A Short Instruction for nnbarrier

Hengjun Zhao

School of Computer and Information Science
Southwest University, Chongqing, 400715, P. R. China
zhaohj2016@swu.edu.cn

1 Introduction

In our paper *Synthesizing Barrier Certificates Using Neural Networks* accepted by HSCC'20, we developed a tool named nnbarrier that can automatically learn a barrier ceritificate represented by a neural network for the safety verification of a continuous dynamical system. Here we give a short instruction to the use of nnbarrier, covering the system requirements, installation process, the structure of source codes, sample inputs, and user-defined inputs. We will emphasize what parts that were presented in the submitted paper will be covered in the instruction, for the purpose of repeatability evaluation. If there is any problem in using nnbarrier, please contact zhaohj2016@swu.edu.cn.

2 Installation

2.1 System Requirements

It is assumed that you have Python Version 3.x installed on your system. We have tested nnbarrier on Ubuntu Linux, Mac OS, and Windows (see Table 1).

2.2 Dependent Packages

It is assumed that you have the python package manager Pip installed for Python 3.x, which will facilitate the installation of dependent packages greatly.

Essentially, to run nnbarrier without visualization, the only two packages you need to install are Pytorch and NumPy. It seems that numpy will be automatically installed when installing Pytorch. Please visit https://pytorch.org/ for the installation instructions for the popular machine learning platform Pytorch. For example, with the combination Mac+Python 3.7+Pip (without cuda GPU support), the latest stable version Pytorch 1.3 can be installed by simply run

```
pip3 install torch torchvision
```

If you would like to visualize the generated barrier function together with the considered system, i.e. the system dynamics, the domain, the initial set, and the unsafe/safe region, then some additional graphics packages are required. For visualization of 2D systems, matplotlib needs to be installed, and you are referred to

https://matplotlib.org/users/installing.html for the instructions. In the implementation of nnbarrier, visulaization of 3D systems is supported by Mayavi, a 3D scientific data visualization library. Please visit http://docs.enthought.com/mayavi/mayavi/installation.html#installing-with-pip for the installation of mayavi and its dependencies (e.g. PyQt5). In our testing, on most platforms the visualization-required packages can be installed with the following commands easily:

```
pip install matplotlib
pip install mayavi
pip install PyQt5
```

where pip can be actually pip3. However, we do met some problems occasionally. If you failed to get this done in the end, nnbarrier can still be run by commenting the statements for visualization, which will be explained later.

In summary, we have tested nnbarrier using the following combinations

Table 1. Tested platforms and packages for nnbarrier

OS	Python	Pip	Pytorch	Visualization
Ubuntu 18.04.02	3.6.7	9.0.1	1.2.0	matplotlib+mayavi+PyQt5
Ubuntu 18.04.02	3.6.9	19.3.1	1.3.1	matplotlib+mayavi+PyQt5
Windows 10 1903	3.7.3	19.3.1	1.3.1	matplotlib+mayavi+PyQt5
Mac OS 10.11.6	3.7.6	19.3.1	1.3.1	matplotlib+mayavi+PySide2

2.3 Obtain the nnbarrier Package

Suppose that you have Git installed on your system. Then the nnbarrier package can be obtained via

```
git clone https://github.com/zhaohj2017/HSCC20-Repeatability
```

Sturcture of the Package. The cloned directory HSCC20-Repeatability consists of 10 Python source files and one file folder as listed below:

- acti.py: self-defined activation functions (i.e. *Bent-ReLU*) for neural networks
- ann.py: generating a multi-layer neural network model (NN for short)
- data.py: generating batches of training data
- loss.py: given a NN and a training data set, computes a loss value
- lrate.py: self-defined learning rate adjusting strategy
- main.py: the main file to run
- opt.py: a set of optimizers provided by Pytorch to train the neural network
- plot.py: visualization of 2D systems
- plot3d.py: visualization of 3D systems
- train.py: the training loop that iterates through batches, epochs, and restarts

- cases: a file folder consisting of all the problem definitions for the case studies in our paper

Inside cases, there are 5 sub-folders corresponding to the examples in the paper:

- eg1_prajna_original: the classical problem from [3], corresponding to the running example Example 1 and its continuations in our paper
- eg2_prajna_modified: modified version of the problem from [3], corresponding to Example 2 in our paper
- eg3_darboux: the *Darboux-type* barrier certificate problem from [4], corresponding to Example 3 in our paper
- eg4_exponential: the *exponential* barrier certificate problem from [2], corresponding to Example 4 in our paper
- eg5_obstacle: the aircraft obstacle avoidance problem modified from [1], corresponding to Example 5 in our paper

Each of the 5 sub-folders consists of 2 Python source files with the same names but different contents:

– prob.py: specify the safety verification problem of the corresponding example
– superp.py: specify the super-parameters for training the neural network of the corresponding problem

These are the two files that needs modification for solving user-defined problems.

Test nnbarrier. Suppose you are using Linux or Mac and located in the HSCC20-Repeatability directory. Execute the command

```
cp ./cases/eg1_prajna_original/*.py .
```

to copy the problem definition and parameter specification files to current directory. Then nnbarrier can be invoked by executing

```
python main.py
```

where python can actually be python3 on your platform. If all the packages aforementioned are correctly installed, then the training process starts (see Fig. 1). Upon termination you will see a popup window illustrating the plotted barrier certificate, as presented in our paper.

3 Sample Inputs

Here we explain more details about the source files, focusing on main.py, and prob.py and superp.py from the sub-folder eg1_prajna_original of Example 1. You may skip this and go directly to the next section for repeatability evaluation, but going through the core codes quickly may enable modifications or extensions of the reported case studies in our paper.

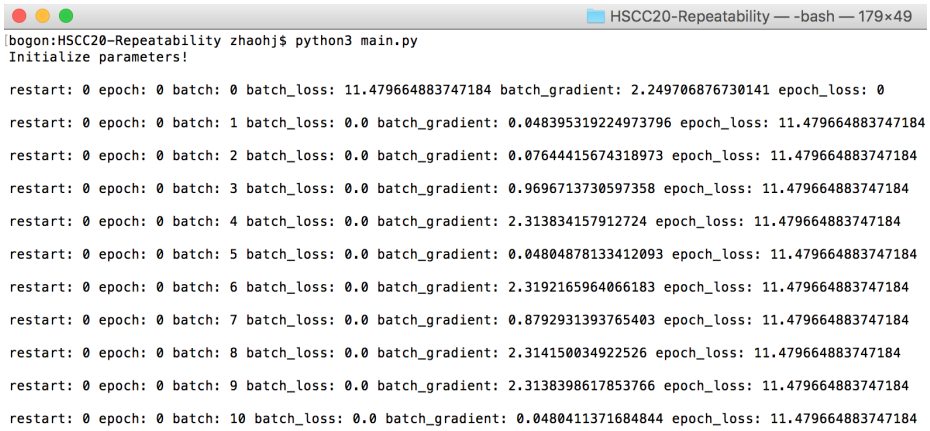

Fig. 1. The training process showing the number of restarts, epochs, batches, current gradient, and the current loss

3.1 main.py

Line

```
20 model = ann.gen_nn()
```

is to build a NN model, the number of layers, neurons, and the type of activation functions of which are specified in superp.ty; Lines

```
27 time_start_data = time.time()
28 batches_init, batches_unsafe, batches_domain = data.
                                    gen_batch_data()
29 time_end_data = time.time()
```

is to generate batches of training data and measure the time cost; Lines

```
32 time_start_train = time.time()
33 train.itr_train(model, batches_init, batches_unsafe,
                                    batches_domain)
34 time_end_train = time.time()
```

are to train the NN model on the generated training data and measure the time cost; Lines

```
36 print("\nData generation totally costs:", time_end_data
                                    - time_start_data)
37 print("Training totally costs:", time_end_train -
                                    time_start_train)
```

are to output the measured time costs; Lines

```
9 import plot
```

and

```
44 plot.plot_barrier(model)
```

are for visualization of the generated barrier certificates, which should be *commented* if you failed to install the required graphics packages successfully.

3.2 prob.py

Line

```
15 DIM = 2
```

is to set the dimension of the considered system; Lines

```
21 INIT = [[1, 2], \
22              [-0.5, 0.5], \
23          ]
24 INIT_SHAPE = 2 # 2 for circle
```

are to set the interval ranges and the real shape of the initial set, where 1 denotes (super-)rectangle and 2 denotes circle or sphere; Lines

```
30 UNSAFE = [[-1.4, -0.6], \
31              [-1.4, -0.6], \
32          ]
33 UNSAFE_SHAPE = 2 # 2 for circle
```

and lines

```
39 DOMAIN = [[-3, 2.5], \
40              [-2, 1], \
41          ]
42 DOMAIN_SHAPE = 1 # 1 for rectangle
```

are to set the interval ranges and shapes for the unsafe region and the domain of the system, respectively; Line

```
49 def cons_init(x):
50     return torch.pow(x[:, 0] - 1.5, 2) + \
51               torch.pow(x[:, 1], 2) <= 0.25 + \
52                  superp.TOL_DATA_GEN
```

is to set the inequality constraint representing the circle region of the initial set; Lines

```
55 def cons_unsafe(x):
```

and

```
59 def cons_domain(x):
```

are defining the inequality constraints representing the unsafe and domain areas, respectively; Lines

```
67 def vector_field(x):
68     # the vector of functions
69     def f(i, x):
70         if i == 1:
71             return x[:, 1] # x[:, 1] stands for x2
72         elif i == 2:
73             return - x[:, 0] - x[:, 1] + \
74                 torch.pow(x[:, 0], 3) / 3.0
75                 # x[:, 0] stands for x1
76         else:
77             print("Vector function error!")
78             exit()
79
80     vf = torch.stack([f(i + 1, x) for i in range(DIM)],
                                      dim=1)
81     return vf
```

are defining the continuous dynamics of the considered system, thus finishing the formulation of the safety verification problem in Example 1.

3.3 superp.py

Line

```
15 VERBOSE = 1 # set to 1 to display epoch and batch losses
                                    in the training process
```

is to set the VERBOSE option to 1 so the training information will be displayed as shown in Fig. 1; Lines

```
22 N_H = 1 # then number of hidden layers
23 D_H = 5 # the number of neurons of each hidden layer
```

are to set the number of hidden layers and the number of neurons of each layer in the generated NN model (here we assume that the number of neurons are the same in different layers); Line

```
28 BENT_DEG = 0.0001
```

is to set the constant parameter in our designed *Bent-ReLU* activation functions; Lines

```
33 TOL_INIT = 0.02
34 TOL_SAFE = 0.02
35 TOL_LIE = 0.01
36 TOL_BOUNDARY = 0.05
```

are to set the four tolerances in our designed loss functions; Line

```
63 EPOCHS = 10
```

is to set the number of training epochs; Lines

```
69 ALPHA = 0.1 # initial learning rate
70 BETA = 0 # if beta equals 0 then constant rate = alpha
71 GAMMA = 0 # when beta is nonzero, larger gamma gives
                                    faster drop of rate
```

are to setting the parameters of our designed learning rate adjusting strategy; Line

```
77 TOL_MAX_GRAD = 6
```

is to set the maximum gradient value for our gradient control strategy; Lines

```
84 DATA_EXP_I = np.array([5, 5])
85     # for sampling from initial; length = prob.DIM
86 DATA_LEN_I = np.power(2, DATA_EXP_I)
87     # the number of samples for each dimension of domain
```

are to set the number of samples, which is an interger power of 2, for each dimension of the system for training data generation from the initial set; Lines

```
88 BLOCK_EXP_I = np.array([3, 3])
89     # 0 <= BATCH_EXP <= DATA_EXP
90 BLOCK_LEN_I = np.power(2, BLOCK_EXP_I)
91     # number of batches for each dimension
```

are to set the number of batches, which is an interger power of 2, for each dimension of the system for generating small batches of training data from the initial set; besides, we have similar lines for generating training data from the unsafe and domain regions respectively, thus finishing the specification of super-parameters for NN training.

3.4 User-defined inputs

4 Repeatability Evaluation

5 cases, figures, tables

4.1 Tables

- What elements of the paper are included in the REP (e.g.: specific figures, tables, etc.).
- Instructions for installing and running the software and extracting the corresponding results.

4.2 Figures

4.3 Fine-Tuning Figure and Table

References

1. Barry, A., Majumdar, A., Tedrake, R.: Safety verification of reactive controllers for uav flight in cluttered environments using barrier certificates. In: 2012 IEEE International Conference on Robotics and Automation, ICRA 2012. pp. 484–490. nstitute of Electrical and Electronics Engineers Inc. (2012)
2. Liu, J., Zhan, N., Zhao, H., Zou, L.: Abstraction of elementary hybrid systems by variable transformation. In: FM 2015: Formal Methods - 20th International Symposium, Oslo, Norway, June 24-26, 2015, Proceedings. pp. 360–377. Springer (2015)
3. Prajna, S., Jadbabaie, A.: Safety verification of hybrid systems using barrier certificates. In: Proceedings of the 7th International Workshop on Hybrid Systems: Computation and Control HSCC. pp. 477–492 (2004)
4. Zeng, X., Lin, W., Yang, Z., Chen, X., Wang, L.: Darboux-type barrier certificates for safety verification of nonlinear hybrid systems. In: Proceedings of the 13th International Conference on Embedded Software. pp. 11:1–11:10. EMSOFT '16, ACM (2016)