Neural Network Reliability Verification

Assignment 1

Jongbok Won (2025321347)
Department of Artificial Intelligence

October 3, 2025

1 Introduction

This document is the abstract report for the first assignment of "Neural Network Reliability Verification". The goal of this assignment is to implement FGSM (Fast Gradient Sign Method) and PGD (Projected Gradient Descent) adversarial attacks in Python for the MNIST and CIFAR-10 datasets, and to test the robustness of custom-designed deep learning models.

This project is based on an object-oriented design to encapsulate the attack logic. A key focus was placed on automating the entire workflow - from data loading, model training, and evaluation to performing attacks and saving the results in .json format - through the execution of a single script.

2 Core Implementation

2.1 Model Architecture

As per the assignment instructions, two unique CNN models were designed for each dataset.

- SourceNet (for MNIST)
  - A Simple `CNN` model suitable for processing grayscale images (1×28×28).
  - It extracts features using two `Conv2d` layers and `MaxPool2d` layers. After a `Flatten` layer, it classifies the image to 10 classes using two `Linear` layers. `ReLU` was used as the activation function.
- ColorNet (for CIFAR-10)
  - A model designed for processing color images (3×32×32).
  - It has deeper convolutional blocsk than `SourceNet`, with each block containing `BatchNorm2d` and `LeakyReLU` to promote more stable and faster training.
  - `Dropout` was applied in the classifier section to prevent overfitting.

2.2 Attack Algorithm Implementation

The four attack techniques specified in the requirements (Targeted/Untargeted FGSM, Targeted/Untargeted PGD) were implemented using an object-oriented approach to enhance code reusability and maintainability.

- FGSM Attacker (`FgsmAttacker`):
  - The `FgsmAttacker` class stores the model as an instance attribute.
  - **Targeted Attack**: It calculates the sign of the gradient to **minimize** the loss function (CrossEntropyLoss) and perturbs the image by `epsilon` to make it misclassified as the target label.

$$x_{\text{adv}} = x - \epsilon \cdot \text{sign}(\nabla_x J(\theta, x, y_{\text{target}})) \tag{1}$$

  - **Untargeted Attack**: It calculates the sign of the gradient to **maximize** the loss function, perturbing the image to move it as far as possible from its original label.

$$x_{\text{adv}} = x + \epsilon \cdot \text{sign}(\nabla_x J(\theta, x, y)) \tag{2}$$

- PGD Attacker (`PgdAttacker`)
  - The PGD attack is an iterative version of FGSM.
  - The `PgdAttacker` class performs the FGSM attack for a specified number of iterations (`num_steps`).

$$x_{\text{adv}}^{(t+1)} = \Pi_\epsilon \left(x_{\text{adv}}^{(t)} + \alpha \cdot \text{sign}(\nabla_x J(\theta, x_{\text{adv}}^{(t)}, y)) \right) \tag{3}$$

  - In all attack implementations, the `torch.clamp` function was used to ensure that pixel values remain within the valid image domain of `[0,1]`.

$$x_{\text{adv}}^{(t+1)} = \Pi_\epsilon \left(x_{\text{adv}}^{(t)} - \alpha \cdot \text{sign}(\nabla_x J(\theta, x_{\text{adv}}^{(t)}, y_{\text{target}})) \right) \tag{4}$$

3 Repository Structure and Execution

3.1 Structure of Repository

The repository is systematically organized by functionality:

- `run_experiment.py`: The main script to execute the entire experiment.
- `source/attacks/`: Attack algorithm implementations like `fgsm_attacker.py` and `pgd_attacker.py`.
- `source/models/`: Model architecture definitions like `mnist_net.py` and `cifar_net.py`.
- `source/core/`: Model training and evaluation logic, such as `runner.py`.

- `source/data/`: Data loading and pre-processing scripts like `data_provider.py`.
- `results/`: Directory where experiment configurations and results are saved as JSON files.
- `checkpoints/`: Directory where trained model weights are saved.

3.2 How to Run

- **Environment Setup**: Install the required Python packages using the `environment.yaml`. Using a conda environment would be recommended.
- **Execution**: Run the `run_all_experiments.sh` shell script to automatically and sequentially execute all experiments for both MNIST and CIFAR-10.

```
1) chmod +x run_all_experiments.sh
2) ./run_experiments.sh
```

- **Check Results**: Upon completion, JSON files containing hyperparameter settings, clean accuracy, and post-attack accuracy for each experiment will be generated in the `results/` directory.

4 Conclusion

In conclusion, this project successfully fulfills the core requirements of the assignment by implementing FGSM and PGD adversarial attacks and testing them on custom-built neural networks. The object-oriented design effectively modularizes the attack logic, and the automated execution scripts streamline the entire experimental process, ensuring reproducibility.

Beyond the functional implementation, this project underscores the importance of software reliability. A crucial aspect of this is fulfilling the assignment's requirement for a "test code" that verifies the implementation's correctness. While a manual demonstration can show functionality, adopting a formal testing framework like **pytest** would provide a more robust and systematic validation. This approach would not only guarantee the correctness of the attack logic in modules like `FgsmAttacker` but also serve as a critical safety net to prevent regressions during future modifications.

Therefore, this work represents not just the completion of an academic task, but the development of a functional, well-structured, and reliable software solution for studying the vulnerabilities of neural networks.