Course Project for RIAI 2018

The goal of the project is to design a precise and scalable automated verifier for proving the robustness of fully connected feedforward neural networks with rectified linear activations (ReLU) against adversarial attacks. We will consider adversarial attacks based on the L_∞-norm based perturbations. This type of attack tries to find a misclassified image inside a L_∞-norm based ε-ball around the original image. We will focus on leveraging both the Interval domain [3] and Linear programming [2] for designing our verifier. There exists a tradeoff between the precision of an abstract domain and its speed. For example, certification with the Interval domain is fast but the results can often be imprecise, i.e., domain fails to prove robustness when the network is robust for a given L_∞-norm based perturbation. On the other hand, Linear programming enables more precise verification than the Interval domain but it can be slow compared to the Interval domain for large networks. The challenge is thus to combine the speed of Interval domain with the precision of Linear programming to achieve a better tradeoff for the certification of neural network robustness.

Project Task

You are provided with a virtual machine containing a Python code skeleton that implements a sound Interval domain based verifier for feedforward neural networks. The credentials for using the virtual machine are:

Username: riai2018
Password: Project@2018

The directories in the virtual machine are organized as follows:

mnist_images: contains 100 files from the MNIST testset formatted to be used by our analyzer. Each file contains 785 rows containing intervals. The first row of the file denotes the label of the image and the remaining rows describe the pixel intensity in the range [0,1].

mnist_nets: contains 10 trained fully connected feedforward networks with ReLU activation for the MNIST dataset. The number of hidden layers and the number of hidden units per layer in a network can be deduced from its name, e.g., the file “mnist_relu_3_50.txt” contains 3 hidden layers each containing 50 hidden units.

ELINA: contains source code of the interval domain implementation from the ELINA library.

Gurobi810: contains the code for the Gurobi linear solver.

analyzer: This directory contains the verifier code in the file “analyzer.py”. The function “analyze” runs interval analysis for robustness verification.

The verifier addresses the following problem:

Inputs:

- A fully connected feedforward neural network
- An image
- An ε value

Output:

The verifier runs the analysis on the original image first to determine if the neural network classifies the image correctly. If the image is not classified correctly then the verifier outputs an error message and exits. Otherwise, the verifier runs the Interval analysis on the input image perturbed by the provided epsilon and outputs:

- "verified" if the verifier successfully proves that network is robust for the given image and ε; otherwise, it outputs "cannot be verified".

Additionally, the verifier also outputs the timing information in seconds.

Instructions:

1. Import the virtual machine and configure the settings such as the amount of RAM:
    a. https://docs.oracle.com/cd/E26217_01/E26796/html/qs-import-vm.html
    b. We recommend at least 2 GB of RAM.
2. Go to the "ELINA" folder and execute "make" followed by "sudo make install".
3. Go the "analyzer" folder and run the commands specified in the "setup_gurobi.sh" file to setup the necessary paths for the Gurobi linear solver.
4. In the "analyzer" folder, run the analysis by typing "python3 analyzer.py --netname --image epsilon".

Your task is to improve the precision of the provided verifier by modifying the "analyze" function. That is, the verifier should verify the robustness for more general cases (e.g. bigger ε values). To this end, you must augment the verifier with Linear programming by designing heuristics that use the Interval domain for analyzing some parts of the neural network and Linear programming for the remaining ones. An example of a heuristic is to use the Interval domain for analyzing the first k layers then use the analysis results for analyzing the remaining layers with a Linear solver (for another heuristic the Linear solver can be applied before the Interval domain). Your verifier must be sound: it must never output "verified" when the network is not robust. Moreover, it must remain efficient: we will use a time limit of 7 minutes per verification task.

Calling ELINA API for updating Intervals with the bounds from the linear solver:

If you want to update the bounds for dimension "i" from [c,d] to newly refined bounds [a,b], you need to intersect the abstract element "element" with two constraints "x0-a>=0" and "b-x0>=0". The ELINA code that does this is below:

```
#create an array of two linear constraints
lincons0_array = elina_lincons0_array_make(2)

#Create a greater than or equal to inequality for the lower bound
lincons0_array.p[0].constyp = c_uint(ElinaConstyp.ELINA_CONS_SUPEQ)
linexpr0 = elina_linexpr0_alloc(ElinaLinexprDiscr.ELINA_LINEXPR_SPARSE, 1)
```

```
cst = pointer(linexpr0.contents.cst)

#plug the lower bound “a” here
elina_scalar_set_double(cst.contents.val.scalar, -a)
linterm = pointer(linexpr0.contents.p.linterm[0])

#plug the dimension “i” here
linterm.contents.dim = ElinaDim(i)
coeff = pointer(linterm.contents.coeff)
elina_scalar_set_double(coeff.contents.val.scalar, 1)
lincons0_array.p[0].linexpr0 = linexpr0

#create a greater than or equal to inequality for the upper bound
lincons0_array.p[1].constyp = c_uint(ElinaConstyp.ELINA_CONS_SUPEQ)
linexpr0 = elina_linexpr0_alloc(ElinaLinexprDiscr.ELINA_LINEXPR_SPARSE, 1)
cst = pointer(linexpr0.contents.cst)

#plug the upper bound “b” here
elina_scalar_set_double(cst.contents.val.scalar, b)
linterm = pointer(linexpr0.contents.p.linterm[0])

#plug the dimension “i” here
linterm.contents.dim = ElinaDim(i)
coeff = pointer(linterm.contents.coeff)
elina_scalar_set_double(coeff.contents.val.scalar, -1)
lincons0_array.p[1].linexpr0 = linexpr0

#perform the intersection
element = elina_abstract0_meet_lincons_array(man,True,element,lincons0_array)
```

You will have to put in your values of "i", "a", and "b" in the above code snippet.

You can work on the project in groups consisting of at most 2 students. To register as a group, please send the group name along with the name and emails of the group members to: gsingh@inf.ethz.ch

Grading

The project 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 1 point if your verifier outputs “verified” when the network is not robust for the provided image and ε
- If there is a timeout on a verification task, then the result will be considered as “cannot be verified”

The inputs your verifier will be tested on satisfy the following conditions:

- neural networks that have at least 3 and at most 9 fully connected feedforward layers, each layer consisting of between 20 to 2000 neurons
- images from the MNIST dataset
- ε values ranging between 0.005 and 0.1

Requirements

1. The implementation must be in Python and it should be single-threaded.
2. The Interval domain implementation must use the Python interface of ELINA[1].
3. The Linear solver must be based on the Python interface of the Gurobi framework [2]. The virtual machine will have the linear solver installed. You are free to use any feature of the Linear solver.
4. No other abstract domains are allowed.
5. No additional libraries are allowed besides the ones provided with the skeleton.
6. All submitted projects will be graded by running them on a 3.3 GhZ Skylake machine. We will provide each group 1 hour access to the machine to check how the performance on our machine compares with yours.

Dates

→ Project announcement: October 29, 2018
→ Group registration deadline: November 3, 2018 (to register, send an email to: gsingh@inf.ethz.ch)
→ Group announcement (via email): November 5, 2018
→ Release of Skeleton: November 5, 2018
→ There will be discussion on the encoding of the neural network robustness based on Linear solvers during the 2nd half of the lecture on November 5, 2018.

Project submission (via email): Please submit your modified “analyzer.py” file along with a one page write up about your heuristics. Please make sure that your code runs and reproduces your results on the virtual machine that we provide. The deadline is 11:59 PM, December 20, 2018

References

[1] ELINA: ETH library for numerical analysis. http://elina.ethz.ch/
[2] Gurobi optimization. http://www.gurobi.com/