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 and user-defined inputs. We will emphasize what parts that were presented in the submitted paper will be covered in this 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

The package nnbarrier was developed in Python, so it is assumed that you have Python Version 3.x installed on your system. Please see Table 1 for the tested Python versions and the hosting operating systems (OS).

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 with the installation of the latest versions of PyQt5, which were resolved by updating Pip and switching to root privilege. 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 Subsection 4.1.

In summary, we have tested nnbarrier using the following combinations

Table 1. Tested platforms and Python 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 directory 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 directory consisting of specifications of all the examples in our paper, organized by 5 sub-directories:
  - 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-directories 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 need modifications for solving user-defined problems, c.f. Section 4.

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 .
```

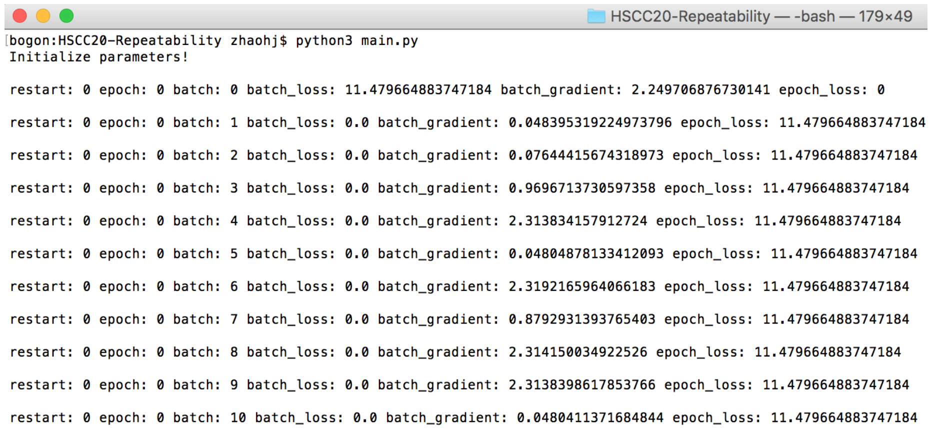
to copy the problem definition and parameter specification files to current directory. Then nnbarrier can be invoked by executing

```
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 Repeatability Evaluation

In this section we present some instructions for repeatability evaluation on our reported results in the paper. We will cover Examples 1-5 and the related Figure 5, Figure 7, Figure 8, Table 1, and Figure 6.



```
HSCC20-Repeatability — -bash — 179×49
bogon:HSCC20-Repeatability zhaohj$ python3 main.py
Initialize parameters!

restart: 0 epoch: 0 batch: 0 batch_loss: 11.479664883747184 batch_gradient: 2.249706876730141 epoch_loss: 0

restart: 0 epoch: 0 batch: 1 batch_loss: 0.0 batch_gradient: 0.048395319224973796 epoch_loss: 11.479664883747184

restart: 0 epoch: 0 batch: 2 batch_loss: 0.0 batch_gradient: 0.07644415674318973 epoch_loss: 11.479664883747184

restart: 0 epoch: 0 batch: 3 batch_loss: 0.0 batch_gradient: 0.9696713730597358 epoch_loss: 11.479664883747184

restart: 0 epoch: 0 batch: 4 batch_loss: 0.0 batch_gradient: 2.313834157912724 epoch_loss: 11.479664883747184

restart: 0 epoch: 0 batch: 5 batch_loss: 0.0 batch_gradient: 0.04804878133412093 epoch_loss: 11.479664883747184

restart: 0 epoch: 0 batch: 6 batch_loss: 0.0 batch_gradient: 2.3192165964066183 epoch_loss: 11.479664883747184

restart: 0 epoch: 0 batch: 7 batch_loss: 0.0 batch_gradient: 0.8792931393765403 epoch_loss: 11.479664883747184

restart: 0 epoch: 0 batch: 8 batch_loss: 0.0 batch_gradient: 2.314150034922526 epoch_loss: 11.479664883747184

restart: 0 epoch: 0 batch: 9 batch_loss: 0.0 batch_gradient: 2.3138398617853766 epoch_loss: 11.479664883747184

restart: 0 epoch: 0 batch: 10 batch_loss: 0.0 batch_gradient: 0.0480411371684844 epoch_loss: 11.479664883747184
```

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

3.1 Repeatability of Figures 5, 7, and 8

Suppose you are using Linux or Mac and have changed into the directory HSCC20-Repeatability.

– Execute the command

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

  followed by

  `python main.py`

  to create Figure 5;
– Execute the command

  `cp ./cases/eg2_prajna_modified/*.py .`

  followed by

  `python main.py`

  to create Figure 7;
– Execute the command

  `cp ./cases/eg5_obstacle/*.py .`

  followed by

  `python main.py`

  to create Figure 8.

3.2 Repeatability of Table 1

First, we emphasize that due to the issue of input format transformation, we haven't completed the automatic integration of the iSat3[1] SMT solver into nnbarrier at the moment, so only training time costs in Table 1 will be available for repeatability evaluation. In other words, we are concentrating on the $T_{\mathsf{nnbarrier}}$ column in Table 1 of our paper.

Suppose you are using Linux or Mac and have changed into the directory HSCC20-Repeatability. For repeatability of $T_{\mathsf{nnbarrier}}$, you could follow the three steps:

1. Execute the command

```
cp ./cases/<sub-folder-name>/*.py .
```

   where `<sub-folder-name>` is one of the 4 sub-directories: eg2_prajna_modified, eg3_darboux, eg4_exponential, or eg5_obstacle
2. In the current directory, modify the VERBOSE option in the superp.py file

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

   by setting it to **0**
3. Execute the command

```
python main.py
```

   Then you will get the measured time costs (in seconds) shown, for example, as

```
Data generation totally costs: 0.149125337600708
Training totally costs: 12.78989315032959
```

We have the following remarks on the repeatability of $T_{\mathsf{nnbarrier}}$:

1. The time costs reported in the paper are obtained on the platform running Ubuntu 18.04.02 with Intel i7-8550u CPU and 32GB memory; the installed packages are Python 3.6.7 and Pytorch 1.2.0;
2. The time costs reported in the paper are averaged over 5 separate runs for Examples 2-5:
3. It can be seen from Table 2 that the time costs of different runs for the same example varies significantly. It is a reasonable phenomenon since the weights and biases of a NN are randomly initialized. Such randomization makes the exact repeatability of the reported time costs impossible. However, we believe that the averaged time costs give the expected length of time you need to wait before a result can be returned. Anyway, in all our tests, the training process for all examples will terminate within 5 restarts. So please pay a little patience if you cannot obtain a result immediately when you run nnbarrier.

[1] `https://projects.informatik.uni-freiburg.de/projects/isat3/`

Table 2. Training time costs (in seconds) over 5 runs for Examples 2-5

	Run 1	Run 2	Run 3	Run 4	Run 5	Average
Example 2	3252.37	540.90	287.36	3653.01	921.61	1731.03
Example 3	14.22	776.57	34.99	13.38	868.09	341.45
Example 4	234.65	1340.01	314.03	1249.72	49.00	637.48
Example 5	1527.77	7254.64	1083.90	487.04	5473.04	3165.28

3.3 Repeatability of Figure 6

Suppose you are using Linux or Mac and have changed into the directory HSCC20-Repeatability.

Pre-train. First, execute the command

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

Next modify the following 3 lines in superp.py in the current directory

```
TOL_INIT = 0.02
TOL_SAFE = 0.02
TOL_LIE = 0.01
```

to set all the three tolerances to **0**. Then run

```
python main.py
```

to obtain a generated barrier certificate and a pre-trained model which will be stored in the pre-trained.pt file in the current directory. At the same time, a picture similar to the left part of Figure 6 will be produced.

Fine-tune. In the following, modify the superp.py file in the current directory

```
FINE_TUNE = 0 # set to 1 for fine-tuning a pre-trained model
```

by setting the FINE_TUNE option to **1**. Next modify the following line in superp.py in the current directory

```
TOL_SAFE = 0
```

to increase the tolerance for safety by setting `TOL_SAFE = 0.05`. Then run

```
python main.py
```

again to obtain a fine-tuned barrier certificate and a fine-tuned NN model which will again be stored in the pre-trained.pt file in the current directory. At the same time, a picture similar to the right part of Figure 6 will be produced.

The above fine-tuning operations can be iteratively performed by increasing the tolerance values gradually.

4 Input Formats

In this section we explain more details about the source files, focusing on main.py, as well as prob.py and superp.py from the sub-directory eg1_prajna_original of Example 1. Going through the core codes quickly may enable modifications or extensions of the reported case studies in our paper.

4.1 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; Line

```
40 torch.save(model.state_dict(), 'pre-trained.pt')
```

is to save the trained NN model to a file named pre-trained.pt for later use, which has been talked about in Subsection 3.3; 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.

4.2 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; Lines

```
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
```

are 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.

4.3 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; Line

```
17 FINE_TUNE = 0 # set to 1 for fine-tuning a pre-trained
                                        model
```

is to set the FINE_TUNE option to indicate whether we are training a pre-trained NN model, extracted from the pre-trained.pt file saved on disk. Lines

```
22 N_H = 1 # the 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 hidden 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 set 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.

4.4 User-defined inputs

If you have your own safety verification problem for a constrained continuous system, and would like to solve it using nnbarrier, what you need to do is just rewriting prob.py and superp.py to formulate your problem and set the super-parameters for NN training, following the above explanations.

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)