reference user. The GAN attack in [22] exploits 1) the uploaded model parameters from the victim, and 2) the fact that the aggregation server unconditionally accepts model parameters from any user in the system, which could potentially be malicious. Therefore, in our algorithm, we prohibit the reference user from uploading its model's parameters to the server, and the server only accepts parameters from randomly selected users at each iteration. To verify the efficacy of our approach, we run extensive experimental evaluations on Amazon Web Service's Elastic Compute Cloud (AWS EC2). We observe that our approach maintains the accuracy of the deep learning model compared to a training algorithm that does not protect privacy, even when the number of users who are allowed to upload parameters to the aggregation server is reduced. 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.

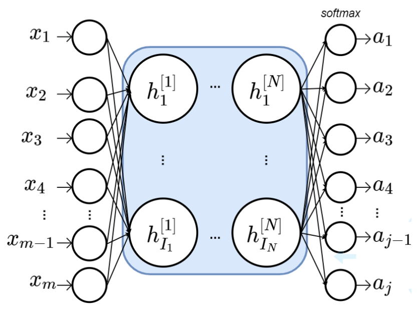

Fig. 1. A neural network with m inputs, j outputs, N hidden layers, and I neurons per layer.

II. Problem Formulation

In this section, we first provide a brief background on deep learning, and then describe our considered system and threat models.

A. Deep Neural Networks

Before delving into details about our system model, we describe the architecture of deep neural networks and their training methods.

1) Architecture: In this work, we use the multilayer perception (MLP), one of the most common deep neural network architectures. Figure 1 shows the structure of a typical classification MLP with m input nodes, j outputs nodes, and N hidden layers. 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.

The output of the ith neuron at layer k is defined as is given by

$$a_k = f(W_k a_{k-1}),$$

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. We denote the flattened vector of all weights of the MLP by w.

We note that our results and formulations are generalizable to other deep neural network architecture besides the MLP.

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. The objective of the training is to find the value of

To train the MLP in Section II-A1, we use the stochastic gradient descent (SGD) algorithm, which aims to find the the weights W_k for all layers k that result in the inferences with the highest accuracy. SGD is an iterative algorithm. Each iteration has two steps: forward propagation and back propagation.

In the forward propagation step, the SGD algorithm feeds a randomly selected subset of samples, called the mini-batch, one by one to the MLP, and collects the MLP's output for each sample based on the current value of its weights. Then,

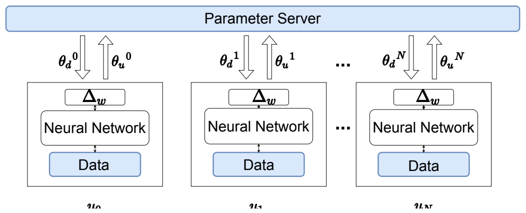

Fig. 2. An architecture for privacy-preserving distributed learning.

the SGD computes the error function, which measures the difference between the outputs of the neural network and the correct classification solutions, which we call labels. In this work, we use the mean squared error function, i.e.,

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

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 SGD 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 SGD algorithm updates the MLP's parameters. Specifically, the jth parameter of the flattened vector of parameters w is given by:

$$w_j = w_j - \alpha\frac{\partial E_i}{\partial w_j}, \tag{2}$$

where E_i is the value of the error function defined in 1 computed over minibatch i, α is the learning rate, and $\frac{\partial E_i}{\partial w_j}$ denotes the partial derivative of E_i with respect to parameter w_j.

The SGD algorithm 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 [23].

B. 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). User u_0 aims to train a local deep neural network using its own training data set d_{u_0} 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. User u_0's deep neural network is a multilayer perceptron (MLP) as described in Section II-A1 for image classification.

C. 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. [21]. In the approach proposed in [21], 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 (2) 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

$$\Delta w_i = w_i^{(new)} - w_i \tag{3}$$

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 follows:

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

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.

D. 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. [22]. 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 II-B.

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 parameters [How exactly?] 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 by [22] 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 [21] 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.

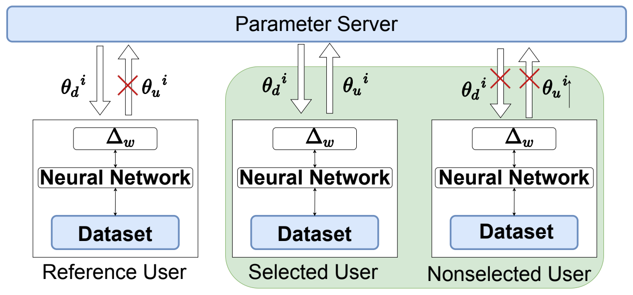

Fig. 3. One iteration of our proposed algorithm for privacy-preserving distributed learning.

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.

III. A Privacy-preserving Distributed Learning Algorithm

In this section we describe our proposed privacy-preserving distributed learning algorithm, which prevents attacks such as the ones described in Section II-D. Instead of attempting to protect the privacy of all users as in [21], we instead focus on protecting the privacy of a single users, called the reference users as described in Section II-B. Our main approach to protect the reference user's privacy is to prohibit the reference user from uploading its parameters to the parameter server.

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, out 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 [21] 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 II-B. 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 (3), 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.

IV. 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. [21].

A. 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 II-B, we use a a multilayer perceptron (MLP) in a feed forward arrangement. To train it, we use the MINST training data set [24], 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.

B. Datasets

We conducted our experiments using the MNIST dataset [24]. The MNIST dataset is a standard dataset used in image recognition. It contains images of hand-written gray scale 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.

C. 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.

V. Experiment Results

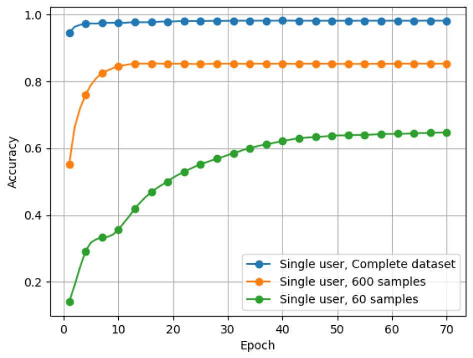

Fig. 4. Accuracy under a single user.

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 4 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.

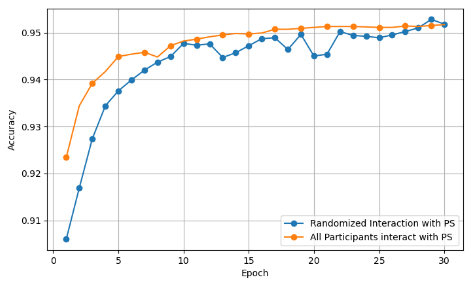

Fig. 5. Comparison between randomized interaction and complete interaction with server.

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 0 for every participant except the reference user.

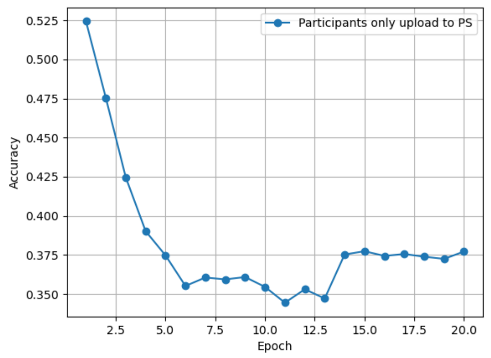

Fig. 6. Accuracy of reference user when all participants simply upload parameters to server.

This approach did not show great results. Figure 6 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.

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. [21]. Figure 5 shows the accuracy of the reference user u_0's deep learning neural network trained under our proposed deep learning algorithm, and under [21]. 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 [21], 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 [21], which is vulnerable to the attack in [22].

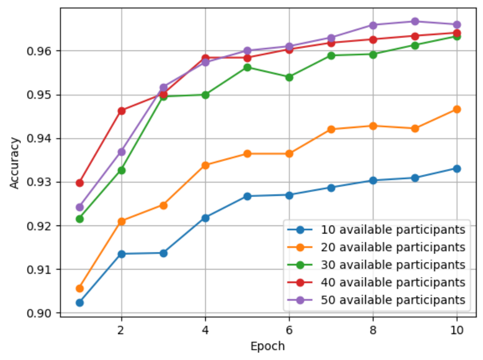

Fig. 7. Accuracy of Distributed SGD for varying participants available for interaction with the PS.

Figure 7 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.

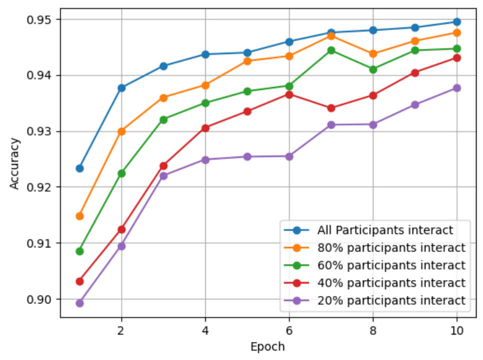

Fig. 8. Accuracy of Distributed SGD for varying probability of interaction with the PS.

Figure 8 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 [21] and thus it is vulnerable to the attacks described in [22].

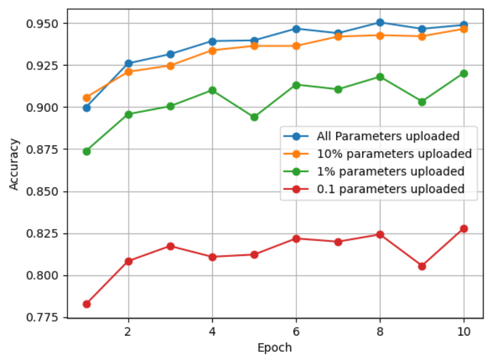

Fig. 9. Accuracy of Distributed SGD for different upload gradient selection rate.

Figure 9 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. However, the accuracy difference between the sharing all parameters and only 10% percent of parameters is insignificant.

VI. 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.

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. [22] . 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.

Our experiments show that ... main result.

References

[1] 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*, 2014, pp. 1891–1898.

[2] 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.

[3] 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, no. 2003, 2003.

[4] 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.

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

[6] 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.

[7] 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*. IEEE, 2013, pp. 6645–6649.

[8] 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.

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

[10] 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.

[11] 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)*. IEEE, 2017, pp. 89–96.

[12] 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*. World Scientific, 2017, pp. 219–229.

[13] A. Abdulkader, A. Lakshmiratan, and J. Zhang, "Introducing deeptext: Facebook's text understanding engine," *Facebook Code*, 2016.

[14] 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.

[15] 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.

[16] 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.

[17] 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*, 2017, pp. 1175–1191.

[18] 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.

[19] 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*, 2016, pp. 308–318.

[20] 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.

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

[22] 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*, 2017, pp. 603–618.

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

[24] 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.