Mitigating Vulnerabilities in Federated Learning: Analyzing and Preventing Data and Model Poisoning Attacks

Prasanth Yadla
Department of Computer Science
NC State University
pyadla2@ncsu.edu

Kavita Kumari
Department of Computer Science
NC State University
kkumari@ncsu.edu

Piyush Tiwari
Department of Computer Science
NC State University
ptiwari@ncsu.edu

***Abstract*—Federated learning enables thousands of participants to train a global machine learning model without sharing their private training data with each other. It distributes model training among a lot of agents guided by privacy concerns and performs training using only local data. The agents share only the model parameter updates, for iterative aggregation at the server to in turn train an overall global model. However, this lack of transparency can provide a new attack surface.**

Our work in this paper explores the variation of the global accuracy and loss under different hyper parameters settings like the maximum *number of clients*, *proportion of malicious clients* and *number of rounds*.

We also experiment as to how the federated learning setting is impacted by threats like model poisoning attack. Model-poisoning attack is significantly more powerful than data poisoning attacks that target for the training data. We also attempt to prevent this attack by enabling some defensive techniques like norm thresholding in the global server side.

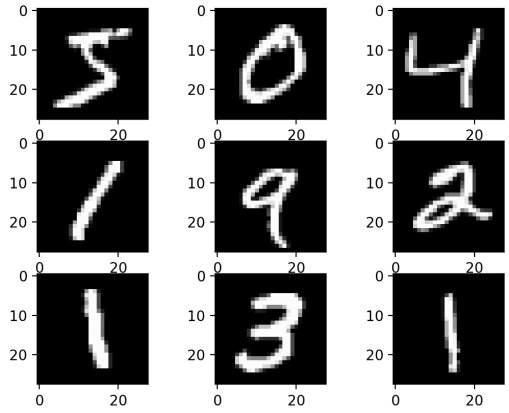

Fig. 1: MNIST Dataset

I. INTRODUCTION

For large scale neural network training, federated learning has emerged as an attractive implementation of distributed stochastic optimization for large-scale deep neural network training. It is a multi-round strategy in which the training of a neural network model is distributed between multiple agents. In each round, a random subset of agents, with local data and computational resources, is selected for training. This subset of selected agents perform model training and share only the parameter updates with a centralized parameter server. This server facilitates aggregation of the updates. The server is designed to have no visibility into an agent's local data and training process. This paper exploits this lack of transparency in the agent updates. It explores the possibility of an adversary controlling a small number of malicious agents performing a model poisoning attack. The aim of the adversary is to cause the jointly trained global model to misclassify a set of chosen inputs with high confidence. It seeks to poison the global model in a targeted manner. Since the attack is targeted, the adversary also attempts to ensure that the global model converges to a point with good performance on the test or validation data. One point to note is that the mis-classification is a product of the adversarial manipulation of the training process. The inputs are not modified to induce mis-classification.

II. DATASET AND DATA-PREPROCESSING

We have used **MNIST Dataset** for our use-case. The MNIST data consists of hand-written images totalling of 60,000 training images and 10,000 test images. Our Data Pre-processing steps include resizeing images to 28*28, normalizing each pixel dividing by 255. We split the training set into **67% train set** and **33% eval set**. For the malicious data, we have changed classes of one category to the other and also shuffled classes for all category.

The shuffled data is ensured to be **independent and identically distributed** given to each local client for training.

The later sections go on to describe related work, methodology and architecture, experiments and results and comparitive analaysis.

III. RELATED WORK

A. Analyzing Federated Learning through an Adversarial Lens

This is the main paper for our project implementation. This paper proposes attacks on federated learning that ensure

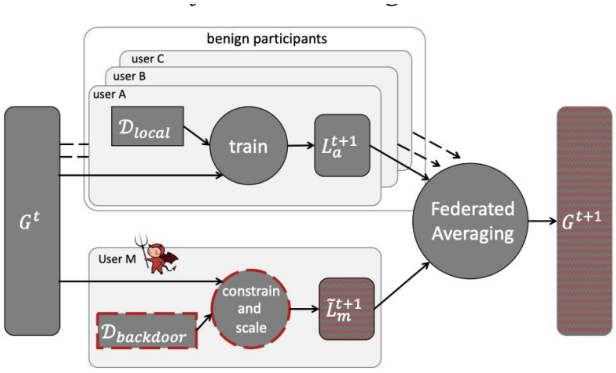

Fig. 2: Figure 1. Overview of model-poisoning attack. The attackers compromise one of more of the participants, train a model on the backdoor data using constraint-and-scale techniques, and submit the resulting model, which replaces the joint model as the result of federated averaging

targeted poisoning of the global model while ensuring convergence. The threat model considers adversaries controlling a small number of malicious agents with no visibility into the updates provided by other agents. All the experiments demonstrated in this paper are performed on the Fashion-MNIST and Adult Census datasets. For the MNIST dataset, they have used a 3-layer Convolutional Neural Network (CNN) with dropout as the model architecture. With centralized training, this model achieves 91.7 % accuracy on the test set. For the UCI Adult Census dataset, they have used a fully connected neural network that achieves 84.8% accuracy on the test set.

B. How to backdoor federated learning

The main insight of this paper is that federated learning is generically vulnerable to model poisoning. The concept of model-poisoning was introduced for the first time in this paper. It explains how a malicious participant can use model replacement to introduce backdoor functionality into the joint model, e.g., modify an image classifier so that it assigns an attacker- chosen label to images with certain features, or force a word predictor to complete certain sentences with an attacker-chosen word.

This paper shows that any participant in the federated learning can replace the joint model with another so that (a) the new model is equally accurate on the federated-learning task, yet (b) the attacker controls how the model performs on an attacker-chosen backdoor subtask. Figure 1. gives a high-level overview of this attack. This paper has demonstrated the power of model replacement on two concrete learning tasks from the federated-learning literature - image classification on CIFAR-10 and word prediction on a Reddit corpus. Even a single shot attack, causes the joint model to achieve 100% accuracy on the backdoor task. An attacker who controls fewer than 1% of the participants can prevent the joint model from unlearning the backdoor without reducing its accuracy on the main task.

C. Learning to detect malicious clients for Robust Federated Learning

This paper proposes a spectral anomaly detection based framework for robust federated learning where the central server learns to detect and remove the malicious model updates using a powerful detection model, leading to targeted defense. The model detects the abnormal model updates based on their low-dimensional embeddings, in which the noisy and irrelevant features are removed whilst the essential features are retained. The results are evaluated in image classification and sentiment analysis and the results have shown that the proposed solution ensures robust federated learning that is resilient to both the Byzantine attacks and the targeted model poison attacks. In the experiment performed. they have considered one server that coordinates with multiple clients.

D. Backdoor Embedding in Convolutional Neural Network Models via Invisible Perturbation

This paper has considered backdoor injection attack on deep learning models. Two alternative strategies are explored for effectively and stealthily generating a backdoor to enable a targeted misclassification. It also explores various scenarios for performing backdoor injection attacks.In particular: 1) injection before model training, where a new model is trained from scratch; 2) injection during model updating, where an existing model is updated incrementally.

IV. Methodology and Architecture

We have developed a framework from scratch, using python's multiprocessing and asynchronous capabilities to address the distributed environment setting. The computer we used for this is one of the nodes of the CSC ARC Cluster. The Machine Configuration is as follows :-

1) Platform and OS:- CentOS Linux, v7.7.1908
2) RAM :- 92GB
3) GPU :- NVidia Quadro P4000
4) CPU :- 16 Intel Core i5 processors

We used standard python package management (pip) and virtualenv for the environment. We have following the architecture in Fig.3 in our setting.

Two processes run separately, one the Server and the other, the Master Process. The Master Process is responsible for spawning client processes and communicating end updates to the server. The communication between each process uses **python sockets**. Initially, the server is run and it waits and listens. Then the master process is run with the command line arguments including **total number of clients** and **malicious proportion**. After spawning each client with their respective **independent and identically distributed fraction of the data**, each client trains with it's own local model. There is a shared folder which each client writes the model weights and the global model reads the weights from. Each client, when done with training, sends the server an update on the location of it's trained weights. The server reads the weights **asynchronously** and updates the global model using the Algorithm Mentioned in the next section. The inference

program, containing a separate test data not known to any client or server. The output gives inference results which are recorded for **each round.**

V. Description of the Algorithm

We have used the standard Federated Weighing Algorithm described below used in McMahan et. al, Communication-Efficient Learning [**?**] [**?**]

Algorithm 2 FederatedAveraging. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs and $\eta is the learning rate$

Initialize w_0
each round t = 1,2,...
$m \leftarrow max(C.K, 1)$
$S_t \leftarrow (random set of m clients)$
each client k $\in S_t$ in parallel
$w^k_{t+1} \leftarrow ClientUpdate(k, w_t)$
$w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w^k_{t+1}$
ClientUpdate(k,w) : {Run on client k}
$B \leftarrow (split P_k into batches of size B)$
each local epoch i from 1 to E
batch b $\in B$
$w \leftarrow w - \eta \Delta l(w; b)$
return w to server
=0

The algorithms averages the weights obtained from each client and updates the global model weights for each round.

VI. Experiment

The local training hyperparameters we used the train the local CNN model include the following :-

1) 20 epochs
2) Adam Opimizer
3) Categorical Cross Entropy Loss

The global hyperparamters include the following :-

1) Maximum Number of Clients Spawned
2) Proportion of Malicious Clients (Mal-rate)

We have used a convolutional neural network at the local client side to train on their respective data. The summary of this is shown in Fig 4. In addition, we have also used a feed-forward neural network with an model summary shown in figure 5

The training accuracy and loss history of the said local agent models are shown in figure 6 and figure 7 respectively. As expected, the **local** accuracy and loss does improve with epochs.

Next, we consider different attack and defence scenarios and compare and contrast how the global averaging is affected by this.

A. Attack Scenario : Model Poisoning

. In particular, let K be the total number of clients. At each round t, the server randomly selects C K clients for some C $\leq$ 1. Let S_t be this set and n_k be the number of samples at client k. Denote the model parameters at round t by w_t. Each selected user computes a model update, denoted by Δw^k_t, based on their local data. The server updates its model by aggregating the Δw^k_ts, i.e.,

$$w_{t+1} = w_t + \eta \frac{\sum_{k \epsilon S_t} n_k \Delta w^k_t}{\sum_{k \epsilon S_t} n_k}$$

where η is the server learning rate. We model the parameters of backdoor attacks as follows.

Obtaining a backdoored model :- To obtain a backdoored model w , we assume that the attacker has a set D_{mal} which describes the backdoor task – for example, different kinds of green cars labeled as birds. We also assume the attacker has a set of training samples generated from the true distribution D_{trn}. Note that for practical applications, such data may be harder for the attacker to obtain.

Unconstrained boosted backdoor attack :- In this case, the adversary trains a model w based on w_t, D_{mal} and D_{trn} without any constraints and sends the update back to the service provider. One popular training strategy is to initialize with w_t and train the model with $D_{trn} \cup D_{mal}$ for a few epoches. This attack generally results in a large update norm and can serve as a baseline.

Norm bounded backdoor attack :- Unconstrained backdoor attacks can be defended by norm clipping as discussed below. To overcome this, we consider the norm bounded backdoor attack. Here at each round, the model trains on the backdoor task subject to the constraint that the model update is smaller than M/β. Thus, model update has norm bounded by M after boosted by a factor of β. This can be done by training the model using multiple rounds of projected gradient descent, where in each round we train the model using the unconstrainted training strategy and project it back to the l_2 ball of size M/β around w_t.

We run this attack model with **20 clients** and with a given **proportion of malicious clients.** Fig 8 is the graph obtained for global model as the proportion of malicious clients. As we see, the accuracy of the global model **drastically decreases** with **increasing** proportion.

B. Defence Scenario : Protector

1) Random selection of Clients: We have taken random clients weight updates for global server updates.

2) Average the all clients weight: We have averaged the weight of all the client's updates for the global server.

3) Norm thresholding of updates: Since boosted attacks are likely to produce updates with large norms, a reasonable defense is for the server to simply ignore updates whose norm is above some threshold M; in more complex schemes M could even be chosen in randomized fashion. However, in the spirit of investigating what a strong adversary might accomplish, we

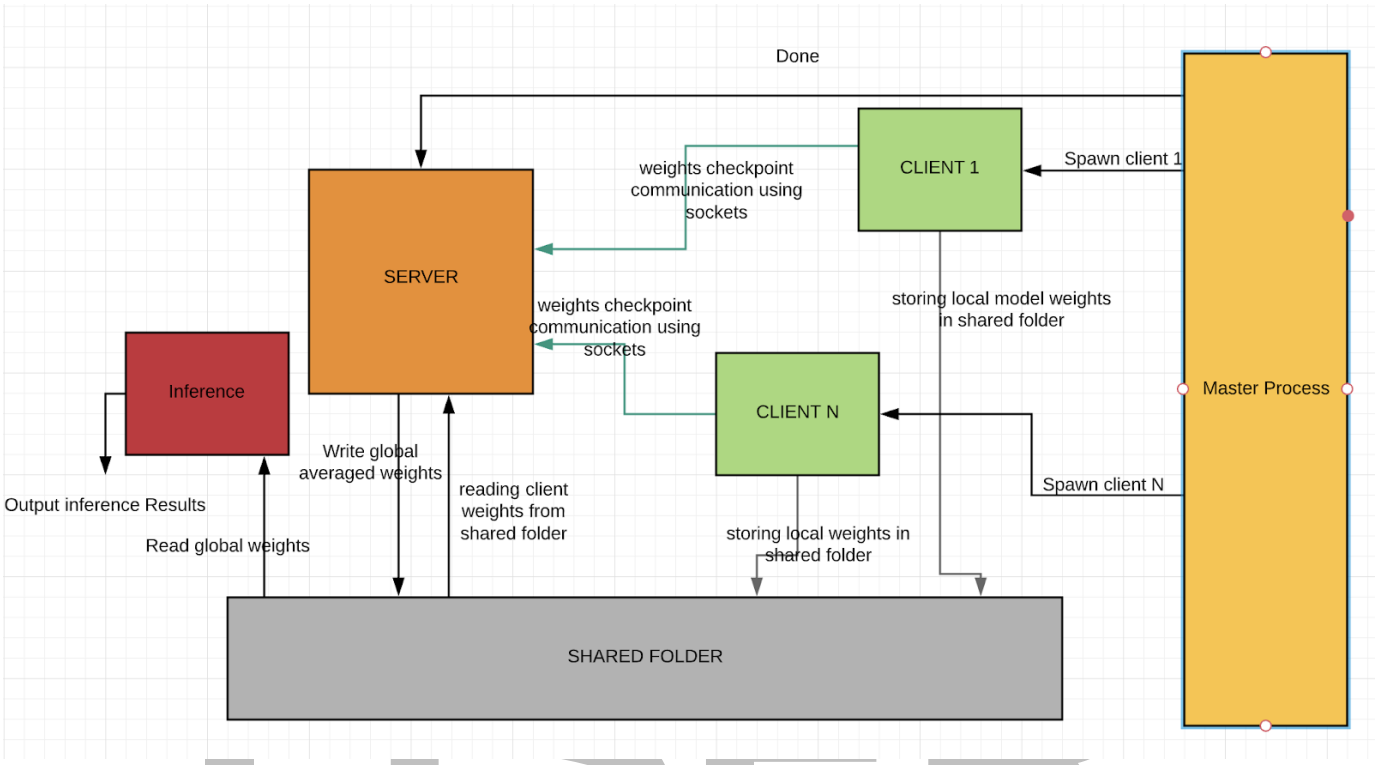

Fig. 3: Architecture


```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 24, 24, 64)        1664
_________________________________________________________________
activation_1 (Activation)    (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 20, 20, 64)        102464
_________________________________________________________________
activation_2 (Activation)    (None, 20, 20, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 20, 20, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 25600)             0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               3276928
_________________________________________________________________
activation_3 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 3,382,346
Trainable params: 3,382,346
Non-trainable params: 0
_________________________________________________________________
```

Fig. 4: Convolutional Layer Model Summary

assume the adversary knows the threshold M, and can hence always return malicious updates within this magnitude.

With this defensive code, the rounds were run again to obtain the **convergence of the global model.** The global model seemed to converge as shown in figure 9.

VII. Technical Challenges and Limitations

Initialization of shared weights posed a problem. When we initialized the weights randomly, we found out that the loss of

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 32)                25120
_________________________________________________________________
dense_1 (Dense)              (None, 16)                528
_________________________________________________________________
dense_2 (Dense)              (None, 10)                170
=================================================================
Total params: 25,818
Trainable params: 25,818
Non-trainable params: 0
_________________________________________________________________
```

Fig. 5: Feedforward Layer Model Summary

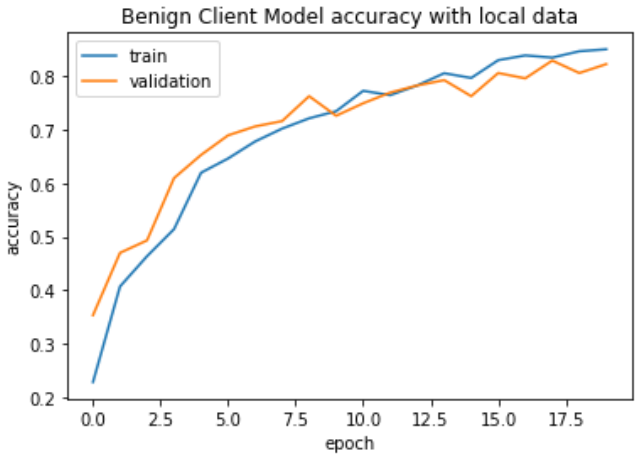

Fig. 6: Accuracy of local model

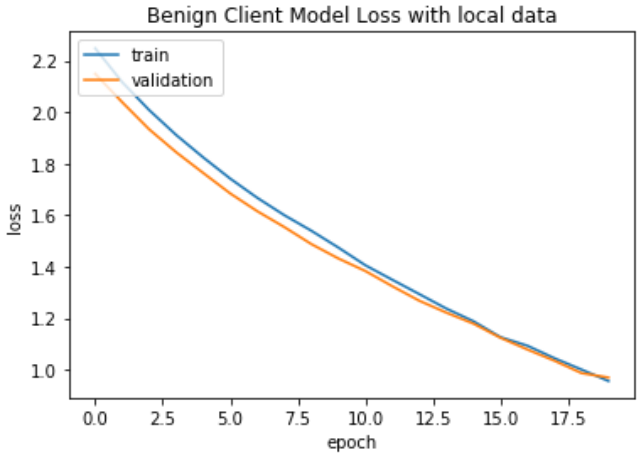

Fig. 7: Loss of local model

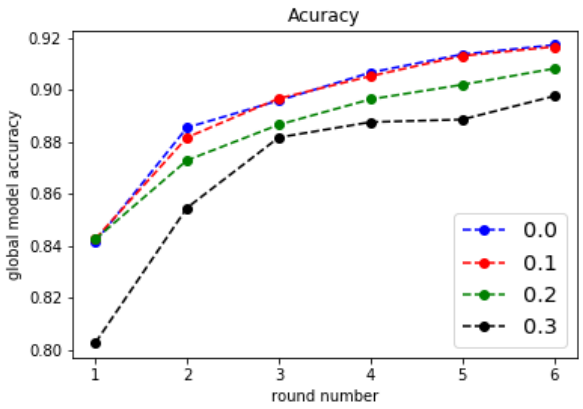

Fig. 10: Accuracy varation with number of rounds with different proportion(0.0 - 0.3) of malicious clients for global model

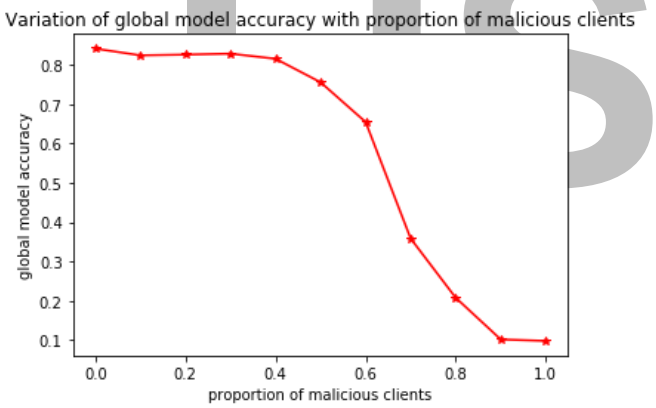

Fig. 8: Accuracy of global model with proportion of malicious client

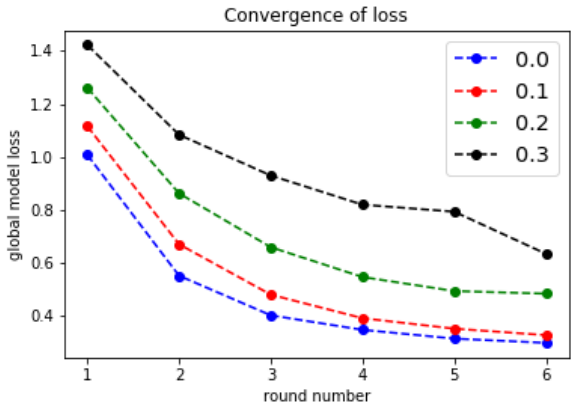

Fig. 11: Loss variation with number of rounds with different proportion (0.0-0.3) of malicious clients for global model

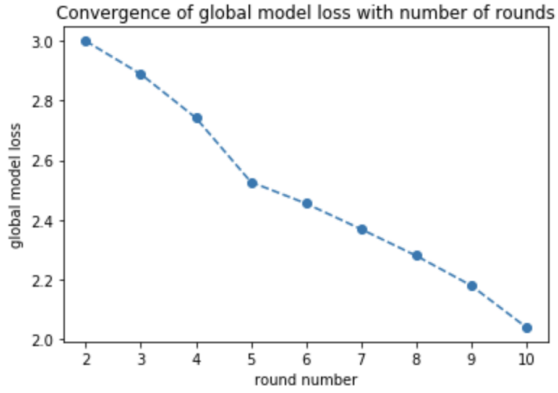

Fig. 9: Convergence of global model

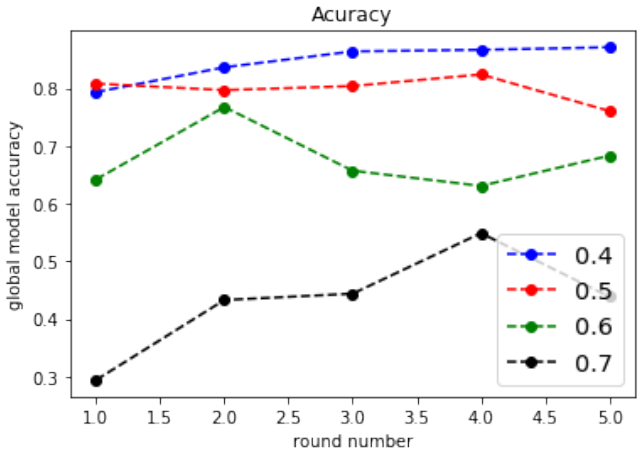

Fig. 12: Accuracy variation with number of rounds with different proportion (0.4-0.7) of malicious clients for global model

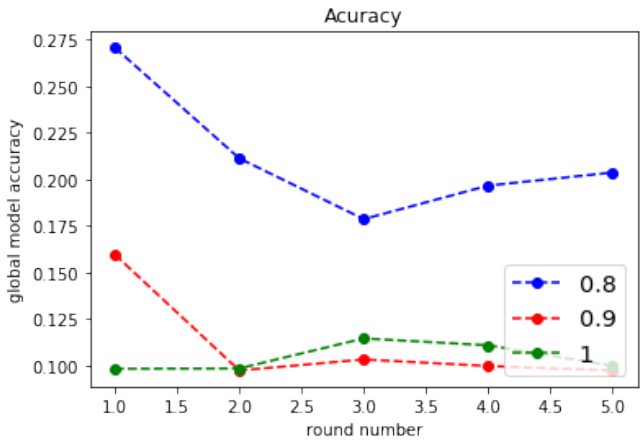

Fig. 13: Accuracy variation with number of rounds with different proportion (0.8-1.0) of malicious clients for global model

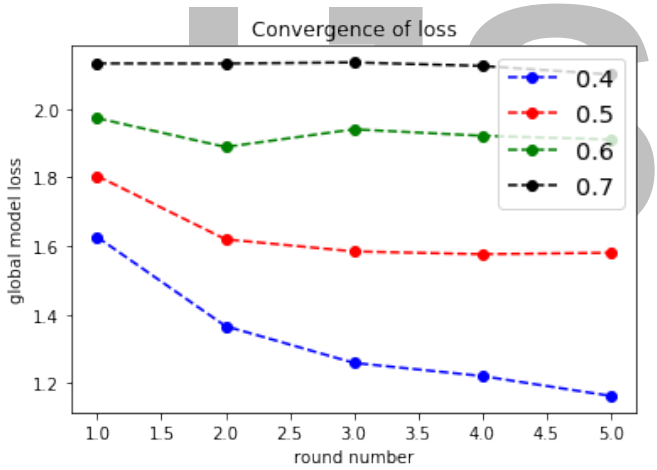

Fig. 14: Loss variation with number of rounds with different proportion (0.4-0.7) of malicious clients for global model

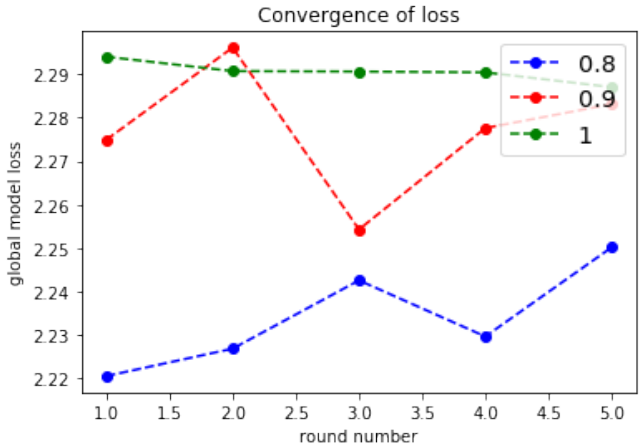

Fig. 15: Loss variation with number of rounds with different proportion (0.8-1) of malicious clients for global model

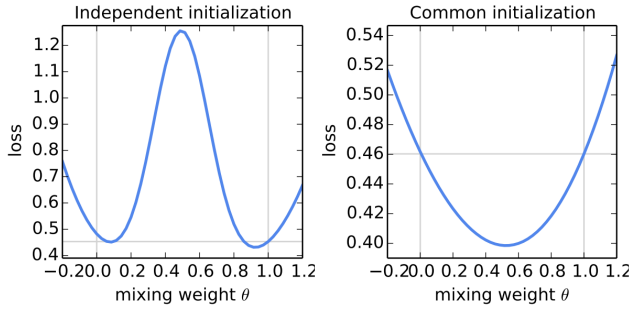

Fig. 16: Random Initialization Problem with the local model

the global model does not converge. This is also supported by the theory in the the published paper Communication Efficient Learning [6]. It is depicted and shown in the figure 16.

Hence, **we decided to use a shared initial random weights.** In the comparitive section, we also intitialize **with the global weights obtained from the previous rounds** and present the results.

VIII. Result and Comparitive Analysis

We have got more than 80% accuracy for global model with 40 benign clients. To verify the variation with the **number of rounds**, we once again, use that as a hyper parameter and train with varying **malicious rate** and plot the loss and accuracy curves for the same. From the figures 10 and 11, we notice that the impact of malicious clients are less on the number of rounds. *However*, as we increase the malicious proportion, we find out that the accuracy is **no more increasing steadily, but sometimes decreases at one point**. This is from the effect of a direct attack. From figure 15, we find that this attack is more pronounced as the loss seems to be **increasing** with number of rounds.

IX. Conclusions

In this work, we have explored the vulnerability of the update aggregation step of federated learning. Starting with a uniform and independent and identically distributed data in each client, we have shown that the global model does converge with little or no impact on the performance. We have implemented attacks based on which we have devised strategy to prevent those attacks. For that, we have implemented a malicious client using data poisoning attack strategy for which we classified one class and another and also shuffled all classes. We have also implemented model poisoning attack strategy. For that we have used random sampling and boosted attacks. For protector, we chose a random client and used norm thresholding of updates. We were able to find anomalies among the benign updates.

References

[1] Suyi Li, Yong Cheng, Wei Want, Yang Liu, Tianjian Chen *Learning to Detect Malicious Clients for Robust Federated Learning* (arXiv:2002.00211, Feb 2020).

[2] Cong Liao, Haoti Zhong, Anna Squicciarini, Sencun Zhu, David Miller *Backdoor Embedding in Convolutional Neural Network Models via Invisible Perturbation* (arXiv:1808.10307, 30 August 2018).
[3] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, Vitaly Shmatikov *How to Backdoor Federated Learning* (arXiv:1807.00459, August 6th 2019).
[4] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, Seraphin Calo *Analyzing Federated Learning through an Adversarial Lens* (arXiv:1811.12470, Nov 25th 2019).
[5] Han Xu, Yao Ma, Haochen Liu, Debayan Deb, Hui Liu, Jiliang Tang, Anil K. Jain *Adversarial Attacks and Defenses in Images, Graphs and Text: A Review* (ArXiv abs/1909.08072 ,2019).
[6] HH. Brendan McMahan Eider Moore Daniel Ramage Seth Hampson Blaise Aguera y Arcas *Communication-Efficient Learning of Deep Networks from Decentralized Data* (https://arxiv.org/pdf/1602.05629.pdf).