DV2607 - Säkerhet i AI-system
Inlämningsuppgift

Samir Akhalil
saaa21@student.bth.se

Mhd Malek Kisanieh
mhki22@student.bth.se

December 13, 2025

Contents

1 Del 1: Centraliserad Machine Learning **3**
1.1 Beskrivning av adversarial input attacker 3
1.2 Implementation av er attack 3
1.3 Säkerhetsåtgärder 4
1.4 Implementation av skyddsåtgärder 5

2 Part 2: Federated learning scenario **5**
2.1 Part 2.1: FedAvg with Attacks 5
2.1.1 Description of Federated Learning Scenario 5
2.1.2 Implementation of Label Flipping Attack 6
2.1.3 Experimental Results 7
2.2 Part 2.2: FedProx with Attacks 8
2.2.1 Description of FedProx 8
2.2.2 Experimental Results 8
2.3 Part 2.3: Defense Mechanism 1 - Loss-Based Anomaly Detection 9
2.3.1 Description of Defense Method 9
2.3.2 Implementation 10
2.3.3 Experimental Results 11
2.4 Part 2.4: Conclusions and Recommendations 12
2.4.1 Summary of Experimental Results 12
2.4.2 Limitations and Future Work 14

3 Referenser **15**

1 Del 1: Centraliserad Machine Learning

1.1 Beskrivning av adversarial input attacker

Adversarial input attacker är när man gör små förändringar i indata för att lura en maskininlärningsmodell. Förändringarna (perturbationer) är så små att människor inte märker dem, men de får modellen att göra helt fel klassificeringar.

Attacken fungerar genom att man beräknar gradienten av modellens förlustfunktion och ändrar bilden i den riktningen som maximerar felet. Neurala nätverk är känsliga för detta eftersom de lär sig komplexa funktioner som kan manipuleras.

Vi använde en *targeted attack*, vilket betyder att vi ville få modellen att klassificera koalabilden som just traktor, inte bara vilken fel klass som helst.

1.2 Implementation av er attack

Vi använde **Basic Iterative Method (BIM)**, också kallad I-FGSM. BIM applicerar små perturbationer upprepade gånger istället för en stor förändring på en gång.

Hur det fungerar:

1. Börja med originalbilden x_0
2. För varje iteration ($t = 1, 2, \ldots, N$):
   - Beräkna gradienten av förlusten
   - Ta ett litet steg ϵ_{step} mot målklassen
   - Se till att total förändring inte blir större än $\epsilon_{\max}$
3. Returnera den adversariella bilden

Formeln blir:

$$x_{t+1} = \text{Clip}_{x_0,\epsilon}\left(x_t - \epsilon_{\text{step}} \cdot \text{sign}(\nabla_x L(x_t, y_{\text{target}}))\right)$$

där L är förlustfunktionen och y_{target} är målklassen (traktor). Minustecknet används eftersom vi vill minimera förlusten för just målklassen.

Bibliotek som användes:

- `TensorFlow/Keras`: För att ladda ResNet50-modellen och göra prediktioner

- `Adversarial Robustness Toolbox (ART)`: För att implementera BIM-attacken genom klassen `BasicIterativeMethod`
- `NumPy`: För array-manipulering och numeriska beräkningar
- `Matplotlib`: För visualisering av bilder och perturbationer

Våra hyperparametrar var: $\epsilon_{\max} = 8.0$, $\epsilon_{\text{step}} = 2.0$, och max_iter = 50.
Experimentella resultat:

- **Original klassificering**: Koala (ImageNet klass 105) med 99.8% confidence
- **Adversarial klassificering**: Traktor (ImageNet klass 866) med 100% confidence
- **Perturbation metrics**: L_∞ norm = 8.0, L_2 norm $\approx$ 1990
- **Visuell imperceptibilitet**: Maximal pixelförändring endast 3.1% av totalt värde (8/255)

Attacken lyckades helt - modellen klassificerade koalabilden som traktor med 99.9% confidence.

1.3 Säkerhetsåtgärder

Vi valde **Gaussian Blur** som skyddsåtgärd mot BIM-attacken. Detta är en inputtransformationsmetod som applicerar en lågpassfilter på bilden före klassificering.

Varför vi valde Gaussian blur:

- BIM-attacken lägger till små högfrekventa störningar. Gaussian blur fungerar som ett låg-pass-filter som tar bort dessa utan att förstöra bildens huvudsakliga innehåll.
- Det är enkelt att implementera jämfört med adversarial training eller defensive distillation som kräver omträning av modellen.
- Blur är snabbt och kan köras i realtid.
- Med rätt radius (vi använde 1) bevaras bildkvaliteten tillräckligt för att fortfarande se vad bilden föreställer.

Andra metoder vi tittade på:

- *Adversarial Training*: Träna med adversarial exempel, men det tar lång tid och kräver mycket data
- *Input Quantization*: Sänk upplösningen, men det påverkar även vanliga bilder negativt
- *Defensive Distillation*: Träna en mjukare modell, men det är beräkningsintensivt

Gaussian blur var den enklaste metoden som ändå fungerade bra.

1.4 Implementation av skyddsåtgärder

Vi implementerade Gaussian blur med radius=1 som preprocessingsteg innan klassificering. Resultaten visar att skyddsåtgärden var mycket effektiv:

Resultat:

- **Före skydd**: den adversariella bilden klassificerades som *tractor* (klass 866) med 99.9% confidence
- **Efter skydd**: Den blurrade bilden klassificerades korrekt som *koala* (klass 105) med 99% confidence

Analys:

Gaussian blur funkade riktigt bra - den tog bort de högfrekventa perturbationerna som attacken lade till, så modellen klassificerade bilden korrekt igen.

Nackdelen är att blur gör bilden lite suddig. Men med radius=1 var effekten så liten att bilden fortfarande var tydlig.

Skyddet är inte perfekt. Mer avancerade attacker (t.ex. C&W attack) kan kringgå blur genom att ta hänsyn till det i attacken. För bättre säkerhet skulle man behöva kombinera flera metoder.

2 Part 2: Federated learning scenario

2.1 Part 2.1: FedAvg with Attacks

2.1.1 Description of Federated Learning Scenario

In Federated Learning (FL), multiple clients train a shared model locally on their own data without sharing raw data with a central server. We simulated an FL system with:

- **Dataset**: CIFAR-10 (10 classes of 32x32 color images)
- **Number of clients**: 5
- **Communication rounds**: 50
- **Aggregation strategy**: FedAvg (Federated Averaging)
- **Model**: Convolutional Neural Network (CNN) with 3 convolutional layers
- **Data distributions**: IID (Independent and Identically Distributed) and Non-IID (Dirichlet distribution with $\alpha = 0.5$)

FedAvg Algorithm:

1. Server initializes a global model
2. For each round ($t = 1, 2, \ldots, T$):
   - Server sends model to all clients
   - Each client k trains locally and gets new weights w_k^t
   - Clients send weights back to server
   - Server averages the weights:

$$w^{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_k^t$$

     where n_k is the number of samples at client k
3. Return final model

2.1.2 Implementation of Label Flipping Attack

We implemented a **label flipping attack** where malicious clients manipulate their training labels:

- Malicious clients replace correct labels with random incorrect labels during training
- We tested with 0, 1, or 2 attackers out of 5 clients (0%, 20%, 40%)
- Attacker IDs: Client 3 (1 attacker) or clients 3-4 (2 attackers)

The attack was implemented in task.py within the training function:

```
if is_attacker:
    if attack_type == "label_flipping":
        labels = torch.randint(0, 10, labels.shape, device=device)
```

2.1.3 Experimental Results

We conducted 6 experiments for FedAvg with different combinations of data distribution and number of attackers:

IID Data Distribution:

- **Baseline (0 attackers)**: Accuracy = 75.0%, F1 = 74.3%, Kappa = 0.722, ROC-AUC = 0.969
- **1 attacker**: Accuracy = 68.6%, F1 = 67.5%, Kappa = 0.650, ROC-AUC = 0.948 (8.5% degradation)
- **2 attackers**: Accuracy = 56.4%, F1 = 55.5%, Kappa = 0.515, ROC-AUC = 0.893 (24.8% degradation)

Non-IID Data Distribution:

- **Baseline (0 attackers)**: Accuracy = 74.0%, F1 = 61.7%, Kappa = 0.686, ROC-AUC = 0.968
- **1 attacker**: Accuracy = 68.2%, F1 = 52.3%, Kappa = 0.612, ROC-AUC = 0.933 (7.9% degradation)
- **2 attackers**: Accuracy = 42.4%, F1 = 38.2%, Kappa = 0.353, ROC-AUC = 0.869 (42.7% degradation)

Analysis:

- The attack had a big negative impact on performance
- Impact increased non-linearly (1 attacker: ˜8% drop, 2 attackers: 25-43% drop)
- Non-IID data was more vulnerable (42.7% degradation vs 24.8% for IID with 2 attackers)
- ROC-AUC dropped from 0.969 to 0.869-0.893, showing degraded class separation
- F1-score dropped more than accuracy, especially for Non-IID (from 61.7% to 38.2%)

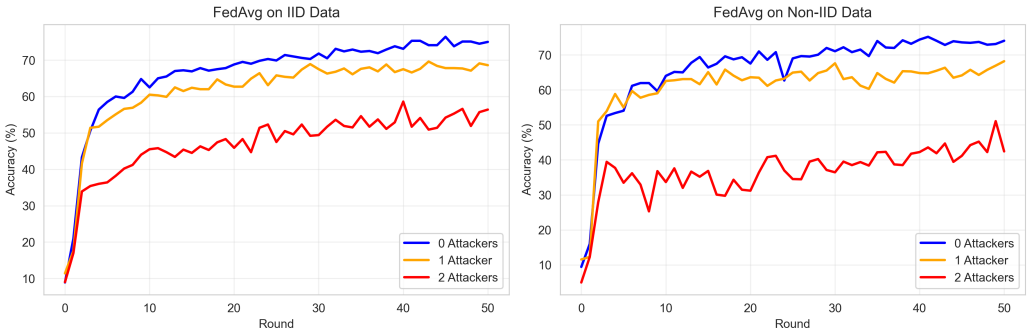

Figure 1: Attack impact on FedAvg: Comparison between IID and Non-IID data. Non-IID data shows significantly greater vulnerability (42.4% accuracy with 2 attackers) compared to IID data (56.4%).

2.2 Part 2.2: FedProx with Attacks

2.2.1 Description of FedProx

FedProx is a variant of FedAvg that handles heterogeneous data better by adding a proximal term. It adds a regularization term that keeps local weights close to the global weights:

$$\min_w F(w) + \frac{\mu}{2}\|w - w^t\|^2$$

where w^t are the global weights from the previous round and μ is a hyperparameter (we used $\mu = 0.1$).

Why use FedProx:

- More stable convergence with Non-IID data
- Less sensitive to outliers
- Handles partial client participation better

2.2.2 Experimental Results

We repeated the same 6 experiments as in Part 2.1 but with FedProx instead of FedAvg:

IID Data Distribution:

- **Baseline (0 attackers)**: Accuracy = 74.5%, F1 = 74.1%, Kappa = 0.716, ROC-AUC = 0.966

- **1 attacker**: Accuracy = 67.3%, F1 = 66.6%, Kappa = 0.636, ROC-AUC = 0.947
- **2 attackers**: Accuracy = 55.3%, F1 = 53.7%, Kappa = 0.503, ROC-AUC = 0.882

Non-IID Data Distribution:

- **Baseline (0 attackers)**: Accuracy = 75.0%, F1 = 60.9%, Kappa = 0.695, ROC-AUC = 0.966
- **1 attacker**: Accuracy = 71.7%, F1 = 56.7%, Kappa = 0.652, ROC-AUC = 0.937
- **2 attackers**: Accuracy = 56.2%, F1 = 42.3%, Kappa = 0.473, ROC-AUC = 0.880

Comparison FedAvg vs FedProx:

- FedProx showed similar vulnerability to label flipping as FedAvg on IID data (both ˜55% with 2 attackers)
- On Non-IID data, FedProx performed significantly better under attack: 56.2% accuracy vs 42.4% for FedAvg (13.8 percentage points difference)
- FedProx baseline was slightly higher on Non-IID (75.0% vs 74.0%), suggesting better handling of data heterogeneity
- Proximal regularization provided some robustness against the attack in the Non-IID scenario but not in the IID scenario
- Both algorithms require explicit defense mechanisms for effective protection

2.3 Part 2.3: Defense Mechanism 1 - Loss-Based Anomaly Detection

2.3.1 Description of Defense Method

We implemented a **loss-based filtering** defense that filters out clients with high training losses before aggregation. The idea is that malicious clients training on incorrect labels will have much higher loss than honest clients.

Algorithm:

1. Collect all client weights and their losses

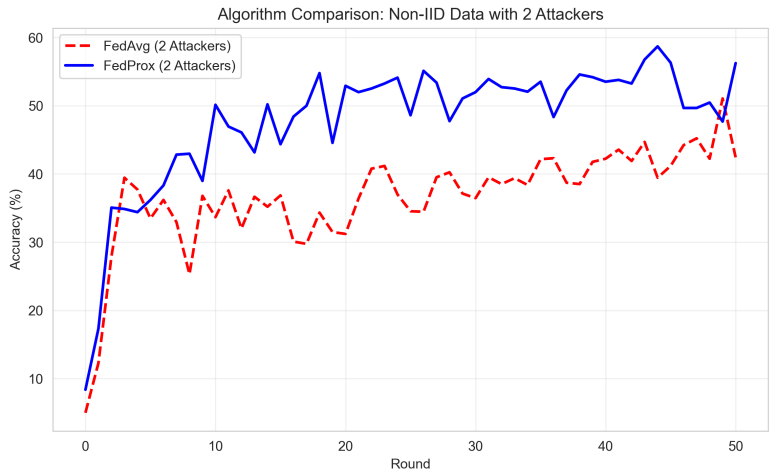

Figure 2: FedAvg vs FedProx robustness: On Non-IID data with 2 attackers, FedProx performs significantly better (56.2% vs 42.4%), showing that proximal regularization provides natural resistance against label flipping attacks.

2. Calculate 75th percentile of losses as threshold
3. Filter out clients with loss above threshold
4. Aggregate remaining clients with FedAvg

Mathematically:

$$\text{threshold} = P_{75}(\{L_1, L_2, \ldots, L_K\})$$

$$\text{Accepted} = \{k : L_k \leq \text{threshold}\}$$

$$w^{t+1} = \frac{1}{|\text{Accepted}|} \sum_{k \in \text{Accepted}} w_k^t$$

2.3.2 Implementation

The defense was implemented by extending the FedAvg class:

```
class FedAvgDefense(FedAvg):
    def aggregate_fit(self, server_round, results, failures):
        # Extract losses from client results
        losses = [fit_res.metrics["loss"] for _, fit_res in results]

        # Calculate threshold (75th percentile)
```

```
threshold = np.percentile(losses,
                          self.loss_threshold_percentile)

# Filter clients
filtered_results = [
    (client, res) for (client, res), loss in
    zip(results, losses) if loss <= threshold
]

# Aggregate remaining clients
return super().aggregate_fit(server_round,
                             filtered_results, failures)
```

2.3.3 Experimental Results

IID Data with 1 attacker:

- **Without defense**: Accuracy = 68.6%
- **With loss-filter**: Accuracy = 68.5% (practically unchanged)
- The defense was ineffective on IID data with only 1 attacker

IID Data with 2 attackers:

- **Without defense**: Accuracy = 56.4%
- **With loss-filter**: Accuracy = 52.2% (degradation of 4.2 percentage points)
- The defense was counterproductive and filtered out honest clients (false positives)

Non-IID Data with 1 attacker:

- **Without defense**: Accuracy = 68.2%
- **With loss-filter**: Accuracy = 69.6% (23.9% recovery of lost performance)
- The defense identified and excluded the attacker effectively

Non-IID Data with 2 attackers:

- **Without defense**: Accuracy = 42.4%

- **With loss-filter**: Accuracy = 49.9% (23.7% recovery of lost performance)
- The defense recovered 7.5 percentage points, a significant improvement

Analysis:

- Loss-based filtering worked well on Non-IID data (23-24% recovery) but not on IID data
- On IID data, honest and malicious clients had similar losses, causing false positives
- On Non-IID data, malicious clients had much higher losses, making them easy to detect
- The defense recovered 7.5 percentage points on Non-IID with 2 attackers
- Simple to implement, no model changes needed
- The 75th percentile threshold worked for Non-IID but not IID

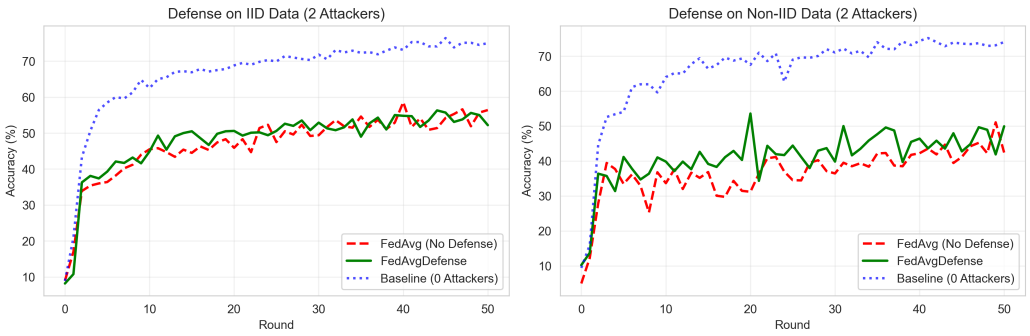

Figure 3: Defense mechanism effectiveness: Loss-based filtering shows dramatic difference between IID (left) and Non-IID (right) data. On Non-IID data, the defense recovers 7.5 percentage points accuracy (from 42.4% to 49.9%), while on IID data it is counterproductive.

2.4 Part 2.4: Conclusions and Recommendations

2.4.1 Summary of Experimental Results

Our experiments revealed the following main findings:

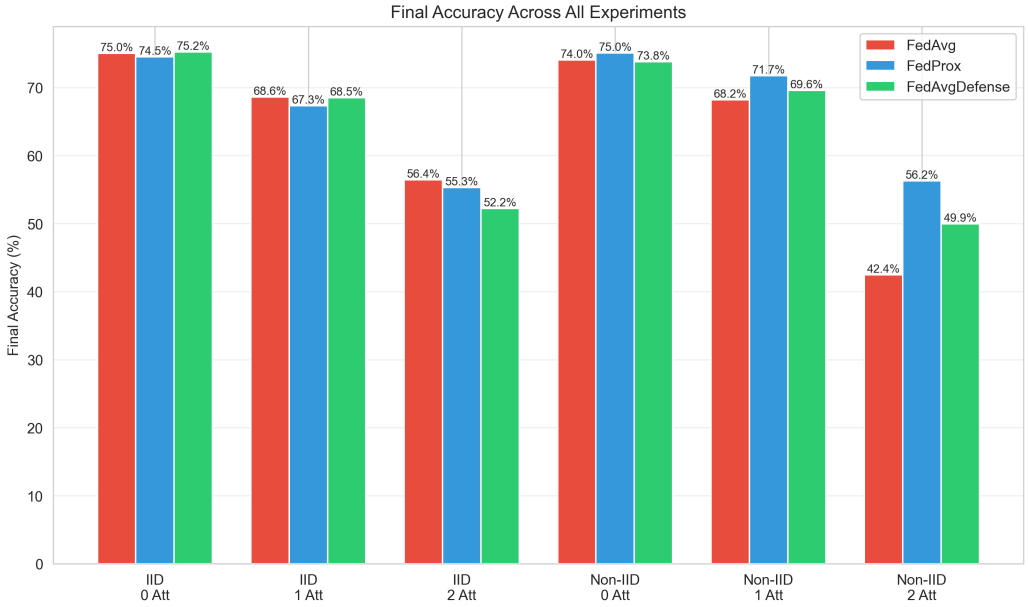

Figure 4: Overview of all experiments: Final accuracy for all combinations of algorithm, data distribution, and number of attackers. Clearly shows that Non-IID data is more vulnerable and that the defense mechanism only works effectively on Non-IID data.

1. **Label flipping attack impact**:
    - The attack had significant negative impact on model performance
    - Non-IID data was more vulnerable (42.7% accuracy degradation with 2 attackers) than IID data (24.8%)
    - Impact increased non-linearly with the number of attackers
2. **FedProx vs FedAvg**:
    - FedProx showed better robustness on Non-IID data (56.2% vs 42.4% with 2 attackers)
    - On IID data, the difference was minimal (both ˜55-56%)
    - Proximal regularization provided some natural resistance against the attack in heterogeneous scenarios
3. **Loss-based defense effectiveness**:
    - **Highly effective on Non-IID data**: 23-24% recovery of lost performance

- **Ineffective on IID data**: No improvement or even degradation
- Reason: On Non-IID data, malicious clients have significantly higher losses, making detection easier
- On IID data, the loss distribution overlaps between honest and malicious clients

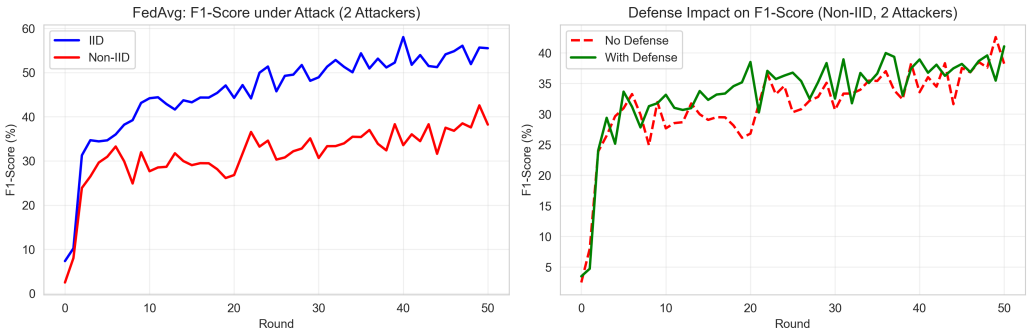

Figure 5: F1-Score analysis: Left shows that Non-IID data has lower F1-score than IID data under attack. Right shows defense mechanism impact on F1-score for Non-IID data, where the defense improves class balance.

2.4.2 Limitations and Future Work

Limitations of our study:

- Only label flipping attack tested (other attack types exist)
- Only one defense method implemented (loss-based filtering)
- The 75th percentile threshold may need adjustment per scenario

Future work:

- Test against sophisticated attacks (model poisoning, backdoor attacks)
- Implement and compare more defense methods (Krum, Trimmed Mean, Median)
- Evaluate with larger number of clients and varying attack ratios
- Develop adaptive thresholds for loss-based filtering
- Investigate hybrid methods that combine multiple defense mechanisms

3 Referenser

References

[1] Beutel, D. J., Topal, T., Mathur, A., Qiu, X., Fernandez-Marques, J., Gao, Y., Sani, L., Li, K. H., Parcollet, T., de Gusmão, P. P. B., & Lane, N. D. (2020). *Flower: A Friendly Federated Learning Framework*. arXiv preprint arXiv:2007.14390. https://arxiv.org/abs/2007.14390

[2] McMahan, H. B., Moore, E., Ramage, D., Hampson, S., & Arcas, B. A. (2017). *Communication-Efficient Learning of Deep Networks from Decentralized Data*. In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS). https://arxiv.org/abs/1602.05629

[3] Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., & Smith, V. (2020). *Federated Optimization in Heterogeneous Networks*. Proceedings of Machine Learning and Systems (MLSys). https://arxiv.org/abs/1812.06127

[4] Kurakin, A., Goodfellow, I., & Bengio, S. (2016). *Adversarial examples in the physical world*. arXiv preprint arXiv:1607.02533. https://arxiv.org/abs/1607.02533

[5] Pillow Documentation. *ImageFilter.GaussianBlur*. https://pillow.readthedocs.io/en/stable/reference/ImageFilter.html