RIAI 2020 Course Project

The goal of the project is to design a precise and scalable automated verifier for proving the robustness of fully connected and convolutional neural networks with rectified linear activations (ReLU) against adversarial attacks. We will consider adversarial attacks based on the L_∞-norm perturbations. A network is vulnerable to this type of attack if there exists a misclassified image inside an L_∞-norm based ε-ball around the original image.

We will leverage the DeepPoly relaxation [1] for building our verifier. The ReLU transformer introduced in [1] and explained in the lecture is a DeepPoly transformer with a minimum area, if one considers a single neuron at a time. However, there may exist transformers with larger area, which can verify more images than the transformer with minimum area. Concretely, the ReLU transformer is parametrized by the slope λ of the linear lower bound and it is sound for any value λ in [0, 1]. This value is chosen using a predefined formula as explained in the lecture. In this project, your task is to improve the ReLU transformer in DeepPoly. Your goal is to learn a value λ for each neuron in the network which maximizes the precision of the verifier. Given a test case (an image and ε value) and a network, your algorithm should first produce new ReLU transformer (by learning λ) for each neuron in the network and then run the verification procedure using these transformers.

You are **allowed** to learn an ensemble or a set of different λ, run DeepPoly verification using each λ separately, and then use the information from all runs to prove the robustness. Formally, you are allowed to verify an image if you can prove that all outputs in the intersection of final convex shapes computed using different sets of λ are correctly classified. You are **not allowed** to improve the precision by combining (e.g. intersecting) the polyhedra from different λ *before* the final layer of the network.

Project Task

The input to your verifier is a neural network and a test case. The output is the result of your verification procedure (verified/not verified). Below we give more details.

Networks: We will run your verifier on 7 fully connected and 3 convolutional networks with ReLU activations. The networks are trained using different training methods (standard training, PGD, DiffAI). The architectures and weights of these networks will be provided together with the code release. The architectures are encoded as PyTorch classes in `networks.py`. All trained networks are provided in the folder `mnist_nets`.

Example Test Cases: The example test cases consist of 20 files from the MNIST test set formatted to be used by the verifier. Each file contains 785 rows. The first row of the file denotes the label of the image and the remaining rows describe the image pixel intensity in the range [0,1]. All example test cases are stored at folder `mnist_specs`. The epsilon value of a test case can be deduced from its name, e.g., the file `img0_0.001.txt` defines the 0.001-ball around image `img0`. We provide those example test cases for you to develop your verifier and they are **not the same** as the ones we will use for the final grading. Note that images have pixel intensities between 0 and 1, e.g. if perturbation is 0.2 and pixel has value 0.9 then you only

have to verify range [0.7, 1.0] for intensities of this pixel, instead of [0.7, 1.1]. Filegt.txt contains ground truth for each example.

Verifier: The directory `verifier` contains the code of the verifier (file `verifier.py`). The verifier addresses the following problem:

Inputs:
- A fully connected or convolutional neural network
- A test case

Output:
- `verified`, if the verifier *successfully* proves that network is robust for the given test case
- `not verified`, if the verifier *fails* to prove that network is robust for the given test case

Your verifier will be executed using the following command, where <network file> is replaced by a path to the network file and <test case file> is replaced by a file containing the test case:

```
$ python3 verifier.py --net <network file> --spec <test case file>
```

Your task is to implement a sound and precise verifier by modifying the "analyze" function. That is, the verifier should verify the robustness. To this end, you must build upon the DeepPoly relaxation [1] as explained before. Your verifier must be **sound**: it must never output `verified` when the network is not robust to a test case. It should be **precise:** it should try to verify as many test cases as possible while keeping soundness and scalability (we will use a time limit of 3 minutes per test case).

Testing Conditions and Grading

Your verifier will be tested on the following conditions:
- Inputs to the networks are images from the MNIST dataset.
- ε values range between 0.005 and 0.2.
- Configuration of the machine used for testing: Intel Xeon E5-2690 v4 CPUs and 512GB RAM. We will use 14 threads and memory limit 64GB to verify each test case.
- Your verifier will be executed using Python 3.7

The verifier will be graded based on the precision and soundness of your verifier:
- You start with 0 points.
- **You receive 1 point** for any verification task for which your verifier correctly outputs `verified` within the time limit.
- **You will be deducted 2 points** if your verifier outputs `verified` when the network is not robust for the provided test case.
- If your verifier outputs `not verified` you receive 0 points. This means that the maximum number of points that can be achieved by any solution may be less than 100.
- If there is a timeout or memory limit exceeded on a verification task, then the result will be considered as `not verified`.

We do not require your verifier to be floating-point sound and we guarantee that full points can be obtained using torch.float32. If you have questions about the grading, please send an email to mislav.balunovic@inf.ethz.ch.

Requirements

You should obey the following rules otherwise you may get **0 points** for the project:

1. The implementation must be in Python 3.7.
2. You must use the DeepPoly relaxation. No other relaxations are allowed.
3. You are not allowed to check for counter-examples using any kind of adversarial attack.
4. The only allowed libraries are PyTorch 1.7.0, Torchvision 0.8.1, Numpy 1.19.4, Scipy 1.5.3 and Python Standard Library. Other libraries are not allowed and will not be installed on the evaluation machine.

Deadlines

Event	Deadline
Project announcement	November 4, 2020
Skeleton code release	5:59 PM CET, November 4, 2020
Group registration	5:59 PM CET, November 11, 2020 (use this form to register; send an email to jingxuan.he@inf.ethz.ch if you want to be matched with other students)
Preliminary submission (optional)	5:59 PM CET, December 1, 2020
Preliminary feedback	5:59 PM CET, December 3, 2020
Final submission	**5:59 PM CET, December 18, 2020**

Groups can submit their project by the preliminary submission deadline to receive feedback. We will run your verifier on 25 out of 100 test cases which will be used for the final grading and report to you, for each test, the ground truth, output and the runtime of your verifier. The feedback will be sent by 11:59 PM CET, December 3, 2020. Your preliminary submission results do not affect your final project score.

Submission

After the group registration deadline on November 11, each group is going to receive an invitation to GitLab repository of name **ddd-riai-project-2020** where **ddd** here is group number that will be assigned to you. This repository will contain template code, networks and test cases (content is the same as the zip file released on the course website on November 4).

You should commit your solutions to the assigned repository. After the final submission deadline, we will treat the assigned repository as your submission and evaluate your solution in the repository on the final test cases. Note that we are **not going to accept** any submission by email. Below are detailed instructions for final submission. You should obey the rules in the instructions otherwise you may be **deducted points**:

1. Before the final submission deadline, you should prepare your code in the **master** branch of the assigned repository. The code should:
    a. only print `verified` or `not verified`. If additional messages are printed, we will take the last line as your output.
    b. follow the **Requirements** section of this document. Especially, you should only use the allowed libraries. In the final evaluation, importing libraries not allowed will crash the verifier as they are not installed in the evaluation environment, resulting in **0 points**.
2. In addition to the code, you should commit a 1 page write up about your verifier in PDF format.
3. At the deadline time, we will pull your solution and put it into the **final_submission** branch. You **should not commit** to the final_submission branch. After the final evaluation, we will put the evaluation result into the **final_submission** branch.

References

[1] Singh, Gagandeep, Timon Gehr, Markus Püschel, and Martin Vechev. "An abstract domain for certifying neural networks." *Proceedings of the ACM on Programming Languages* 3, no. POPL (2019) https://www.sri.inf.ethz.ch/publications/singh2019domain