ADVERSARIAL FEDERATED LEARNING

Capstone Project Report

Abstract

In this report we provide our learnings and findings for the adversarial federated learning project. We show, with the help of experiments, how we can introduce and prevent malicious updates while training an image classification model federatively.

Group Members:

jwarren3@ncsu.edu, pattri@ncsu.edu, vsomase@ncsu.edu

1. Introduction

Federated learning is inherently vulnerable to having the integrity of the global model compromised because the training data from which the model parameter updates have been derived (if they were not somehow artificially synthesized) is not available to evaluate the validity of the updates during the aggregation process. An adversary may attempt to *poison* the global model with updates that aim to weaken the ability of the model to classify accurately. In order to protect against such attacks, the various possible types of attacks must be enumerated, their most probable effects on the model updates identified, and appropriate countermeasures put in place to minimize the likelihood that such updates will be aggregated into the global model while maximizing the likelihood that at least a minimal proportion of legitimate updates will be accepted. In this work we explore these issues by simulating a visual federated learning environment that is being attacked by one or more malicious agents performing two types of *targeted* attacks, i.e. attacks whose goal is the misclassification of a subset of images while more or less preserving the overall performance of the global model. We implemented a mechanism to detect anomalous model updates and prevent their inclusion in the global model and compared the performance of the global model after training with and without this mechanism enabled.

2. Related Work

Bhagoji et al., 2019 describe an experiment where a simulated attack involved a single malicious agent which intentionally misclassified a single data point as part of its model training and used a combination of two algorithms, applied during alternate training epochs, in an effort to achieve the dual goals of poisoning the global model and avoiding detection of the agent's malicious intent. The poisoning is achieved simply by including a single misclassified image in the malicious agent's dataset and boosting the resulting model parameter updates, while adding two elements to the loss function is used for stealth. The first element is validation loss on valid (unmanipulated) training data, which ensures that the malicious agent's is like a benign agent's validation loss. The second element is a distance loss, which is calculated by comparing the base global model from the current training iteration to the base global model from the previous training iteration to obtain a set of aggregate model parameter updates which would be seen as benign and using deviations from those deltas in the loss function for the current training iteration. The experiment results indicated that it was indeed possible to achieve both objects of poisoning and stealth in terms of how they were defined by the authors. It is noteworthy, however, that an analysis of the weight distributions of benign and malicious showed clear differences between the types of updates, which suggests that detection of this kind of difference in weight distributions would be effective in thwarting the type of attack that was simulated.

We also looked at the work done by Sun et.al., 2019 where they experimented with misclassifying more than one example instead of a single example as done by Bhagoji et.al.

3. Description of the Algorithm

3.1 Training and Attack

Attacking is simulated by creating two PySyft federated loaders, one containing valid (benign) training data and one containing the same training data in the same sequence, but with either one or all instances of a class having had their class labels switched (malicious data). The training logic iterates through both loaders but distributes batches to virtual benign or malicious agents from the benign or malicious data loaders, respectively. Training occurs in a sequence of multiple "timestamp" iterations, at the beginning of which the global model is distributed to all agents to be used for training as each agent receives its first batch of training data. Each agent's local model is incrementally updated during multiple training epochs

per timestamp. At the end of each timestamp, the model update parameters are calculated by comparing the trained agent models to the global model that was originally dispatched to the agents.
Before the model parameter updates are aggregated together and applied to the global model, the malicious agents' updates are boosted by a factor of 10, which is evenly distributed among the malicious agents. For instance, with only one malicious agent, the single agent's model parameter updates would get a 10-fold boost while with two agents, each would get a 5-fold boost, etc. Once the updates have been aggregated by weighted average, they are applied to the global model before it is dispatched to the agents for the next timestamp iteration of training.

3.2 Prevention

The malicious agent tries to attack the base model by boosting the weight deltas which means they should be relatively high compared to benign agent weight deltas for them to have an impact when all the agents weight deltas is aggregated. In our experiment, we use a two-step process to detect the anomalous agents which are explained below.

Step 1: Average Distance Calculation

- For each layer Euclidean Distance between the weight deltas is measured by matrix multiplication method using the following function.
  torch.cdist(x1,x2,p=2.0,compute_mode='use_mm_for_euclid_dist_if_necessary')
- X1 and x2 are the weight deltas between two agents within a layer.
- p is the value for the p-norm distance to calculate between each vector pair $\in[0,\infty]$
- Once we have the Euclidean distance, we find the mean of this distance, which signifies on an average how different are the means between two agents in a single layer.

Step 2: Ranking System

- Score the agents in each layer on how close or far away they are from other agents.
- Punishment metric used,
  - Each agent ranks the other agents based on how far they are with respect to it.
  - The closer one gets a better rank than the one that's far.
  - Ranks are divided into three groups, top 30%, average 40%, and bottom 30%
  - Punishment grows from top to bottom by group (remains the same within the group).

  Let us see an example to understand it better, Consider 3 agents [x1,x2,x3]

  cdist(x1,x2) = 10 and cdist(x1,x3) = 5
  cdist(x2,x1) = 10 and cdist(x2,x3) = 3
  cdist(x3,x1) = 5 and cdist(x3,x2) = 3

  Here we can see that x1 and x2 both give higher rank to x3.
  x2 gets the higher rank only by x3.
  x1 gets the lowest rank by both x2 and x3.
  Since x3 has the highest rank overall and falls in "top 30%" it gets least punished, x2 on the other hand fall in "average 40%" so gets slightly higher punishment and x1 for being the lowest rank get heavily punished.
- Once the score is calculated for each agent across all the layers in a time stamp.
  - Check if the score is less than a THRESHOLD = Mean+SD,

    If Check is true,

    Tag agent as non-Malicious

Else,

Tag Agent as Malicious

- mean+SD gave an understanding of how big the bigger score from the smaller ones is, which helped to set the threshold.

4. Experiment, results and comparative analysis

We are using Torchvision's Fashion-MINST dataset and a simple convolutional network for training a model that is used for image classification. The neural network model is based on torch's nn class. It has 2 convolutions layers, each followed by max pooling and Relu layers. The final activation function is log Softmax. We used Google Colab to run all our experiments that we divided into 4 scenarios. The aim of all the experiments was to train the model federatively, using 10 virtual workers or agents, and introduce and prevent malicious attacks by one or more agents.

Scenario 1:

In this scenario we used 1 malicious agent and 9 benign agents. The dataset was equally distributed among all agents and the malicious agent's training data was modified for targeted attack by changing labels for ALL class 5 examples to class 7. Between timestamps the datasets were also shuffled to make sure benign and malicious agents get a new set of training examples, while making sure that malicious agent's data is *corrupted* by changing its class from 5 to 7.

The weight delta boosting factor for the malicious agent was calculated as $\frac{\sum_0^n n_k}{n_k}$, where n is the total number of examples across all agents and n_k is the number of examples available for each agent. Because the weights are equally distributed the booting factor in this case is 10. Two experiments were performed in this scenario. 1). with attack prevention turned OFF and 2). with attack prevention turned ON.

Scenario 1: Attack prevention turned OFF

Figure 1 shows results obtained in this scenario. The green line represents the global model's accuracy on the test dataset and the red line represents the global model's misclassification accuracy on Class 5. The model accuracy and misclassification rates are defined as follows:

$$Misclassification\ rate = \frac{number\ of\ misclassified\ Class\ 5 \rightarrow 7\ images}{total\ number\ of\ Class\ 5\ images} * 100$$

$$Gloabl\ model\ accuracy = \frac{number\ of\ correct\ predictions}{total\ number\ of\ images\ in\ test\ dataset} * 100$$

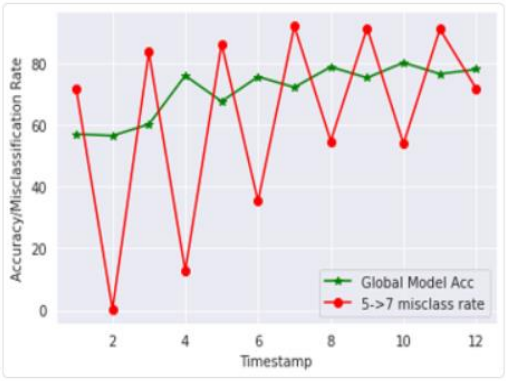

Figure 1: Scenario 1. Attack prevention OFF

At timestamp 1 the global model is very confident (about 75%) in its misclassification and the overall accuracy is only about 60%. In timestamp 2 the weight deltas from benign agents overpower the effect of malicious agent's delta and bring the misclassification rate very low. The oscillating behavior is repeated with global model's misclassification confidence stabilizing in higher timestamps. However, even after 12 timestamps the global model's misclassification doesn't stabilize completely. The overall accuracy on the test dataset reaches 80% at the end of training.

Scenario 1: Attack prevention turned on

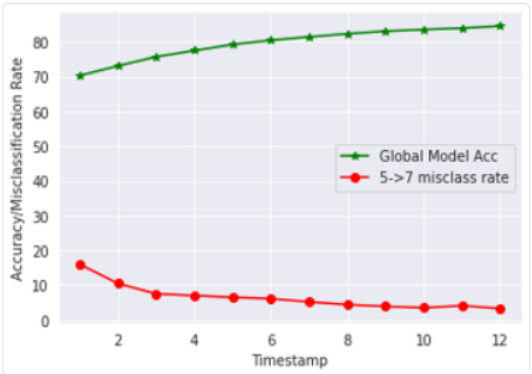

Figure 2: Scenario 1. Attack prevention ON

In this scenario the behavior is completely different. Our attack prevention model detects malicious updates and prevents them from affecting the global model. See figure 2. The global model is somewhat confident in misclassifying class 5 to class 7 on the test data at timestamp 1 but it quickly drops in its misclassification confidence in higher timestamps, reaching a low confidence of about 5% at the end of timestamp 12. The overall global model's accuracy also follows a nice upwards trend and reaches 85% at the end of timestamp 12. This graph proves that attack prevention worked, and the global model was not affected by the malicious agent.

We also looked at the weight delta distributions of the malicious agents and compared it with the averaged weight deltas of all benign agents. Figure 3a shows the weight delta distribution. The orange line in the graph shows the distribution of malicious agent's weights and green line shows the distribution of average of all benign agents. The distributions are quite different and the malicious agent's weight deltas are more distributed because of the boosting factor while the benign agent deltas are clustered in a much narrow range. Figure 3b shows the averaged delta for malicious and benign agents with timestamps. Both the deltas follow the same behavior with a large delta difference between timeframes initially and gradually decreasing in higher timestamps. This is because the model converges in higher timestamps so there are fewer changes in the weights and hence the deltas are small in subsequent timestamps.

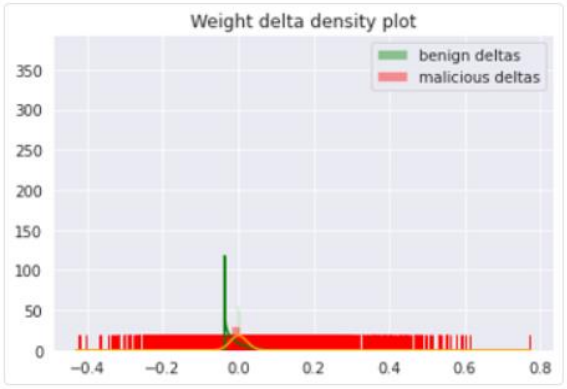

Figure 3a: Weight delta distribution

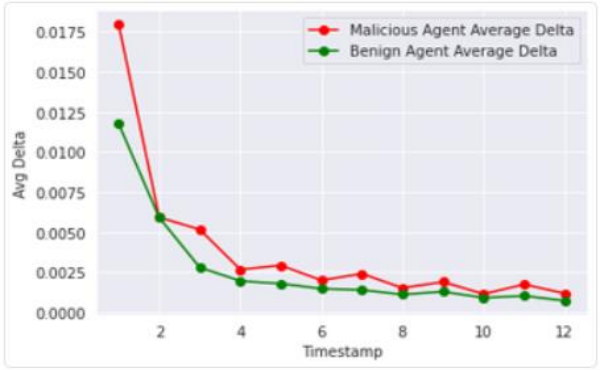

Figure 3b: Average weight deltas with timestamps

Scenario 2:

In this scenario we changed the number of malicious agents to 2 and lowered benign agents to 8. The booting factor in this case for each malicious agent was 5. As with last scenario we ran it with different settings: with attack prevention turned off and with attack prevention turned on.

Scenario 2: Attack prevention turned off

Figure 4 shows results obtained in this scenario. At timestamp 1 the global model is very confident (about 75%) in its misclassification and the overall accuracy is only about 60%. In timestamp 2 the weight deltas from benign agents completely overpower the effect of malicious agent's delta and bring the misclassification rate very low. This behavior is repeated initially but the misclassification confidence stabilizes in higher timestamps. After timestamp 8 the global model is pretty confident in its misclassification confidence and at the end of 12 timestamp its 85% confident in misclassification. This proved that the attack worked, and the global model was altered to misclassify a whole class while keep the overall accuracy to an acceptable value of about 80%.

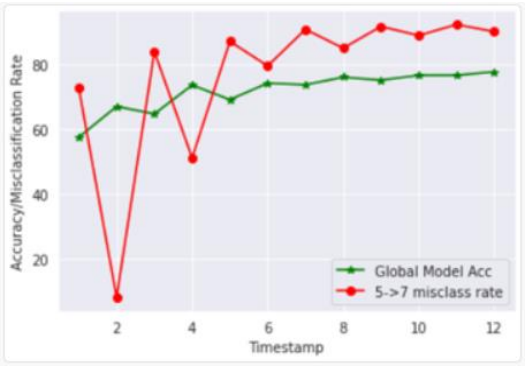

Figure 4: Scenario 2. Attack prevention OFF

Scenario 2: Attack prevention turned on

See figure 5. Attack prevention model detects malicious updates and it quickly nullifies the effect of the malicious agent's weight deltas. In higher timestamps the misclassification confidence is very low, settling to about 5% at the end of timestamp 12. The overall global model's accuracy also follows a nice upwards trend and reaches 85% at the end of timestamp 12. This graph proves that attack prevention worked, and the global model was not affected by the malicious agent.

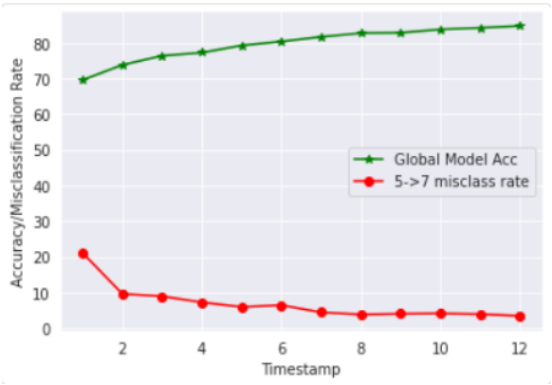

Figure 5: Scenario 2. Attack prevention ON

Scenario 3:

In this scenario we changed the number of malicious agents to 3 and benign agents to 7. The boosting factor in this case was around 3.34

Scenario 3: Attack prevention turned off

See Figure 6. After timestamp 5 the global model stabilizes in its misclassification confidence and at the end of 12 timestamp it is 95% confident in misclassification. This proves that the attack worked. Although, the global model's overall accuracy on the test data suffered and could only improve upto 77% at the end of timestamp 12.

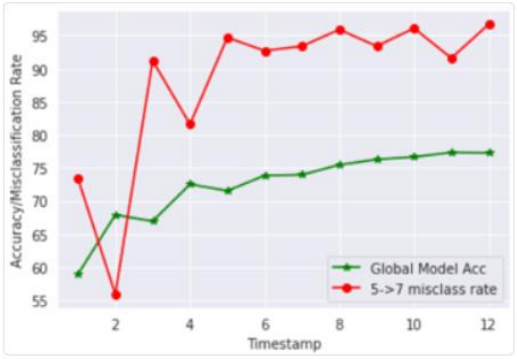

Figure 6: Scenario 3. Attack prevention OFF

Scenario 3: Attack prevention turned on

See figure 7. With attack prevention the behavior is completely different when compared with scenario 1 & 2. The attack prevention model is not able to detect all malicious updates and is unable to prevents them from affecting the global model. The global model is still able to misclassify class 5 examples with very high confidence in timestamps 5, 10 and 12. This is because in these timestamps the attack prevention model could not detect all malicious weight delta updates. For some timeframes like 2, 5, 9 and 11 it was able to detect all malicious agent updates and was able to prevent them from affecting the global model, and that is why for those timestamps the misclassification confidence is very low. The overall accuracy on test data set reaches a respectable value of 80%.

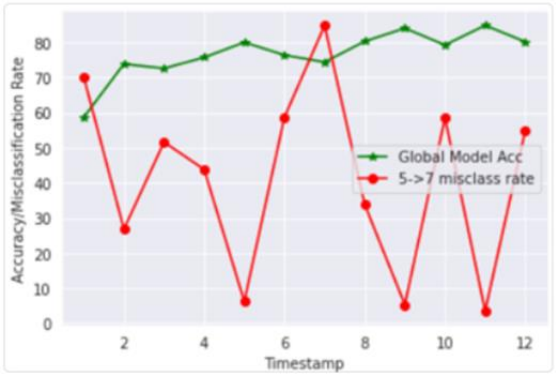

Figure 7: Scenario 3. Attack prevention ON

Scenario 4:

This scenario is different because the definition of targeted attack was changed here. Instead of misclassifying all examples of Class 5 the attack was narrowed with the malicious agent trying to force global model to misclassify ONE single image. Number of malicious agents: 1, benign agents: 9. The other change in this scenario was that there was no shuffling of the datasets between timeframes. This was done to ensure the malicious agent can send malicious updates for the same targeted example at the end of each timeframe. The boosting factor in this case was around 10.

Scenario 4: Attack prevention turned off

Figure 8 shows results obtained in this scenario. In this scenario the misclassification confidence was defined as:

$$Misclassification\ rate(confidence) = \begin{cases} 100, if\ global\ model\ predicts\ Class\ 5 \rightarrow 7 \\ Probability\ of\ predicting\ class\ 7 * 100 \end{cases}$$

$$Gloabl\ model\ accuracy = \frac{number\ of\ correct\ predictions}{total\ number\ of\ images\ in\ test\ dataset} * 100$$

The graph in figure 7 shows that the global model is confident in its misclassification for the ONE targeted example half of the time. The global model's accuracy on test dataset reaches 85% at the end of training.

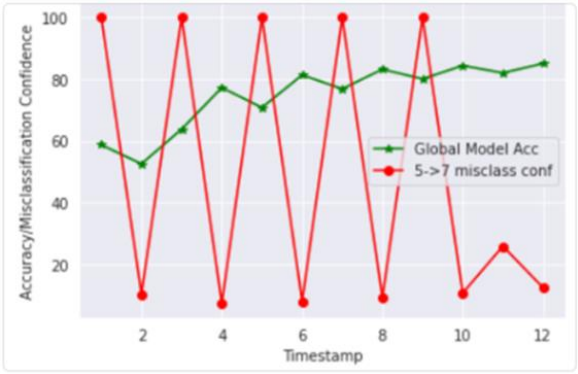

Figure 8: Scenario 4. Attack prevention OFF

Scenario 4: Attack prevention turned on

As shown in figure 9 the global model can correctly misclassify the targeted image on timestamp 1 but in subsequent timeframes the attack prevention algorithm kicks in and prevents the global model to misclassify the image with less and less confidence, eventually going down to less than 5%.

Figure 9: Scenario 4. Attack prevention ON

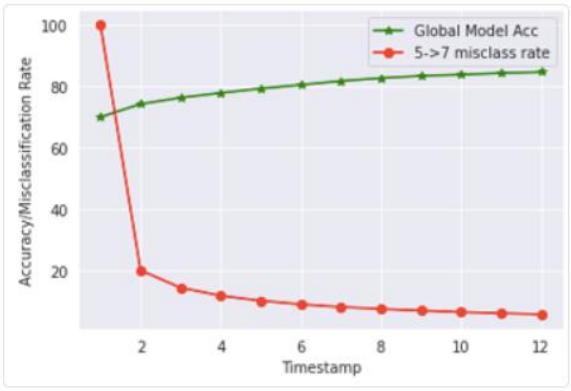

Figure 9: Weight delta distribution

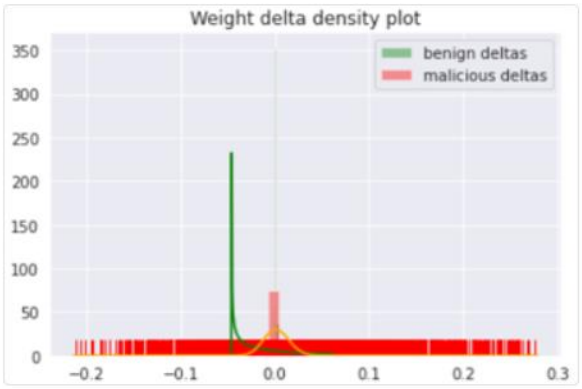

Figure 10: Weight delta distribution

Figure 10 shows the weight delta distribution difference between malicious and averaged benign agent weight deltas. This graph is for timestamp 1.

5. Conclusions

The metrics gathered from our experiments indicate that we were successful in constructing model updates that resulted in targeted misclassification by the global model and we were successful in detecting this type of malicious update and preventing it from being applied to the global model. An interesting observation we made during the training runs where attack prevention was disabled was the oscillation of the misclassification success rates between timestamp iterations, where the misclassification success rate was relatively high after odd-numbered timestamps and relatively low after even-numbered timestamps. We believe that this is due to the relatively low model parameter update deltas from the malicious agents in iterations where the misclassification success rate on the incoming global model was high, thus allowing the updates from the benign updates to weaken the updates from the malicious agents. This hypothesis is supported by the observed averages of the absolute-value deltas depicted in figure 3b. While attacks and attack defenses were successfully demonstrated, this was done under controlled conditions which maximized the probability of success. In a real-world situation, the ratio of malicious agents to benign agents would be much lower, making it much more difficult for the updates from malicious agents to overpower the ones from benign agents. Likewise, real-world malicious agents would employ strategies to conceal their intentions, making them much more difficult to detect and neutralize. Adversarial federated learning promises to be rich in opportunities to devise increasingly sophisticated methods of attack and attack prevention and the experiments carried out in this work barely scratch the surface in terms of what is possible.

6. References

Bhagoji, A. N., Chakraborty, S., Mittal, P., and Calo, S. Analyzing Federated Learning through an Adversarial Lens. In *Proceedings of the 36th International Conference on Machine Learning*, 2019 pp 634-643.

Davy Preuveneers, Vera Rimmer, Ilias Tsingenopoulos, Jan Spooren, Wouter Joosen and Elisabeth Ilie-Zudor. Chained anomaly detection models for federated learning: an intrusion detection case study. www.mdpi.com/journal/applsci

Suyi Li1, Yong Cheng, Wei Wang1, Yang Liu2, Tianjian Chen2. Learning to Detect Malicious Clients for Robust Federated Learning. arXiv:2002.00211v1 [cs.LG] 1 Feb 2020

Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H Brendan McMahan. Can you really backdoor federated learning? arXiv preprint arXiv:1911.07963, 2019.