CIS 6261: Trustworthy Machine Learning
Project Option 1: Instructions

February 15, 2023

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 likely want to also implement other (stronger) attacks to make sure that your defense is robust to them.) Your solution will be evaluated against attack examples provided but also other (unknown) attacks.

(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.

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.npz`. This file contains the data used to train the model as well as disjoint validation and test set, which you can use.
- `target-model.h5`. This is the saved target model file that your will protect in part 1.
- `advexp{0,1}.npz`. These files contain adversarial examples crafted for the target model.
- `examples/advexp_*.png`. These files contain examples of some of the adversarial examples images.

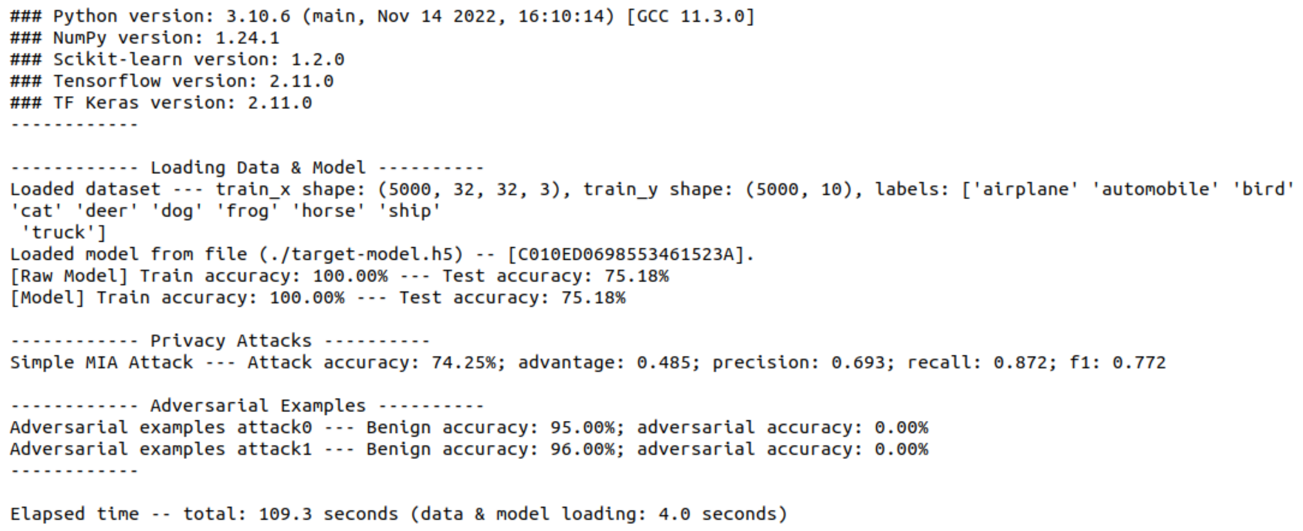
Note: You should use Python3 and Tensorflow 2. You may use HiPerGator or your own system.

Getting started

To get started I suggest that you study the project files and run `part1.py` (i.e., `python3 part1.py`). You can use HiPerGator or your own system. (You may need to install additional python packages to run it. I suggest 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 `advexp{0,1}.npz`.

Here is the output I get on my machine:



```
### Python version: 3.10.6 (main, Nov 14 2022, 16:10:14) [GCC 11.3.0]
### NumPy version: 1.24.1
### Scikit-learn version: 1.2.0
### Tensorflow version: 2.11.0
### TF Keras version: 2.11.0
------------

------------ Loading Data & Model ----------
Loaded dataset --- train_x shape: (5000, 32, 32, 3), train_y shape: (5000, 10), labels: ['airplane' 'automobile' 'bird'
'cat' 'deer' 'dog' 'frog' 'horse' 'ship'
 'truck']
Loaded model from file (./target-model.h5) -- [C010ED0698553461523A].
[Raw Model] Train accuracy: 100.00% --- Test accuracy: 75.18%
[Model] Train accuracy: 100.00% --- Test accuracy: 75.18%

------------ Privacy Attacks ----------
Simple MIA Attack --- Attack accuracy: 74.25%; advantage: 0.485; precision: 0.693; recall: 0.872; f1: 0.772

------------ Adversarial Examples ----------
Adversarial examples attack0 --- Benign accuracy: 95.00%; adversarial accuracy: 0.00%
Adversarial examples attack1 --- Benign accuracy: 96.00%; adversarial accuracy: 0.00%
------------

Elapsed time -- total: 109.3 seconds (data & model loading: 4.0 seconds)
```

As you can see the model is highly vulnerable to both privacy attacks (e.g., we would like the simple attack or even stronger attacks to get an accuracy close to 50%) and adversarial examples (e.g., we would like to see an adversarial accuracy close to the benign accuracy).

You should spend some time studying the provided files. The model was trained on CIFAR-10 data (`train_x`, `train_y`) and it is overfitted as you can see. For part 1, your defense should protect this specific model and not train a new one. This means you can implement your defense by defining a new prediction function (see `basic_predict()` in `part1.py`). The code in `part1.py` uses the `predict_fn` (defined on line 129) to access the model and evaluate the attacks. Therefore, if you replace `predict_fn` with your own function, the provided code will evaluate the effectiveness of your defense.

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 too 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 andf 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 and from the Internet. 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 [**?**] 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 (`train_x`, `train_y` are limited to 5000 examples on purpose).

[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 two minutes on machines without a GPU. After your defense is implemented, it should not take more than 30 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 Timeline

With the mid-semester report due on 3/31, here is a recommended timeline and steps (these are only guidelines):

- Week of 2/20: 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 2/27: 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 3/6: Test the feasibility of your defense ideas. Implement the attacks identified earlier so you can test your defense against them.
- Week of 3/13: *Spring Break*
- Week of 3/20: Implement the chosen defense. Start writing the mid-semester report.
- Week of 3/27: Finalize implementation of defense. Test it extensively. Finish writing the mid-semester report and submit it.
- Week of 4/3: Switch to part 2. Identify a dataset and model architecture. Train the model and ensure it performs as expected.
- Week of 4/10: Implement your defense on the newly trained model. Prepare and run evaluation code. Start writing the final report.
- Week of 4/17: Finalize the experiments. Finishing writing the report. Record your project presentation. Submit the report, presentation recording, and code/files.