A Robust Optimisation Perspective on Counterexample-Guided Repair of Neural Networks

David Boetius **Stefan Leue** **Tobias Sutter**

Universität Konstanz

Hi everyone, I'm David Boetius and I welcome you to my presentation on the paper "A Robust Optimisation Perspective on Counterexample-Guided Repair of Neural Networks" that I wrote together with Stefan Leue and Tobias Sutter.

Safety-Critical Neural Network Applications

Autonomous Driving

Figure 6: Screen shot of the simulator in interactive mode. See Section 7.1 for explanation of the performance metrics. The green area on the left is unknown because of the viewpoint transformation. The highlighted wide rectangle below the horizon is the area which is sent to the CNN.

(Bojarski et al. 2016)

Aircraft Control

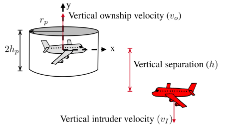


Figure 1: Three input variables of VerticalCAS neural network and ownship puck defined by r_p and h_p on ownship centered coordinate frame.

(Julian et al. 2019)

Medical Applications

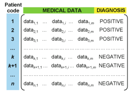


Fig. 4. **Example of training database structure.** Each row refers to a different patient labeled with a numerical code. The element $data_{k,i}$ refers to the i-th medical data (symptom, laboratory data, etc.) of the k-th patient.

(Amato et al. 2013)

Today, we see more and more approaches that suggest using neural networks in safety critical domains, such as autonomous driving or medical applications. But, for deploying neural networks in these applications, we require safety guarantees on the networks involved. One approach for obtaining strong safety guarantees is to use formal methods.

Formal Methods for Safe Deep Learning

Specifications

- formal description of safety constraints

Formal methods, typically build upon a specification which is a formal description of safe behaviour.
In our setting, a specification consists of a set of properties, which each consists of an input set X phi and an output set Y phi. Typically the input set is a hyperrectangle in the high-dimensional input space while the output set captures, for example, that some output output is the largest output.
If we have a neural network, we want that it satisfies each of the properties in a specification. That means that for all of the inputs in the property input set, the network produces an output in the property output set.

Formal Methods for Safe Deep Learning

Specifications

- formal description of safety constraints

$$\varphi = (\mathcal{X}_\varphi, \mathcal{Y}_\varphi)$$

Formal methods, typically build upon a specification which is a formal description of safe behaviour.

In our setting, a specification consists of a set of properties, which each consists of an input set X phi and an output set Y phi. Typically the input set is a hyperrectangle in the high-dimensional input space while the output set captures, for example, that some output output is the largest output.

If we have a neural network, we want that it satisfies each of the properties in a specification. That means that for all of the inputs in the property input set, the network produces an output in the property output set.

Formal Methods for Safe Deep Learning

Specifications

- formal description of safety constraints

$$\varphi = (\mathcal{X}_\varphi, \mathcal{Y}_\varphi)$$

$$\begin{aligned} &\mathrm{net}_{\boldsymbol{\theta}} \vDash \varphi \\ \Leftrightarrow\ &\forall \mathbf{x} \in \mathcal{X}_\varphi : \mathrm{net}_{\boldsymbol{\theta}}(\mathbf{x}) \in \mathcal{Y}_\varphi \end{aligned}$$

Formal methods, typically build upon a specification which is a formal description of safe behaviour.
In our setting, a specification consists of a set of properties, which each consists of an input set X phi and an output set Y phi. Typically the input set is a hyperrectangle in the high-dimensional input space while the output set captures, for example, that some output output is the largest output.
If we have a neural network, we want that it satisfies each of the properties in a specification. That means that for all of the inputs in the property input set, the network produces an output in the property output set.

Formal Methods for Safe Deep Learning

Specifications

- formal description of safety constraints

$$\varphi = (\mathcal{X}_\varphi, \mathcal{Y}_\varphi)$$

$$\mathrm{net}_{\boldsymbol{\theta}} \vDash \varphi$$
$$\Leftrightarrow \forall \mathbf{x} \in \mathcal{X}_\varphi : \mathrm{net}_{\boldsymbol{\theta}}(\mathbf{x}) \in \mathcal{Y}_\varphi$$

Verification

- given a neural network $\mathrm{net}_{\boldsymbol{\theta}}$ and a property φ, either ...
  1. Prove $\mathrm{net}_{\boldsymbol{\theta}} \vDash \varphi$, or
  2. Provide a counterexample

If we have such a formal specification, we can use a verifier to either prove that the network satisfies the specification or to derive a counterexample, which is an input that shows that the network does not satisfy the specification.
And if the network does not satisfy the specification, we can use repair to modify the network parameters to make the network satisfy the specification. An important secondary goal here is to maintain correct functionality which could, for example, mean maintaining accuracy on some data set.
In our paper, we look at a specific algorithm for neural network repair…

Formal Methods for Safe Deep Learning

Specifications

– formal description of safety constraints

$$\varphi = (\mathcal{X}_\varphi, \mathcal{Y}_\varphi)$$

$$\mathrm{net}_{\boldsymbol{\theta}} \vDash \varphi$$
$$\Leftrightarrow \forall \mathbf{x} \in \mathcal{X}_\varphi : \mathrm{net}_{\boldsymbol{\theta}}(\mathbf{x}) \in \mathcal{Y}_\varphi$$

Verification

– given a neural network $\mathrm{net}_{\boldsymbol{\theta}}$ and a property φ, either ...

1. Prove $\mathrm{net}_{\boldsymbol{\theta}} \vDash \varphi$, or
2. Provide a counterexample

Repair

– Make network satisfy specification

– Modify network parameters

– Maintain correct functionality

If we have such a formal specification, we can use a verifier to either prove that the network satisfies the specification or to derive a counterexample, which is an input that shows that the network does not satisfy the specification.

And if the network does not satisfy the specification, we can use repair to modify the network parameters to make the network satisfy the specification. An important secondary goal here is to maintain correct functionality which could, for example, mean maintaining accuracy on some data set.

In our paper, we look at a specific algorithm for neural network repair...

Counterexample-Guided Repair

```
while network is unsafe do
    find counterexamples
    remove counterexamples
```


…that's called counterexample-guided repair. The basic idea of counterexample-guided repair is to iterate finding counterexamples and removing counterexamples. Actually, this approach is very popular and also empirically successful, but on theoretical side of things, we know very little about this approach.
In particular, we don't know whether it is guaranteed to terminate which is what we are mainly studying in our paper.

Counterexample-Guided Repair

```
while network is unsafe do
    find counterexamples
    remove counterexamples
```

Practice: ✓ (Pulina and Tacchella 2010; Goodfellow et al. 2015; Guidotti et al. 2019; Dong et al. 2021; Goldberger et al. 2020; Sivaraman et al. 2020; Tan et al. 2021; Bauer-Marquart et al. 2022)

Theory: **???**

…that's called counterexample-guided repair. The basic idea of counterexample-guided repair is to iterate finding counterexamples and removing counterexamples. Actually, this approach is very popular and also empirically successful, but on theoretical side of things, we know very little about this approach.

In particular, we don't know whether it is guaranteed to terminate which is what we are mainly studying in our paper.

Counterexample-Guided Repair

while *network is unsafe* **do**
  find counterexamples
  remove counterexamples

Practice: ✓ (Pulina and Tacchella 2010; Goodfellow et al. 2015; Guidotti et al. 2019; Dong et al. 2021; Goldberger et al. 2020; Sivaraman et al. 2020; Tan et al. 2021; Bauer-Marquart et al. 2022)

Theory: **Termination?**

…that's called counterexample-guided repair. The basic idea of counterexample-guided repair is to iterate finding counterexamples and removing counterexamples. Actually, this approach is very popular and also empirically successful, but on theoretical side of things, we know very little about this approach.

In particular, we don't know whether it is guaranteed to terminate which is what we are mainly studying in our paper.

while *network is unsafe* **do**
    find counterexamples
    remove counterexamples

So, to gain a theoretical understanding of counterexample-guided repair,
one helpful insight is that you can describe both finding counterexamples and removing counterexamples as solving optimisation problems.
However, when looking closer, we can realise that the whole algorithm is trying to solve an optimisation problem, but one with infinitely many constraints — a robust optimisation problem.
And actually removing counterexamples corresponds to what is called a scenario problem of this robust optimisation problem and counterexample-guided repair itself corresponds to solving a sequence of scenario problems, which is also a popular approach for solving robust optimisation problems.
Using this link between robust optimisation and counterexample-guided repair…

$$V : \begin{cases} \underset{\mathbf{x}}{\text{minimise}} \;\; f_{\text{Sat}}(\text{net}_{\boldsymbol{\theta}}(\mathbf{x})) \\ \text{subject to} \;\; \mathbf{x} \in \mathcal{X}_\varphi \end{cases}$$

while *network is unsafe* **do**
    find counterexamples
    remove counterexamples

$$CR : \begin{cases} \underset{\boldsymbol{\theta} \in \mathbb{R}^p}{\text{minimise}} \;\; J(\boldsymbol{\theta}) \\ \text{subject to} \;\; f_{\text{Sat}}\big(\text{net}_{\boldsymbol{\theta}}\big(\mathbf{x}^{(i)}\big)\big) \geq 0 \\ \qquad\qquad\qquad\qquad \forall i \in \{1, \ldots, N\} \end{cases}$$

So, to gain a theoretical understanding of counterexample-guided repair, one helpful insight is that you can describe both finding counterexamples and removing counterexamples as solving optimisation problems.
However, when looking closer, we can realise that the whole algorithm is trying to solve an optimisation problem, but one with infinitely many constraints — a robust optimisation problem.
And actually removing counterexamples corresponds to what is called a scenario problem of this robust optimisation problem and counterexample-guided repair itself corresponds to solving a sequence of scenario problems, which is also a popular approach for solving robust optimisation problems.
Using this link between robust optimisation and counterexample-guided repair…

$$V : \begin{cases} \underset{\mathbf{x}}{\text{minimise}} \;\; f_{\text{Sat}}(\text{net}_{\boldsymbol{\theta}}(\mathbf{x})) \\ \text{subject to } \; \mathbf{x} \in \mathcal{X}_{\varphi} \end{cases}$$

$$R : \begin{cases} \underset{\boldsymbol{\theta} \in \mathbb{R}^p}{\text{minimise}} \;\; J(\boldsymbol{\theta}) \\ \text{subject to } \; f_{\text{Sat}}(\text{net}_{\boldsymbol{\theta}}(\mathbf{x})) \geq 0 \quad \forall \mathbf{x} \in \mathcal{X}_{\varphi}. \end{cases}$$

while *network is unsafe* **do**
  find counterexamples
  remove counterexamples

$$CR : \begin{cases} \underset{\boldsymbol{\theta} \in \mathbb{R}^p}{\text{minimise}} \;\; J(\boldsymbol{\theta}) \\ \text{subject to } \; f_{\text{Sat}}\big(\text{net}_{\boldsymbol{\theta}}\big(\mathbf{x}^{(i)}\big)\big) \geq 0 \\ \qquad\qquad\qquad\qquad \forall i \in \{1, \ldots, N\} \end{cases}$$

So, to gain a theoretical understanding of counterexample-guided repair,
one helpful insight is that you can describe both finding counterexamples and removing counterexamples as solving optimisation problems.
However, when looking closer, we can realise that the whole algorithm is trying to solve an optimisation problem, but one with infinitely many constraints — a robust optimisation problem.
And actually removing counterexamples corresponds to what is called a scenario problem of this robust optimisation problem and counterexample-guided repair itself corresponds to solving a sequence of scenario problems, which is also a popular approach for solving robust optimisation problems.
Using this link between robust optimisation and counterexample-guided repair…

$$V : \begin{cases} \underset{\mathbf{x}}{\text{minimise}} & f_{\text{Sat}}(\text{net}_{\boldsymbol{\theta}}(\mathbf{x})) \\ \text{subject to} & \mathbf{x} \in \mathcal{X}_{\varphi} \end{cases}$$

$$R : \begin{cases} \underset{\boldsymbol{\theta} \in \mathbb{R}^p}{\text{minimise}} & J(\boldsymbol{\theta}) \\ \text{subject to} & f_{\text{Sat}}(\text{net}_{\boldsymbol{\theta}}(\mathbf{x})) \geq 0 \quad \forall \mathbf{x} \in \mathcal{X}_{\varphi}. \end{cases}$$

while *network is unsafe* **do**

find counterexamples

remove counterexamples

$$CR : \begin{cases} \underset{\boldsymbol{\theta} \in \mathbb{R}^p}{\text{minimise}} & J(\boldsymbol{\theta}) \\ \text{subject to} & f_{\text{Sat}}\left(\text{net}_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right)\right) \geq 0 \\ & \forall i \in \{1, \ldots, N\} \end{cases}$$

So, to gain a theoretical understanding of counterexample-guided repair, one helpful insight is that you can describe both finding counterexamples and removing counterexamples as solving optimisation problems.
However, when looking closer, we can realise that the whole algorithm is trying to solve an optimisation problem, but one with infinitely many constraints — a robust optimisation problem.
And actually removing counterexamples corresponds to what is called a scenario problem of this robust optimisation problem and counterexample-guided repair itself corresponds to solving a sequence of scenario problems, which is also a popular approach for solving robust optimisation problems.
Using this link between robust optimisation and counterexample-guided repair…

Termination Results

Model	Specification	
Linear Regression Model	Linear	✓
Linear Classifier, ReLU Neuron	Linear	✓
Neural Network	Bounded Input Set	?
Neural Network	Unbounded Input Set	✗

. . . we were able to derive several termination results for counterexample-guided repair. First of all, we were able to show that counterexample-guided repair is actually guaranteed to terminate when repairing linear regression models, linear classifiers and also single ReLU neurons under mild assumptions on the specification. On the other hand, we were also able to show that counterexample-guided repair is not guaranteed to terminate when repairing neural networks to conform to properties with unbounded input sets. Now, in practice, specifications actually have bounded input sets, so our theoretical results do not yet address the main application, but still they provide insights into the termination of counterexample-guided repair for the first time.

These results assume a verifier that always computes most-violating counterexamples, but most available verifiers instead compute arbitrary counterexamples. We also had a look at using these, how we call them, "early-exit" verifiers and found that actually counterexample-guided repair is not guaranteed to terminate when using such verifiers regardless of the model class.

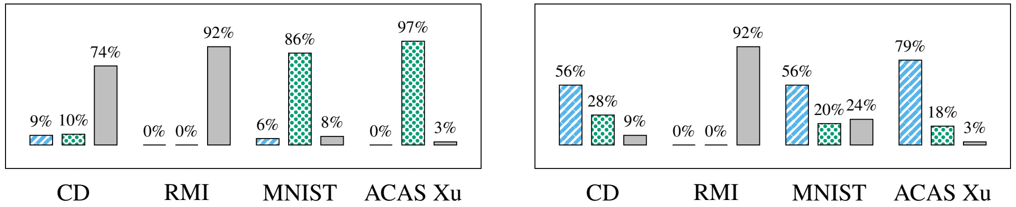
Termination Results

Model	Specification	
Linear Regression Model	Linear	✓
Linear Classifier, ReLU Neuron	Linear	✓
Neural Network	Bounded Input Set	?
Neural Network	Unbounded Input Set	✗
When using an Early-Exit Verifier		✗

…we were able to derive several termination results for counterexample-guided repair. First of all, we were able to show that counterexample-guided repair is actually guaranteed to terminate when repairing linear regression models, linear classifiers and also single ReLU neurons under mild assumptions on the specification. On the other hand, we were also able to show that counterexample-guided repair is not guaranteed to terminate when repairing neural networks to conform to properties with unbounded input sets. Now, in practice, specifications actually have bounded input sets, so our theoretical results do not yet address the main application, but still they provide insights into the termination of counterexample-guided repair for the first time.

These results assume a verifier that always computes most-violating counterexamples, but most available verifiers instead compute arbitrary counterexamples. We also had a look at using these, how we call them, "early-exit" verifiers and found that actually counterexample-guided repair is not guaranteed to terminate when using such verifiers regardless of the model class.

Optimal vs. Early-Exit Verifier

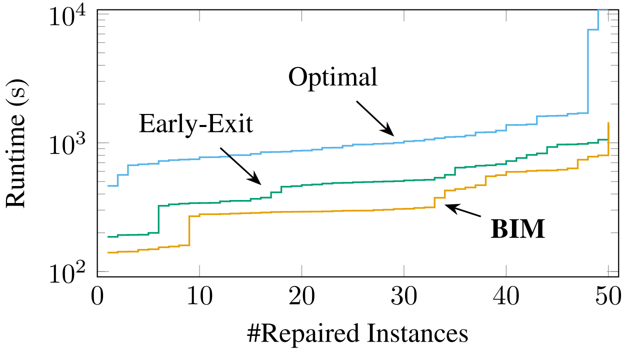


(a) Which is faster in terms of runtime? **(b)** Which is faster in terms of repair steps?

optimal verifier, early-exit verifier, same speed

This motivated us to evaluate the use of early-exit verifiers empirically and we found that using an early-exit verifier in practice provides speed advantages compared to computing most-violating counterexamples while we did not observe non-termination in our experiments.

Using Falsifiers for Repair



We also studied another approach for speeding up repair that uses falsifiers which are techniques that try to generate counterexamples faster while not being able to prove property satisfaction. We found that certain falsifiers can also significantly accelerate repair.

Repairing Linear Regression Models

Algorithm	Success Rate $\varepsilon = 100$	$\varepsilon = 150$
Ouroboros	$30\,\%$	$77\,\%$
SpecRepair	$58\,\%$	$94\,\%$
Quadratic Programming	$\mathbf{72}\,\%$	$\mathbf{97}\,\%$

Lastly, our insights into repairing linear regression models allow us to derive a new repair algorithm that is based on quadratic programming and this new algorithm surpasses existing algorithms for repairing linear regression models.

Conclusion

1 **while** *network is unsafe* **do**
2 find counterexamples
3 remove counterexamples

To summarise, we study counterexample-guided repair and established a link between counterexample-guided repair and robust optimisation. This allows us to gain insights into the termination of counterexample guided repair for repairing neural networks for the first time.
We complement our theoretical results with experiments on accelerating repair and on a new algorithm for repairing linear regression models that is enabled by our theoretical results that surpasses existing repair algorithms for these models.
This concludes my presentation, thank you for watching.

Conclusion

$$V : \begin{cases} \underset{\mathbf{x}}{\text{minimise}} \; f_{\text{Sat}}(\text{net}_{\boldsymbol{\theta}}(\mathbf{x})) \\ \text{subject to } \; \mathbf{x} \in \mathcal{X}_{\varphi} \end{cases}$$

$$R : \begin{cases} \underset{\boldsymbol{\theta} \in \mathbb{R}^p}{\text{minimise}} \; J(\boldsymbol{\theta}) \\ \text{subject to } \; f_{\text{Sat}}(\text{net}_{\boldsymbol{\theta}}(\mathbf{x})) \geq 0 \quad \forall \mathbf{x} \in \mathcal{X}_{\varphi}. \end{cases}$$

while *network is unsafe* **do**
- find counterexamples
- remove counterexamples

$$CR : \begin{cases} \underset{\boldsymbol{\theta} \in \mathbb{R}^p}{\text{minimise}} \; J(\boldsymbol{\theta}) \\ \text{subject to } \; f_{\text{Sat}}\left(\text{net}_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right)\right) \geq 0 \\ \qquad \forall i \in \{1, \ldots, N\} \end{cases}$$

Model	Specification	
Linear Regression Model	Linear	✓
Linear Classifier, ReLU Neuron	Linear	✓
Neural Network	Bounded Input Set	?
Neural Network	Unbounded Input Set	✗
When using an Early-Exit Verifier		✗

To summarise, we study counterexample-guided repair and established a link between counterexample-guided repair and robust optimisation. This allows us to gain insights into the termination of counterexample guided repair for repairing neural networks for the first time.
We complement our theoretical results with experiments on accelerating repair and on a new algorithm for repairing linear regression models that is enabled by our theoretical results that surpasses existing repair algorithms for these models.
This concludes my presentation, thank you for watching.

Conclusion

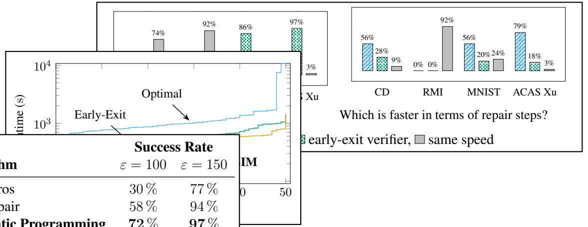
$$V : \begin{cases} \text{minimise}_{\mathbf{x}} & f_{\text{Sat}}(\text{net}_{\boldsymbol{\theta}}(\mathbf{x})) \\ \text{subject to} & \mathbf{x} \in \mathcal{X}_\varphi \end{cases}$$

$$R : \begin{cases} \text{minimise}_{\boldsymbol{\theta} \in \mathbb{R}^p} & J(\boldsymbol{\theta}) \\ \text{subject to} & f_{\text{Sat}}(\text{net}_{\boldsymbol{\theta}}(\mathbf{x})) \geq 0 \quad \forall \mathbf{x} \in \mathcal{X}_\varphi. \end{cases}$$

while *network is unsafe* **do**
- find counterexamples
- remove counterexamples

$$CR : \begin{cases} \text{minimise}_{\boldsymbol{\theta} \in \mathbb{R}^p} & J(\boldsymbol{\theta}) \\ \text{subject to} & f_{\text{Sat}}(\text{net}_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})) \geq 0 \\ & \forall i \in \{1, \ldots, N\} \end{cases}$$

Model	Specification	
Linear Regression Model	Linear	✓
Linear Classifier, ReLU Neuron	Linear	✓
Neural Network	Bounded Input Set	?
Neural Network	Unbounded Input Set	✗
When using an Early-Exit Verifier		✗



Algorithm	Success Rate $\varepsilon = 100$	$\varepsilon = 150$
Ouroboros	30 %	77 %
SpecRepair	58 %	94 %
Quadratic Programming	**72 %**	**97 %**

To summarise, we study counterexample-guided repair and established a link between counterexample-guided repair and robust optimisation. This allows us to gain insights into the termination of counterexample guided repair for repairing neural networks for the first time.
We complement our theoretical results with experiments on accelerating repair and on a new algorithm for repairing linear regression models that is enabled by our theoretical results that surpasses existing repair algorithms for these models.
This concludes my presentation, thank you for watching.

References

Amato, Filippo, Alberto López, Eladia M. Peña-Méndez, Petr Vaňhara, Aleš Hampl, and Josef Havel (2013). "Artificial neural networks in medical diagnosis". In: *J. Appl. Biomed.* 11.2, pp. 47–58. URL: https://doi.org/10.2478/v10136-012-0031-x.

Bauer-Marquart, Fabian, David Boetius, Stefan Leue, and Christian Schilling (2022). "SpecRepair: Counter-Example Guided Safety Repair of Deep Neural Networks". In: *SPIN*. Vol. 13255. Lecture Notes in Computer Science, pp. 79–96. URL: https://doi.org/10.1007/978-3-031-15077-7_5.

References (cont.)

Bojarski, Mariusz, 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 (2016). "End to End Learning for Self-Driving Cars". In: *CoRR* abs/1604.07316. URL: `http://arxiv.org/abs/1604.07316`.

Dong, Guoliang, Jun Sun, Jingyi Wang, Xinyu Wang, and Ting Dai (2021). "Towards Repairing Neural Networks Correctly". In: *QRS*, pp. 714–725. URL: `https://doi.org/10.1109/QRS54544.2021.00081`.

Goldberger, Ben, Guy Katz, Yossi Adi, and Joseph Keshet (2020). "Minimal Modifications of Deep Neural Networks using Verification". In: *LPAR*. Vol. 73. EPiC Series in Computing, pp. 260–278. URL: `https://easychair.org/publications/paper/CWhF`.

References (cont.)

- Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy (2015). "Explaining and Harnessing Adversarial Examples". In: *ICLR (Poster)*. URL: http://arxiv.org/abs/1412.6572.
- Guidotti, Dario, Francesco Leofante, Armando Tacchella, and Claudio Castellini (2019). "Improving reliability of myocontrol using formal verification". In: *IEEE Trans. Neural Syst. Rehabilitation Eng.* 27.4, pp. 564–571. URL: https://doi.org/10.1109/TNSRE.2019.2893152.
- Julian, Kyle D., Shivam Sharma, Jean-Baptiste Jeannin, and Mykel J. Kochenderfer (2019). "Verifying Aircraft Collision Avoidance Neural Networks Through Linear Approximations of Safe Regions". In: *CoRR* abs/1903.00762. URL: http://arxiv.org/abs/1903.00762.

References (cont.)

- Pulina, Luca and Armando Tacchella (2010). "An Abstraction-Refinement Approach to Verification of Artificial Neural Networks". In: *CAV*. Vol. 6174. Lecture Notes in Computer Science, pp. 243–257. URL: https://doi.org/10.1007/978-3-642-14295-6_24.
- Sivaraman, Aishwarya, Golnoosh Farnadi, Todd D. Millstein, and Guy Van den Broeck (2020). "Counterexample-Guided Learning of Monotonic Neural Networks". In: *NeurIPS*. URL: https://proceedings.neurips.cc/paper/2020/hash/8ab70731b1553f17c11a3bbc87e0b605-Abstract.html.
- Tan, Cheng, Yibo Zhu, and Chuanxiong Guo (2021). "Building verified neural networks with specifications for systems". In: *APSys*, pp. 42–47. URL: https://doi.org/10.1145/3476886.3477508.