Security & Privacy of Machine Learning — Assignment 3 Write-Up

Daan Brugmans (S1080742)

May 31, 2024

My Jupyter notebook and report can also be found publicly on GitHub at this URL. My implementation of the assignment consists of two parts: the setup, which contains all implementations of everything I need in order to answer the assignment questions, and the assignment questions themselves. I will adhere to this structure for this write-up as well.

1 Setup

1.1 Imports, Preparation, and Metrics

For the code regarding Federated Learning (FL), dr. Picek has provided a library that implements a fully functioning FL environment. I make use of this code and it is found in the `src` folder. I use this code to set up an FL environment. The `FLSettings` class consists of a collection of variables that we use throughout the codebase to tweak FL settings, such as the number of malignant clients or which aggregation function to use. I use PyTorch as the main environment for building and training a deep learning model, since that is the deep learning framework I am most comfortable with. At the start of the notebook, I execute some preparatory code. I set a seed for *torch*, *NumPy*, and Python itself for improved reproducibility, and I also define a function for visualizing model state dicts. I include an implementation for the Attack Success Rate as a metric.

1.2 Data

For loading the data, I have implemented a function that automatically loads the CIFAR-10 dataset and returns dataloaders for a train, validation, and test set.

1.3 Model

The class `NeuralModel` is a collection of all processes and objects that are needed for training a neural network. It contains an instance of the pre-trained `ResNet18Light` network, its loss function, its optimizer, the number of training epochs, and dataloaders for the train, validation, and test sets. I have chosen to implement it this way, so that all code related to the neural network and its architecture is encapsulated within a single class. In my opinion, this makes performing varying attacks very clean: with only a few rows of code, I am able to instantiate, train, and test a new model. This makes the experiments easy to read and hides away set implementation details.

1.4 Attacks and Defenses

I define classes for BadNet Attacks and Blend Attacks, in addition to functions for the Scaling Attack and Clipping Defense. I made the implementation for the Blend Attack myself, using template code from the week 8 tutorial, while the BadNet attack makes use of existing functionality in `src`. I also made the implementation for the clipping function myself, while the scaling attack code comes from the tutorial on Federated Learning.

1.5 Federated Learning

To simulate an FL scenario, I define a collection of code blocks. I define a single function called `learn_federated` that executes an entire FL scenario with a single function call. I also define the different aggregation functions that we will use (FedAvg and Krum) and use the code in `src` to setup an FL environment.

2 Assignment Questions

2.1 Backdoor Attack in Federated Learning

2.1.1 Implement a source-agnostic backdoor attack in FL using the Blend attack with the Hello Kitty image. Below you find a list of parameters/settings to use for this attack. Limit your implementation to these values. You will be investigating the performance of the attack with different number of malicious clients in the network, i.e., 1, 2, or 3. In all cases, a total of 6 clients compose the network. For example, in the first setting you have 1 malicious and 5 benign clients. However, we ask you to select only a subset from all the clients each round to perform the local training. This subset selection should be random, but you are free how you implement this selection procedure. In the end of the training, i.e., after 5 rounds of FL, plot the final ASR (y-axis) and the global (poisoned) model's task accuracy (also y-axis) versus the number of malicious clients (x-axis). Either make two plots, one for ASR and one for accuracy, or combine the results in one. The following parameters/settings should be used:

- **Model**: Load the pre-trained ResNet18Light model from the Federated Learning tutorial.
- **Poisoning Rate**: Every malicious client should use a poisoning rate of 50% of the local dataset.
- **Global Aggregation Rounds**: 5
- **Local Training Epochs**: 2
- **Backdoor Target Class**: 0 (airplane)
- **Number Selected Clients Per Round**: 5
- **Aggregation Method**: FedAvg

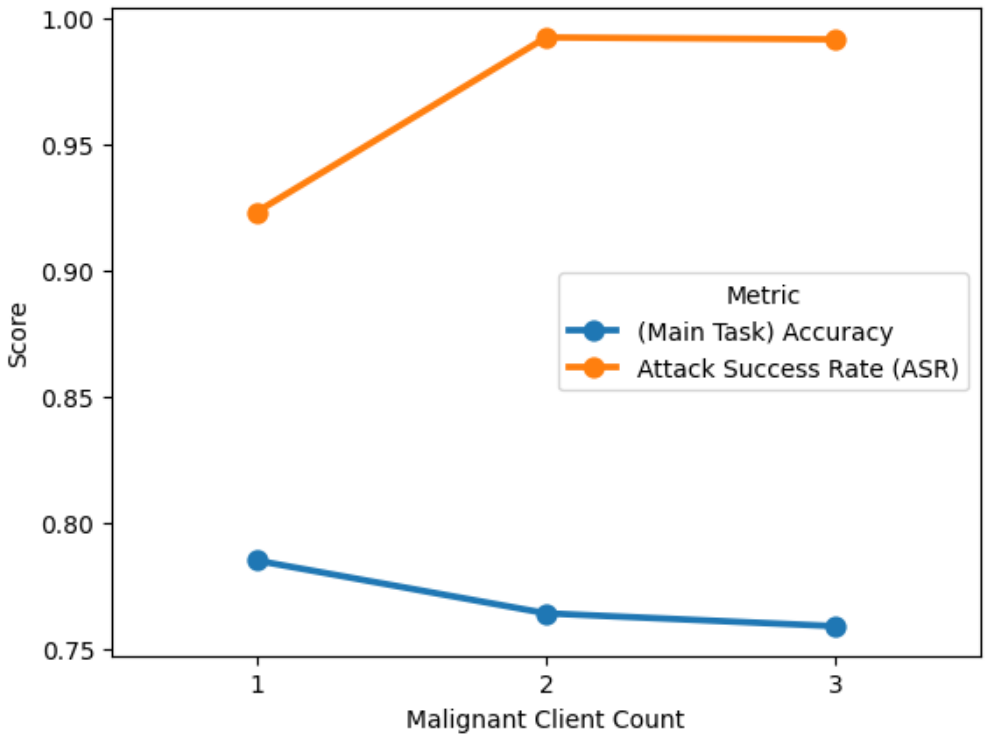

Figure 1: The Main Task Accuracy and Attack Success Rate of the global model with a varying amount of malignant clients.

See figure 1.

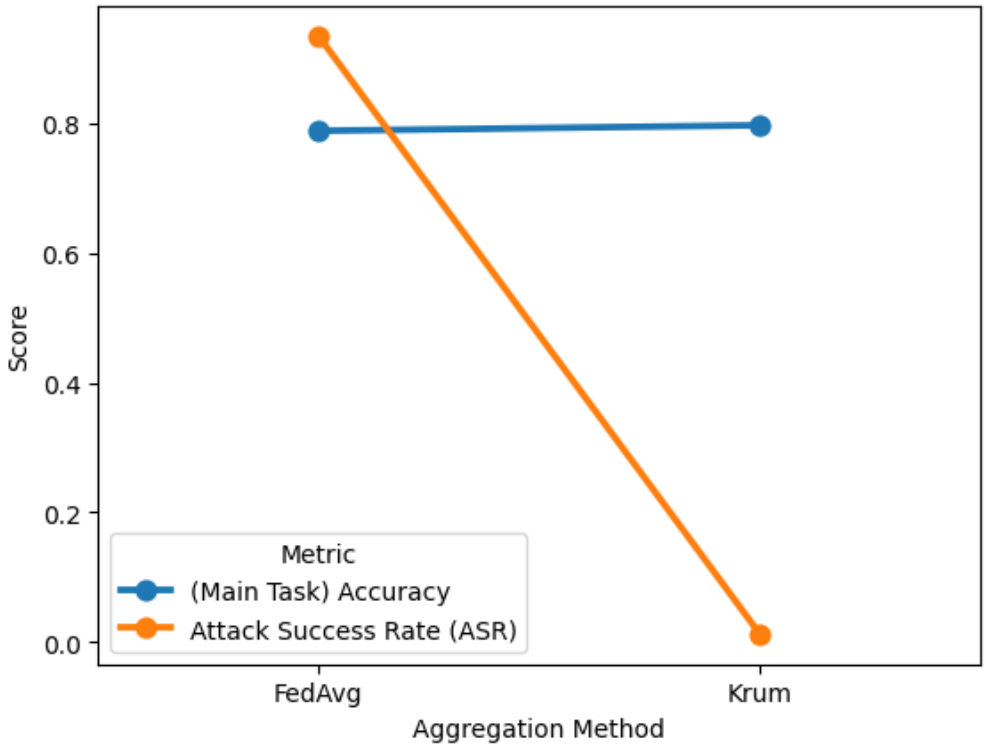

Figure 2: The Main Task Accuracy and Attack Success Rate of the global model with a varying aggregation methods.

2.1.2 In question 1(a) we asked you to use a subset of clients each round. What do you think is the influence of the number of selected clients on the performance of the backdoor attack in the federated setting? Share your conclusions.

I think that the influence of the number of selected clients on the performance of the backdoor attack will vary significantly depending on what we assume that the share of malignant clients will be. If the share of malignant clients is small, then I expect the procedure to be a mildly useful defense, as the global model has an opportunity to recuperate from learning the backdoor every once in a while when a malignant user is excluded. However, if the share of malignant clients is bigger, or even approaching a 50/50 split between benign and malignant clients, then I think that the procedure may only make the backdoor worse, since there is a real chance that there will be round where the global model aggregates from mostly or exclusively malignant clients. This dynamic holds true for when the number of selected clients changes: the smaller the number of selected clients is, the more likely it is that the model aggregates from mostly malignant clients for some rounds, and the more likely it is that the backdoor will be inserted.

2.1.3 In question 1(a) we asked you to perform Federated Learning using the FedAvg method for aggregation. Now using the same settings, but with just 1 malicious client, perform federated learning by making use of the Krum aggregation method. Plot the final ASR (y-axis) and global (poisoned) model task accuracy (y-axis) for both the FedAvg and Krum methods (x-axis). Compare the methods and share your conclusions on how both methods affect the ASR and accuracy. For this homework question we ask you to assume 1 malicious client, so f = 1. Also, only consider the subset of local models each round for aggregation.

When Krum aggregates the weights, it chooses only a subset of the local models to update the global model. It chooses this subset by calculating the Euclidean distances between the local models' weights. Benign models will be close to each other in the Euclidean weight space, since they were trained on similar, unaltered data. Malignant models will be further removed from other models, since they were trained on data that was altered. Having learned from altered data, the weights of the malignant models will look more different from each other and from the benign models than the weights from benign models will look from other benign models. By comparing these weights using the Euclidean distance, we find the models whose weights are most similar to one another, since they were trained on similar data,

and Krum then picks these models to aggregate local weights from. This way, Krum excludes local models which are further removed from other models in the Euclidean space, which are likely to be malignant/backdoored.

In figure 2, we can see that the accuracy for the global models is roughly the same when using either FedAvg or Krum. However, while FedAvg results in a model with an ASR of over 90%, the ASR drops to near-zero when Krum is used as an aggregation method. This is because of the reason explained above: Krum excludes the malignant model from the aggregation process by recognizing that it is far removed from the benign models in a Euclidean space. The malignant model's exclusion from the aggregation prevents the insertion of the backdoor, which is why the ASR is so low when using Krum. FedAvg does not have a check for potentially malignant models, so they are always included in the aggregation, meaning that the backdoor will be learned by the model with every aggregation round.

2.2 Distributed Backdoor Attack

2.2.1 Implement the Distributed Backdoor Attack (DBA). The network of clients is composed of three malicious clients and three benign clients. The three malicious clients will split a red 6 x 6 square trigger vertically into three equal parts. Use the following parameters/settings for your attack: (see list below) Plot the final ASR (y-axis) and global (poisoned) model task accuracy (y-axis). Compare your results with your plot from question 1(a). How does DBA perform compared to the previous FL attack setting? Which FL attack setup performs best and why do you think this is the case? Share your conclusions.

- **Model**: Load the pre-trained ResNet18Light model from the Federated Learning tutorial.
- **Poisoning Rate**: Every malicious client should use a poisoning rate of 50% of the local dataset.
- **Global Aggregation Rounds**: 5
- **Local Training Epochs**: 2
- **Backdoor Target Class**: 0 (airplane)
- **Number Selected Clients Per Round**: 6
- **Aggregation Method**: FedAvg

As of right now, I do not have an implementation for this question.

2.3 Clipping Defense

For this part you are going to perform the clipping defense, but first you will perform the scaling attack as explained in the lecture and tutorial. After applying the scale update to the malicious client and aggregating the results, you will investigate the effect of the clipping defense.

2.3.1 Take exactly the same settings as question 2.1.a but limit it to just 1 malicious client. Use the blend attack and at each round let the malicious client apply the scale update with a scaling factor of $\gamma = \frac{n}{\mu}$. Here n is the number of clients and μ is the learning rate which is also the number of malicious clients and thus now set to 1. Finally, plot the ASR (y-axis) and global model accuracy (y-axis) for just the "scaling approach" (x-axis). Store these results, as you will reuse them in a new plot for the next question.

See figure 3.

2.3.2 Enforce an upper boundary for the L_2-norms of model updates, restricting the Euclidean distances between the global and the respective local models. The model shall be down-scaled if the L_2-norm exceeds the boundary (clipping).

See figure 4 for the accuracy and ASR, figure 5 for the L_2-norms before clipping, and figure 6 for the L_2-norms after clipping.

From the results shown in figure 4, I would consider the defense to be unsuccessful. Regardless of clipping being applied as a defense, the scaling attack succeeds at inserting a backdoor into the global model. The clipping defense has not changed this fact: the ASR is practically perfect, but the accuracy is abhorrent. However, when the logs

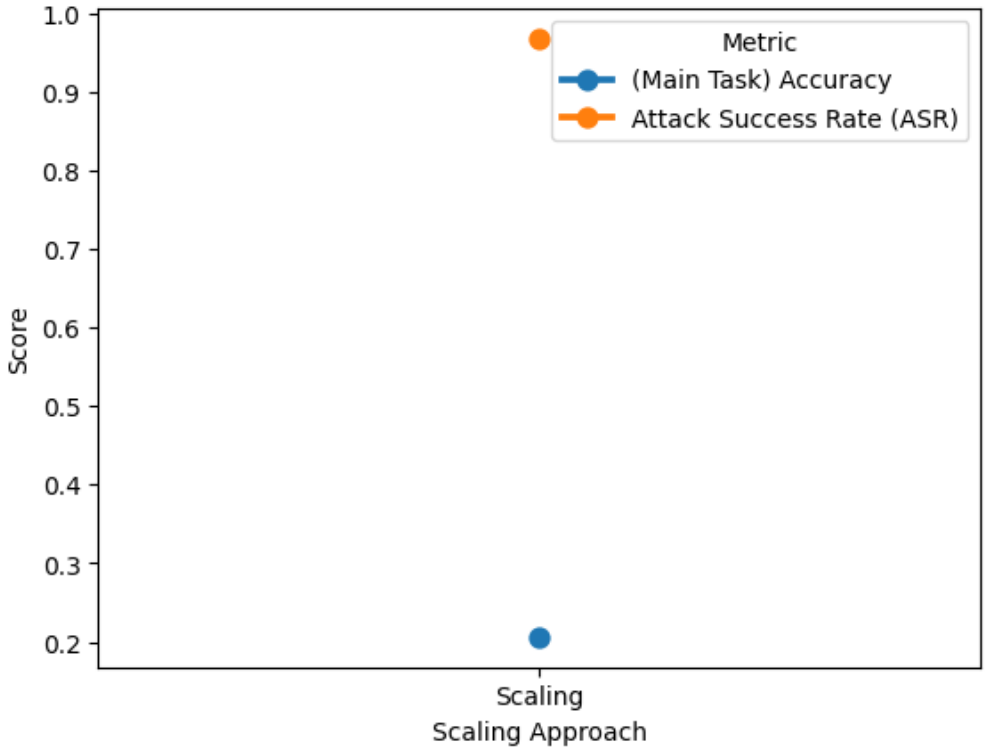

Figure 3: The Main Task Accuracy and Attack Success Rate of the global model with a malignant user that used a scaled Blend attack.

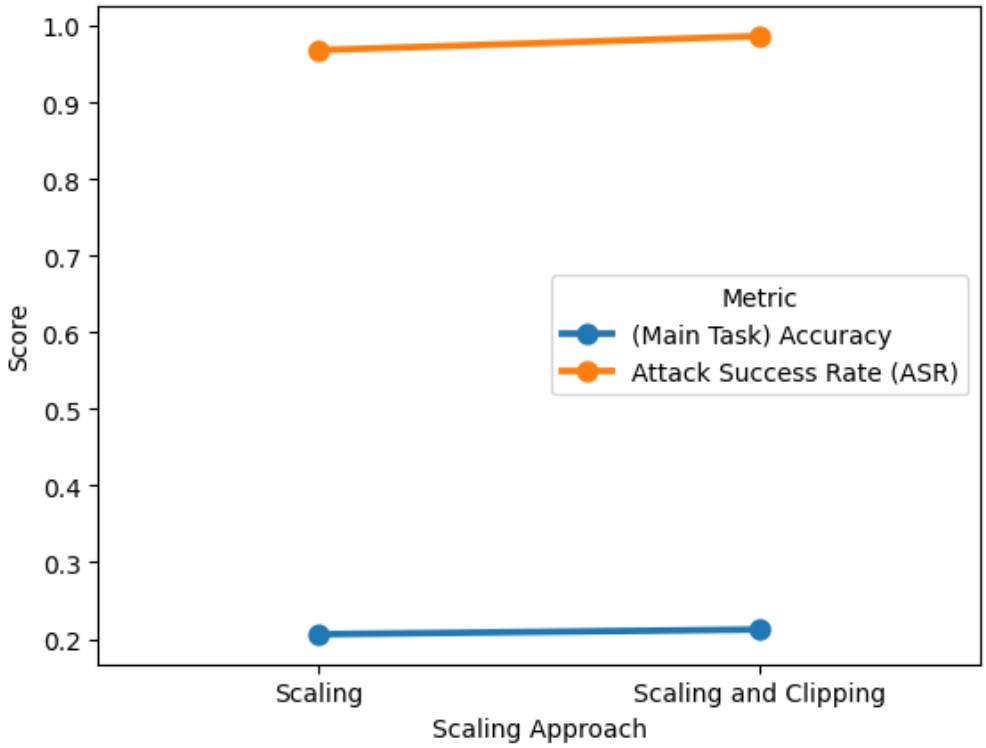

Figure 4: The Main Task Accuracy and Attack Success Rate of the global model with a clipping defense.

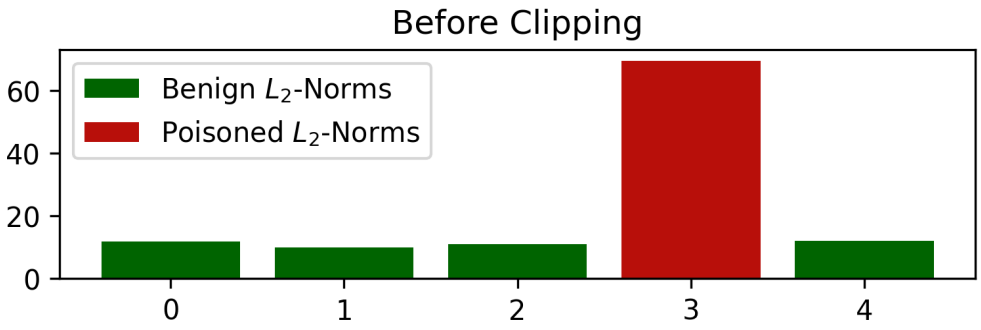

Figure 5: The L_2 norms of the local models at the final aggregation round before clipping.

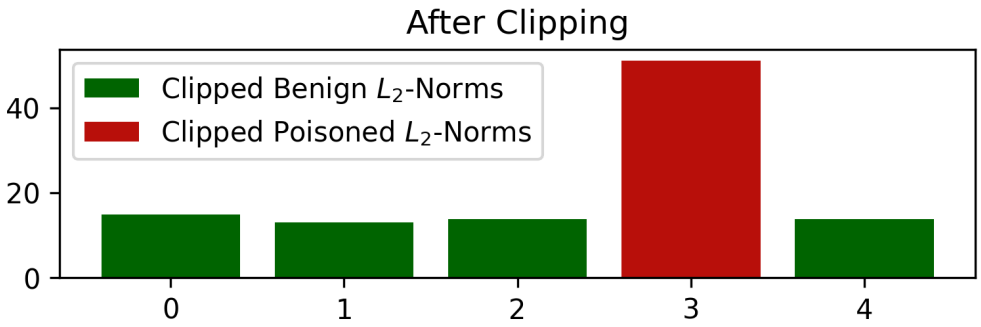

Figure 6: The L_2 norms of the local models at the final aggregation round after clipping.

of the aggregation rounds are inspected, one can see that some rounds end with an accuracy much higher than the final accuracy, even reaching over 50% accuracy. This would imply that the clipping defense does have some effect, but that this effect is too weak to properly mitigate the scaling attack in its entirety.

3 Quality Discussion

I would argue that the quality of my results are good enough to showcase the varying FL scenarios and how they differ from one another. We can easily interpret the effects of how the number of malignant clients influences the global model's ability to learn the backdoor, how different aggregation methods influence this ability to learn the backdoor, how the scaling attack affects the global model, and how the clipping defense mitigates the scaling attack. I was able to achieve this level of quality by making use of both the code that I had already written for assignment 2 and the code provided by the lecturers in the `src` file.