DNNF

David Shriver

Aug 02, 2022

GETTING STARTED

DNNF is a tool for applying falsification methods such as adversarial attacks to the checking of DNN correctness problems. Adversarial attacks provide a powerful repertoire of scalable algorithms for property falsification. DNNF leverages these techniques by employing reductions to automatically transform correctness problems into equivalent sets of adversarial robustness problems, to which these attacks can then be applied.

CHAPTER

ONE

INSTALLATION

DNNF can be installed using pip, manually from source, or using docker. We also provide a pre-configured VirtualBox VM, containing the tool and data used for the evaluation in Reducing DNN Properties to Enable Falsification with Adversarial Attacks (pre-print).

1.1 Pip Install

DNNF can be installed using pip by running:

```
$ pip install dnnf
```

This will install the latest release of DNNF on PyPI. To install the most recent changes from GitHub, run:

```
$ pip install git+https://github.com/dlshriver/DNNF.git@main
```

Note: Installation with pip will not install the TensorFuzz falsification backend. Currently this backend is only available through manual installation or the provided docker image.

1.2 Source Install

The required dependencies to install DNNF from source are:

- python3

The optional tensorfuzz backend also requires:

- git
- python2.7
- virtualenv

If you do not plan to use tensorfuzz, then these dependencies are not required. Please ensure that the required dependencies are installed prior to running the installation script. For example, on a fresh Ubuntu 20.04 system, the dependencies can be installed using apt as follows:

```
$ sudo add-apt-repository ppa:deadsnakes/ppa
$ sudo apt-get update
$ sudo apt-get install git python2.7 python3.7 virtualenv
```

To install DNNF in the local directory, download this repo and run the provided installation script, optionally specifying which backends to include during installation:

```
$ ./install.sh [--include-cleverhans] [--include-foolbox] [--include-tensorfuzz]
```

We have successfully tested this installation procedure on machines running Ubuntu 20.04 and CentOS 7.

1.3 Using Docker

We also provide a pre-built docker image containing DNNF, available on Docker Hub. To use this image, run the following:

```
$ docker pull dlshriver/dnnf
$ docker run --rm -it dlshriver/dnnf
(.venv) dnnf@hostname:~$ dnnf -h
```

To build a docker image with the latest changes to DNNF, run:

```
$ docker build . -t dlshriver/dnnf
$ docker run --rm -it dlshriver/dnnf
(.venv) dnnf@hostname:~$ dnnf -h
```

CHAPTER

TWO

BASIC USAGE

DNNF can be run on correctness problems specified using ONNX and DNNP. DNNP is the same property specification language used by the DNNV verifier framework. A description of this specification language can be found in the DNNV documentation.

To execute DNNF, first activate the virtual environment with:

```
$ . .venv/bin/activate
```

This is only required if DNNF was installed from source. The virtual environment should open automatically if using the docker image or the provided VM.

The DNNF tool can then be run as follows:

```
$ dnnf PROPERTY --network NAME PATH
```

Where `PROPERTY` is the path to the property specification, `NAME` is the name of the network used in the property specification (typically `N`), and `PATH` is the path to a DNN model in the ONNX format.

To see additional options, run:

```
$ dnnf -h
usage: dnnf [-h] [--long-help] [-V] [--seed SEED] [-v | -q] [-N NAME PATH]
            [--save-violation PATH] [--vnnlib] [-p N_PROC] [--n_starts N_STARTS]
            [--cuda] [--backend BACKEND [BACKEND ...]] [--set BACKEND PARAM VALUE]
            property

dnnf - deep neural network falsification

positional arguments:
  property

optional arguments:
  -h, --help            show this help message and exit
  --long-help           show a longer help message with available falsifier backends and␣
↪exit
  -V, --version         show program's version number and exit
  --seed SEED           the random seed to use (default: None)
  -v, --verbose         show messages with finer-grained information (default: False)
  -q, --quiet           suppress non-essential messages (default: False)
  -N, --network NAME PATH
  --save-violation PATH
                        the path to save a found violation (default: None)
```

(continues on next page)

(continued from previous page)

```
  --vnnlib              use the vnnlib property format (default: None)
  -p, --processors, --n_proc N_PROC
                        The maximum number of processors to use (default: 1)
  --n_starts N_STARTS   The default number of random starts per sub-property (can be set␣
↪per backend with --set) (default: -1)
  --cuda                use cuda (default: False)
  --backend BACKEND [BACKEND ...]
                        the falsification backends to use (default: ['pgd'])
  --set BACKEND PARAM VALUE
                        set parameters for the falsification backend (default: None)
```

To see the currently available falsification backends, use the `--long-help` option.

2.1 Running on Benchmarks

We provide several DNN verification benchmarks in DNNP and ONNX formats in dlshriver/dnnv-benchmarks. This benchmark repository includes both DNNF-GHPR and the DNNF-CIFAR-EQ benchmarks introduced by DNNF!

To execute DNNF on a problem in one of the benchmarks, first navigate to the desired benchmark directory in `benchmarks` (e.g., `DNNF-GHPR`, `DNNF-GHPR`). Then run DNNF as specified above. For example, to run DNNF with the Projected Gradient Descent adversarial attack from cleverhans on an DNNF-GHPR property and network, run:

```
$ cd benchmarks/DNNF-GHPR
$ dnnf properties/dronet_property_0.py --network N onnx/dronet.onnx --backend cleverhans.
↪projected_gradient_descent
```

Which will produce output similar to:

```
Falsifying: Forall(x, (((0 <= x) & (x <= 1) & (N[(slice(2, -3, None), 1)](x) <= -2.
↪1972245773362196)) ==> ((-0.08726646259971647 <= N[(slice(2, -1, None), 0)](x)) &␣
↪(N[(slice(2, -1, None), 0)](x) <= 0.08726646259971647))))

dnnf
  result: sat
  falsification time: 0.6901
  total time: 2.3260
```

The available backends for falsification are:

- CleverHans
  - `cleverhans.carlini_wagner_l2`
  - `cleverhans.fast_gradient_method`
  - `cleverhans.hop_skip_jump_attack`
  - `cleverhans.projected_gradient_descent`
  - `cleverhans.spsa`
- FoolBox
  - `foolbox.ATTACK` where `ATTACK` is the name of an adversarial attack from this list
- TensorFuzz

- `tensorfuzz`

Attack specific parameters can be set using the `--set BACKEND NAME VALUE` option. For example, to set the `nb_iter` parameter of the `cleverhans.projected_gradient_descent` attack to 40 steps, you can specify `--set cleverhans.projected_gradient_descent nb_iter 40`.

If a property uses parameters, then the parameter value can be set using `--prop.PARAMETER=VALUE`, e.g., `--prop.epsilon=1`, similar to DNNV.