Readme Document for DeepPAC

Please Refer to ICSE 2022 Artifact Evaluation Abstract PDF first.

1 Guide to use the Artifact

All experiments are conducted on a Windows 10 PC with Intel i5-11500, RTX 3060, and 16G RAM. So we recommend the reproduction environment should have similar hardware resources, and run Windows 10 system. The GPU version of PyTorch and corresponding CUDA-Toolkit are needed.

1.1 Artifact Installation

We assume that you extract the compressed file into `~/DeepPAC`.

1.1.1 Physical Machine Installation

We recommend you use the Conda[1] for dependency installation. We have provided a `requirements.txt` to help you install all the dependency as follow:

```
conda create -n DeepPAC --file requirements.txt \
-c conda-forge -c gurobi -c pytorch
```

Or you can use the following code to install the dependency.

```
conda config --add channels conda-forge
conda create -n DeepPAC python=3.7.8
conda activate DeepPAC
conda install pkgconfig
conda install pytorch torchvision torchaudio cudatoolkit=11.1 -c pytorch
conda install pillow scipy numpy matplotlib foolbox tqdm scikit-learn
conda install cvxopt cvxpy-base cvxpy glpk blas libblas libcblas
conda config --add channels http://conda.anaconda.org/gurobi
conda install gurobi
```

We use the Gurobi as the linear programming (LP) solver, which is academic free. You will need a normal academic license for Gurobi [2].

[1] Miniconda: https://docs.conda.io/en/latest/miniconda.html
[2] Gurobi License: https://www.gurobi.com/academia/academic-program-and-licenses/

1.1.2 Using the Docker Image

For the artifact evaluation, we provides a docker image for Linux that has already initialized with all dependencies. The exported docker container can be found in https://drive.google.com/file/d/1LiyMdNrwB5AYVSCgnErRralTybEdTNGz/view?usp=sharing. However, some extra steps are needed for preparing the docker environment. The overall requirements are listed below:

- Linux Kernel: $>$ 3.10
- Docker: $>=$ 19.03
- RAM: 16GB (or better)
- GPU: Nvidia, Architecture $>=$ Kepler, 6GB memory at least and supports CUDA 11.1
- Nvidia Driver: $>=$ 418.81.07

Nvidia Support for Containers DeepPAC utilises the GPU to sample on large DNNs, so the Nvidia Container Toolkit is required for Docker. The tutorial can be found here[3]. Note you can run the docker image without GPU support and only use CPU. However, it will be extremely slow on large DNNs like ResNets.

Gurobi Licenses for Containers The Gurobi package is already included in the docker image but without a valid license. Gurobi is academic free and you can apply a container academic license on the Gurobi Web License Manager[4] after registered. When you have the license file (`gurobi.lic`), you will need to mount it in `/opt/gurobi` in the docker container. The tutorial can be found in section 4.3 in Gurobi documentation[5]. Note you have to connect to a recognized academic institution network such as a university network for the license. If you have problems with Gurobi, we also provide an open-source solver CBC. However, CBC could be much slower than Gurobi (up to 52 times slower).

Create the Docker Container To run the DeepPAC in docker, you can then import the docker image from the tar ball and create a container correspondingly.

```
sudo gunzip -c DeepPAC.tar.gz | sudo docker \
import - deeppac:v1
sudo docker run -it --gpus all \
--security-opt seccomp:unconfined deeppac:v1 bash
cd ~/DeepPAC
```

[3] https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html#docker

[4] Gurobi WLS: https://license.gurobi.com/manager/licenses

[5] https://www.gurobi.com/wp-content/plugins/hd_documentations/documentation/9.5/quickstart_linux.pdf

Note the flag secomp:unconfined boost the performance of python programs in containers. DeepPAC could be 100% slower without the flag.

1.2 Usage

Check the guidelines and augments of DeepPAC using following code:

```
cd ~/DeepPAC
conda activate DeepPAC
python . -h
```

We provide some examples to facilitate the use of DeepPAC. You can modify the augments of the examples to verify other networks on different inputs with different parameters. You can find the corresponding network files in `models` folder according to Sect. 1.4.

MNIST Case Verify the PAC-model robustness of the CNN1 on the first image of MNIST training set with the radius 1, error rate 0.01, and confidence 0.999. The number of samples used for first focused learning phase is 2000, and 7000 for second phase. We use Gurobi as LP solver.

```
python . -ncf models.mnist_ERAN_convs \
-nc ConvSmallNetwork \
-m ./models/convSmallRELU__Point.pth -d mnist \
-r 1 -eps 0.01 -eta 0.001 -ind 0 -train \
-FT 2000 -ST 7000 -solver gurobi
```

CIFAR-10 Case Verify the PAC-model robustness of the Resnet18 on the 10th image of CIFAR-10 testing set with the radius 4, error rate 0.01, and confidence 0.99. The number of samples used for first focused learning phase is 20000, and 8000 for second hase. Here we use GPU to sample on Resnet18 and set the batchsize to be 200. We use CBC as LP solver this time.

```
python . -ncf models.resnet -nc ResNet18 \
-m ./models/Cifar_ResNet18_adv_trim.pth -d cifar10 \
-r 4 -eps 0.01 -eta 0.01 -ind 9 -gpu \
-FT 20000 -ST 8000 -bsize 200 -solver cbc
```

Imagenet Case Verify the PAC-model robustness of the Resnet50a on a given church image with the radius 3, error rate 0.005, and confidence 0.999. The number of samples used for stepwise splitting is 18000. Here we use GPU to sample on Resnet50 and set the batchsize to be 200.

```
python . -ncf torchvision.models -nc resnet50 \
-m ./models/imagenet_linf_4.pth -d imagenet \
-img ./ImageNet_Samples/church_0.jpeg \
-r 3 -eps 0.005 -eta 0.001 -gpu \
-b 18000 -bsize 200
```

1.3 Reproduce the Result

All experiments are conducted on a physical machine with Intel I5-11500, Nvidia RTX 3060 and 16GB RAM, and the results are presented in `logs` folder. The running time is in the bottom of the log file. Note different running environment may bring a little change in the results. We provide 5 python files to reproduce all experiments.

MNIST Maximum Robustness Radius The first experiment calculates the maximum robustness radius $r_{\max}$ for different networks on MNIST dataset (See Fig 6). We use `mnist_experiments.py` to reproduce this experiment. It has four augments

- -n: the network to be verified (CNN1-6, FNN1-6),
- -gpu: set to use GPU,
- -inc: set to verify the properties with the radius $r_{\max} + 1$,
- -solver: choose the LP solver (gurobi, cbc).

For instance, examine the results of the maximum robustness radius of FNN1 as below:

```
python mnist_experiments.py -n FNN1 -solver gurobi
### 25 PAC-Model Robust out of 25 cases.
python mnist_experiments.py -n FNN1 -inc -solver gurobi
### 0 PAC-Model Robust out of 25 cases.
```

The result shows the maximum robustness radius of FNN1 in our previous experiment is correct. It costs nearly 2400 seconds per run in our settings.

CIFAR Robustness Rate The second experiment estimates the robustness rate of CIFAR-10 CNNs on 100 images with radius $2, 4, 6, 8$ (See Fig.7). We use `cifar_batch.py` to reproduce this experiment. It has four augments

- -n: the network to be verified (CNN1-3),
- -gpu: set to use GPU.
- -r: the radius of the L_∞ ball $(2, 4, 6, 8)$,
- -solver: choose the LP solver (gurobi, cbc).

Simply run the python file with augments to estimate the robustness rate. It costs nearly 20000 seconds per run in our settings.

CIFAR Verification with Different Parameters The third experiment tries to calculate the maximum robustness radius of the same image with different parameters (See Tab.2). We use `cifar_different_parameters.py` to reproduce this experiment. It has four augments

- -n: The network to be verified (ResNet18, ResNet50, ResNet152),
- -gpu: set to use GPU,
- -ind: the image index (0-4),
- -solver: choose the LP solver (gurobi, cbc).

Simply run the python file with augments to check the maximum robustness radius with different parameters. It costs nearly 2300 seconds per run in our settings.

Imagenette Targeted Verification The fourth experiment tries to verify 20 images with the radius 4 on a subset of imagenet dataset (imagenette) using targeted score difference function (See Targeted Method of Tab.3). We use `imagenette_experiments.py` to reproduce this experiment. It has two augments

- -n: the network to be verified (ResNet50a, ResNet50b),
- -gpu: set to use GPU.

Simply run the python file with augments to check the verification results. It costs nearly 28000 seconds per run in our settings.

Imagenet Untargeted Verification The fifth experiment tries to verify 50 images with the radius 4 on imagenet dataset using untargeted score difference function (See Untargeted Method of Tab.3). We use `imagenet_experiments.py` to reproduce this experiment. It has two augments

- -n: The network to be verified (ResNet50a, ResNet50b),
- -gpu: set to use GPU.

Simply run the python file with augments to check the verification results. It costs nearly 31600 seconds per run in our settings.

1.4 Supplemental Files

Networks We place the model files and the PyTorch network class files in the `models` folder:

- `mnist_network_FNN.py` PyTorch MNIST FNN class file.
- `mnist_ERAN_convs.py` PyTorch MNIST CNN class file.

- `mnist_FNN_i.*` MNIST FNNi model files.
- `convSmallRELU__Point.*` MNIST CNN1 model files.
- `convSmallRELU__PGDK.*` MNIST CNN2 model files.
- `convSmallRELU__DiffAI.*` MNIST CNN3 model files.
- `convMedGRELU__Point.*` MNIST CNN4 model files.
- `convMedGRELU__PGDK_w_0.1.*` MNIST CNN5 model files.
- `convMedGRELU__PGDK_w_0.3.*` MNIST CNN6 model files.
- `cifar_ERAN_convs.py` PyTorch CIFAR-10 class file.
- `cifar_convSmallRELU__PGDK.*` CIFAR-10 CNN1 model files.
- `cifar_convMedGRELU__PGDK_w_0.0078.*` CIFAR-10 CNN2 model files.
- `cifar_convMedGRELU__PGDK_w_0.0313.*` CIFAR-10 CNN3 model files.
- `vgg.py` PyTorch CIFAR-10 VGG class file.
- `resnet.py` PyTorch CIFAR-10 ResNet class file.
- `Cifar_ResNet18_adv_trim.pth` CIFAR-10 ResNet18 model file.
- `Cifar_ResNet50_adv_trim.pth` CIFAR-10 ResNet50 model file.
- `Cifar_ResNet152_adv_trim.pth` CIFAR-10 ResNet152 model file.
- `Cifar_VGG16_adv_trim.pth` CIFAR-10 VGG16 model file.
- `imagenet_linf_4.pth` ImageNet ResNet50a model file.
- `imagenet_linf_8.pth` ImageNet ResNet50b model file.

In our evaluation, we use the `.pth` files to load PyTorch models. PROVERO uses `.pyt` files. ERAN uses both `.pyt` files and `.onnx` files.

CSVs for PROVERO and ERAN We extracts the data we use in our evaluation (first 100 images in the dataset) as `.csv` files in the fold `extras`. If you want to evaluate the results on PROVERO and ERAN, you can replace the original `.csv` files with our `.csv` files. Note that PROVERO and ERAN are not included in our Artifact, which can be found here [6] [7].

[6] ERAN: https://github.com/eth-sri/eran
[7] PROVERO: https://github.com/teobaluta/provero

2 FAQs and Trouble Shooting

It is too slow to produce the result

First, check if you are giving large `FThreshold`, `SThreshold` or `budget` to DeepPAC. The larger the sampling number, the harder the optimization problem is. It is a trade-off between efficiency and precision to choose appropriate sampling numbers. Second, make sure you have Gurobi installed with a valid license and use GPU to sample on large DNNs. Use `-solver gurobi -gpu` to make DeepPAC run with Gurobi and GPU. For container, please make sure the `seccomp:unconfined` flag is passed to docker.

Out of (CUDA) memory

First, using large sampling number consumes more memory to store the intermediate result. Check if the `FThreshold`, `SThreshold` or `budget` is too large. Second, check if the `bsize` is too large, reduce the batch size may solve the `CUDA error: Out of Memory`.

Fail to find GPU in docker

Use `nvidia-smi` to check if the driver has successfully initialized in container. If the driver is functional, please reboot the host and restart the container. You may need to reinstall the Nvidia Container Toolkit if the driver failed.

How to verify other networks in DeepPAC?

The DeepPAC supports networks defined in Pytorch. You can pass the network class file, the name of the network class, and the statedict model file to DeepPAC using the augment `-ncf networkclassfile -nc networkclass -m model`.

How to apply DeepPAC in other datasets?

The algorithm and theorem behind DeepPAC can transfer to any datasets. However, we use the fixed optimization policy like focused learning and stepwise splitting to apply DeepPAC in three given dataset (MNIST, CIFAR-10, and Imagenet). As future work, we are seeking to apply these optimizations automatically for an arbitrarily given dataset.

3 Acknowledgement

We use networks from the `eth-sri/eran` and `MadryLab/robustness` repository and train our CIFAR-10 resnets use the code from `kuangliu/pytorch-cifar` repository.