Prolip Artifact Instructions

Artifact Instructions

0.1 Description

The PROLIP algorithm analyzes the behavior of a neural network in a user-specified box-shaped input region and computes: (i) lower bounds on the probabilistic mass of such a region with respect to the generative model (ii) upper bounds on the Lipschitz constant of the neural network in this region, with the help of the local Lipschitzness analysis.

0.2 Virtual Machine Setup

Open VirtualBox. Click on File –> Import Appliance and choose "prolip.ova" as the file to import. This will set up the virtual machine in Virtual Box with 2048 MB of memory and 1 CPU allocated to it. Click "Start" to boot the virtual machine. Sign in to User with the password "password". There is no password for the root user if you require root access. Open the terminal and navigate to the "∼/prolip" directory.

0.3 Artifact Directory Structure

- `experiment.py` is used to interact with our tool through the command-line, and it is responsible for generating the output .csv and .png files for each experiment.
- `boxprop.py` contains the "Box" class, which represents a box-shaped input region for a neural network layer. The "Box" class contains functions to propagate the "Box" through various types of layers in a neural network, and these functions are called from `onnx_to_boxprop.py`.
- `lipprop.py` contains the "Lip" class, which represents a range of Jacobian matrices. The "Lip" class contains functions to propagate the Jacobian range through various types of layers in a neural network, and these functions are called from `boxprop.py`.
- `onnx_to_boxprop.py` takes an ONNX model and a "Box" object as inputs and calls methods from `boxprop.py` to propogate the "Box" through the ONNX network.
- `pretrained_generators` contains one CIFAR-10 generator model and one MNIST generator model.
- `pretrained_classifiers` contains two CIFAR-10 classifier models and one MNIST classifier model.

- `precomputed_results` contains the expected results from running the experiments listed below in section 0.5.
- `requirements.txt` lists the required python dependencies.

0.4 Understanding the Command-Line Tool

`experiment.py` takes multiple arguments to handle different experiment scenarios: "python3 experiment.py `--genname` <path name to generator network> `--clfname` <path name to classifier network> `--boxsizes` <list of ints> `--numcenters` <int greater than 0> `--randomseed` <int> `--outfile` <name of output files>"

- `<genname>`: path name to ONNX generator model (must have .onnx extension)
- `<clfname>`: path name to ONNX classifier model (must have .onnx extension)
- `<boxsizes>`: list of integers specifying the different box sizes to run the `PROLIP` algorithm on (default is 0.0001 0.001 0.1)
- `<numcenters>`: specifies the number of random centers of boxes to run the `PROLIP` algorithm on (default is 1)
- `<randomseed>`: specifies the seed for generating random tensors corresponding to the centers of the random boxes (default is 0)
- `<outfile>`: name of output .png and .csv files (default is "out")

0.5 Commands for Running Experiments from the Paper

- `python3 experiment.py --genname` ./pretrained_generators/mnist/mnist_g.onnx `--clfname` ./pretrained_classifiers/mnist/mnist_f.onnx `--numcenters` 5 `--outfile` mnist
- `python3 experiment.py --genname` ./pretrained_generators/cifar/cifar_g.onnx `--clfname` ./pretrained_classifiers/cifar/small_cifar_f.onnx `--numcenters` 5 `--outfile` small_cifar
- `python3 experiment.py --genname` ./pretrained_generators/cifar/cifar_g.onnx `--clfname` ./pretrained_classifiers/cifar/large_cifar_f.onnx `--numcenters` 5 `--outfile` large_cifar

Note that in our paper, we run the experiments on a GCP VM instance with 208 GB RAM and 32 vCPU's. The first two commands will run properly with the current virtual machine configuration; however, the third command requires much more RAM to run. You will need to allocate at least 100 more GB of memory to the virtual machine to run the third command properly.

Each command generates a .csv file and a .png file. The .csv file contains the randomly generated centers of the boxes, the box sizes, the Lipschitz constants for the corresponding boxes, and the run times for the corresponding run of the `PROLIP` algorithm. The .png file contains a bar graph displaying the run times of the `PROLIP` algorithm corresponding to Figure 5 in the paper.

The first command will produce the results of running the `PROLIP` algorithm on the MNIST program, and it should take a total of 45 seconds to run. The second command will produce the

results of running the PROLIP algorithm on the small CIFAR-10 program, and it should take a total of 2.5 minutes to run. The third command will produce the results of running the PROLIP algorithm on the large CIFAR-10 program, and it should take 2.5 hours to run on hardware that meets the requirements stated earlier.

0.6 GitHub Repository

https://github.com/ksarangmath/prolip