CIS 6261: Trustworthy Machine Learning
Project Option 1: Instructions

October 3, 2025

Introduction

Please read the instructions carefully.

In this project develop a defense technique to increase adversarial robustness and reduce privacy leakage of machine learning models, while minimally degrading their prediction quality (e.g., accuracy).

The project has two parts.

(1) You are given a target model trained on CIFAR-10 images and you have to protect it *without* retraining it (see ground rules below). You are provided with some attack examples that you can use to evaluate your approach. You will also want to also implement other (stronger) attacks to make sure that your defense is robust to them. We will evaluate your solution against attack examples provided but also other attacks (unknown to you).

(2) You will train a model on a dataset of your choice and apply your defense (and all that you have learned from part 1) to ensure that your model is robust and that its privacy leakage is limited. You will write a report evaluating your approach and explaining why it meets the goal.

A primary goal of this project is to get you thinking about *adaptive* attacks and defenses. We will evaluate not only your results but also your approach (i.e., its suitability, novelty, efficiency, etc.).

Project Files

The project archive contains the following files:

- `part1.py`. This file is the main Python code file for part 1. You will add to it to evaluate your approach.
- `utils.py`. This file defines useful functions.
- `data/`. This directory contains the data used to train the model as well as disjoint validation and test set, which you can use.
- `target_model.pt`. This is the saved target model file that your will protect in part 1. You should **not** modify this file.
- `advexp0.npz`. A file contain adversarial examples crafted for the target model.
- `examples/advexp_*.png`. These files contain examples of some of the adversarial examples images.

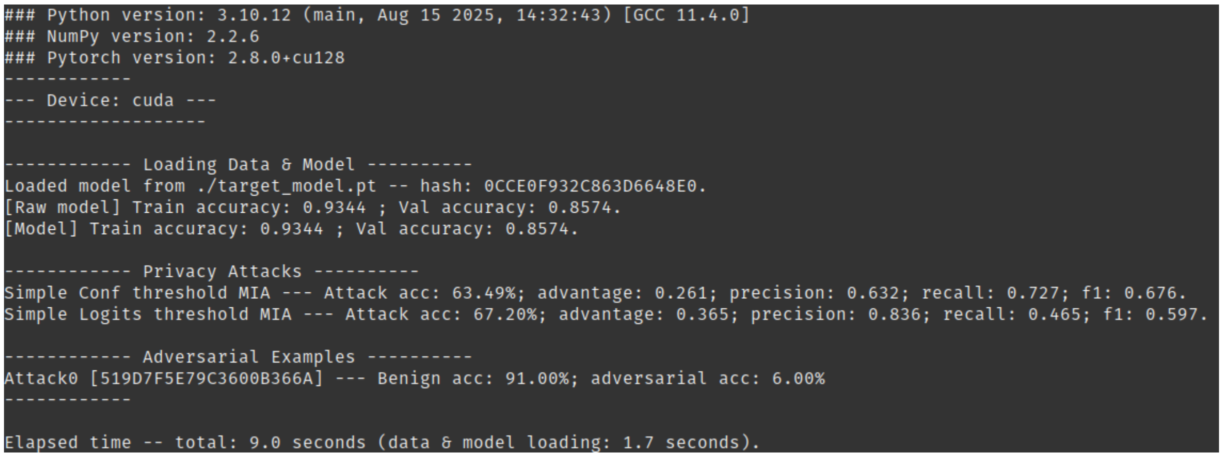
Note: You should use Python3 and PyTorch 2.X. You may use HiPerGator (if available) or your own system. The code is written so it runs on GPU (cuda) if available, and CPU otherwise.

Getting started

To get started I suggest that you study the project files and run `part1.py` (i.e., `python3 part1.py`). (You may need to install additional python packages to run it. I suggest using/creating a virtual environment for the course.)

The code will load the target model for you and evaluate its accuracy on the train and test datasets. It will then run and evaluate a membership inference attack on the target model. Finally, it will evaluate the adversarial robustness of the target model to the adversarial examples provided `advexp0.npz`.

Here is the output I get on my machine:



```
### Python version: 3.10.12 (main, Aug 15 2025, 14:32:43) [GCC 11.4.0]
### NumPy version: 2.2.6
### Pytorch version: 2.8.0+cu128
------------
--- Device: cuda ---
-------------------

------------ Loading Data & Model ----------
Loaded model from ./target_model.pt -- hash: 0CCE0F932C863D6648E0.
[Raw model] Train accuracy: 0.9344 ; Val accuracy: 0.8574.
[Model] Train accuracy: 0.9344 ; Val accuracy: 0.8574.

------------ Privacy Attacks ----------
Simple Conf threshold MIA --- Attack acc: 63.49%; advantage: 0.261; precision: 0.632; recall: 0.727; f1: 0.676.
Simple Logits threshold MIA --- Attack acc: 67.20%; advantage: 0.365; precision: 0.836; recall: 0.465; f1: 0.597.

------------ Adversarial Examples ----------
Attack0 [519D7F5E79C3600B366A] --- Benign acc: 91.00%; adversarial acc: 6.00%
------------

Elapsed time -- total: 9.0 seconds (data & model loading: 1.7 seconds).
```

As you can see the model is vulnerable to privacy attacks (e.g., we would like even stronger attacks to get an advantage near 0) and adversarial examples (e.g., we would like to see the adversarial accuracy near the benign accuracy).

You should spend some time studying the provided files. The model was trained on CIFAR10 data (`train_x`, `train_y`) and it is a little bit overfitted. For part 1, your defense should protect this specific model and **not** train a new one. This means you should implement your defense by defining a new prediction function (see `basic_predict()` in `part1.py`). The code in `part1.py` uses the `predict_fn` to access the model and evaluate the attacks. Therefore, if you replace `predict_fn` with your own function, the provided code will now evaluate the effectiveness of your defense.

If your defense changes the model's raw predictions (and it almost certainly will need to) the train/val accuracies of the raw model and the model (with your defense) will change.

Your goal for part 1 is to design a defense that maximally reduces the effectiveness of adversarial examples and membership inference attacks *while also* minimally reducing the accuracy of the model on adversarial examples and on the test data (e.g., test accuracy should not decrease much).

For part 1 you should start by adding code to `part1.py` to implement your defense. You should start working on part 2 only after you have made substantial progress in part 1.

1 Designing your Defense

You are free to use any approach that you think will work, but I do expect you to justify it. Your will explain your choices in the mid-semester report and final report.

You can take inspiration from techniques we talk about during the lectures and you can (you *should*) consider techniques from the research literature. However, you should abide by proper academic citation practices and ensure that you cite resources that you use (and do not claim credit for ideas/work that are not your own). You can cite publications like so Vapnik et al. [1] and non-publications (e.g., websites) like so[1].

2 Ground Rules

Please follow these rules. If you have a doubt whether something is allowed, be sure to ask.

- You cannot replace the provided model by training a new model from scratch (the idea is to protect the model provided). You may fine-tune the model (e.g., to implement adversarial training), but you should do so **without** modifying the saved model file. Also, you cannot expand the model's training set.

[1]You can use Google Scholar: https://scholar.google.com

- Your defense will slow down inference speed, but it has to run in reasonable time. For example, running `part1.py` takes less than three minutes on machines without a GPU. After your defense is implemented, it should not take more than 20 minutes. (This is so we can actually run your code ourselves and it finishes in reasonable time.)
- You can use both the training data and the validation data (`val_x`, `val_y`). However, you should **not** use the test data to avoid "overfitting" your solution on it.
- You will modify `part1.py` but you should **not** alter its attack evaluation logic.

3 What to do for Part 2?

The goal of part 2 is to implement a defense on a model of your choice on a dataset of your choice, provided the complexity is at least that of CIFAR10 (MNIST would be considered too simple). I suggest using an image dataset not too different from CIFAR10 (e.g., CIFAR100, ImageNette, EuroSAT, etc.) so can leverage work you did in part 1, but the choice is ultimately yours.[2]

As in part 1 the approach for the defense is up to you. But what we are looking for is that the approach is well justified. You should explain (in your final report) why you think your approach will be robust and how you evaluated it. For example, you may want to implement multiple strong adversarial examples and privacy attacks and evaluate them. So that you have some experimental evidence to justify that your defense is working as you expect.

4 Timeline

This is a recommended timeline and steps (these are only guidelines):

- Week of 10/6: explore the provided code and file. Meet with your group and setup an environment to run it. Start of thinking of your approach. Make sure you understand what you can and cannot do and how your approach will be evaluated.
- Week of 10/13: do literature search and brainstorming for ideas. Come up with an approach for the defense. Also identify two or three adversarial examples and (or) membership inference attacks that you can use.
- Week of 10/20: Test the feasibility of your defense ideas. Implement the attacks identified earlier so you can test your defense against them.
- Week of 10/27: Implement the chosen defense. Start writing the mid-semester report.
- Week of 11/3: Finalize implementation of defense. Test it extensively. Start work on part 2. Identify a dataset and model architecture. Train the model and ensure it performs as expected.
- Week of 11/10: Finish writing the mid-semester report and submit it. Continue to work on part 2. Be sure to consider training time approaches since you are allowed to use those for part 2.
- Week of 11/17: Implement your defense on the newly trained model. Prepare and run evaluation code. Start writing the final report.
- Week of 11/24: *Thanksgiving*
- Week of 12/1: Finalize the experiments. Finishing writing the report. Record your project presentation. Submit the report, presentation recording, and code/files.

References

[1] Vladimir Vapnik, Esther Levin, and Yann Le Cun. Measuring the vc-dimension of a learning machine. *Neural computation*, 6(5):851–876, 1994.

[2] I do not recommend you pick a dataset and model architecture that you do not the resources to comfortably work with. For example, training a large Vision Transformer from scratch on ImageNet-21K or LAION-5B would be too ambitious for most groups in this course.