RIAI 2021 project

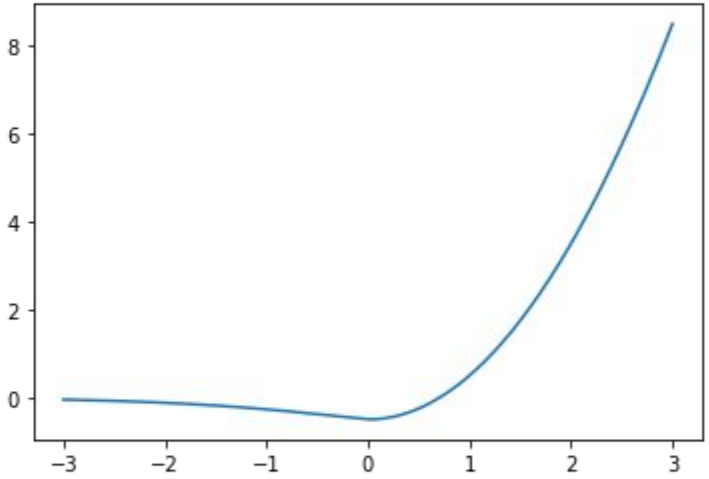
Project: Design precise and scalable verifier

- Build on DeepPoly
- Fully-connected networks
- L-infinity perturbations
- **New activation function**: Sigmoid-Parabola Unit (SPU)

Sigmoid-Parabola Unit (SPU)

SPU activation is defined piecewise:

$$\mathrm{SPU}(x) = \begin{cases} x^2 - 0.5, & \text{for } x \geq 0 \\ \mathrm{sigmoid}(-x) - 1, & \text{for } x < 0 \end{cases}$$



This project: DeepPoly SPU transformer

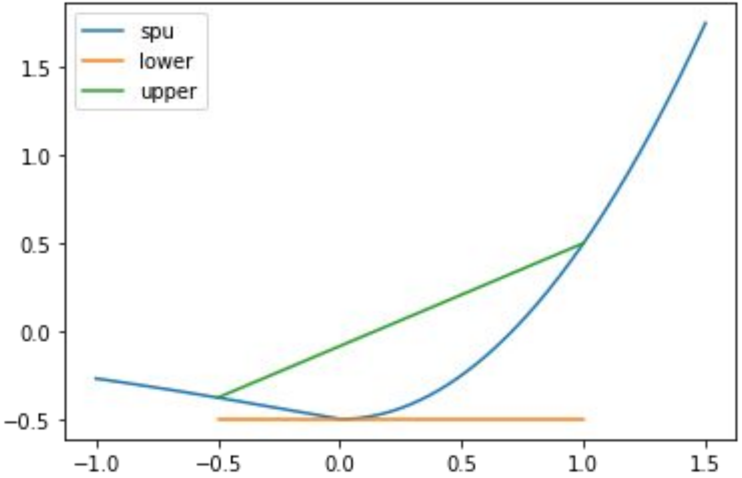
In this project, your goal is to come up with a DeepPoly transformer for SPU.

At a high level, given some interval [l, u] your transformer should compute **sound** linear lower bound $w_L x + b_L$ and linear upper bound $w_U x + b_U$, meaning that:

$$\text{for all } x \text{ in } [l, u],\ w_L x + b_L \leq \mathrm{SPU}(x) \leq w_U x + b_U.$$

Example of linear bounds

Example of a sound linear lower bound and linear upper bound for [l, u] = [-0.5, 1]



Goal

Goal is to create a **sound** transformer which certifies as many images as possible when used as part of the end-to-end DeepPoly certification.

Two examples of heuristics already seen in the lecture for ReLU:

1. **Box:** corresponds to $w_L = 0$ and $w_U = 0$.
2. **Min-area heuristic:** bounds with the minimum area between the bound and the function

Note: These two transformers are generally non-comparable: for different examples, different heuristic might be better.

Networks

- We will run your verifier on 10 fully connected networks with SPU 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.

Example Test Cases

- The example test cases consist of 20 files from the MNIST test set formatted to be used by the verifier (2 examples per network).

- Each file contains 785 rows, describing label of the image and pixel intensities in the image. The epsilon value of a test case can be deduced from its name, e.g., the file img0_0.003.txt defines the 0.003 L-infinity -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.

Verifier

- The directory verifier contains the code of the verifier (file verifier.py). The verifier addresses the following problem:
- Inputs: Network + Test Case
- Output: “verified” or “not verified”
- Example command:

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

Test Conditions and Grading

- 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 1 minute per test case).

Test Conditions and Grading

- Inputs to the networks are images from the MNIST dataset.
- Perturbation radiuses range between 0.001 and 0.3.
- 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

Grading

- 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.

Requirements

- The implementation must be in Python 3.7.
- You must use the DeepPoly relaxation. No other relaxations are allowed.
- You are **not allowed** to check for counter-examples using any kind of adversarial attack.
- The only allowed libraries are PyTorch 1.10.0, Torchvision 0.11.1, Numpy 1.19.5 and Python Standard Library. Other libraries are not allowed and will not be installed on the evaluation machine.

Deadlines

Project announcement	November 3
Code release	11:59 PM CET, November 3
Preliminary submission (optional)	5:59 PM CET, December 1
Preliminary feedback	5:59 PM CET, December 3
Final submission	**5:59 PM CET, December 20**

Submission details

Each group is going to receive an invitation to GitLab repository of name **ddd-riai-project-2021** 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 3). See details on submission procedure in the official project description.

Preliminary submission

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 5:59 PM CET, December 3, 2021. Your preliminary submission results do not affect your final project score.

Next week: Project Q&A in Exercise session

- Monday, 12-14 via Zoom
  Wednesday, 12-14 via Zoom (usual exercise slots)

- Please prepare questions beforehand.

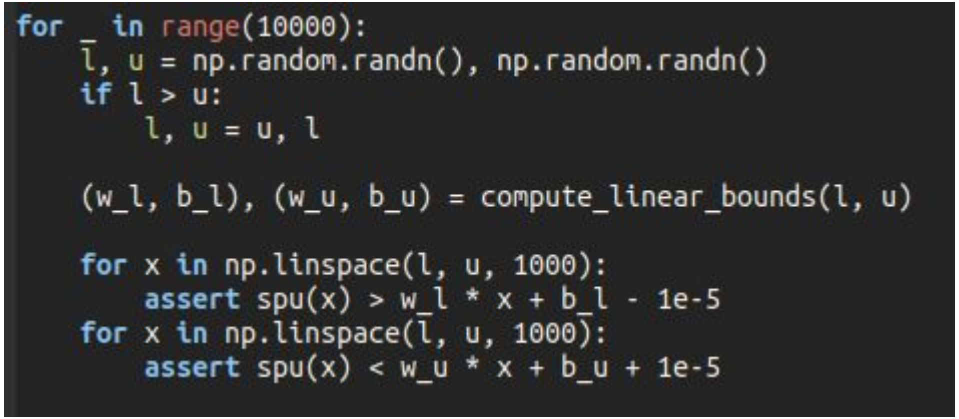
Advices

Key to get the good solution is combining both good **research** and **engineering** practices:

- Testing soundness of SPU transformer
- Testing soundness of end-to-end verifier
- Coming up with several different heuristics
- Measuring performance of different heuristics

Advices

Testing soundness of your SPU DeepPoly transformer:



```
for _ in range(10000):
    l, u = np.random.randn(), np.random.randn()
    if l > u:
        l, u = u, l

    (w_l, b_l), (w_u, b_u) = compute_linear_bounds(l, u)

    for x in np.linspace(l, u, 1000):
        assert spu(x) > w_l * x + b_l - 1e-5
    for x in np.linspace(l, u, 1000):
        assert spu(x) < w_u * x + b_u + 1e-5
```

Advices

Testing soundness of your end-to-end DeepPoly verifier:

1. Sample an image from MNIST dataset
2. Run adversarial attack
3. Run your DeepPoly verifier
4. If you reported “verified”, and attack found an adversarial example, your verifier is unsound

Advices

- We strongly encourage you to have a preliminary submission at December 1st.
- Try the code and start working on the project so you can ask questions in the exercise session next week.

Good luck!

Questions: Via Moodle (preferred) or send an e-mail to mislav.balunovic@inf.ethz.ch