Automated Repair of Neural Networks

Author: Dor Cohen
Advisor: Prof. Ofer Strichman

Industrial Engineering & Management, Information Management Engineering
Technion — Israel Institute of Technology

June 11, 2022

Table of Contents

Introduction

- Artificial Neural Networks (NNs) have been widely used recently
- NNs excel at solving image and natural language processing tasks
- However, they were also found to be susceptible to **Adversarial Attacks**
- This drawback limits the usage of NNs in **safety-critical** systems

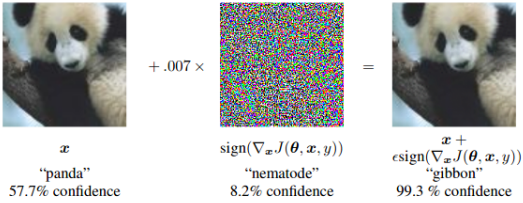


Figure: Depicts the effect of adding noise to an input example - this slight modification shifted the network prediction. Image is borrowed from [goodfellow2015explaining]

Verification of Neural Networks

- As a result, tools for verifying the safety of NNs were developed (e.g., [katz2017reluplex])
- Such tools provide a **safety certificate** given NN and desired properties
- Or, a counter-example in case the NN does not satisfy the specification

Verification of Neural Networks

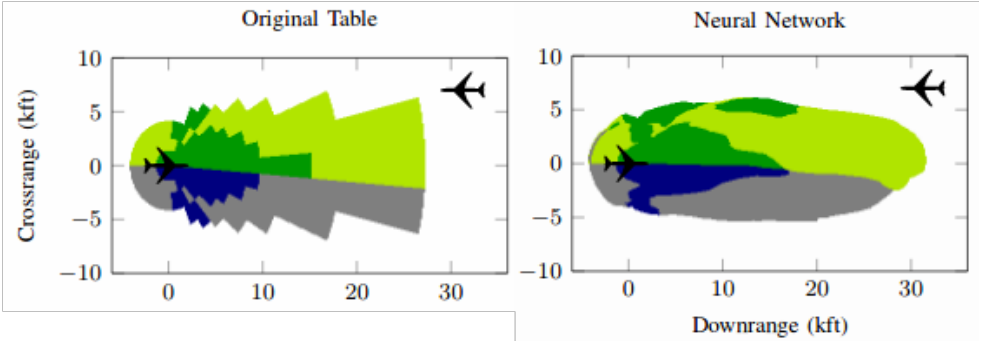


Figure: Taken from [julian2016policy] - depicted at left are the decision boundaries derived from the original *ACAS Xu* hash table. At the right figure, we can observe the trained NN boundaries. Each color depicts a different action that the aircraft should carry, with respect to the intruder position. It can be seen how the NN imposes unsafe regions, in contrast to the original table.

Table of Contents

Repair of Neural Networks

Problem definition: Consider a NN model denoted by $f_{w_A}(x)$ (model A). Assume that $f_{w_A}(x)$ is not safe w.r.t. a required specification φ. In the task of provably repairing a NN, we search for a new model $f_{w_B}(x)$ (model B) which satisfies φ and its decision boundaries are similar as possible to model A.

- Can be achieved naively via verification and re-training, until convergence
- In this work, we attempt to mitigate the repair process by utilizing tools from the field of *formal verification*

Table of Contents

Preliminaries - Linear classification for visual recognition

- Class score as a weighted sum of all of its pixel values
- E.g., we'd expect a ship classifier to have positive values in its blue channel weights

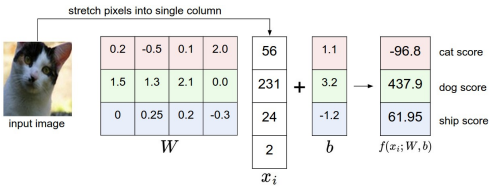


Figure: Taken from CS231N - Stanford's course for Visual Recognition with NNs - This is an example of mapping an image to class scores by using a linear classifier.

Preliminaries - Neural Network (NN) definition

- A mapping: $f_w : X \rightarrow Y$
  - $X \in \mathbb{R}^n$, $Y \in \mathbb{R}^m$, $n, m \in \mathbb{N}$ and w denotes the set of function **parameters**.
- Usually a composition of k functions: $f_w(x) = f^k(...(f^2(f^1(x))))$,
  - where k is referred to as the **number of layers**
- Typically, each layer performs a linear transformation followed by non-linear: $f^i(x) \equiv g_i(W_i x + b_i)$,
  - g_i denotes the non-linear operation (ReLU is a common one: $g(x) = max(0, x)$)
  - The shapes of W_i, b_i indicate the number of neurons for specific layer i

Preliminaries - Neural Network (NN) example

- For example, a two-layer NN with d ReLU neurons, an input $x \in \mathbb{R}^n$ and output $y \in \mathbb{R}^m$ will have the following form:

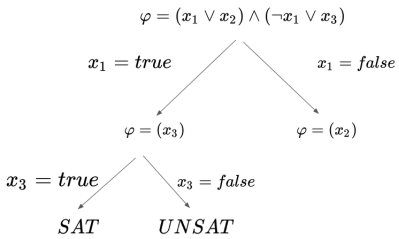
$$f_w(x) = W_2 * max(0, W_1 x + b_1) + b_2 \tag{1}$$

- where $W_1 \in \mathbb{R}^{d \times n}$, $b_1 \in \mathbb{R}^d$, $W_2 \in \mathbb{R}^{m \times d}$, $b_2 \in \mathbb{R}^m$

SAT Problem

- A Boolean formula φ is said to be in *CNF* if it's a **logical conjunction** of a set of clauses $C_1, ..., C_n$, where each clause C is a **logical disjunction** of a set of literals
- SAT: deciding if there exists an **assignment** that satisfies φ

- Example: $\varphi = (x_1 \vee x_2) \wedge (\neg x_1 \vee x_3)$
- Satisfying assignment: $\{x_1 = 1, x_2 = 1, x_3 = 1\}$, $\rightarrow \varphi$ is SATISFIABLE
- Solved by DPLL in practice (a backtracking based search algorithm)



Satisfiability modulo theories (SMT)

- Extends the SAT problem by supporting more theories (e.g., linear real arithmetic)
- Fundamentally solved with DPLL (similar to SAT) and a corresponding theory solver

- For instance, consider the real variables x, y in this formula: $\varphi = (y > 3x) \wedge (x >= 3)$
- Can re-write as: $\varphi = A \wedge B$
- At first, let's try to assign $B = \mathit{True}$
- A theory solver is invoked and assigns $x = 3$, which satisfies $B = \mathit{True}$
- After simplification, we get $y > 9$
- Finally, a possible solution is: $x = 3, y = 10$

(NN) Verification

- Verification methods allow to prove whether functions fulfill a certain **specification**
- Let f_w be such a function, with a given parameter set w
- A general specification can be formulated in first-order logic as follows:

$$\forall x \;\; C_{in}(x) \implies C_{out}(f_w, x) \tag{2}$$

where C_{in} and C_{out} denote input and output constraints functions respectively.

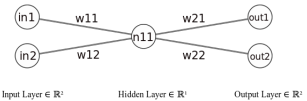
NN Repair

- Consider the case where f_w has an **unsafe** property
- We can slightly modify Eq. 2 to search for a new model $f_{\hat{w}}$
- The new model shall satisfy the specification, assuming such model exists
- That is by adding an existential quantifier:

$$\exists \hat{w} \;\; \forall x \;\; C_{in}(x) \implies C_{out}(f_{\hat{w}}, x) \tag{3}$$

Example - SMT formula to repair an NN

Consider the following NN, it has two inputs ($in1, in2$), a hidden layer with one **ReLU** neuron ($n11$), two outputs ($out1, out2$) and 6 weights ($w11, w12, w21, w22, bias11, bias21, bias22$):



Assume that this NN violates a required **specification**: $\forall in_1 \in [0, 0.5] \ .out_1 > out_2$. We then search for a repaired NN, by choosing to free two weights (while the rest remain constant):

$$
\begin{aligned}
&\exists w_{21}, bias_{21} \ (\\
&\quad \forall in_1, in_2 \ (\\
&\quad n_{11} = \text{if-then-else}(in_1 * w_{11} + in_2 * w_{12} + bias_{11} > 0, in_1 * w_{11} + in_2 * w_{12} + bias_{11}, 0) \wedge \\
&\quad out_1 = w_{21} * n_{11} + bias_{21} \wedge \\
&\quad out_2 = w_{22} * n_{11} + bias_{22} \wedge \\
&\quad (in_1 >= 0) \wedge (in_1 <= 0.5) \implies (out_1 > out_2) \\
&) \ )
\end{aligned}
$$

Adversarial Robustness property

Adversarial robustness defines **invariance to noise** in a δ neighborhood of x_0 as follows:

$$\forall x.\ \|x - x_0\| \leq \delta \implies D(f_w(x)) = D(f_w(x_0)), \tag{4}$$

where D denotes the decision function, usually $D(f(x)) = \underset{j}{\text{argmax}}(f(x))$ where j denotes the model predicted category.

Table of Contents

Repair w.r.t. Adversarial Robustness property

To generate a repaired network f_{w_B} which satisfies the safety properties, we search for a **subset** of weights $w_{B_{free}} \subseteq w_B$, and corresponding values, such that

$$\forall x. \quad \|x - x_0\| \leq \delta \implies D(f_{w_B}(x)) = D(x_0), \tag{5}$$

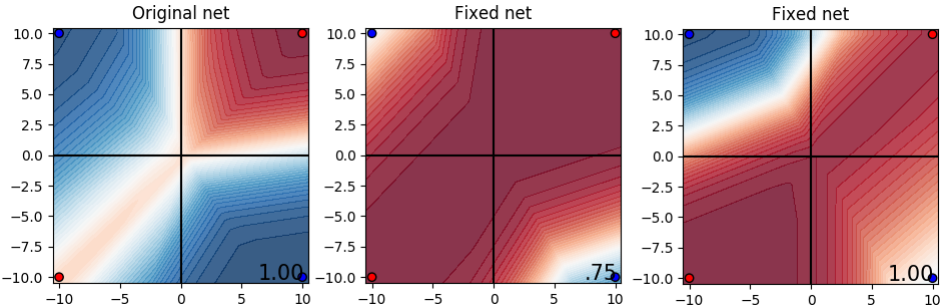
holds, where f_{w_B} is a NN in which the weights in $w_{B_{free}}$ are set to those values, and the rest are set to the values of the corresponding weights in the original NN.

Notice: we have many options for choosing $w_{B_{free}}$

Table of Contents

Initial Experiments

At first, we attempt to repair the following network (depicted on leftmost figure) to satisfy the **Adversarial Robustness** property w.r.t. point (-10, -10):



This experiment demonstrates that we should also preserve similarity

Experimental Setup

For the main part, we experiment with the following **neural networks** and their corresponding **data sets**:

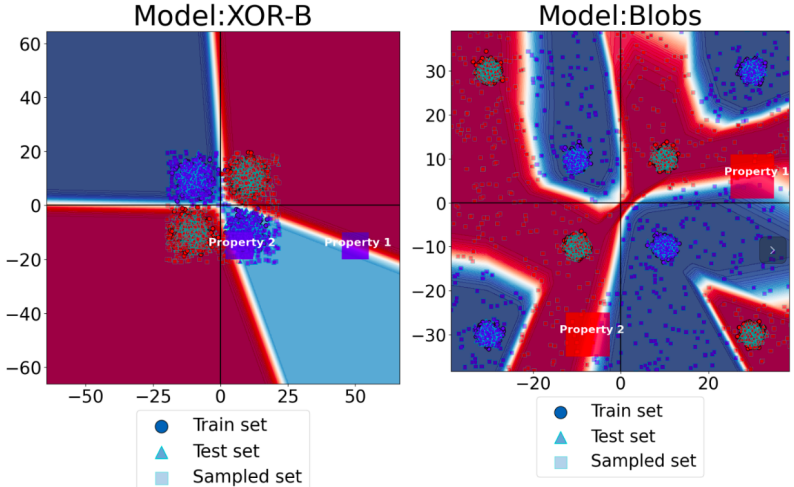
Model Name	Hidden Layers Sizes	Parameters	Train Set Size	Train Set Accuracy	Test Set Size	Test Set Accuracy	Sampled Set Size	Sampled Set Accuracy
XOR-B	[4]	22	1562	0.99743	1600	0.99125	500	1.0
Blobs	[10, 10]	162	6000	1.00000	4000	1.00000	1000	1.0

Further, we require these **properties** to hold:

Model Name	Property	Coordinate	Delta	Class	Norm	Violated
XOR-B	Property 1	[50, -15]	5	Blue (1)	L1	Yes
XOR-B	Property 2	[7, -15]	5	Blue (1)	L1	Yes
Blobs	Property 1	[30, 6]	5	Red (0)	L1	Yes
Blobs	Property 2	[-7.5, -30]	5	Red (0)	L1	Yes

Experimental Setup

Depicted below are the **decision boundaries** and **data sets** for each model, along with the required **specification**:



Experimental Setup - Technical Details

- **Z3** SMT solver and its **Python** wrapper to search for new weights assignments
- **PyTorch** with *Adam* optimizer to train the networks
- Experiments were performed on a computer with Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, using 8Gb of RAM, running Linux Ubuntu version 16.04.1
- Each repair trial was given a timeout of 600 seconds

Experiments - Similarity Heuristics

- As our initial experiments demonstrate, we shall employ **similarity-preserving** heuristics to keep the repaired NN decision boundaries as close as possible to the originals
- For this purpose we use three different heuristics which are illustrated below

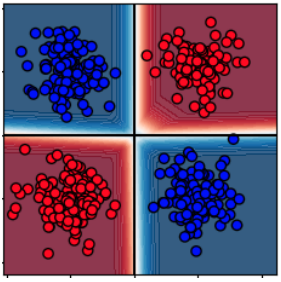


Figure: **Samples** - Enforce the model prediction on certain input coordinates.

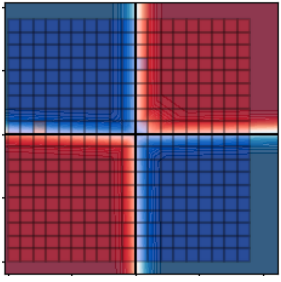


Figure: **Grid** - Divide the decision boundary into cells and enforce the same prediction for each cell.

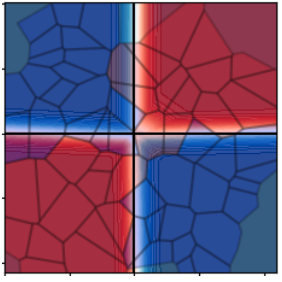
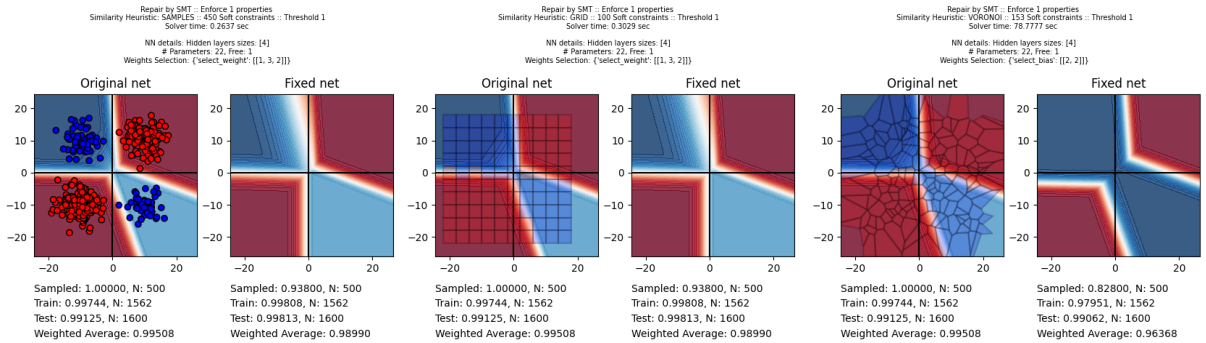


Figure: **Voronoi** - Build the cells according to Voronoi decomposition.

Experiments - Similarity Heuristics

- We perform extensive experiments to compare our proposed **similarity heuristics**.
- Following are visualizations for the repair of model *XOR-B*'s 1st property via each heuristic



Each similarity-preserving heuristic yields n additional constraints, in our experiments we require that **at least** $m \leq n$ constraints hold (we later use **threshold** to denote m).

Experiments - Similarity Heuristics

Attempts to repair each of ***XOR-B*** properties via each of our proposed heuristics:

Heuristic	Soft Constraints	Thresholds	Accuracy Before Repair	Max. Accuracy	Min. Accuracy	Average Accuracy	Trials	SAT	UNSAT	Timeout	Total Solver Time (sec.)
Weight selection tactic: **Individual weights**											
1st Property											
GRID	[100]	[1, 2]	99.5085%	85.9640%	37.5205%	60.1614%	44	9	26	9	6397
SAMPLES	[450]	[1, 250, 400]	99.5085%	86.9470%	37.5205%	71.8988%	66	15	51	0	32
VORONOI	[153]	[1, 2]	99.5085%	85.9640%	37.5205%	55.9803%	44	9	13	22	13994
2nd Property											
GRID	[100]	[1, 2]	99.5085%	**98.9896%**	37.5205%	65.5980%	44	10	24	10	6234
SAMPLES	[450]	[1, 250, 400]	99.5085%	**98.9896%**	37.5205%	83.6997%	66	24	42	0	32
VORONOI	[153]	[1, 2]	99.5085%	96.3681%	37.5205%	60.6827%	44	10	12	22	13843
Weight selection tactic: **Arbitrary combinations of weights, neurons or layers**											
1st Property											
GRID	[100, 400]	[1, 2, 10]	99.5085%	85.9640%	37.5205%	60.9448%	38	15	0	23	13812
SAMPLES	[450]	[1, 250, 400]	99.5085%	**98.6073%**	37.5205%	84.4578%	36	32	4	0	171
VORONOI	[268, 250, 153]	[1, 2, 10]	99.5085%	86.2370%	37.5205%	60.9030%	39	15	0	24	14424

The ***Samples*** similarity heuristic proves to be the most effective and efficient, as noticed by average accuracy and total solving time.

Experiments - Repair with *Samples* Similarity Heuristic

Aggregated search results for the repair of ***XOR-B*** network, with ***Samples*** similarity heuristics, results are grouped by soft constraints threshold value:

Threshold / Soft Constraints	Accuracy Before Repair	Max. Accuracy	Min. Accuracy	Average Accuracy	Trials	SAT	UNSAT	Timeout	Skipped	Total Solver Time (sec.)
1st Property										
Weight selection: **Combinations of two weights**										
1/1562	99.50847%	99.20808%	33.72474%	65.91046%	231	156	61	14	0	9127
325/1562	99.50847%	99.12616%	37.73894%	68.05714%	231	103	0	53	75	42027
750/1562	99.50847%	**99.26270%**	48.90770%	71.35950%	231	92	0	11	128	18004
1000/1562	99.50847%	98.60732%	61.22338%	80.55633%	230	74	11	6	139	13321
1500/1562	99.50847%	**99.26270%**	93.11851%	96.96887%	230	18	51	5	156	9529
2nd Property										
Weight selection: **Individual weights**										
1/1562	99.50847%	98.98962%	38.09394%	74.23539%	22	10	12	0	0	141
325/1562	99.50847%	97.95194%	37.52048%	74.13435%	22	10	12	0	0	75
750/1562	99.50847%	97.95194%	49.80885%	77.06475%	22	9	13	0	0	63
1000/1562	99.50847%	97.95194%	64.09066%	89.54806%	22	8	14	0	0	68
1500/1562	99.50847%	**99.31731%**	96.36810%	97.50137%	22	4	18	0	0	64
1561/1562	99.50847%	NA	NA	NA	22	0	22	0	0	27
1st And 2nd Property										
Weight selection: **Combinations of two weights**										
1/1562	99.50847%	97.21464%	20.04369%	57.71723%	231	119	95	14	0	8933
325/1562	99.50847%	90.44238%	36.15511%	63.79459%	231	76	0	43	112	34731
750/1562	99.50847%	**97.35117%**	48.90770%	68.24854%	231	65	0	11	155	11555
1000/1562	99.50847%	90.44238%	63.62643%	80.96122%	231	40	16	9	166	11202
1500/1562	99.50847%	97.05079%	96.28618%	96.66849%	231	2	37	1	191	3676

Produce safe networks by freeing up to two weights, with only a mild loss of accuracy

Experiments - Repair with *Samples* Similarity Heuristic

Aggregated search results for the repair of the ***Blobs*** network, with the ***Samples*** similarity heuristic, results are grouped by soft constraints threshold value:

Threshold / Soft Constraints	Accuracy Before Repair	Max. Accuracy	Min. Accuracy	Average Accuracy	Trials	SAT	UNSAT	Timeout	Skipped	Total Solver Time (sec.)
1st Property (individual weights)										
1/6000	100.00000%	99.43636%	64.10000%	89.52500%	107	48	44	15	0	18968
500/6000	100.00000%	99.41818%	86.50909%	95.03438%	107	32	0	16	59	11723
1000/6000	100.00000%	**99.50000%**	86.50909%	94.11420%	107	32	0	0	75	2046
2000/6000	100.00000%	99.43636%	86.47273%	94.44403%	107	32	0	0	75	2330
5500/6000	100.00000%	99.34545%	90.49091%	95.55016%	107	29	3	0	75	3518
2nd Property (individual weights)										
1/6000	100.00000%	99.72727%	51.12727%	86.66129%	143	31	95	17	0	23884
500/6000	100.00000%	99.72727%	86.15455%	92.72323%	143	18	0	13	112	9573
1000/6000	100.00000%	99.72727%	86.30909%	93.59798%	143	18	0	0	125	1406
2000/6000	100.00000%	**99.75455%**	64.67273%	90.52121%	143	18	0	0	125	1789
5500/6000	100.00000%	99.63636%	90.47273%	94.50160%	143	17	1	0	125	2580
1st And 2nd Property										
Combinations of size 2, assembled from top-5 repair setups of individual weights										
1/6000	100.00000%	99.49091%	62.45455%	89.15565%	45	33	12	0	0	5137
500/6000	100.00000%	99.10909%	74.16364%	93.26263%	45	9	0	24	12	15823
1000/6000	100.00000%	**99.56364%**	86.30909%	93.08636%	45	4	0	5	36	3477
2000/6000	100.00000%	86.89091%	86.30909%	86.55455%	45	3	0	1	41	747
5500/6000	100.00000%	98.81818%	97.35455%	98.23636%	45	3	0	0	42	638

Information from repairing single properties might be useful to further attempts of repairing multiple properties simultaneously. ***Blobs*** network has 162 parameters, and 13041 options to choose 2 parameters.

Experiments - Comparison to baseline

- We compare the results of repairing ***XOR-B*** and ***Blobs*** to a naive baseline
- Baseline: a repair loop which alternates between verification and re-training

Depicted are the accuracy measures for each of the networks and their repaired properties:

Model	XOR-B			Blobs		
Property	1st	2nd	Both	1st	2nd	Both
Our method	**99.2627%**	**99.3173%**	**97.3511%**	**99.5000%**	**99.7545%**	**99.5636%**
Naive method	97.2678%	96.4469%	94.5724%	97.3533%	97.8333%	96.0733%

Our method outperforms the naive method in terms of accuracy across all configurations

Automated Repair of Neural Networks

- Challenge in enumerating all possible weight combinations (n choose k)
- We suggest to perform the search in a greedy manner
- Provide a heuristic which attempts to minimize the UNSAT formulas we tackle

Thesis includes an algorithm to automatically repair NNs, a brief description:

Greedy repair (sketch)

1. Traverse a **search tree** where its nodes are combinations of free weights, and the tree level indicates the number of free weights.
2. Each **node** traversed corresponds to a **trial** which attempts to repair the network by freeing the given subset of weights.
3. Stop traversing according to timeout, return the most similar network as measured by accuracy.

Table of Contents

Contributions

- Demonstrate how off-the-shelf SMT solvers can be utilized to repair NNs
- Propose and experiment with similarity-preserving heuristics
- Perform additional experiments with the winning heuristic (Samples), and compare it to a naive baseline
- Suggest a greedy search algorithm to automatically repair NNs, and further strategies to improve its efficiency

Future work

- Perform additional experiments with larger benchmarks such as *Acas XU* networks, compare the results to related techniques
- Suggest further improvements to the algorithm
- Experiment with additional SMT solvers, attempt to fine-tune solvers to this specific task

Table of Contents

References


Ian J. Goodfellow and Jonathon Shlens and Christian Szegedy
Explaining and Harnessing Adversarial Examples
arXiv 1412.6572, 2015


Guy Katz and Clark Barrett and David Dill and Kyle Julian and Mykel Kochenderfer
Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks
arXiv 1702.01135, 2017


Julian, Kyle D and Lopez, Jessica and Brush, Jeffrey S and Owen, Michael P and Kochenderfer, Mykel J
Policy compression for aircraft collision avoidance systems
2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)

The End