VeRe: Verification Guided Synthesis for Repairing Deep Neural Networks

Jianan Ma[1,2], Pengfei Yang[3], Jingyi Wang[2,4*], Youcheng Sun[5],
Cheng-Chao Huang[6] and Zhen Wang[1]
[1]Hangzhou Dianzi University [2]Zhejiang University [3]SKLCS, Institute of Software, CAS
[4]ZJU-Hangzhou Global Scientific and Technological Innovation Center
[5]University of Manchester [6]Nanjing Institute of Software Technology, ISCAS

ABSTRACT

In this note, we give a brief introduction to VeRe for its artifact evaluation. VeRe is a verification-guided neural network repair framework. The main function of VeRe is to symbolically calculate the repair significance of neurons based on linear relaxation and furthermore optimize the parameters of problematic neurons to repair erroneous behaviors. This note is to help users quickly understand the basic information of VeRe (including the operating environment required) and reproduce our previous results. Our artifact is available at Figshare and the docker image can also be obtained in Docker Hub.

1 INTRODUCTION

VeRe is a framework for repairing (removing backdoors, correcting safety property violations) neural networks, which is an an implementation of the algorithms in our technical article titled "VeRe: Verification Guided Synthesis for Repairing Deep Neural Networks". VeRe is now open source under the Apache-2.0 License, and we intend to apply the Artifacts Available and Artifacts Reusable badges in the artifact evaluation.

The main function of VeRe is to repair neural networks that have been subjected to backdoor attacks or violate certain safety properties, while preserving the original performance of the model as much as possible. We provide the technical paper in PDF format in our artifact. If you are interested in how VeRe works, please refer to our technical paper.

2 REQUIREMENTS

VeRe is developed based on Python 3.9.18, which is compatible with both Windows and Linux. Users can install all dependencies via Conda. For the artifact evaluation, we provide a docker image for Linux that has already been initialized with all dependencies. However, some extra steps are needed for preparing the docker environment. VeRe requires a GPU for calculating repair significance and fixing problematic neurons, so the Nvidia-docker2 is required for Docker. Ensure that your Docker version is not higher than 19.03. The tutorial can be found here[1]. In addition, please ensure that your CUDA version is greater than or equal to 12.0 to successfully run the image.

3 USAGE

Our artifacts can be obtained from: link_to_figshare. You can load the artifact (Docker image) using the following command:

```
sudo docker load -i vere.tar
```

[1]https://cnvrg.io/how-to-setup-docker-and-nvidia-docker-2-0-on-ubuntu-18-04/

Besides, if you want to obtain the docker image from Docker Hub, you can use the following command to pull our image:

```
sudo docker pull mjnn/vere:1.0
```

After obtaining the image, the first step is to import the Docker image from the tarball and create a corresponding container to run VeRe.

```
sudo docker run --runtime=nvidia -it \
--name Artifact-vere mjnn/vere:1.0
```

Use the following two commands to enter the folders for the backdoor removal experiments and the experiment of correcting safety property violation, respectively:

```
cd root/abLocal/backdoor/
cd root/abLocal/safety/
```

4 REPRODUCTION

All the experiments are conducted on a physical machine with Intel(R) Xeon(R) Gold 6248R CPU @ 3.00GHz, Nvidia RTX 4090 and 256GB RAM, and the results are presented in the "result" folder. Note that a different running environment may bring a little change to the results.

4.1 Backdoor Removal

4.1.1 Main Experiments. This experiment removed the implanted backdoor in the DNNs and evaluated the performance of the repaired model, including accuracy and success rate against backdoor attacks on the generalization set. We create a subfolder under the "result" directory based on the real-time execution time, and store the repaired model and log files inside them. We conducted experiments in two popular backdoor attack scenarios: BadNets [2] and Blend [1]. Specifically, we utilized `badnets_repair.py` and `blend_repair.py` to replicate the experimental results. Additionally, experiments on the Imagenette [3] dataset were implemented with the extra code `imagenette_repair.py`. We provide explanations for all arguments, using `badnets_repair.py` as an example:

- `--dataset`: This argument indicates on which dataset the neural network to be repaired is trained.
- `--interval`: The number of sub-intervals split from the neuron range.
- `--N`: The number of available poisoned samples.
- `--N_clean`: The number of available clean samples.
- `--layer_round`: The maximum number of iterations.
- `--device`: Device name, used to specify the computing device, default is "cuda:0".
- `--repair_method`: Used to specify the method employed during the repair phase, default is "vere".
- `--seed`: This argument is used to specify the random seed.

For instance, you can use the following two commands to examine the results of backdoor removal for the VGG13 model trained on the CIFAR10 dataset poisoned by two different backdoor attacks:

```
python3 badnets_repair.py --dataset CIFAR10 \
--N 1000 --N_clean 1000 --interval 10 \
--layer_round 20 --repair_method vere \
--seed 0 --device cuda
python3 blend_repair.py --dataset CIFAR10 \
--N 1000 --N_clean 1000 --interval 10 \
--layer_round 20 --repair_method vere \
--seed 0 --device cuda
```

In our settings, the experiment takes approximately 50 to 500 seconds, depending on the selected dataset. Please note that we use the RTX 4090 as the computing device, and using different GPUs may result in varying time costs.

Moreover, you can specify the `repair_method` parameter as other repair methods such as "`rm_finetune`" to examine the experimental results of combining our fault localization method with other repair methods. For example, you can replicate the experiment using our fault localization method followed by the RM's fine-tuning repair method to remove backdoors in the VGG13 model poisoned by Badnets attacks on the CIFAR10 with the following command:

```
python3 badnets_repair.py --dataset CIFAR10 \
--N 1000 --N_clean 1000 --interval 10 \
--layer_round 20 --repair_method rm_finetune \
--seed 0 --device cuda
```

4.1.2 Experiments on the Imagenette Dataset. For the experiments on the Imagenette dataset, you can use `imagenette_repair.py` to examine the backdoor removal results, including both Badnets and Blend attacks. Specifically, we have added an argument as follows:

- `--attack`: This argument is used to specify the type of backdoor attack (Badnets, Blend).

For instance, you can use the following two commands to examine the results of backdoor removal for the VGG16 model trained on the Imagenette dataset poisoned by Badnets and Blend attacks:

```
python3 imagenette_repair.py --attack Badnets \
--N 100 --N_clean 100 --interval 10 \
--layer_round 20 --seed 0 --device cuda
python3 imagenette_repair.py --attack Blend \
--N 100 --N_clean 100 --interval 10 \
--layer_round 20 --seed 0 --device cuda
```

This experiment typically takes several hundred seconds in our setup. Please note that, due to the higher resolution of samples and the larger model size in this experiment, a GPU with 24GB or more of VRAM is required.

4.1.3 Experiments on the Other Activation Function. In this experiment, the ReLU activation function in each neural network model is replaced with the Sigmoid activation function. This experiment conduct on two datasets, CIFAR10 and SVHN, along with Badnets and Blend attacks. We use `otheract_repair.py` to reproduce this experiment.

For instance, you can use the following command to examine the results of backdoor removal for the the VGG13 model using the Sigmoid activation function, trained on the CIFAR10 and SVHN poisoned by Badnets attacks:

```
python3 otheract_repair.py --attack Badnets \
--dataset CIFAR10 --N 1000 --N_clean 1000 \
--interval 10 --layer_round 20 --seed 0 \
--device cuda
python3 otheract_repair.py --attack Badnets \
--dataset SVHN --N 1000 --N_clean 1000 \
--interval 10 --layer_round 20 --seed 0 \
--device cuda
```

4.2 Correcting Safety Property Violation.

This experiment corrected the violated safety properties within the ACAS Xu networks and assessed the repair success rate, generalization, drawdown of the model, and time overhead. Specifically, we created a folder with the same name for each network and saved the repaired network along with logs in the "result" folder within each respective subfolder.

We use `main.py` in each subfolder to reproduce the relevant experimental results. It includes the following parameters:

- `--net`: This argument is used to specify the name of the network to be repaired (e.g., $N_{1,9}$).
- `--interval`: The number of sub-intervals split from the neuron range.
- `--N`: The number of available counterexamples.
- `--N_drawdown`: The number of available correct samples.
- `--layer_round`: The maximum number of iterations.
- `--device`: Device name, used to specify the computing device, default is "cuda:0".
- `--seed`: This argument is used to specify the random seed.

For example, you can use the following three commands to reproduce the experimental results for correcting the safety properties $(2, 7, 8)$ for networks $\left(N_{3,1}, N_{1,9}, N_{2,9}\right)$, respectively:

```
python3 n31/main.py --net n31 --interval 10 \
--N 10000 --N_drawdown 10000 \
--layer_round 20 --seed 0 --device cuda
python3 n19/main.py --net n19 --interval 10 \
--N 10000 --N_drawdown 10000 \
--layer_round 20 --seed 0 --device cuda
python3 n29-8/main.py --net n29 \
--interval 10 --N 10000 --N_drawdown 10000 \
--layer_round 20 --seed 0 --device cuda
```

In our settings, correcting safety properties on different networks typically takes from a few seconds to tens of seconds, with reduced time overhead for fewer available samples.

REFERENCES

[1] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. 2017. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526* (2017).
[2] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2019. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access* 7 (2019), 47230–47244.
[3] Jeremy Howard. 2019. The Imagenette dataset. https://github.com/fastai/imagenette