TP3: Understanding Federated learning Attacks and Counteract Schemes

Ahmad Dabaja, Caina Figueiredo Pereira and Rachid Elazouzi

Introduction

In the first practical session (TP1), you implemented a complete Federated Learning project using the FedAvg algorithm, gaining hands-on experience with the FLOWER framework. In the second practical session (TP2), you investigated the effects of data heterogeneity and client drift and explored two advanced optimization techniques, FedProx and SCAFFOLD, designed to improve robustness and convergence in non-IID scenarios.

In this third session (TP3), we shift our focus to the security challenges of Federated Learning. While FL offers strong privacy guarantees by keeping data local, it remains vulnerable to adversarial behaviors by participating clients. In particular, we will explore two common types of attacks: data poisoning and model poisoning, which aim to degrade the global model's performance for malicious goals.

You will implement both types of attacks by modifying the client logic and simulate their impact on the training process. To defend against such attacks, we will also study and implement two robust aggregation schemes, FedMedian and Krum, which aim to mute or filter out malicious updates during aggregation. Finally, we will analyze the trade-offs these defenses introduce, especially under varying degrees of data heterogeneity.

Objectives

This TP aims to introduce you to security threats in Federated Learning and guide you through the implementation of common attack strategies and corresponding defense mechanisms. Specifically, you will:

- Understand the concepts of **data poisoning** and **model poisoning** in Federated Learning, including their goals, mechanisms, and effects on global model performance.
- Implement malicious client behaviors that perform either data or model poisoning within a standard Federated Learning pipeline.

- Evaluate the impact of different proportions of malicious clients (e.g., 0%, 25%, 50%) on model performance using the FedAvg aggregation scheme.
- Implement and test two robust aggregation methods — **FedMedian** and **Krum** — and compare their effectiveness in mitigating poisoning attacks.
- Investigate how these defense mechanisms behave under different levels of **data heterogeneity** (using varying Dirichlet α values) and reflect on the trade-off between robustness to attacks and tolerance to legitimate variations among clients.

Data Poisoning and Model Poisoning

Federated Learning systems are particularly vulnerable to adversarial behavior due to their decentralized nature and limited observability of local training processes. In this section, we explore two common types of attacks that can severely degrade global model performance: **data poisoning** and **model poisoning**.

Understanding the Attacks

Data Poisoning

Data poisoning occurs when a client manipulates its local dataset to mislead the global learning process. The attacker injects malicious examples or modifies labels in a way that skews the local model updates and ultimately harms the global model.

Common techniques:

- *Label flipping:* Changing labels of certain classes (e.g., flipping class A to class B).
- *Random noise injection:* Adding unstructured noise to input data or labels.

Model Poisoning

Model poisoning involves directly altering the model parameters or gradients sent to the server during aggregation. Unlike data poisoning, the client may train normally and then modify the update before submitting it, or amplify its gradients to dominate the global aggregation.

Common techniques:

- *Update injection:* Adding a crafted malicious direction to the model update before sending it.
- *Gradient scaling:* Multiplying model weights or gradients by a large factor to influence global updates.

Both types of attacks can severely reduce model accuracy, slow convergence, or introduce malicious behaviors (e.g., misclassification triggers), especially when a high percentage of participating clients are compromised.

Implementing Data Poisoning

To simulate data poisoning, you will modify the loss calculation used by a client. The simplest form of attack is **label flipping** by changing all labels of the classes. To simulate a data poisoning attack:

- Modify your client class (or your model class based on your code design) from TP1.
- During training, keep the input data unchanged, but alter the labels just before calculating the loss.
- For instance, apply label flipping by replacing all labels in the batch or randomly remapping labels.
- Only the loss calculation should be impacted; the model should still see the original inputs.

Implementing Model Poisoning

In this task, you will simulate clients that perform standard training but modify the resulting model update before sending it back to the server.

Instructions:

- Use the same client class from TP1.
- At the end of the local trainng, before returning updated model parameters:
  - Add a random noise vector or reverse the sign of updates.
  - Multiply all model weights or gradients by a constant factor (e.g., 5 or 10).
- Return the poisoned parameters to the server.

Design tip: You may choose to keep a unified client class with an additional parameter `attack_type {none, data, model}` to highlight the client type and logic or implement three separate classes for each client. This depends on your preferred code structure.

Running FedAvg Under Attack Scenarios

To observe the effect of poisoning attacks on a standard federated setup, run FedAvg simulations under the following conditions:

- **Malicious Client Ratios:** Run separate experiments where 0%, 25%, and 50% of the clients are malicious.
- **Attack Types:** For each ratio, test both data poisoning and model poisoning.
- **Metrics:** Plot and compare the loss and accuracy curves over the training rounds.
- **Conclusion:** Analyze how each type of attack affects model performance. Which attack is more destructive? Is the impact linear with the percentage of malicious clients?

Hint: To inject malicious behavior, you should modify the `run_client` script to accept an additional argument specifying the type of client (e.g., healthy, data-poisoning, or model-poisoning). Based on this argument, the corresponding client class with its attack mode should be launched. Additionally, you will update the `run_simulation` script to randomly select a subset of clients to behave maliciously, according to a specified malicious client proportion provided as a hyperparameter.

In these attacks, we consider two scenarios: the synchronous attack and the asynchronous attack. Different types of attack will be explained in the session dedicated to this TP.

FedMedian: A Robust Aggregation Strategy

What is FedMedian and Why Does It Work?

FedMedian is a robust aggregation method designed to defend Federated Learning systems against malicious or corrupted client updates. Unlike FedAvg, which uses the weighted average of client updates, FedMedian aggregates updates by computing the *coordinate-wise median* of the submitted model parameters.

The idea is based on a simple statistical principle: the median is more robust to outliers than the mean. In the presence of malicious clients who may send poisoned or extreme updates, the median can dampen their effect and preserve the integrity of the global model.

Why it works:

- In a distributed setting, if fewer than 50% of the clients are malicious, then for each model parameter, the coordinate-wise median is guaranteed to reflect one of the honest client updates.
- This strategy prevents extreme values (e.g., from model poisoning) from skewing the global update.

For further information on FedMedian, you can refer to this paper.

Implementing FedMedian

To implement FedMedian, you will modify the `aggregate_fit` method in your `CustomStrategy` class from TP1.

1. Collect all the client model updates as lists of NumPy arrays.
2. Stack the corresponding tensors from all clients and compute the coordinate-wise median:

$$w_{\text{median}}[i] = \text{median}\left(w_1[i], w_2[i], ..., w_K[i]\right)$$

   for every parameter tensor i in the model. where K is the number of clients in aggregation.
3. Convert the resulting list of NumPy arrays back to Flower's `Parameters` format and return it as the new global model.

Note: Unlike FedAvg, you do not need to weight client updates by data size — FedMedian treats all client updates equally.
Hint: You can use NumPy's `np.median(..., axis=0)` function to compute the coordinate-wise median.

Running Experiments with FedMedian

Now that FedMedian is implemented, evaluate its robustness under attack.

- Run three experiments with different ratios of malicious clients: 0%, 25%, and 50%.
- Simulate a low heterogeneity environment by setting the Dirichlet parameter to $\alpha = 10$.
- In each case, compare the behavior of FedMedian to that of FedAvg.
- Repeat the experiments for both **data poisoning** and **model poisoning**.
- Using the `DataVisualizer` class, plot and compare:
  - Training loss
  - Validation accuracy
- Reflect on the following questions:
  - How does FedMedian perform compared to FedAvg when no malicious clients are present?
  - At which malicious client ratio does FedAvg start to fail?
  - Does FedMedian succeed in stabilizing the training?

Krum: Distance-Based Robust Aggregation

What is Krum and Why Does It Work?

Krum is a robust Federated Learning aggregation strategy designed to resist Byzantine attacks by filtering out suspicious or inconsistent client updates.

The key idea behind Krum is to select the update that is most "in agreement" with the majority of the other updates. It assumes that malicious updates will look different from the majority, while honest clients will produce similar gradient directions.

How Krum works:

1. For each client update, compute the Euclidean distance between it and all other client updates.
2. For every client, compute a *score* by summing the distances to its $n-f-2$ closest neighbors (where n is the total number of clients, and f is the number of suspected malicious clients).
3. Select the client update with the lowest score.
4. Use this single update to replace the global model (i.e., no averaging).

Why it works:

- Honest client updates tend to cluster close together.
- Malicious updates usually deviate significantly, so their scores will be higher.
- Selecting the most "central" update reduces the influence of malicious clients.

For more information on Krum, you can refer to its original paper.

Implementing Krum

To implement Krum, you will again modify the `aggregate_fit` method in your custom strategy class.

1. Collect all client updates as lists of NumPy arrays.
2. Flatten each update into a single vector (you can use `np.concatenate()`).
3. Compute pairwise Euclidean distances between all client updates.
4. For each client, sum the distances to its $n - f - 2$ closest neighbors and assign it a score.
5. Select the update with the lowest score.
6. Use that update as the new global model.
7. Convert the selected NumPy update back to Flower's `Parameters` format.

Running Experiments with Krum

Repeat the same attack scenarios using Krum.

- Run three simulations with 0%, 25%, and 50% of malicious clients.
- Simulate a low heterogeneity environment by setting the Dirichlet parameter to $\alpha = 10$.
- Repeat for both **data poisoning** and **model poisoning** attacks.
- Using `DataVisualizer` class, Plot and compare Krum with FedAvg and FedMedian in terms of:
  - Accuracy and loss
  - Convergence speed

Notes: Choose f according to the expected number of malicious clients (e.g., $f = 5$ if 50% of 10 clients are malicious).

Impact of Data Heterogeneity on Robust Aggregation Schemes

Motivation

While robust aggregation schemes like FedMedian and Krum can mitigate the effect of malicious clients, they may also inadvertently penalize honest clients when data heterogeneity is high.

In real-world FL deployments, different clients often hold data from different distributions (i.e., non-IID). This can make some honest clients appear "abnormal," similar to malicious ones, and cause them to be ignored or down-weighted by robust schemes. Understanding this trade-off is crucial for designing effective defenses.

Experiment Setup

In this final experiment, you will fix the percentage of malicious clients to 25%, and vary the level of data heterogeneity using the Dirichlet distribution parameter $\alpha \in \{10, 1, 0.1\}$. For each level of heterogeneity:

- Simulate federated learning under:
  - FedAvg (baseline)
  - FedMedian
  - Krum
- Use the same type of poisoning (choose either data poisoning or model poisoning).

- Keep all other hyperparameters (e.g., learning rate, number of rounds, number of clients) constant.

Evaluation and Comparison

After running the experiments:

- Use the `ResultsVisualizer` to plot accuracy and loss curves.
- Analyze and discuss:
  - How does increasing heterogeneity affect the performance of each scheme?
  - Do FedMedian or Krum mistakenly reject honest clients with outlier behavior?
  - Under which conditions does each scheme work best?

This step will help you understand the challenge of distinguishing between malicious updates and honest updates from statistically different clients — a key issue in the design of secure and robust Federated Learning systems.

Summary

In this third practical session, you explored the vulnerabilities of Federated Learning to adversarial attacks and studied techniques to mitigate them. Building on your previous implementations from TP1 and TP2, you gained hands-on experience with both offensive and defensive strategies in the FL setting.

You have:

- **Learned about FL attacks:** You studied two common attacks in FL — *data poisoning*, where clients manipulate their labels, and *model poisoning*, where clients alter their model updates.
- **Implemented Malicious Clients:** You modified the client logic to simulate both types of attacks and explored their effects on training convergence and accuracy.
- **Analyzed Attack Impact:** You ran simulations with different proportions of malicious clients (0%, 25%, and 50%) and visualized how the global model deteriorated under poisoning.
- **Implemented FedMedian:** You implemented a robust aggregation rule based on the coordinate-wise median, and compared its resilience to poisoning against FedAvg.
- **Implemented Krum:** You added another robust strategy that selects the most representative client update using distance metrics, and tested its behavior in presence of attackers.

- **Explored Heterogeneity vs Robustness:** You investigated how high data heterogeneity can confuse robust defenses, particularly under non-IID distributions, by fixing the malicious rate to 25% and varying the Dirichlet α.

By completing this TP, you developed a foundational understanding of adversarial threats in Federated Learning, along with practical knowledge of defensive techniques. This experience prepares you to reason critically about trust, robustness, and fairness in decentralized systems.