USING VERIFICATION TOOLS TO PROVE SAFETY OF A NEURAL NETWORK-BASED CYBER-PHYSICAL SYSTEMS (CPS)

Tanmay Khandait
Student ID : 1219385830
CIDSE, Arizona State University
tkhandai@asu.edu
Safe Autonomy for CPS (2021 Fall)

December 10, 2021

ABSTRACT

Neural Networks are being increasingly used in the domain of perception, prediction and control of various cyber-physical systems (CPS). This has also led to an increased research in the verification and testing community to develop tools that can verify the properties of a neural network. This becomes extremely useful when deployed in systems which have a huge cost in case of a failure, aka, safety-critical systems. In this project, we utilise the ACAS Xu benchmark and try to find a single falsifying input that can falsify maximum number of neural networks from the dataset.

1 Introduction

Neural Networks are increasingly being used for perception, predication and control of various cyber-physical systems [1, 2, 3]. These cyber-physical systems are usually systems which are in constant interaction with humans and/or are often expensive in their development. Such systems are often referred to as safety critical systems. It thus becomes important to find regions or inputs that can lead to the failure of the system.

This becomes all the more important when we look at applications that are autonomous in nature. One such application is the Airborne Collision Avoidance System X Unmanned (ACAS Xu) benchmark. This goal of this system is to reduce mid-air collision by proving horizontal maneuver advisories[4, 5]. Initially, this system was based on a large lookup table that mapped the sensor values of the aircraft to the advisories. This however consumed a large amount of space in the vicinity of 2GB[6]. This becomes a challenge when deploying on any hardware. To solve this problem, a neural network solution was proposed. Further in time, this large single neural network was broken down into multiple neural networks which had similar behavior. This dramatically reduced the space to just around 3MB of memory, while performing better than its counterpart. The major advantage was that neural networks tends to capture the continuous behavior of the system more efficiently than a lookup table, which essentially is discrete in its nature. The implication of this is that the neural networks could capture and provide a solution to values not present in the lookup table more efficiently than some interpolation methods.

ACAS-Xu Benchmark: They are a set of 45 neural networks designed, where each of these networks have 5 nodes in the input layer, 50 neurons in each of the 6 hidden layers which have ReLU activation, and 5 nodes in the output layer, totalling up to 300 hidden neurons in total. Each of the 5 nodes in the input and output nodes have a physical meaning attached in it. The input nodes represent the sensor readings from the unmanned aircraft (see Figure 1):

- ρ represents the distance of the ownship to the intruder aircraft.
- θ represents the angle to the intruder relative to the ownship heading direction.
- ψ represents angle in which the intruder is heading towards with respect to the ownship's heading direction.
- V_{own} represents the speed of the ownship.;
- V_{int} represents the speed of the intruder.

The 5 outputs represent the horizontal maneuver advisories to the aircraft and are labeled as Clear of Conflict (COC), Weak Left (WL), Weak Right (WR), Strong Left (SL) and Strong Right (SR) [7]. This neural network architecture has been shown in Figure 2.

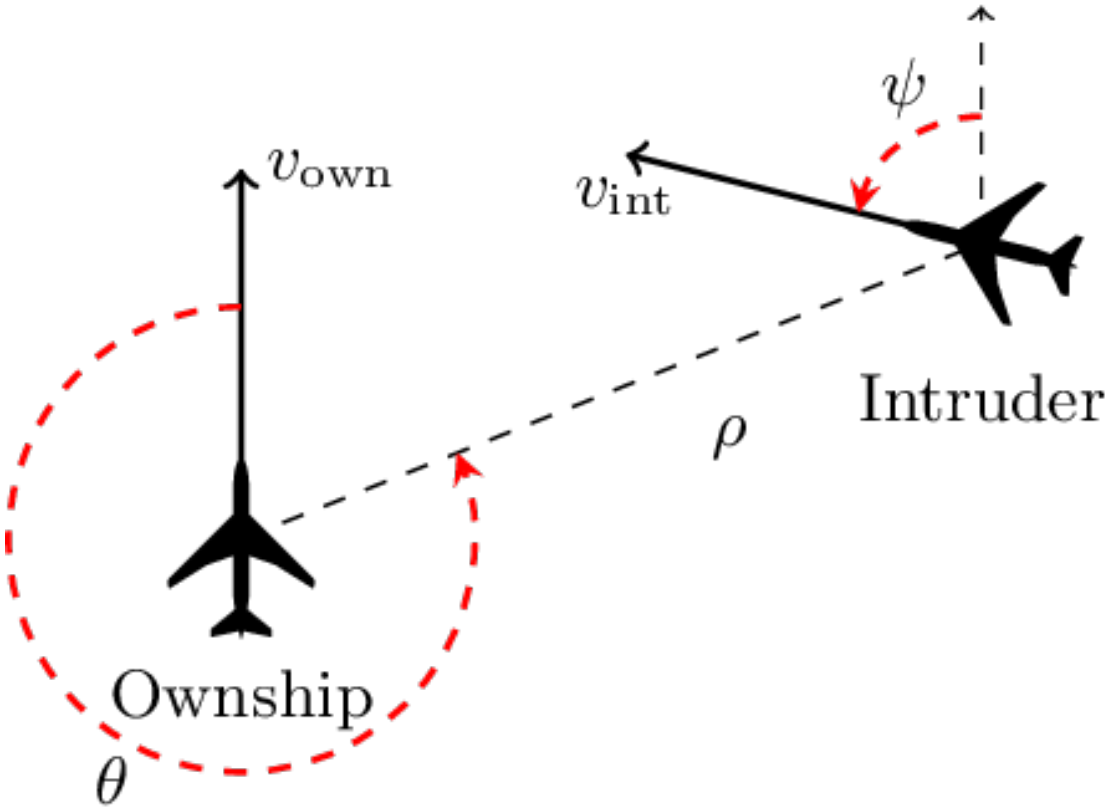

Figure 1: Physical meaning of inputs to the neural network

Maneuveur Advisories: Given a neural network, the output from ACAS-Xu benchamrk are the advisories. The idea is to formulate these advisories as test-case scenarios hich have been defined as 10 properties for the output. These properties are relationships between the outputs. We specifically use the property ϕ_7 which has a desirable property that the if the vertical separation between the ownship and intruder is large, the scores for strong left (SL) and strong right (SR) are never minimal, that is they are never chosen [4]. This can be defined as

$$\phi_7 = ((\text{SR} > \text{COC} \wedge \text{SL} > \text{COC}) \vee (\text{SR} > \text{WL} \wedge \text{WL} > \text{COC}) \vee (\text{SR} > \text{WR} \wedge \text{SL} > \text{WR}))$$

The input domain for the property ϕ_7 is as follows:

- $\rho \in [0, 60760]$
- $\theta \in [-\pi, \pi]$
- $\psi \in [-\pi, \pi]$
- $V_{own} \in [100, 1200]$
- $V_{int} \in [0, 1200]$

With this discussion, it is evident that neural network controllers do have some interesting application in safety-critical systems and that efforts must be put in to verify them. In this project, we try to come up with a pipeline that can find falsifying inputs to ACAS Xu benchmark. We define the literature review in section 2. We then proceed to show the architecture and the experimental setup in section 3. Sections 4 and 4 present the evaluations and results from the work done. We then move on to the concluding notes.

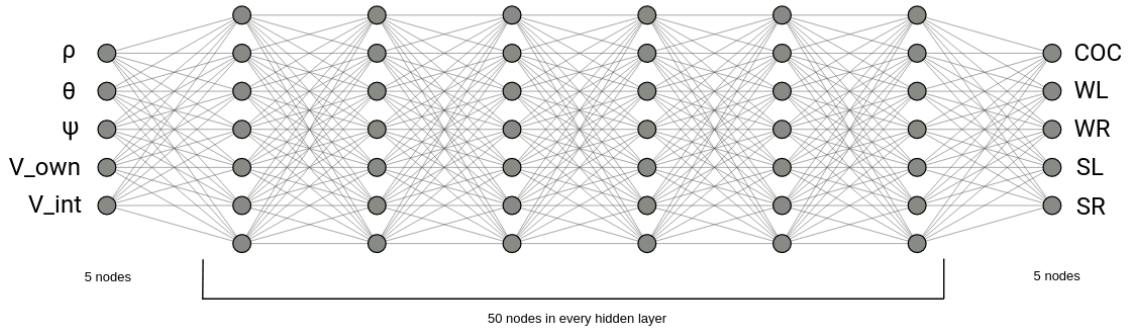

Figure 2: NN architecture for ACAS-Xu Benchmark

2 Literature review

Significant amount of research has been happening in verifying the ACAS Xu networks. This project is focused on at algorithms that are search algorithms directed by optimization algorithms. The SHERLOCK algorithm works by posing the problem as a range estimation problem and reducing the verification problems to this algorithm [8]. To solve the range estimation problem, it performs a series of local searches by solving a linear program and global search by solving Mixed integer linear program. The Branch-and-Bound algorithms [9] solves the verification problem by posing it as an optimization problem. The upper and lower bounds of a region serve as an estimate of the global maximum and the global minimum once the algorithm terminates. The algorithm works by splitting the input domain into smaller and smaller subregions until the upper and lower bounds differ by a certain scalar δ.

Reluplex [4] and Marabou [10] are SMT based solver that work by first inferring bounds on each node and then applies a simplex search over the linear constraints. This helps in directing the search over the non-linear constraints [4, 10]. These two tools have been used as benchmark for verifying neural networks.

Part-X [11] is one of the new algorithms that works by finding falsifying regions by directing the search using Bayesian optimization over the input domain.

3 Work Done

The expected deliverable from this project is try to apply Part-X algorithm to find falsifying inputs over the 45 neural networks.

Goal: Theoretically, I want to find inputs that falsifies all the networks.

ACAS Xu benchmarks, which were initially look-up tables, aimed at covering all the corner cases so that the system could be verified. Moving on, neural networks were used to learn this lookup table with the aim of covering much more corner cases that could falsify the system. However, if we still end up finding some inputs that can falsify all the networks, it could potentially contribute to re-investigating these cases.

3.1 Architecture

There are two parts to this work. First is formulating the blackbox function, which in this case is the neural network that are meant to satisfy the properties (in our project ϕ_7). Building a robustness metric for calculating the robustness of a neural network is still under research. Various methods calculate this metric by searching over a neighborhood, I took a rather different approach.

Falsification Metric: I consider the number of neural networks that are not satisfying the predicate ϕ_7. Formally,

$$\text{Falsification Metric:} \quad R = -1 * \Sigma_1^{45} + 20 \neg \phi_7^{\text{NN}^i}$$

We can easily notice if property ϕ_7 is not satisfied on any of the networks, then the score will be -25 and if all the property phi_7 is satisfied on all the networks, we get a score of 20. This particular value of 20 has no certain significance, and was completely based on observation. The falsification occurs when the ϕ_7 is violated in at least 21 neural networks. The idea is now to minimize this metric R.

The second part is to pass this blackbox function to the Part-X optimizer and search for falsifying inputs. The architecture is shown in Figure 3.

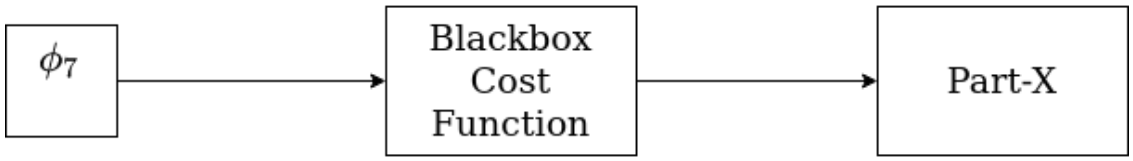

Figure 3: The specification phi_7 is given to the Blackbox model to build the Cost Function R. This blackbox is given to the Part-X algorithm with the aim to find falsifying inputs for the given blackbox.

3.2 Setting up ACAS Xu Benchmark

The ACAS Xu benchmarks provide 45 trained neural nets in the for of .nnet file[1]. Using a python script, these networks were loaded along with their weights and converted to tensorflow models[2]. These models were then converted into a multi-input multi-ouptut model. A X-input X-output model basically builds a model which has X neural nets. This is particularly suitable for us because we want to give the same input point to the 45 neural networks to test against the property ϕ_7. Figure 4 shows a 2-input 2-output model, where as in our actual experiment, a 45-input 45-output model is used (not shown here due to space constraints).

3.3 Setting up Part-X

The cost function was implemented and is passed as a parameter to the Part-X optimizer. Other parameters for part-X were determined based on the multiple iterations and seeing the behavior of Part-X after the first branching and classification of the region is done.

There were some issues with the Part-X package in terms of choosing the kernels for gaussian process regression and optimizer for calculating expected improvement in Bayesian Optimization. After testing, the matern kernel was chosen with parameter $nu = 2.5$ and the optimizer was L-BFGS-B. For calculating the expected improvement in bayesian optimization, dual-annealing (a global search method) was used.

4 Experimental Evaluation

Two sets of experiments were performed.

Experiment 1 - Uniform Random Sampling: In this experiment, we generate a set of 2000 points using a uniform random sampler across the input domain. This acts like a benchmark for further experiments.

Experiment 2 - Part-X: In this experiment, we run the Part-X algorithm for a budget of 2000 points using the following parameters to the optimizer:

- Initialization Budget = 50
- Bayesian optimization Budget = 30
- Continued Sampling Budget = 50
- Maximum Budget = 2000
- Monte Carlo Quantile Estimation Parameters: R= 30, M = 10000
- Number of macro-replications = 1

All these experiments were run on 20 core CPU.

[1]Models available here: `https://github.com/NeuralNetworkVerification/Marabou/tree/master/resources/nnet/acasxu`

[2]Original Script here: `https://github.com/rnbguy/acasxu_tf_keras/blob/master/gen_tf_keras.py`

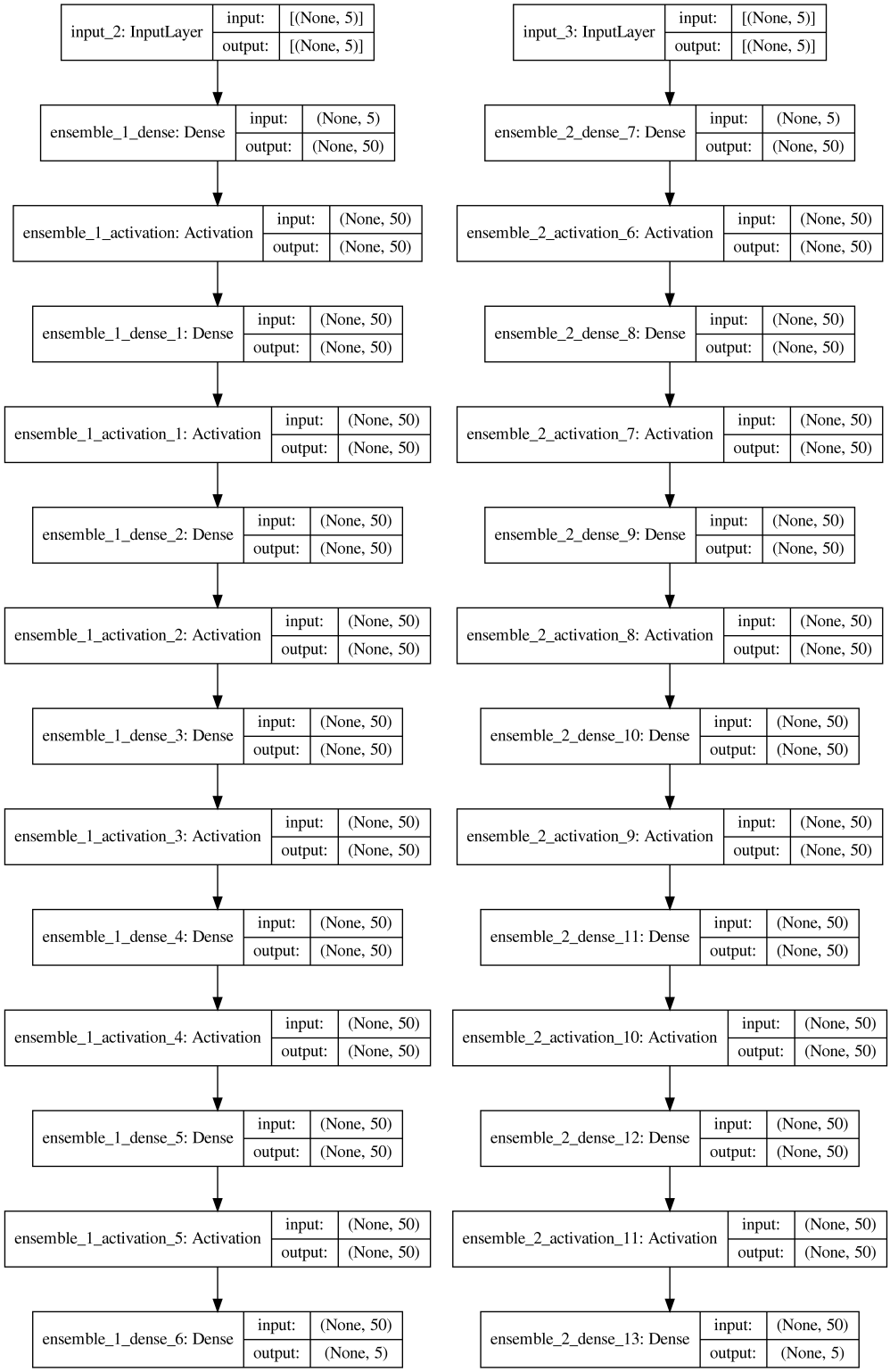

Figure 4: 2-Input 2-Output model. This model basically takes two neural networks. Actual test were run on a 45-input 45-output model.

Uniform Random			Part-X		
Mean Nets Falsified	Min Nets Falsified	Max Nets Falsified	Mean Nets Falsified	Min Nets Falsified	Max Nets Falsified
14.169	11	25	16	11	27

Table 1: Comparison of Uniform Random vs Part-X.

4.1 How to run the code?

Please make sure that you have poetry[3] installed. Enter the nn_example folder and first run poetry update. This will make the directory ready and install all the dependencies. Then go ahead and run the command poetry run python exp_1.py to run the experiment 1 or run the command poetry run python exp_2.py to run the experiment 2.

5 Key Results and Discussion

As we see in Table 1, Part-X is able to find falsifying inputs to a mean of 16 neural networks in comparison to 14.17 in UR. It was observed that Part-X classified an entire region as greater than or equal to 0. This was an expected output since the Part-X is able to sample more intelligently, in comparison to uniform sampling. Moreover, the fact that Part-X classifies a region positively with 95% confidence gives us enough reason to look at other regions. At this point, not much can be inferred from the Part-X results, since the budget allocated was less.

I was hoping Part-X to perform magnitudes better than the uniform random. But I believe this has something to do with the fault in the way I have described the robustness measure. However, I plan to research deeper into this and get to the crux of whats happening.

Something interesting is that in most of the falsifying cases, the intruder and own ship are moving in parallel and away from each other. Physically, this case will not lead to a collision irrespective of moving straight or taking a weak or strong turn, however more in depth analysis needs to be done to understand the implications of such results.

6 Status and Lessons Learned

Currently, as we presented in the section 5, we have some promising preliminary results from the part-X optimizer on the ACAS Xu benchmark. Though the results section is not as strong as it is intended to be, the results from Part-X seem to be poinint got some direction in the are o repair of neural networks. The end goal from this project was to use the negatively classified regions found by Part-X to repair the neural network. However, due to time and resources constraint, I was unable to get to the repair problem. The major challenges were:

- One of the major challenge was to come up with a way to assess the falsification metric for the problem at hand. This was one the most time taking tasks.
- The blackbox model took around 2 seconds to do a function evaluation. This led to longer run times. Since Part-X is still in its developmental steps, a lot of aspects of the code were to be experimented in order to come up with the parameters and setting for the results presented in the report.
- Lack of knowledge and exposure to falsification of neural networks forced me to spend a lot of time studying and understanding topics in this area.

The key learning from this project include:

- Formulation of the problem at hand. In this case, a CPS was compiled and falsifying inputs were found to the neural networks.
- Exposure to various papers in the field of verification and repair of neural network.

7 Relevance to Course

In this project, we used Part-X to find falsifying inputs of a safety critical neural network controller.

[3]Find poetry Documentation here: `https://python-poetry.org/`

8 Conclusion

Neural Network controllers are being increasingly used in safety-critical systems and this calls the need for verifying and validating them. Through this project, I wanted to get familiar with the research happening in the field of verifying of neural network. This was established using the ACAS Xu system coupled with Part-X optimizer. This project opens up avenues in various directions, one of them being the repair problem.

References

[1] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2722–2730, 2015.

[2] Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos A. Theodorou, and Byron Boots. Agile autonomous driving using end-to-end deep imitation learning. In *Robotics: Science and Systems*, 2018.

[3] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars, 2016.

[4] Guy Katz, Clark Barrett, David Dill, Kyle Julian, and Mykel Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks, 2017.

[5] Michael Marston and Gabe Baca. Acas-xu initial self-separation flight tests. 2015.

[6] Kyle D. Julian, Jessica Lopez, Jeffrey S. Brush, Michael P. Owen, and Mykel J. Kochenderfer. Policy compression for aircraft collision avoidance systems. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pages 1–10, 2016.

[7] Mykel J. Kochenderfer and James P. Chryssanthacopoulos. Robust airborne collision avoidance through dynamic programming. 2011.

[8] Souradeep Dutta, Xin Chen, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Sherlock - a tool for verification of neural network feedback systems: Demo abstract. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '19, page 262–263, New York, NY, USA, 2019. Association for Computing Machinery.

[9] Rudy Bunel, Ilker Turkaslan, Philip H.S. Torr, Pushmeet Kohli, and M. Pawan Kumar. A unified view of piecewise linear neural network verification. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 4795–4804, Red Hook, NY, USA, 2018. Curran Associates Inc.

[10] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, David L. Dill, Mykel J. Kochenderfer, and Clark Barrett. The marabou framework for verification and analysis of deep neural networks. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification*, pages 443–452, Cham, 2019. Springer International Publishing.

[11] Giulia Pedrielli, Tanmay Khandait, Surdeep Chotaliya, Quinn Thibeault, Hao Huang, Mauricio Castillo-Effen, and Georgios Fainekos. Part-x: A family of stochastic algorithms for search-based test generation with probabilistic guarantees, 2021.