Verification of ACAS Xu using Alpha-Beta-Crown and Nnenum

Văduva Vlad-Andrei, Granu Dragoș Vlad, Iordache Constantin, Hrițu Toma

Coordinator: Conf. dr. Erașcu Mădălina

January 24, 2024

Contents

1 Abstract

The purpose of verifying the ACAS Xu benchmark is to assure the correctness of the results output by the benchmark. Such results can have real life consequences. Therefore, it is important to ensure the validity of these results in order to further improve and develop the benchmark.

2 Introduction[1]

ACAS Xu is an air-to-air collision avoidance system designed for unmanned aircraft that issues horizontal turn advisories to avoid an intruder aircraft. Due the use of a large lookup table in the design, a neural network compression of the policy was proposed. The purpose of this paper is to document the process of verifying the ACAS Xu benchmark using two VNN tools, alpha-beta crown and nnenum.

3 ACAS Xu [2]

ACAS XU refers to the Automatic Collision Avoidance System - Experimental Upgrade. ACAS XU is a set of experimental enhancements to the existing ACAS II (Traffic Alert and Collision Avoidance System II) used in aviation. ACAS is a safety system designed to reduce the risk of mid-air collisions between aircraft.

ACAS II is based on the Traffic Advisory (TA) and Resolution Advisory (RA) concepts. When ACAS II detects a potential collision threat, it provides pilots with advisories to take evasive action, such as a vertical resolution advisory to climb or descend.

ACAS XU represents advancements and upgrades to the existing system, incorporating improved algorithms and technologies to enhance collision avoidance capabilities.

3.1 Architecture Elements

ACAS Xu uses FC and Relu for its architecture.
FC is a Fully Connected layer inside the neural network. This is one one the layer types that is used as a basis in almost all neural networks. In an FC layer, all the neurons of the input are connected to every neuron of the output layer. In an FC layer, a weighted linear transformation is applied to the input neurons and then pass the output through a non-linear activation function.
ReLU[5] (Rectified Linear Unit) is an activation function that introduces the property of non-linearity to a deep learning model and solves the vanishing gradients issue.

4 Tools

4.1 Alpha-Beta-Crown [3]

Alpha-Beta-CROWN is a neural network verification tool based on an efficient linear bound propagation framework and branch and bound. It can be accelerated efficiently on GPUs and scales well for big convolutional neural networks.
Supported neural network architectures:

- Layers: fully connected, convolutional, pooling, transposed convolution
- Activation functions: ReLU, sigmoid, tanh, arctan, sin, cos, tan
- Residual connections and other irregular graphs

Prerequisites: Conda, miniconda works as well.
Miniconda Installation: We create a folder named miniconda3:

```
mkdir -p ~/miniconda3
```

We must download miniconda from the main site:

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -O ~/miniconda3/miniconda.
```

We run the installation script:

```
bash ~/miniconda3/miniconda.sh -b -u -p ~/miniconda3
```

After we installed the prerequisites, we can proceed with alpha-beta-crown installation: We clone the GitHub repository:

```
git clone --recursive https://github.com/Verified-Intelligence/alpha-beta-CROWN.git
```

This way, we also clone auto-LiRPA sub-module, which will then be installed with the command

```
 python setup.py install
```

Remove the old environment, if necessary:

```
conda deactivate
```

```
conda env remove --name alpha-beta-crown
```

Install all dependents into the alpha-beta-crown environment:

```
conda env create -f complete_verifier/environment.yaml --name alpha-beta-crown
```

Activate the environment

```
conda activate alpha-beta-crown
```

To run alpha-beta-CROWN, we need to run abcrown.py, located in the complete_verifier folder:

```
    python abcrown.py --config exp_configs/vnncomp23/acasxu.yaml
```

Alpha-beta-CROWN will begin to verify the benchmark and at the end it will output a summary as such:

```
Result: safe in 5.5502 seconds
############ Summary ############
Final verified acc: 74.73118279569893% (total 186 examples)
Problem instances count: 186 , total verified (safe/unsat): 139 , total falsified (unsafe/sat): 47 , timeout: 0
mean time for ALL instances (total 186):1.9213825520794554, max time: 56.16105675697327
mean time for verified SAFE instances(total 139): 2.330431156021228, max time: 56.16105675697327
mean time for verified (SAFE + UNSAFE) instances (total 186): 1.9213826553795927, max time: [4.987591743469238, 1.0515899658203125, 1.1373426
914215088, 1.1383116245269775, 0.9410784244537354, 1.0703227519989014, 0.8995537757873535, 0.8275392055511475, 0.7465686798095703, 1.03259205
81817627, 1.2449369430541992, 1.0392775535583496, 1.0992765426635742, 1.6034088134765625, 1.3225817680358887, 1.7448487281799316, 1.872863769
53125, 2.144280433654785, 1.1417665481567383, 1.228705883026123, 1.079270839691162, 1.18998384475708, 1.3613700866699219, 1.5798993110656738,
 1.8165299892425537, 1.8878631591796875, 2.9490511417388916, 1.408735990524292, 1.1001667976379395, 1.1289374828338623, 1.1405465602874756, 1
.3316984176635742, 1.901839017868042, 2.3600962162017822, 1.992403268814087, 2.4158427715301514, 1.1141650676727295, 1.1582543849945068, 1.13
18187713623047, 1.070127248764038, 1.2881455421447754, 1.627547264099121, 1.6446032524108887, 2.1017229557037354, 1.6175827980041504, 3.92093
3723449707, 1.8419489860534668, 1.8132131099700928, 2.144282579421997, 56.16105675697327, 49.79389214515686, 17.78862476348877, 1.43561410903
93066, 2.144057512283325, 0.5397024154663086, 0.5348045825958252, 0.304715633392334, 1.2868564128875732, 1.0832734107971191, 0.84339928627014
16, 0.3096454143524170, 0.5361270904541016, 0.32764410972595215, 0.31276535987854004, 0.30327343940734863, 0.32598447799682617, 0.552764415740
9668, 0.7603826522827148, 0.6221842765808105, 0.6466856002807617, 0.5361919403076172, 0.6036214828491211, 0.3056817054748535, 0.5362575054168
701, 0.6216149330139160, 1.3513917922973633, 1.4622435569763184, 1.4699289798736572, 0.5561351776123047, 0.31014204025268555, 0.54496622085571
29, 0.5411732196807861, 0.3111126422882080, 0.6180894374847412, 1.4705798625946045, 1.2620623111724854, 0.5442743301391602, 0.5394802093505859
, 0.5355291366577148, 0.5391213893890381, 0.3042013645172119, 0.5446953773498535, 0.5640759468078613, 3.1384294033050537, 1.6805551052093506,
 1.3555181026458740, 0.6106760501861572, 0.6187770366668701, 0.5591793060302734, 0.5940217971801758, 0.5739064216613770, 0.5577661991119385, 0.
5532572269439697, 0.5681223869323730, 0.5690176486968994, 0.5656690597534180, 0.6843051910400391, 0.3178544044494629, 0.5580251216888428, 0.629
2526721954346, 0.3193233013153076, 0.5771796703338623, 0.5788996219635010, 0.5630974769592285, 0.5721838474273682, 0.6546056270599365, 0.55610
60905456543, 0.3171894550323486, 0.5462350845336914, 0.5434105396270752, 0.5622584819793701, 0.5890798568725586, 0.5958678722381592, 0.57717
46635437012, 0.5772185325622559, 0.5706882476806641, 0.5563659667968750, 0.5688810348510742, 0.5535593032836914, 0.5597419738769531, 0.6780183
3152771, 0.3133671283721924, 0.3093545436859131, 0.5492215156555176, 0.5477604866027832, 35.472904443740845, 18.174123287200928, 7.1825716495
51392, 5.550166606903076]
mean time for verified UNSAFE instances (total 47): 0.7116434726309269, max time: 22.987380981445312
safe (total 139), index: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 3
1, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 51, 52, 53, 65, 73, 90, 91, 92, 93, 94, 95, 99, 100, 101, 102, 103, 104, 105, 106,
 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134,
135, 136, 137, 138, 139, 140, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 1
66, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 184, 185]
unsafe-bab (total 5), index: [46, 47, 49, 83, 182]
unsafe-pgd (total 42), index: [48, 50, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 66, 67, 68, 69, 70, 71, 72, 74, 75, 76, 77, 78, 79, 80, 81
, 82, 84, 85, 86, 87, 88, 89, 96, 97, 98, 141, 142, 143, 183]
```

Here, "total verified" and "total falsified" are the number of instances in which the benchmark returned a correct and incorrect answer, respectively. We can also notice a timeout of 0, which means that none of the instances timed out during the verification process.

4.2 Nnenum [4]

NNenum is a high performance neural network verification tool. It uses multiple layers of abstractions in order to quickly verify neural networks without the cost of completeness. During VNN-COMP 2020, nnenum was the fastest running tool. Even though every contestant ran their tool on their own hardware, nnenum's speed is attributed to several algorithmic optimizations[6].
Prerequisites:

- Docker
- Python

To install nnenum, we first must clone the GitHub repository from the following address: `https://github.com/stanleybak/nnenum`. Next, we need to run the following command to build the tool using Docker:

```
docker build . -t nnenum_image
```

This will install all the prerequisites such as the required Python libraries.
Now that nnenum is installed, we can run it. First we need to run the following command:

```
docker run -it nnenum_image bash
```

This will start the shell in which nnenum is going to run.
In order to run onnx and vnnlib files one-by-one, we can use the follwing command from within this shell:

```
python3 -m nnenum.nnenum examples/acasxu/data/ACASXU_run2a_3_3_batch_2000.onnx
examples/acasxu/data/prop_9.vnnlib
```

Alternatively, to run all the onnx and vnnlib files for ACAS Xu, we need to run the Python script located at examples/acasxu/acasxu_all.py using the following command:

```
python3 acasxu_all.py
```

After running, nnenum saves the results in `examples/acasxu/results/full_acasxu.dat`, which is a file that contains the verdict for every instance and looks like this:

```
1_1 1 holds 0.9043560099999013
1_2 1 holds 0.7880940459999692
1_3 1 holds 0.9245396309997886
5_9 1 holds 2.6333773339997606
1_1 2 holds 0.9800955360001353
1_2 2 violated 0.6960424779999812
1_3 2 violated 1.2031206550000206
1_4 2 violated 0.8025689860000966
1_5 2 violated 0.6941987039999731

...

5_7 4 holds 0.5372130199998537
5_8 4 holds 0.5712843749997774
5_9 4 holds 0.54370307399995
1_1 5 holds 1.5234227890000511
1_1 6 holds 5.243353473999832
1_9 7 violated 0.7887507830000686
2_9 8 violated 0.68229071199994
3_3 9 holds 3.115485338000326
4_5 10 holds 1.0181782939998811
```

"Holds" means that the instance returned the correct answer and "violated" means that the instance returned the wrong answer.

5 Experimental Results

#	Tool	Verified	Falsified	Penalty	Score	Percent
1	nnenum	139	47	0	1860	100%
2	$\alpha - \beta - CROWN$	139	47	0	1860	100%

Alpha-beta-CROWN ran for approximately 5-6 minutes before returning the final answer and nnenum ran for 4 minutes before returning the answer.
From our analysis, both tools have managed to get a perfect score. The column "Verified" indicates how many instances the tool detected where the benchmark gave the correct answer. In the tool results, this is also referred to as "safe".
The column "Falsified" indicates how many instances the tool detected where the benchmark gave the wrong answer. In the tool results, this is also referred to as "unsafe".
None of the tools received a penalty because they both correctly evaluate the answers given by the benchmark, without missing any deadlines. Thus, they both received a 100% rating with a perfect score of 1860. The score is calculated as follows:

$$Verified * 10 + Falsified * 10 - Penalty * 150$$

These results match the official results from VNN-COMP23 [1]. The raw results from our testing can be found at the GitHub repository: `https://github.com/Vaduva-Vlad/ProiectVF`

5.1 Obstacles encountered

Although the process of running both tools is fairly smooth, we encountered a couple of problems along the way:

- Alpha-beta-CROWN runs only on NVidia graphics cards. As such, the tool could only be run by the team members who have an NVidia graphics card.
- Alpha-beta-CROWN had a more complicated and error-prone installation process, requiring the installation of miniconda, Auto-LiRPA and the tool itself.
- Nnenum had to be run twice, because the first run returned an error on one of the instances. However, subsequent runs were successful, so we decided to give nnenum the maximum score.
- We've managed to install nnenum only on Windows, as the Docker installer did not work on Linux and manual installation returned many errors.

6 Conclusions

ACAS Xu is collision avoidance system that aims to increase the safety of air traffic. However, as suggested by our tests, the accuracy of the model (74.73%) should be higher in order to reach this goal. This is why neural network verification is important. It is an important step towards assuring that the neural network is safe to use before integrating it into the aircraft. It is also a cost-saving measure as it reduces the failure rate in physical planes and lowers the number of materials wasted.
Measuring the accuracy of neural network verification tools is important, because it provides a continuous tool-benchmark improvement. In order for the neural network results to be correct, the benchmark verification needs to be accurate as well.

References

[1] C. Brix, S. Bak, C. Liu, and T. T. Johnson, "The fourth international verification of neural networks competition (vnn-comp 2023): Summary and results," 2023.

[2] S. Bak and H.-D. Tran, "Neural network compression of acas xu early prototype is unsafe: Closed-loop verification through quantized state backreachability," 2022.

[3] H. Zhang, "alpha-beta-CROWN." `https://github.com/Verified-Intelligence/alpha-beta-CROWN`, 2023.

[4] S. Bak, "nnenum." `https://github.com/stanleybak/nnenum`, 2023.

[5] B. Krishnamurthy, "An introduction to the relu activation function." `https://builtin.com/machine-learning/relu-activation-function`, 2022.

[6] S. Bak, "nnenum: Verification of relu neural networks with optimized abstraction refinement," in *NASA Formal Methods Symposium*, pp. 19–36, Springer, 2021.