Learning Verifiable Representations

by

Mathias Lechner

May, 2022

A thesis submitted to the
Graduate School
of the
Institute of Science and Technology Austria
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Committee in charge:

Krzysztof Pietrzak, Chair
Thomas A. Henzinger
Christoph Lampert
Radu Grosu
Sepp Hochreiter

Institute of Science and Technology Austria

The thesis of Mathias Lechner, titled *Learning Verifiable Representations*, is approved by:

Supervisor: Thomas A. Henzinger, IST Austria, Klosterneuburg, Austria

Signature: ______________________________

Committee Member: Christoph Lampert, IST Austria, Klosterneuburg, Austria

Signature: ______________________________

Committee Member: Radu Grosu, Vienna University of Technology (TU Wien), Vienna, Austria

Signature: ______________________________

Committee Member: Sepp Hochreiter, Johannes Kepler University (JKU), Linz, Austria

Signature: ______________________________

Defense Chair: Krzysztof Pietrzak, IST Austria, Klosterneuburg, Austria

Signature: ______________________________

Signed page is on file

IST Austria Thesis, ISSN: 2663-337X

ISBN: 978-3-99078-017-6

I hereby declare that this thesis is my own work and that it does not contain other people's work without this being so stated; this thesis does not contain my previous work without this being stated, and the bibliography contains all the literature that I used in writing the dissertation.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee, and that this thesis has not been submitted for a higher degree to any other university or institution.

I certify that any republication of materials presented in this thesis has been approved by the relevant publishers and co-authors.

Signature: ______________________________

Mathias Lechner
May, 2022

Signed page is on file

Abstract

Deep learning has enabled breakthroughs in challenging computing problems and has emerged as the standard problem-solving tool for computer vision and natural language processing tasks. One exception to this trend is safety-critical tasks where robustness and resilience requirements contradict the black-box nature of neural networks. To deploy deep learning methods for these tasks, it is vital to provide guarantees on neural network agents' safety and robustness criteria. This can be achieved by developing formal verification methods to verify the safety and robustness properties of neural networks.

Our goal is to design, develop and assess safety verification methods for neural networks to improve their reliability and trustworthiness in real-world applications. This thesis establishes techniques for the verification of compressed and adversarially trained models as well as the design of novel neural networks for verifiably safe decision-making.

First, we establish the problem of verifying quantized neural networks. Quantization is a technique that trades numerical precision for the computational efficiency of running a neural network and is widely adopted in industry. We show that neglecting the reduced precision when verifying a neural network can lead to wrong conclusions about the robustness and safety of the network, highlighting that novel techniques for quantized network verification are necessary. We introduce several bit-exact verification methods explicitly designed for quantized neural networks and experimentally confirm on realistic networks that the network's robustness and other formal properties are affected by the quantization.

Furthermore, we perform a case study providing evidence that adversarial training, a standard technique for making neural networks more robust, has detrimental effects on the network's performance. This robustness-accuracy tradeoff has been studied before regarding the accuracy obtained on classification datasets where each data point is independent of all other data points. On the other hand, we investigate the tradeoff empirically in robot learning settings where a both, a high accuracy and a high robustness, are desirable. Our results suggest that the negative side-effects of adversarial training outweigh its robustness benefits in practice.

Finally, we consider the problem of verifying safety when running a Bayesian neural network policy in a feedback loop with systems over the infinite time horizon. Bayesian neural networks are probabilistic models for learning uncertainties in the data and are therefore often used on robotic and healthcare applications where data is inherently stochastic. We introduce a method for recalibrating Bayesian neural networks so that they yield probability distributions over safe decisions only. Our method learns a safety certificate that guarantees safety over the infinite time horizon to determine which decisions are safe in every possible state of the system. We demonstrate the effectiveness of our approach on a series of reinforcement learning benchmarks.

Acknowledgements

First of all, I would like to express my complete thanks to my Ph.D. advisor Tom Henzinger. The environment Tom provided enabled me to perform *interest driven* research and allowed me to grow as a computer scientist. Tom's focus on precision showed me how to phrase research questions properly and clearly. He taught me to apply the highest standards to my research and set my goals ambitiously.

I am sincerely thankful to Prof. Radu Grosu for convincing me to pursue a Ph.D. research career, his generous support throughout my studies, and for introducing me to Tom, Daniela, and many of my collaborators. I thank Prof. Daniela Rus, who believed in my abilities early on and taught me how to conduct and manage long-lasting research projects efficiently and successfully. Furthermore, I want to thank Prof. Christoph Lampert and Prof. Krishnendu Chatterjee. Christoph and Krish are immense sources of knowledge in machine learning and formal verification and who I am glad to be at ISTA. I also want to thank Sepp Hochreiter for giving me valuable feedback through all stages of my study. I express my deepest gratitude to Sepp and Christoph for assuring that top-tier machine learning researchers can be found in Austria.

I genuinely thank my friend and long-term collaborator, Ramin Hasani. Ramin shares my research vision, and our discussions have fueled large parts of my research and helped me challenge adversities. I sincerely appreciate my friend and colleague Đorđe Žikelić, without whom this thesis in its presented form would not be possible. My interactions with Đorđe allowed me to understand complex topics thoroughly, leading to many innovative publications.

I thank my fantastic collaborator Alexander Amini. Alexander has been remarkably open for collaborations early on and our discussions while driving to MIT's autonomous driving test track opened new perspectives on research for me. Additionally, I want to thank all my collaborators, especially Mirco Giacobbe, Sophie Grünbacher, Prof. Manuel Zimmer, Prof. Zvonimir Rakamaric, Zahra Babaiee, Felix Naser, Lucas Liebenwein, and Tsun-Hsuan (Johnson) Wang.

Last but not least, I would like to thank my friends, family, and Valentine for their support, not limited to my Ph.D., but throughout life, and made it possible to endure the troubles, challenges, and adversities I encountered during my studies.

Funding sources This research was supported in part by the Austrian Science Fund (FWF) under grant Z211-N23 (Wittgenstein Award) and the ERC-2020-AdG 101020093.

About the Author

Mathias Lechner completed a BSc and MSc in Computer Engineering at the Vienna University of Technology (TU Wien) before joining ISTA in September 2018. His primary research interests are in the intersection of machine learning, robotics, and verification, which he could pursue under the supervision of Tom Henzinger. During his PhD studies, he published papers in the journal Nature Machine Intelligence and high-impact conferences, such as NeurIPS, ICML, and AAAI, drawing from productive collaborations with researchers from MIT, TU Wien, and within ISTA.

List of Collaborators and Publications

1. Chapter 2 is based on the work "Mirco Giacobbe, Thomas A. Henzinger, and **Mathias Lechner** (alphabetically ordered). How Many Bits Does it Take to Quantize Your Neural Network? In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. 2020".
2. Chapter 3 is based on the work "Thomas A. Henzinger, **Mathias Lechner**, and Đorđe Žikelić (alphabetically ordered). Scalable Verification of Quantized Neural Networks. In *AAAI Conference on Artificial Intelligence (AAAI)*. 2021".
3. Chapter 4 contains parts of the work "**Mathias Lechner**, Ramin Hasani, Radu Grosu, Daniela Rus, Thomas A. Henzinger. Adversarial Training is Not Ready for Robot Learning. In *International Conference on Robotics and Automation (ICRA)*. 2021".
4. Chapter 4 contains parts of the work "**Mathias Lechner**, Alexander Amini, Daniela Rus, Thomas A. Henzinger. Revisiting the Adversarial Robustness-Accuracy Tradeoff in Robot Learning. *arXiv preprint arXiv:2204.07373*. 2022".
5. Chapter 5 is based on the work "**Mathias Lechner***, Đorđe Žikelić*, Krishnendu Chatterjee, Thomas A Henzinger. Infinite Time Horizon Safety of Bayesian Neural Networks. In *Conference on Neural Information Processing Systems (NeurIPS)*. 2021".

The following list contains loosely connected works that were published concurrently to the works listed above but are not parts of this thesis:

- **Mathias Lechner***, Đorđe Žikelić*, Krishnendu Chatterjee, Thomas A. Henzinger. Stability Verification in Stochastic Control Systems via Neural Network Supermartingales. In *AAAI Conference on Artificial Intelligence (AAAI)*. 2022.
- Sophie Gruenbacher, **Mathias Lechner**, Ramin Hasani, Daniela Rus, Thomas A. Henzinger, Scott A. Smolka, Radu Grosu. GoTube: Scalable Stochastic Verification of Continuous-Depth Models. In *AAAI Conference on Artificial Intelligence (AAAI)*. 2022.
- Charles J Vorbach*, Ramin Hasani*, Alexander Amini, **Mathias Lechner**, Daniela Rus. Causal Navigation by Continuous-time Neural Networks. In *Conference on Neural Information Processing Systems (NeurIPS)*. 2021.

* denotes equal contributions.

- Zahra Babaiee, Ramin Hasani, **Mathias Lechner**, Daniela Rus, Radu Grosu. On-Off Center-Surround Receptive Fields for Accurate and Robust Image Classification. In *International Conference on Machine Learning (ICML)*. 2021.

- Ramin Hasani*, **Mathias Lechner***, Alexander Amini, Daniela Rus, Radu Grosu. Liquid Time-constant Networks. In *AAAI Conference on Artificial Intelligence (AAAI)*. 2021.

- Sophie Grünbacher, Ramin Hasani, **Mathias Lechner**, Jacek Cyranka, Scott A. Smolka, Radu Grosu. On the Verification of Neural ODEs with Stochastic Guarantees. In *AAAI Conference on Artificial Intelligence (AAAI)*. 2021.

- **Mathias Lechner***, Ramin Hasani*, Alexander Amini, Thomas Henzinger, Daniela Rus, Radu Grosu. Neural Circuit Policies Enabling Auditable Autonomy. In *Nature Machine Intelligence*. 2020.

- **Mathias Lechner**. Learning Representations For Binary-classification without Backpropagation. In *International Conference on Learning Representations (ICLR)*. 2020.

- **Mathias Lechner***, Ramin Hasani*, Daniela Rus, Radu Grosu. Gershgorin Loss Stabilizes the Recurrent Neural Network Compartment of an End-to-end Robot Learning Scheme. In *International Conference on Robotics and Automation (ICRA)*. 2020.

- Ramin Hasani*, **Mathias Lechner***, Alexander Amini, Daniela Rus, Radu Grosu. The Natural Lottery Ticket Winner: Reinforcement Learning by Ordinary Neural Circuits. In *International Conference on Machine Learning (ICML)*. 2020.

- Marek Baranowski, Shaobo He, **Mathias Lechner**, Thanh Son Nguyen, and Zvonimir Rakamaric. An SMT Theory of Fixed-Point Arithmetice. In *International Joint Conference on Automated Reasoning (IJCAR)*. 2020.

- Sophie Gruenbacher, Jacek Cyranka, **Mathias Lechner**, Md. Ariful Islam, Scott Smolka, and Radu Grosu. Lagrangian Reachtubes: The Next Generation. In *IEEE Conference on Decision and Control (CDC)*. 2020.

- **Mathias Lechner***, Ramin Hasani*, Manuel Zimmer, Thomas Henzinger, Radu Grosu. Designing Worm-inspired Neural Networks for Interpretable Robotics Control. In *International Conference on Robotics and Automation (ICRA)*. 2019.

- Ramin Hasani*, Alexander Amini*, **Mathias Lechner**, Felix Naser, Radu Grosu, Daniela Rus. Response Characterization for Auditing Cell Dynamics in Long Short-term Memory Networks. In *International Joint Conference on Neural Networks (IJCNN)*. 2019.

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

BNNs Bayesian neural networks. 7, 69, 72

CMDPs Constraint Markov decision processes. 71

ERM Empirical risk minimization. 45

GAN Generative adversarial networks. 49

IA Interval arithmetic. 39

IFGSM Iterative fast gradient sign method. 23

MDPs Markov decision processes. 71

MILP Mixed-integer linear programming. 3, 9, 13, 77

ML Machine learning. 1

MLP Multi layer perceptron. 1

NN Neural network. 72

NRA-SMT Non-linear real arithmetic satisfiability modulo theory. 9, 77

QF_BV Quantifier-free bit-vector SMT. 9, 14, 31

QNNs Quantized neural networks. 4

RL Reinforcement learning. 50, 71

SAT Boolean satisfiability. 31

SMT Satisfiability modulo theories. 3, 9, 13, 27

CHAPTER 1

Introduction

Machine learning is a set of problem-solving methods that increasingly become better at solving their task with more available data. Instead of being explicitly programmed by a human developer, these algorithms learn their function from the provided data. Deep learning, a subclass of ML, has emerged as being particularly successful at solving difficult problems. Most famously, deep learning has enabled breakthroughs in the long-standing open problems of autonomous driving [Bojarski et al., 2016], protein folding [Jumper et al., 2021], and the board game Go [Silver et al., 2016]. The key components of deep learning are neural networks, which are parametrized functions f_θ that are structurally loosely inspired by the information processing of biological nervous systems, e.g., see Figure 1.1. The parameters θ of these networks are learned by gradient descent-based training algorithms such that the resulting function fits the provided training data with respect to a loss criterion.

Despite their remarkable problem-solving capabilities, deploying neural networks on safety-critical tasks where performance guarantees are necessary is problematic due to three reasons: First, as the function implemented by a network is determined by millions of parameters, interpreting and explaining on what basis the network forms its decision is non-trivial [Olah et al., 2018]. Consequently, established techniques for validating software correctness, such as code audit, are inapplicable to deep learning. Second, neural networks are susceptible to adversarially crafted input perturbations. For instance, changing the pixels values of an image by a few percent can lead to neural networks making completely different decisions, despite the original and the perturbed images being visually indistinguishable for humans [Goodfellow et al., 2014b]. In Figure 1.2 we visualize such an adversarial perturbation on the popular ResNet50 image classifier. Third, neural networks inherit the biases and flaws of the training data. For example, Arjovsky et al. [2019] considers a hypothetical network that needs to classify images of camels and cows. Naturally, most training images of cows have a grassland background, while the images of camels have a desert background. Consequently,

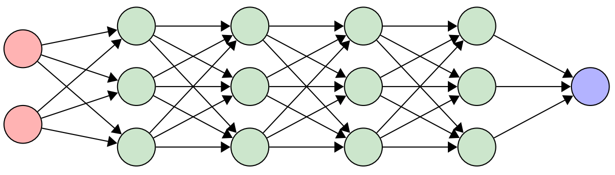


Figure 1.1: Illustration of a Multi-Layer Perceptron (MLP) neural network [Rumelhart et al., 1986]. Red nodes represent inputs; green nodes represent hidden units and blue nodes represent outputs of the network. Typical deep neural networks consist of thousands of inputs and outputs, and millions of hidden units.

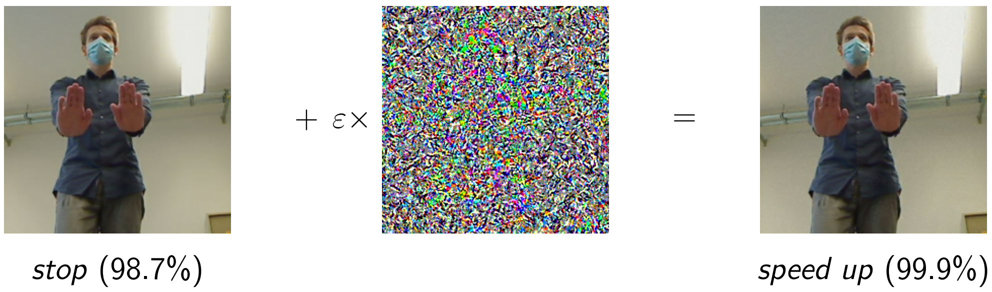


Figure 1.2: Example of an adversarial attack on a ResNet50 [He et al., 2016] running on a mobile robot to classify gesture commands. The image on the left is classified by the network as *"stop"* command (confidence 98.7%), while the image on the right is classified as *"speed up"* command (confidence 99.9%). The mask in the center shows the difference between the two images, amplified for visualization purposes.

the neural network learns to rely on the very predictive background for making a decision and misclassifies images at prediction time of cows with a sandy beach background.

These three challenges highlight the importance of verifying learned networks. An ultimate goal is to obtain formal guarantees about the behavior and correctness of a neural network beyond simply evaluating them on a test dataset. Researchers from machine learning, artificial intelligence, formal methods, and the robotics communities have all tried to tackle this problem from different perspectives. In this thesis, we look at the problem of learning a verifiable network from the various perspectives of these different research communities.

Vision Our goal is to design, develop and assess methods for verifying neural networks while capturing the nuances of the particular types of applications and networks deployed in practice. We uncover several limitations, flaws, and missing aspects regarding realistic use cases of existing literature on the topic. Finally, we establish methods addressing these limiting factors to serve as a foundation for future work on advancing the scale of neural network verification.

In the rest of this chapter, we summarize the main topics of this thesis. We first give some background on the learning and verification of neural networks. We then look at the verification questions and methods covered in this thesis. Finally, we summarize our contributions.

1.1 Learning Representations

A neural network is a function $f_\theta : \mathcal{X} \to \mathcal{Y}$ from the input space $\mathcal{X}$ to the output space $\mathcal{Y}$. Structurally, f_θ comprises of a sequence of layers, e.g., as illustrated in Figure 1.1, which iteratively change the representation of the input data until a final decision is made at the output layer. Therefore, learning f_θ is often called *representation learning*. The most common learning setting is *supervised learning*, in which the training data takes the form of a set $\{(x_1, y_1), \ldots (x_n, y_n)\}$ independently sampled from a probability distribution over the domain $\mathcal{X} \times \mathcal{Y}$. The parameters are learned by minimizing the empirical risk term

$$\frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(f_\theta(x_i), y_i)$$

via stochastic gradient descent, where $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is a given loss function that characterizes how well the network's prediction $f_\theta(x_i)$ matches the ground truth y_i.

Simply minimizing the empirical risk of the training data tells us little about how the network performs, as the network could have just learned to recall the training data. To genuinely measure how well the network generalizes to arbitrary data, some fraction of the training samples are typically held out from the learning process and reserved for testing how well the network performs on non-training data. This procedure allows us to estimate the expected loss on data sampled from the same probability distribution as the training data. However, testing does not tell us how the network performs if the prediction time data distribution differs from the training distribution due to biases in the data collection step. Moreover, the test loss does not capture other potential properties such as fairness, robustness, and safety.

1.2 Verifying Learned Representations

Formal methods are algorithms and proof techniques that, given a specification and an implementation of a system, check whether the implementation fulfills the specification [Clarke et al., 2018]. Researchers have adopted these methods to neural network settings. Most notably, mixed-integer linear programming (MILP) and satisfiability modulo theories (SMT) have been employed to check whether trained neural networks satisfy specifications on the input and output variables of the network [Ehlers, 2017, Katz et al., 2017, Tjeng et al., 2019]. MILP and SMT are methods for determining if a feasible solution to a set of constraints exists. MILP and the various theories within SMT differ in what data types and variable operations are allowed to specify the constraints. For instance, while MILP allows only conjunctions of linear inequalities over integer and real variables, SMT is derived from decidable subsets of first-order logic.

MILP or SMT-based approaches verify a neural network by representing the semantics of a neural network together with the specification as a set of constraints. A MILP or SMT solver then checks the constraints for satisfiability.

MILP and SMT approaches have been particularly successful for networks with ReLU activations, i.e., $x \mapsto \max(x, 0)$, by exploiting their piecewise linear structure and employing efficient linear programming techniques for reasoning over the linear components of the network [Ehlers, 2017, Katz et al., 2017, Tjeng et al., 2019]. [Katz et al., 2017] have shown that verifying a ReLU network with linear inequality specification is NP-complete in the number of ReLU nodes in the network, making it an inherently complex problem.

More recently, abstract interpretation [Cousot and Cousot, 1977] has become a common approach to neural network verification. These approaches avoid the NP-hardness of the verification problem at the cost of algorithmic completeness [Singh et al., 2019, Gowal et al., 2019, Wang et al., 2021]. In particular, incomplete methods can prove some, but not all, positive instances, i.e., that the network fulfills the specification. The key idea of abstract interpretation is to represent the network by another mathematical object that can be checked more efficiently than the original network. An essential condition of the abstraction is that it must over-approximate the input-output behavior of the network, i.e., if the abstraction fulfills the specification, the original network must also meet it. The drawback of such approaches is that the abstraction might be too loose and violate the specification, even though the network satisfies the specification.

In the realm of neural network verification, interval arithmetic [Gowal et al., 2019], hybrid automata [Xiang et al., 2018], zonotopes [Singh et al., 2018], polyhedra [Gehr et al., 2018, Singh et al., 2019], convex relaxations [Dvijotham et al., 2018], and polynomial [Zhang et al.,

Operation	float (32 bit)	float (16 bit)	integer (32 bit)	integer (8 bit)
Add	0.9pJ	0.4 pJ	**0.1 pJ**	0.03pJ
Multiply	4 pJ	1 pJ	3 pJ	**0.2 pJ**

Table 1.1: Rough hardware energy consumption for various arithmetic operations on a 45nm CMOS node in picojoules (pJ) [Horowitz, 2014]. Quantized neural networks primarily use 8-bit integer multiply and 32-bit integer add operations (highlighted in bold) [Jacob et al., 2018].

2018] abstract domains have been introduced. Abstraction-refinement procedures [Clarke et al., 2000] tackle the incompleteness issue by tightening the abstraction whenever a spurious counterexample violates the specification. Naturally, these techniques have been adopted to neural network verification settings [Bunel et al., 2018, Wang et al., 2018a, 2021].

In this thesis, we are the first who investigate the problem of verifying quantized neural networks and how it differs from the verification of standard neural networks.

1.3 Quantized Neural Networks

Quantized neural networks (QNNs) are neural networks that represent their parameters and computations as low-bit integer variables, e.g., typically 8-bit integers [Hubara et al., 2016, Li et al., 2017, Chen et al., 2017, Hubara et al., 2017, Jacob et al., 2018]. The compressed size and reduced precision of the quantized model significantly improve the computational efficiency of running a network, e.g., see Table 1.1, while usually having only minor effects on the model's accuracy. Consequently, quantizing trained networks before deploying them in embedded applications has been adopted as a de-facto standard in the industry. For instance, the first generation of Google's Tensor processing units (TPUs) only supported running quantized neural networks [Jouppi et al., 2017]. Similarly, neural processing units (NPUs) for accelerating neural networks in phones and autonomous driving hardware [1] are often optimized for quantized networks.

In Chapter 2, we are the first to study how formal properties of a network, such as certified adversarial robustness, are affected by quantization. We show that the robustness of a network is non-monotonic in the numerical precision used to run the network. Consequently, when we quantize a provably robust floating-point network, we cannot conclude the formal robustness of the quantized model. To formally verify some properties of quantized networks, we propose a bit-vector SMT encoding that captures the exact semantics of the integer networks. Using our SMT encodings and existing methods for verifying non-quantized networks, we experimentally study how the robustness of networks with floating-point and quantized at various bit-levels, i.e., 6 to 10 bits, differ in practice. For the non-quantized, 7-bit, and 8-bit quantized network triple, we observed all of the eight possible combinations of robustness-vs-adversarial vulnerability verification outcomes. For example, input samples exist for which the floating-point network is provably robust, while adversarial attacks for the quantized models exist. Counterintuitively but expected from our theoretical insights, in some cases, quantizing with fewer bits makes the network robust again. Conversely, some samples can be certified robust for the quantized networks, while attacks for the non-quantized model exist.

[1] `https://en.wikichip.org/wiki/tesla_(car_company)/fsd_chip`

Moreover, most complete verification methods for non-quantized networks do not prove the robustness of floating-point networks directly; they do so for their real-valued abstractions [Katz et al., 2017]. In essence, no rounding errors caused by the floating-point numerics are considered. Our theoretical insights also show that a verification tool might miss adversarial attacks that exploit floating-point rounding effects in such cases. Concurrent work has demonstrated that there indeed exist such floating-point attack opportunities in practice [Jia and Rinard, 2021].

From a complexity perspective, verifying the robustness of a non-quantized network is NP-hard [Katz et al., 2017]. However, the complexity bit-vector SMT used in our verification method for quantized networks is in general NEXP-complete [Kovásznai et al., 2016]. In Chapter 3, we prove that verifying the robustness of a quantized neural network is PSPACE-hard, i.e., more complex than for non-quantized networks. We show this result by a poly-time reduction from TQBF, i.e., a known PSPACE-complete problem [Arora and Barak, 2009], to the adversarial verification problem of quantized neural networks. From a top-level view, the same effect that allows quantization to express a neural network at a fraction of the original size also allows a more succinct expression of computationally hard problems, thus causing an increase in complexity compared to the verification of real-valued networks.

We propose an optimized bit-vector SMT formulation of quantized networks by leveraging their internal structure to simplify their encoded representation to tackle the complexity barrier. First, we perform a fast abstract interpretation overapproximation of the network's behavior to obtain bounds on the value of each node inside the network. Next, we prune variables and constraints from the SMT formula that the obtained bounds already determine. Moreover, while a naive bit-vector encoding would represent each variable in the SMT formula by the exact amount of bits used in their actual implementation, e.g., 32-bit registers for intermediate results, we use the obtained variable bounds to allocate only the minimum necessary bits for each variable to avoid overflows. In essence, we prune bits that will never change during the network's computations. Finally, as the weights are quantized, there is a high chance of two or more equal weights coming from the same input. We factorize such common subexpressions into a single expression to yield an SMT formula with fewer constraints.

Experimental results on our optimized encodings show that they improve the scalability of SMT-based verification of quantized neural networks by up to three orders of magnitude compared to naive encodings. We discuss these results in Chapter 3.

1.4 Robust Learning

Robust learning describes the problem of training a network that is immune to certain types of attacks [Huber, 1964, Xu et al., 2009, Madry et al., 2018, Zhang et al., 2019, Song et al., 2019, Konstantinov and Lampert, 2019]. Attacks that try to make a neural network produce incorrect outputs by manipulating its input, i.e., often referred to as *adversarial attacks*, are among the most common types of attacks [Goodfellow et al., 2014b, Eykholt et al., 2018]. Formally, an adversarial attack is a norm-bounded noise vector ε that when added to an input sample x changes the classification output of the network, i.e., $f(x) \neq f(x+\varepsilon)$, e.g., as shown in Figure 1.2. A network is said to be robust if no such perturbation ε for the data samples exist within a given radius[2]. The most dominant approach for training robust models

[2]Note that adversarial robustness is not defined for a network alone but only with respect to a set data samples

is to add adversarial perturbations to the training data already during the learning procedure [Madry et al., 2018]. In essence, the risk term that is minimized via stochastic gradient descent becomes the *robust optimization* [Wald, 1945] objective of

$$\frac{1}{n}\sum_{i=1}^{n}\max_{\varepsilon:\|\varepsilon\|\leq\delta}\mathcal{L}(f_\theta(x_i+\varepsilon),y_i), \tag{1.1}$$

where $\delta > 0$ is some attack budget controlling how much each input can be perturbed. While these techniques do not provide absolute guarantees about the robustness of the trained networks, they significantly improve robustness empirically.

Adversarial training methods do not improve the robustness of a network for free but at the cost of lower nominal accuracy [Raghunathan et al., 2019, Zhang et al., 2019]. For instance, the advanced adversarial training algorithm of Zhang et al. [2019], which won the NeurIPS 2018 Adversarial Vision Challenge, yielded a robust network with an accuracy of 89% on the CIFAR-10 dataset. In contrast, standard training algorithms can easily produce non-robust networks with an accuracy above 96% on this dataset. In particular, adversarial training violates some of the assumptions made in machine learning, i.e., the training samples are iid, making the optimization process more difficult. Consequently, when deploying a network in a real-world machine learning application, we are faced with the dilemma of deploying a highly accurate but non-robust or a slightly less accurate but robust network.

In Chapter 4 of this thesis, we study this dilemma empirically and investigate the accuracy-robustness tradeoff in the context of robot learning [Atkeson and Schaal, 1997]. Robotic tasks are inherently closed-loop with several controller-environment interactions per second, unlike the static applications typically considered in machine learning research. For instance, a network running at 10Hz that controls an autonomous vehicle may tolerate occasional failure to detect other cars if the controller does detect them in the next frame. Contrarily, the vehicle will crash eventually if the network consistently fails to detect green cars, even though green cars might be uncommon. Finally, robots that operate in an open domain are expected to encounter situations that deviate from the nominally collected training scenarios and observe out-of-distribution data. Thus, resilience and robustness requirements are vital to robot learning setups.

To characterize these erroneous behaviors in robot learning settings on a more fine-grained level, we introduce three types of error profiles: transient, systematic, and conditional errors. We first generalize adversarial training to a safety-domain optimization scheme allowing for more generic specifications, i.e., arbitrary adversarial domains instead of norm-bounded neighborhoods around samples. We then perform a case study consisting of three real-world robot-learning tasks. We train each network with standard and adversarial/safety-domain training at various levels of the adversarial attack budget. Finally, we evaluate whether the robotic controllers can solve a predefined set of control scenarios.

We observed that the networks trained with standard empirical risk minimization yield the best application performance. Moreover, while the models trained with small adversarial attack budgets still behave acceptably, the performance deteriorates significantly for non-trivial attack budgets. In essence, our results in Chapter 4 suggest that the negative effects on the accuracy outweigh the improved robustness of adversarial training procedures in practice.

1.5 Closed-loop Stability and Safety

Often a trained network does not operate alone but continuously interacts with a system, e.g., closed control loops with neural network control policies. Research from the control theory community has considered the problem of verifying the safety of closed loops with neural network control policies. The two dominant notions of safety considered in the literature are stability, which requires all system trajectories to converge to an equilibrium set and safety in the context of all system trajectories avoiding unsafe states.

Lyapunov functions allow verifying stability specifications of such closed-loop systems [Khalil and Grizzle, 1996]. Finding such a Lyapunov function or proving its existence for a given system is the main challenge of this approach. While older works relied on Lyapunov functions that are given or synthesized via constraint-solving [Berkenkamp et al., 2017], more recently, reformulating the problem of finding a Lyapunov function as a learning problem has emerged as a more scalable alternative [Richards et al., 2018, Chang et al., 2019, Abate et al., 2021]. In particular, the Lyapunov function is parametrized by a neural network and trained to satisfy the conditions of a valid Lyapunov function.

Barrier and positive invariant functions allow verifying safety specifications in the context of the system never to reach an unsafe state [Blanchini and Miani, 2008, Ames et al., 2019]. These functions partition the system's state space into two safe and unsafe states so that no transition from the safe to the unsafe states exists. Naturally, these techniques have been adapted to verify systems' close-loop safety with neural networks as controllers [Peruffo et al., 2021].

Alternative approaches for proving a system's safety are based on bounded analysis. These methods construct overapproximations of all states reachable from given initial states, i.e., reachtubes. As the reachtube is guaranteed to contain all possible system trajectories, we can conclude the system's safety if the reachtube is disjoint from unsafe system states. Bounded reachability analysis over finite time horizons of systems with neural network controllers have been studied in Ivanov et al. [2019], Dutta et al. [2019], Gruenbacher et al. [2020]

In this thesis, we study the safety of Bayesian neural networks in feed-forward and closed-loop settings via positive invariants.

Bayesian Neural Networks Bayesian neural networks (BNNs) model their parameters θ as a probability distribution over the parameter space, i.e., $\theta \sim p(\theta)$ [MacKay, 1992, Hinton and Van Camp, 1993, Barber and Bishop, 1998, MacKay, 1995, Neal, 2012, Blundell et al., 2015, Gal and Ghahramani, 2016, Maddox et al., 2019]. As a result, Bayesian neural networks do not learn a single function f_θ but an entire distribution of networks $f_{\theta\sim p(\theta))}$. Bayesian neural networks can learn a flexible class of output distribution from primitive parameter distributions, e.g., typically Gaussian parameters. Consequently, Bayesian neural networks can capture uncertainties present in the data and model's prediction as illustrated in Figure 1.3.

These capabilities make BNNs an appealing model choice for many real-world applications in the robotic and healthcare domains, where uncertainty is an inherent property of the data [Herzog and Ostwald, 2013, McAllister et al., 2017, Amini et al., 2020b, Michelmore et al., 2020]. However, precisely these domains often involve making safety-critical decisions, requiring safety guarantees on the behavior of the network.

In Chapter 5, we introduce two approaches for verifying the safety of BNNs, one for feed-forward specifications and one for BNNs in closed-loop systems. Our method verifies sure safety, i.e., the safety of every system execution, whereas the existing verification techniques for Bayesian neural networks rely on sampling, which provides only statistical guarantees. Moreover, our method allows verifying BNNs in closed-loop systems that run indefinitely, i.e., over the infinite time horizon, while existing sampling-based approaches can only handle finite time horizons [Cardelli et al., 2019, Wicker et al., 2020, Michelmore et al., 2020].

Bayesian neural networks typically use weight distributions with unbounded support, i.e., *"long-tailed"* distributions such as Gaussian distributions. Consequently, BNN's output distributions are likely also to have unbounded support and assign a non-zero probability density to every possible output decision in $\mathbb{R}$. However, the existence of rare tail events of the output variables is problematic with respect robustness and safety properties of the network. Consequently, typical Bayesian neural networks might be considered unsafe by default under such strict definition of safety. To overcome this issue, we introduce a re-calibration scheme in the form of safe weight sets, which provides safety guarantees as long as the BNN samples its parameters from this set. We propose a method for efficiently computing such safe weight sets for feed-forward BNNs by using constraint solving techniques for deterministic neural networks.

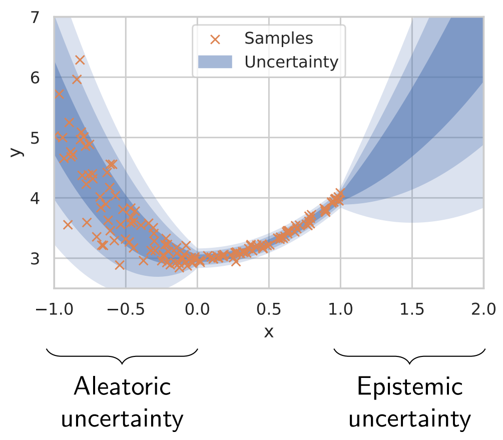


Figure 1.3: Visualization of how aleatoric and epistemic uncertainty emerges when fitting a function to data. Aleatoric uncertainty is caused by ambiguous data, while epistemic uncertainty is caused by a lack of data. Bayesian neural networks can capture both sources of uncertainty.

For proving the safety of BNNs in closed-loop systems, we synthesize a safety certificate in the form of a positive invariant, i.e., a set that the system is guaranteed never to leave as long as the BNN's parameters are sampled from the safe weight set. We represent the potentially infinite positive invariant set efficiently as a deterministic neural network classifier. Next, we propose an algorithm for learning the positive invariant classifier by obtaining the training data as samples from the safety specification and the system's executions. After the training process, our method checks if the trained classifier is a valid positive invariant, i.e., the system never leaves the positive invariant. If a counterexample is found that shows how the classifier violates a positive invariant condition, we add the counterexample to the training data and re-train the invariant classifier. We repeat this process until the system is proven safe by finding a valid positive invariant.

We experimentally evaluate our approach on three benchmark tasks and show that our method can yield non-trivial safe weight sets for Bayesian neural networks in practice.

1.6 Methods

In this section, we summarize the formal methods used throughout the thesis. Formal methods are rigorous mathematical proof techniques used to analyze system properties formally. They can be used to provide formal guarantees about whether a system fulfills its specification, e.g., correctness, safety, and security. Some methods and concepts described here have already been briefly introduced in previous sections.

Satisfiability modulo theories Computation problems can often be represented as satisfiability questions over formal systems of rules and axioms. The most commonly used formal systems are the propositional logic and the first-order logic [Kroening and Strichman, 2016].

Satisfiability problems phrased in propositional logic are generally solvable by SAT-solvers in NP. However, propositional logic reasons only about true or false propositions, which severely limits its expressiveness in practice. For example, while verification problems of binarized neural networks, i.e., networks with weights and activations restricted to the domain $\{-1, 1\}$, can be naturally expressed in propositional logic [Narodytska et al., 2018, Cheng et al., 2018], verification problems that involve variables over the real domain are impossible to express in propositional logic.

First-order logic allows quantifiers, predicates, and functions over arbitrary domains and is significantly more expressive than propositional logic. The main disadvantage of first-order logic is that it is generally undecidable.

Satisfiability modulo theories (SMT) resolve this limitation by introducing theories that restrict first-order logic to a decidable subset. In particular, an SMT theory defines the interpretation of specific predicates and functions and asks whether a given formula is satisfiable with respect to the fixed interpretation. For example, the theory of linear real arithmetic fixes the interpretation of the function $+$ and the predicate $<$ according to their generally known mathematical definitions. Computer programs that check if an SMT formula is satisfiable are called SMT-solvers. Most practically relevant SMT theories further remove quantifiers and fix the interpretation of all predicates and functions for the sake of algorithmic efficiency [Barrett et al., 2017, Kroening and Strichman, 2016, Clarke et al., 2018].

We use SMT with the background theories of quantifier-free bit-vectors (QF_BV) [Barrett et al., 2017] in Chapters 2 and 3. Notably, we phrase the formal verification question of quantized neural networks as a satisfiability problem over the theory of bit-vectors. Our SMT formulas are satisfiable if and only if the specification of the quantized neural network is violated. In Chapter 3 we simplify the formulas by removing the parts of the formula whose value can be determined in advance, leading to a significant speedup in solving time.

In Chapter 5, we show how non-linear real arithmetic SMT (NRA-SMT) and SMT over the Reluplex calculus [Katz et al., 2017] can be used to verify the safety of Bayesian neural networks restricted to weight-sets.

Mixed-integer linear programming Mixed-integer linear programming (MILP) refers to a class of optimization problems over real and integer variables with linear inequality constraints [Dantzig, 2016]. While optimization problems over real variables with linear inequalities can be solved in polynomial time, the addition of integer variables makes the problem NP-complete. Computer programs that can find the solution of a MILP problem, if there is any, are called MILP-solvers.

MILP has been used to verify the robustness of neural networks with piecewise linear activation functions [Tjeng et al., 2019]. Particularly, the network and the negation of the specification are expressed as a set of linear constraints, i.e., a MILP problem instance. A MILP-solver is then asked to find a variable assignment that fulfills the constraints corresponding to the network's semantics but violates the specification.

In Chapter 5, we use mixed-integer linear programming to verify safe-weight sets of Bayesian neural networks. A safe-weight set is a set of weights for which safety is guaranteed as long as the Bayesian neural network samples its weights from this set. In particular, we express the semantics of the network using MILP constraints via the scheme of Tjeng et al. [2019] but let each weight w vary within an interval $[l, u]$ resembling the safe weight set.

Abstract interpretation Abstract interpretation is a method that over-approximates a computer program's behavior soundly, i.e., the approximation must be a superset of all possible executions of the program [Cousot and Cousot, 1977]. The key idea is to interpret the program's semantics over abstract domains instead of concrete program states. For instance, an integer variable z might be interpreted over the two abstract sets "z is zero or positive" and "z is negative". Problems that may be undecidable for the program's concrete semantics, e.g., termination analysis, might be decidable for the abstract interpretation depending on the choice of the abstract domain. The main disadvantage of abstract interpretation methods is that they are incomplete, i.e., the abstraction might contain program executions that are not realizable in the concrete semantics, leading to a pessimistic characterization of the program. Nonetheless, abstract interpretation is a feasible and fast way to obtain information about the behavior of a program. Consequently, abstract interpretation is widely adopted in static code analysis tools and compiler optimization frameworks.

In the context of neural networks, abstract interpretation can reduce the complexity class of neural network verification problems [Wang et al., 2018b, Singh et al., 2018, Bunel et al., 2018]. For example, checking the robustness of a ReLU-network over the network's concrete semantics is NP-hard in the number of neurons [Katz et al., 2017] but only of linear complexity when the network is interpreted over interval domains [Gowal et al., 2019].

In Chapter 4, we use abstract interpretation to learn networks with safety constraints. In particular, we jointly train networks over their concrete semantics on the training data and over their interval arithmetic semantics with respect to the safety objectives. As a result, the network's concrete interpretation fits the training data and is guaranteed never to make unsafe decisions.

Unsound methods Unsound or *incorrect* methods do not provide formal guarantees but ad-hoc estimates of whether a system satisfy given specifications. Despite this downside, unsound methods play an essential role in software development, machine learning, and robotics. Unsound methods do not suffer from undecidability or problematic complexity class issues and quickly scale to complex systems. For example, testing allows us to uncover bugs in programs or estimate the expected performance of machine learning models efficiently without providing any hard formal guarantees.

In Chapter 4, we use unsound methods to estimate the adversarial robustness of large neural networks for which existing sound methods do not scale anymore. Notably, we try to change the prediction of an image classifier by making tiny changes to the input pixels via gradient ascent to maximize the loss term [Goodfellow et al., 2014b]. We run this process for all images in our dataset and measure the attack success rate as a proxy metric of the adversarial vulnerability of a network.

Invariants An invariant is a property of an object that is unaffected when applying certain transformations to the object. For instance, the volume of a cube is invariant regarding rotations and translations applied to the cube. Positive invariants (or inductive invariants) are subsets of invariants that can be proved by mathematical induction, i.e., by proving that the invariant holds in the initial states and that if the invariant holds for a state, then it also holds for all successor states. In program verification, invariants are used to prove that a property holds over the entire execution of a program, i.e., the property is invariant with respect to the transformation of executing the program [Floyd, 1967]. In control theory, invariant sets have been used to prove safety and stability of dynamical systems [Blanchini and Miani, 2008].

In Chapter 5, we use positive invariant sets for certifying the safety of systems involving Bayesian neural networks controllers. We represent the positive invariant by a neural network classifier and train it until the network is indeed an invariant. The step of proving that the network fulfills the conditions of an invariant, i.e., initial case and induction step, is done via MILP.

1.7 Summary of Contributions

In summary, this dissertation makes contributions on the following research topics:

- In Chapters 2 and 3, we introduce the first verification methods for quantized neural networks and present the theoretical study of quantized neural network verification's non-monotonicity and algorithmic complexity.
- In Chapter 4, we empirically study the robustness-accuracy tradeoff of adversarial training methods in robot learning settings and observe that, in practice, the improved robustness of existing adversarial training methods is not worth the reduction in nominal accuracy.
- In Chapter 5, propose a method for proving sure safety of Bayesian neural networks by re-calibrating the networks on safe weight sets. Furthermore, we introduce an algorithm for computing safe weight sets for feed-forward and closed-loop safety specifications.

CHAPTER 2

How Many Bits Does it Take to Quantize Your Neural Network?

In this chapter, we investigate the robustness of quantized networks to adversarial attacks and, more generally, formal verification questions for quantized neural networks.

The formal verification of neural networks has been addressed either by overapproximating—as happens in abstract interpretation—the space of outputs given a space of attacks, or by searching—as it happens in SMT-solving—for a variable assignment that witnesses an attack. The first category include methods that relax the neural networks into computations over interval arithmetic [Pulina and Tacchella, 2010], treat them as hybrid automata [Xiang et al., 2018], or abstract them directly by using zonotopes, polyhedra [Gehr et al., 2018], or tailored abstract domains [Singh et al., 2019]. Overapproximation-based methods are typically fast, but incomplete: they prove robustness but do not produce attacks. On the other hand, methods based on local gradient descent have turned out to be effective in producing attacks in many cases [Moosavi-Dezfooli et al., 2016], but sacrifice formal completeness. Indeed, the search for adversarial attack is NP-complete even for the simplest (i.e., ReLU) networks [Katz et al., 2017], which motivates the rise of methods based on satisfiability modulo theories (SMT) and mixed-integer linear programming (MILP). SMT-solvers have been shown not to scale beyond toy examples (20 hidden neurons) on monolithic encodings [Pulina and Tacchella, 2012], but today's specialized techniques can handle real-life benchmarks such as, neural networks for the MNIST dataset. Specialized tools include DLV [Huang et al., 2017], which subdivides the problem into smaller SMT instances, and Planet [Ehlers, 2017], which combines different SAT and LP relaxations. Reluplex takes a step further augmenting LP-solving with a custom calculus for ReLU networks [Katz et al., 2017]. At the other end of the spectrum, a recent MILP formulation turned out effective using off-the-shelf solvers [Tjeng et al., 2019]. Moreover, it formed the basis for Sherlock [Dutta et al., 2018], which couples local search and MILP, and for a specialized branch and bound algorithm [Bunel et al., 2018].

All techniques mentioned above do not reason about the machine-precise semantics of the networks, neither over floating- nor over fixed-point arithmetic, but reason about a real-number relaxation. Unfortunately, adversarial attacks computed over the reals are not necessarily attacks on execution architectures, in particular, for quantized networks implementations. We show, for the first time, that attacks and, more generally, robustness and vulnerability to attacks do not always transfer between real and quantized networks, and also do not always transfer monotonically with the number of bits across quantized networks. Verifying the real-valued relaxation of a network may lead scenarios where

(i) specifications are fulfilled by the real-valued network but not for its quantized implementation (false negative),

(ii) specifications are violated by the real-valued network but fulfilled by its quantized representation (false negatives), or

(iii) counterexamples witnessing that the real-valued network violated the specification, but do not witness a violation for the quantized network (invalid counterexamples/attacks).

More generally, we show that all three phenomena can occur non-monotonically with the precision in the numerical representation. In other words, it may occur that a quantized network fulfills a specification while both a higher and a lower bits quantization violate it, or that the first violates it and both the higher and lower bits quantizations fulfill it; moreover, specific counterexamples may not transfer monotonically across quantizations.

The verification of real-numbered neural networks using the available methods is inadequate for the analysis of their quantized implementations, and the analysis of quantized neural networks needs techniques that account for their bit-precise semantics. Recently, a similar problem has been addressed for binarized neural networks, through SAT-solving [Narodytska et al., 2018]. Binarized networks represent the special case of 1-bit quantizations. For many-bit quantizations, a method based on gradient descent has been introduced recently [Zhao et al., 2019]. While efficient (and sound), this method is incomplete and may produce false negatives.

We introduce, for the first time, a complete method for the formal verification of quantized neural networks. Our method accounts for the bit-precise semantics of quantized networks by leveraging the first-order theory of bit vectors without quantifiers (QF_BV), to exactly encode hardware operations such as 2'complementation, bit-shift, integer arithmetic with overflow. On the technical side, we present a novel encoding which balances the layout of long sequences of hardware multiply-add operations occurring in quantized neural networks. As a result, we obtain an encoding into a first-order logic formula which, in contrast to a standard unbalanced linear encoding, makes the verification of quantized networks practical and amenable to modern bit-precise SMT-solving. We built a tool using Boolector [Niemetz et al., 2015], evaluated the performance of our encoding, compared its effectiveness against real-numbered verification and gradient descent for quantized networks, and finally assessed the effect of quantization for different networks and verification questions.

We measured the robustness to attacks of a neural classifier involving 890 neurons and trained on the MNIST dataset (handwritten digits), for quantizations between 6 and 10 bits. Each robustness check was running on a single CPU core of a Intel Xeon W-2175 CPU with 64GB memory. First, we demonstrated that Boolector, off-the-shelf and using our balanced SMT encoding, can compute every attack within 16 hours, with a median time of 3h 41m, while timed-out on all instances beyond 6 bits using a standard linear encoding. Second, we experimentally confirmed that both Reluplex and gradient descent for quantized networks can produce false conclusions about quantized networks; in particular, spurious results occurred consistently more frequently as the number of bits in quantization decreases. Finally, we discovered that, to achieve an acceptable level of robustness, it takes a higher bit quantization than is assessed by standard accuracy measures.

Lastly, we applied our method beyond the property of robustness. We also evaluate the effect of quantization upon the gender bias emerging from quantized predictors for a task of predicting students' performance in mathematics exams using synthetic data. More precisely, we computed the maximum predictable grade gap between any two students with identical features except for gender. The experiment showed that a substantial gap existed and was proportionally enlarged by quantization: the lower the number bits the larger the gap.

We summarize our contribution in five points. First, we show that the robustness of quantized neural networks is non-monotonic in the number of bits and is non-transferable from the robustness of their real-numbered counterparts. Second, we introduce the first complete method for the verification of quantized neural networks. Third, we demonstrate that our encoding, in contrast to standard encodings, enabled the state-of-the-art SMT-solver Boolector to verify quantized networks with hundreds of neurons. Fourth, we also show that existing methods determine both robustness and vulnerability of quantized networks less accurately than our bit-precise approach, in particular for low-bit quantizations. Fifth, we illustrate how quantization affects the robustness of neural networks, not only with respect to adversarial attacks, but also with respect to other verification questions, specifically fairness in machine learning.

2.1 Quantization of Feed-forward Networks

A feed-forward neural network consists of a finite set of *neurons* $x_1, \ldots, x_k$ partitioned into a sequence of layers: an *input layer* with n neurons, followed by one or many *hidden layers*, finally followed by an *output layer* with m neurons. Every pair of neurons x_j and x_i in respectively subsequent layers is associated with a *weight* coefficient $w_{ij} \in \mathbb{R}$; if the layer of x_j is not subsequent to that of x_i, then we assume $w_{ij} = 0$. Every hidden or output neuron x_i is associated with a *bias* coefficient $b_i \in \mathbb{R}$. The real-valued semantics of the neural network gives to each neuron a real value: upon a valuation for the neurons in the input layer, every other neuron x_i assumes its value according to the update rule

$$x_i = \mathrm{ReLU}\text{-}N(b_i + \sum_{j=1}^{k} w_{ij} x_j), \tag{2.1}$$

where $\mathrm{ReLU}\text{-}N\colon \mathbb{R} \to \mathbb{R}$ is the *activation function*. Altogether, the neural network implements a function $f\colon \mathbb{R}^n \to \mathbb{R}^m$ whose result corresponds to the valuation for the neurons in the output layer.

The activation function governs the firing logic of the neurons, layer by layer, by introducing non-linearity in the system. Among the most popular activation functions are purely non-linear functions, such as the tangent hyperbolic and the sigmoidal function, and piece-wise linear functions, better known as *Rectified Linear Units* (ReLU) [Nair and Hinton, 2010]. ReLU consists of the function that takes the positive part of its argument, i.e., $\mathrm{ReLU}(x) = \max\{x, 0\}$. We consider the variant of ReLU that imposes a cap value N, known as $\mathrm{ReLU}\text{-}N$ [Krizhevsky and Hinton, 2010]. Precisely

$$\mathrm{ReLU}\text{-}N(x) = \min\{\max\{x, 0\}, N\}, \tag{2.2}$$

which can be alternatively seen as a concatenation of two ReLU functions (see Eq. 2.10). As a consequence, all neural networks we treat are full-fledged ReLU networks; their real-valued versions are amenable to state-of-the-art verification tools including Reluplex, but neither account for the exact floating- nor fixed-point execution models.

Quantizing consists of converting a neural network over real numbers, which is normally deployed on floating-point architectures, into a neural network over integers, whose semantics corresponds to a computation over fixed-point arithmetic [Jacob et al., 2018]. Specifically, fixed-point arithmetic can be carried out by integer-only architectures and possibly over small words, e.g., 8 bits. All numbers are represented in 2's complement over B bits words and F bits are reserved to the fractional part: we call the result a *B-bits quantization in QF arithmetic*. More concretely, the conversion follows from the rounding of weight and bias coefficients to the F-th digit, namely $\bar{b}_i = \mathsf{rnd}(2^F b_i)$ and $\bar{w}_{ij} = \mathsf{rnd}(2^F w_{ij})$ where $\mathsf{rnd}(\cdot)$ stands for any rounding to an integer. Then, the fundamental relation between a quantized value $\bar{a}$ and its real counterpart a is

$$a \approx 2^{-F}\bar{a}. \tag{2.3}$$

Consequently, the semantics of a quantized neural network corresponds to the update rule in Eq. 2.1 after substituting of x, w, and b with the respective approximants $2^{-F}\bar{x}$, $2^{-F}\bar{w}$, and $2^{-F}\bar{b}$. Namely, the semantics amounts to

$$\bar{x}_i = \mathrm{ReLU}\text{-}(2^F N)(\bar{b}_i + \mathsf{int}(2^{-F}\sum_{j=1}^{k}\bar{w}_{ij}\bar{x}_j)), \tag{2.4}$$

where $\mathsf{int}(\cdot)$ truncates the fractional part of its argument or, in other words, rounds towards zero. In summary, the update rule for the quantized semantics consists of four parts. The first part, i.e., the linear combination $\sum_{j=1}^{k}\bar{w}_{ij}\bar{x}_j$, propagates all neurons values from the previous layer, obtaining a value with possibly $2B$ fractional bits. The second scales the result by 2^{-F} truncating the fractional part by, in practice, applying an arithmetic shift to the right of F bits. Finally, the third applies the bias $\bar{b}$ and the fourth clamps the result between 0 and $2^F N$. As a result, a quantize neural network realizes a function $f\colon \mathbb{Z}^n \to \mathbb{Z}^m$, which exactly represents the concrete (integer-only) hardware execution.

We assume all intermediate values, e.g., of the linear combination, to be fully representable as, coherently with the common execution platforms [Jacob et al., 2018], we always allocate enough bits for under and overflow not to happen. Hence, any loss of precision from the respective real-numbered network happens exclusively, at each layer, as a consequence of rounding the result of the linear combination to F fractional bits. Notably, rounding causes the robustness to adversarial attacks of quantized networks with different quantization levels to be independent of one another, and independent of their real counterpart.

2.2 Robustness is Non-monotonic in the Number of Bits

A neural classifier is a neural network that maps a n-dimensional input to one out of m classes, each of which is identified by the output neuron with the largest value, i.e., for the output values $z_1, \ldots, z_m$, the choice is given by

$$\mathrm{class}(z_1, \ldots, z_m) = \arg\max_i z_i. \tag{2.5}$$

For example, a classifier for handwritten digits takes in input the pixels of an image and returns 10 outputs $z_0, \ldots, z_9$, where the largest indicates the digit the image represents. An adversarial attack is a perturbation for a sample input

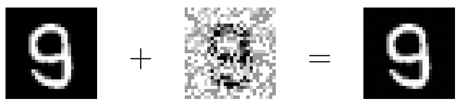


Figure 2.1: Illustration of an adversarial attack on a 8-bit quantized neural network trained on the MNIST dataset. The image of the left is correctly classified as the digit "9", whereas the image of the right is misclassified as "3".

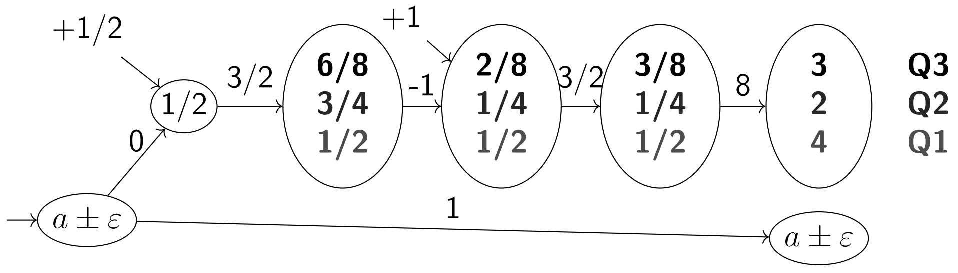


Figure 2.2: Neural network with non-monotonic robustness with respect to the precision it is executed. The labels of the arrows indicate the weight of the connection. An arrow without a source neuron represents a bias term. The network is executed with three different levels of precision Q1, Q2, and Q3 in binary representation. The Q1 representation corresponds to 1 fractional bit, i.e., every neuron value is a fraction of 2, whereas the Q2 and Q3 precision levels corresponds to 2 and 3 fractional bits respectively. The three rows of values within a neuron correspond to the value of the neuron under the precision Q3, Q2, and Q1 from top to bottom respectively.

$$\text{original} + \text{perturbation} = \text{attack}$$

that, according to some notion of closeness, is indistinguishable from the original, but tricks the classifier into inferring an incorrect class. The attack in Figure 2.1 is indistinguishable from the original by the human eye, but induces our classifier to assign the largest value to z_3, rather than z_9, misclassifying the digit as a 3. For this example, misclassification happens consistently, both on the real-numbered and on the respective 8-bits quantized network in Q4 arithmetic. Unfortunately, attacks do not necessarily transfer between real and quantized networks and neither between quantized networks for different precision. More generally, attacks and, dually, robustness to attacks are non-monotonic with the number of bits.

We give a prototypical example for the non-monotonicity of quantized networks in Figure 2.2. The network consists of one input, 4 hidden, and 2 output neurons, respectively from left to right. Weights and bias coefficients, which are annotated on the edges, are all fully representable in Q1. For the neurons in the top row we show, respectively from top to bottom, the valuations obtained using a Q3, Q2, and Q1 quantization of the network (following Eq. 2.4); precisely, we show their fractional counterpart $\bar{x}/2^F$. We evaluate all quantizations and obtain that the valuations for the top output neuron are non-monotonic with the number of fractional bits; in fact, the Q1 dominates the Q3 which dominates the Q2 output. Coincidentally, the valuations for the Q3 quantization correspond to the valuations with real-number precision (i.e., never undergo truncation), indicating that also real and quantized networks are similarly incomparable. Notably, all phenomena occur both for quantized networks with rounding towards zero (as we show in the example), and with rounding to the nearest, which is naturally non-monotonic (e.g., 5/16 rounds to 1/2, 1/4, and 3/8 with, resp., Q1, Q2, and Q3).

Non-monotonicity of the output causes non-monotonicity of robustness, as we can put the decision boundary of the classifier so as to put Q2 into a different class than Q1 and Q3. Suppose the original sample is 3/2 and its class is associated with the output neuron on the top, and suppose attacks can only lay in the neighboring interval $3/2 \pm 1$. In this case, we obtain that the Q2 network admits an attack, because the bottom output neuron can take 5/2, that is larger than 2. On the other hand, the bottom output can never exceed 3 and 4, hence Q1 and Q3 are robust. Dually, also non-robustness is non-monotonic as, for the sample 9/2 whose class corresponds to the bottom neuron, for the interval $9/2 \pm 2$, Q2 is robust while both Q3 and Q1 are vulnerable. Notably, the specific attacks of Q3 and Q1 also do not always coincide as, for instance, 7/2.

Robustness and non-robustness are non-monotonic in the number of bits for quantized networks. As a consequence, verifying a high-bits quantization, or a real-valued network, may derive false conclusions about a target lower-bits quantization, in either direction. Specifically, for the question as for whether an attack exists, we may have both (i) false negatives, i.e., the verified network is robust but the target network admits an attack, and (ii) false positives, i.e., the verified network is vulnerable while the target network robust. In addition, we may also have (iii) true positives with invalid attacks, i.e., both are vulnerable but the found attack do not transfer to the target network. For these reasons we introduce a verification method quantized neural network that accounts for their bit-precise semantics.

2.3 Verification of Quantized Networks using Bit-precise SMT-solving

Bit-precise SMT-solving comprises various technologies for deciding the satisfiability of first-order logic formulae, whose variables are interpreted as bit-vectors of fixed size. In particular, it produces satisfying assignments (if any exist) for formulae that include bitwise and arithmetic operators, whose semantics corresponds to that of hardware architectures. For instance, we can encode bit-shifts, 2's complementation, multiplication and addition with overflow, signed and unsigned comparisons. More precisely, this is the quantifier-free first-order theory of bit-vectors (i.e., QF_BV), which we employ to produce a monolithic encoding of the verification problem for quantized neural networks.

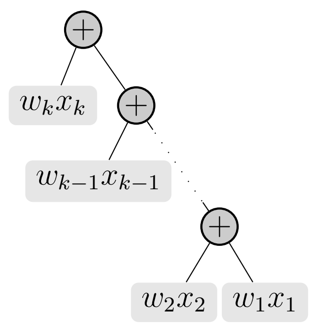
A verification problem for the neural networks $f_1, \ldots, f_K$ consists of checking the validity of a statement of the form

$$\varphi(\vec{y}_1, \ldots, \vec{y}_K) \implies \psi(f_1(\vec{y}_1), \ldots, f_K(\vec{y}_K)), \tag{2.6}$$

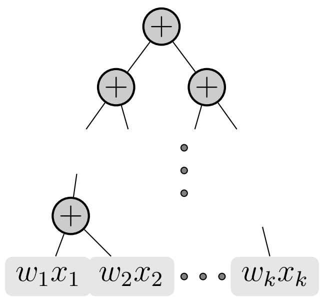
where φ is a predicate over the inputs and ψ over the outputs of all networks; in other words, it consists of checking an input–output relation, which generalizes various verification questions, including robustness to adversarial attacks and fairness in machine learning, which we treat in Section 2.4. For the purpose of SMT solving, we encode the verification problem in Eq. 2.6, which is a validity question, by its dual satisfiability question

$$\varphi(\vec{y}_1, \ldots, \vec{y}_K) \wedge \bigwedge_{i=1}^{K} f_i(\vec{y}_i) = \vec{z}_i \ \wedge \ \neg\psi(\vec{z}_1, \ldots, \vec{z}_K), \tag{2.7}$$

whose satisfying assignments constitute counterexamples for the contract. The formula consists of three conjuncts: the rightmost constraints the input within the assumption, the leftmost



Linear layout
(a)



Balanced layout
(b)

Figure 2.3: Abstract syntax trees for alternative encodings of a long linear combination of the form $\sum_{i=1}^{k} w_i x_i$.

forces the output to violate the guarantee, while the one in the middle relates inputs and outputs by the semantics of the neural networks.

The semantics of the network consists of the bit-level translation of the update rule in Eq. 2.4 over all neurons, which we encode in the formula

$$\bigwedge_{i=1}^{k} x_i = \mathrm{ReLU}\text{-}(2^F N)(x_i') \wedge \ x_i' = \bar{b}_i + \mathtt{ashr}(x_i'', F) \wedge \ x_i'' = \sum_{j=1}^{k} \bar{w}_{ij} x_j. \tag{2.8}$$

Each conjunct in the formula employs three variables x, x', and x'' and is made of three, respective, parts. The first part accounts for the operation of clamping between 0 and $2^F N$, whose semantics is given by the formula $\mathrm{ReLU}\text{-}M(x) = \mathtt{ite}(\mathtt{sign}(x), 0, \mathtt{ite}(x \geq M, M, x))$. Then, the second part accounts for the operations of scaling and biasing. In particular, it encodes the operation of rounding by truncation scaling, i.e., int$(2^{-F}x)$, as an arithmetic shift to the right. Finally, the last part accounts for the propagation of values from the previous layer, which, despite the obvious optimization of pruning away all monomials with null coefficient, often consists of long linear combinations, whose exact semantic amounts to a sequence of multiply-add operations over an accumulator; particularly, encoding it requires care in choosing variables size and association layout.

The size of the bit-vector variables determines whether overflows can occur. In particular, since every monomial $w_{ij} x_j$ consists of the multiplication of two B-bits variables, its result requires $2B$ bits in the worst case; since summation increases the value linearly, its result requires a logarithmic amount of extra bits in the number of summands (regardless of the layout). Provided that, we avoid overflow by using variables of $2B + \log k$ bits, where k is the number of summands.

The association layout is not unique and, more precisely, varies with the order of construction of the long summation. For instance, associating from left to right produces a linear layout, as in Figure 2.3a. Long linear combinations occurring in quantized neural networks are implemented as sequences of multiply-add operations over a single accumulator; this naturally induces a linear encoding. Instead, for the purpose formal verification, we propose a novel encoding which re-associates the linear combination by recursively splitting the sum into equal parts, producing a *balanced layout* as in Figure 2.3b. While linear and balanced layouts are semantically equivalent, we have observed that, in practice, the second impacted positively the performance of the SMT-solver as we discuss in Section 2.4, where we also compare against other methods and investigate different verification questions.

2.4 Experimental Results

We set up an experimental evaluation benchmark based on the MNIST dataset to answer the following three questions. First, how does our balanced encoding scheme impact the runtime of different SMT solvers compared to a standard linear encoding? Then, how often can robustness properties, that are proven for the real-valued network, transferred to the quantized network and vice versa? Finally, how often do gradient based attacking procedures miss attacks for quantized networks?

The MNIST dataset is a well-studied computer vision benchmark, which consists of 70,000 handwritten digits represented by 28-by-28 pixel images with a single 8-bit grayscale channel. Each sample belongs to exactly one category $\{0, 1, \ldots 9\}$, which a machine learning model must predict from the raw pixel values. The MNIST set is split into 60,000 training and 10,000 test samples.

We trained a neural network classifier on MNIST, following a *post-training quantization* scheme [Jacob et al., 2018]. First, we trained, using TensorFlow with floating-point precision, a network composed of 784 inputs, 2 hidden layers of size 64, 32 with ReLU-7 activation function and 10 outputs, for a total of 890 neurons. The classifier yielded a *standard accuracy*, i.e., the ratio of samples that are correctly classified out of all samples in the testing set, of 94.7% on the floating-point architecture. Afterward, we quantized the network with various bit sizes, with the exception of imposing the input layer to be always quantized in 8 bits, i.e., the original precision of the samples. The quantized networks required at least Q3 with 7 total bits to obtain an accuracy above 90% and Q5 with 10 bits to reach 94%. For this reason, we focused our study on the quantizations from 6 and the 10 bits in, respectively, Q2 to Q6 arithmetic.

Robust accuracy or, more simply, *robustness* measure the ratio of robust samples: for the distance $\varepsilon > 0$, a sample a is robust when, for all its perturbations y within that distance, the classifier $\mathrm{class} \circ f$ chooses the original class $c = \mathrm{class} \circ f(a)$. In other words, a is robust if, for all $\vec{y}$

$$|a - \vec{y}|_\infty \leq \varepsilon \implies c = \mathrm{class} \circ f(\vec{y}), \tag{2.9}$$

where, in particular, the right-hand side can be encoded as $\bigwedge_{j=1}^{m} z_j \leq z_c$, for $\vec{z} = f(\vec{y})$. Robustness is a validity question as in Eq. 2.6 and any witness for the dual satisfiability question constitutes an adversarial attack. We checked the robustness of our selected networks over the first 300 test samples from the dataset with $\varepsilon = 1$ on the first 200 and $\varepsilon = 2$ on the next 100. We tested our encoding using the SMT-solver Boolector [Niemetz et al., 2015], Z3 [De Moura and Bjørner, 2008], and CVC4 [Barrett et al., 2011], off-the-shelf.

Our experiments serve two purposes. The first is evaluating the scalability and precision of our approach. As for scalability, we study how encoding layout, i.e., linear or balanced, and the number of bits affect the runtime of the SMT-solver. As for precision, we measured the gap between our method and both a formal verifier for real-numbered networks, i.e., Reluplex [Katz et al., 2017], and the IFGSM algorithm [Zhao et al., 2019], with respect to the accuracy of identifying robust and vulnerable samples. The second purpose of our experiments is evaluating the effect of quantization on the robustness to attacks of our MNIST classifier and, with an additional experiment, measuring the effect of quantization over the gender fairness

SMT-solver	Encoding	6-bit	7-bit	8-bit	9-bit	10-bit
Boolector [Niemetz et al., 2015]	Linear (standard)	3h 25m	oot	oot	oot	oot
	Balanced (ours)	18m	1h 29m	3h 41m	5h 34m	8h 58m
Z3 [De Moura and Bjørner, 2008]	Linear (standard)	oot	-	-	-	-
	Balanced (ours)	oot	-	-	-	-
CVC4 [Barrett et al., 2011]	Linear (standard)	oom	-	-	-	-
	Balanced (ours)	oom	-	-	-	-
Yices2 [Dutertre, 2014]	Linear (standard)	oot	-	-	-	-
	Balanced (ours)	oot	-	-	-	-

Table 2.1: Median runtimes for bit-exact robustness checks. The term oot refers to timeouts, and oom refers to out-of-memory errors. Due to the poor performance of Z3, CVC4, and Yices2 on our smallest 6-bit network, we abstained from running experiments involving more than 6 bits, i.e., entries marked by a dash (-).

QNN encoding	Mean runtime $\pm$ std-dev in minutes	Median runtime (0.1, 0.9 quantile) in minutes
6-bit (linear)	228.0 $\pm$ 54.6	207.1 (184.7,303.7)
6-bit (balanced)	18.4 $\pm$ 1.7	18.2 (16.5,20.0)
7-bit (balanced)	89.7 $\pm$ 12.0	90.0 (73.9,101.2)
8-bit (balanced)	221.9 $\pm$ 52.5	221.9 (153.2,261.3)
9-bit (balanced)	342.4 $\pm$ 64.4	334.6 (267.7,416.1)
10-bit (balanced)	592.9 $\pm$ 135.8	538.5 (473.9,768.6)

Table 2.2: Observed runtime uncertainty of bit-exact robustness checks in minutes. The column in the center shows the mean and the standard deviation. The column of the right shows the median runtime and the 0.1 and 0.9 quantile in parenthesis. Timed out instances and misclassified samples are not included.

of a student grades predictor, also demonstrating the expressiveness of our method beyond adversarial attacks.

As we only compared the verification outcomes, any complete verifier for real-numbered networks would lead to the same results as those obtained with Reluplex. Note that these tools verify the real-numbered abstraction of the network using some form of linear real arithmetic reasoning. Consequently, rounding errors introduced by the floating-point implementation of both, the network and the verifier, are not taken into account.

2.4.1 Scalability and performance

We evaluated whether our balanced encoding strategy, compared to a standard linear encoding, can improve the scalability of contemporary SMT solvers for quantifier-free bit-vectors (QF_BV) to check specifications of quantized neural networks. We ran all our experiments on an Intel Xeon W-2175 CPU, with 64GB memory and 16 hours of time budget per problem instance. We encoded each instance using the two variants, the standard linear and our balanced layout. We scheduled 14 solver instances in parallel, i.e., the number of physical processor cores available on our machine.

While Z3, CVC4 and Yices2 timed out or ran out of memory on the 6-bit network, Boolector could check the instances of our smallest network within the given time budget, independently of the employed encoding scheme. We note that we also tested CVC after enabling a 128 GB swap file and running only single SMT-solver instance on the entire machine to test whether the instances could be solved on a machine with larger memory, but also observed an out-of-memory error in this configuration. Our results align with the SMT-solver performances reported by the SMT-COMP 2019 competition in the QF_BV division [Niemetz et al., 2019]. Consequently, we will focus our discussion on the results obtained with Boolector.

With linear layout Boolector timed-out on all instances but the smallest networks (6 bits), while with the balanced layout it checked all instances with an overall median runtime of 3h 41m and, as shown in Table 2.1, roughly doubling at every bits increase, as also confirmed by the histogram in Figure 2.4. The observed variability and uncertainty of the executed robustness checks are shown in Table 2.2.

Our results demonstrate that our balanced association layout improves the performance of the SMT-solver, enabling it to scale to networks beyond 6 bits. Conversely, a standard linear encoding turned out to be ineffective on all tested SMT solvers. Besides, our method tackled networks with 890 neurons which, while small compared to state-of-the-art image classification models, already pose challenging benchmarks for the formal verification task. In the real-numbered world, for instance, off-the-shelf solvers could initially tackle up to 20 neurons [Pulina and Tacchella, 2010], and modern techniques, while faster, are often evaluated on networks below 1000 neurons [Katz et al., 2017, Bunel et al., 2018].

Additionally, we pushed our method to its limits, refining our MNIST network to a four-layers deep Convolutional network (2 Conv + 2 Fully-connected layers) with a total of 2238 neurons, which achieved a test accuracy of 98.56%. While for the 6-bits quantization we proved robustness for 99% of the tested samples within a median runtime of 3h 39min, for 7-bits and above all instances timed-out. Notably, Reluplex also failed on the real-numbered version, reporting numerical instability.

2.4.2 Comparison to other methods

Looking at existing methods for verification, one has two options to verify quantized neural networks: verifying the real-valued network and hoping the functional property is preserved

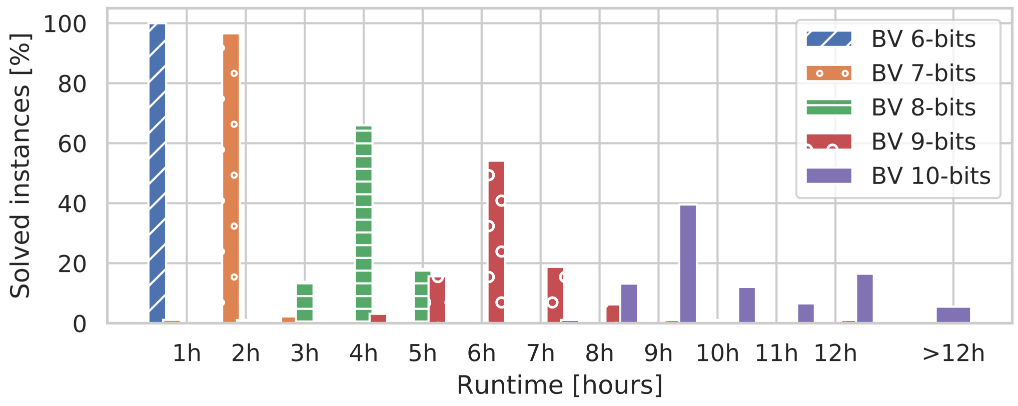


Figure 2.4: Runtimes for bit-exact adversarial robustness checks of a classifier trained on the MNIST dataset using Boolector and our balanced SMT encodings. Runtime roughly doubles with each additional bit used for the quantization.

when quantizing the network, or relying on incomplete methods and hoping no counterexample is missed. A question that emerges is how accurate are these two approaches for verifying robustness of a quantized network? To answer this question, we used Reluplex [Katz et al., 2017] to prove the robustness of the real-valued network. Additionally, we compared to the iterative fast gradient sign method (IFGSM), which has recently been proposed to generate ℓ_∞-bounded adversarial attacks for quantized networks [Zhao et al., 2019]; notably, IFGSM is incomplete in the sense that it may miss attacks. We then compared these two verification outcomes to the ground-truth obtained by our approach.

In our study, we employ the following notation. We use the term "false negative" (i) to describe cases in which the quantized network can be attacked, while no attack exists that fools the real-number network. Conversely, the term "false positive" (ii) describes the cases in which a real-number attack exists while the quantized network is robust. Furthermore, we use the term "invalid attack" (iii) to specify attacks produced for the real-valued network that fools the real-valued network but not the quantized network.

Regarding the real-numbered encoding, Reluplex accepts only pure ReLU networks. For this reason, we translate our $\mathrm{ReLU}\text{-}N$ networks into functionally equivalent ReLU networks, by translating each layer with

$$\mathrm{ReLU}\text{-}N(W \cdot \vec{x} + \vec{b}) = \mathrm{ReLU}\Big(- I \cdot \mathrm{ReLU}(-W \cdot \vec{x} - \vec{b} + N)\Big). \tag{2.10}$$

Out of the 300 samples, at least one method timed out on 56 samples, leaving us with 244 samples whose results were computed over all networks. We note that we limited our evaluation to a single network and only these 244 samples due to the high computational runtime of verifying a sample and the need for verifying robustness for five different quantization levels. Consequently, our results are potentially subject to a high degree of uncertainty.

Table 2.3 depicts how frequently the robustness property could be transferred from the real-valued network to the quantized networks. Not surprisingly, we observed the trend that when increasing the precision of the network, the error between the quantized model and the real-valued model decreases. However, even for the 10-bit model, in 0.8% of the tested samples, verifying the real-valued model leads to a wrong conclusion about the robustness of the quantized network. Moreover, our results show the existence of samples where the 10-bit network is robust while the real-valued is attackable and vice versa. The invalid attacks illustrate that the higher the precision of the quantization, the more targeted attacks need to be. For instance, while 94% of attacks generated for the real-valued network represented valid attacks on the 7-bit model, this percentage decrease to 80% for the 10-bit network.

Next, we compared how well incomplete methods are suited to reason about the robustness of quantized neural networks. We employed IFGSM to attack the 244 test samples for which we obtained the ground-truth robustness and measure how often IFGSM is correct about assessing the robustness of the network. For the sake of completeness, we perform the same analysis for the real-valued network.

Our results in Table 2.4 present the trend that with higher precision, e.g., 10-bits or reals, incomplete methods provide a stable estimate about the robustness of the network, i.e., IFGSM was able to find attacks for all non-robust samples. However, for lower precision levels, IFGSM missed a substantial amount of attacks, i.e., for the 7-bit network, IFGSM could not find a valid attack for 10% of the non-robust samples.

Bits	True negatives	False negatives (i)	False positives (ii)	True positives	Invalid attacks (iii)
6	66.4%	25.0%	3.3%	5.3%	8%
7	84.8%	6.6%	1.6%	7.0%	6%
8	88.5%	2.9%	0.4%	8.2%	10%
9	91.0%	0.4%	0.4%	8.2%	20%
10	91.0%	0.4%	0.4%	8.2%	20%

Table 2.3: Transferability of vulnerability from the verification outcome of the real-valued network to the verification outcome of the quantized model. While vulnerability is transferable between the real-valued and the higher precision networks, (9 and 10-bits), in most of the tested cases, this discrepancy substantially increases when compressing the networks with fewer bits, i.e. see columns (i) and (ii).

Bits	True negatives	False negatives (i)	False positives (ii)	True positives
6	69.7%	1.2 %	-	30.3%
7	86.5%	1.6 %	-	13.5%
8	88.9%	0.8 %	-	11.1%
9	91.4%	0.8 %	-	8.6 %
10	91.4%	0 %	-	8.6 %
$\mathbb{R}$	91.4%	0 %	-	8.6 %

Table 2.4: Transferability of incomplete robustness verification (IFGSM [Zhao et al., 2019]) to ground-truth robustness (ours) for quantized networks. While for the real-valued and 10-bit networks our gradient based incomplete verification did not miss any possible attack, a non-trivial number of vulnerabilities were missed by IFGSM for the low-bit networks. The row indicted by $\mathbb{R}$ compares IFGSM attacking the floating-point implementation to the grouth-truth obtained, using Reluplex, by verifying the real-valued relaxation of the network.

2.4.3 The effect of quantization on robustness

In Table 2.4 we show how standard accuracy and robust accuracy degrade on our MNIST classifier when increasing the compression level. The data indicates a constant discrepancy between standard accuracy and robustness; for real numbered networks, a similar fact was already known in the literature [Tsipras et al., 2018]: we empirically confirm that observation for our quantized networks, whose discrepancy fluctuated between 3 and 4% across all precision levels. Besides, while an acceptable, larger than 90%, standard accuracy was achieved at 7 bits, an equally acceptable robustness was achieved at 9 bits.

One relationship not shown in Table 2.4 is that these 4% of non-robust samples are not equal for across quantization levels. For instance, we observed samples that are robust for 7-bit network but attackable when quantizing with 9- and 10-bits. Conversely, there are attacks for the 7-bit networks that are robust samples in the 8-bit network.

2.4.4 Network specifications beyond robustness

Concerns have been raised that decisions of an ML system could discriminate towards certain groups due to a bias in the training data [Barocas et al., 2017]. A vital issue in quantifying fairness is that neural networks are black-boxes, which makes it hard to explain how each input contributes to a particular decision.

We trained a network on a publicly available synthetically created dataset consisting of 1000 students' personal information and academic test scores [kaggle.com, (accessed December 1, 2021]. The personal features include gender, parental level of education, lunch plans, and whether the student took a preparation course for the test, all of which are discrete variables. We train a predictor for students' math scores, which is a discrete variable between 0 and 100. Notably, the dataset contains a potential source for gender bias: the mean math score among females is 63.63, while it is 68.73 among males. The dataset is randomly split into 100 test and 900 training samples.

The network we trained is composed of 2 hidden layers with 64 and 32 units, respectively. We use a 7-bit quantization-aware training scheme, achieving a 4.14% mean absolute error, i.e., the difference between predicted and actual math scores on the test set.

We define the network as *fair* if the gender of a person influences the predicted math score by at most the bias β. In other words, checking fairness amounts to verifying that

$$\bigwedge_{i \neq \text{gender}} s_i = t_i \wedge s_{\text{gender}} \neq t_{\text{gender}} \implies |f(\vec{s}) - f(\vec{t})| \leq \beta, \tag{2.11}$$

is valid over the variables $\vec{s}$ and $\vec{t}$, which respectively model two students for which gender differs but all other features are identical—we call them twin students. When we encode the dual formula, we encode two copies of the semantics of the same network: to one copy we give one student $\vec{s}$ and take the respective grade g, to the other we give its twin $\vec{t}$ and take grade h; precisely, we check for the unsatisfiability the negation of formula in Eq. 2.11. Then, we compute a tight upper bound for the bias, that is the maximum possible change in predicted score for any two twins. To compute the tightest bias, we progressively increase β until our encoded formula becomes unsatisfiable.

We measure mean test error and gender bias of the 6- to the 10-bits quantization of the networks. We show the results in Table 2.6. The test error was stable between 4.1 and 4.6%

Precision	Standard accuracy	Robust accuracy
6-bit	73.4% ± 2.8	69.7% ± 2.9
7-bit	91.8% ± 1.8	86.5% ± 2.2
8-bit	92.2% ± 1.7	88.9% ± 2.0
9-bit	94.3% ± 1.5	91.4% ± 1.8
10-bit	95.5% ± 1.3	91.4% ± 1.8
$\mathbb{R}$	94.7% ± 1.4	91.4% ± 1.8

Table 2.5: Accuracy of the MNIST classifiers on the 244 test samples for which all quantization levels could be check within the given time budget. The column indicated by $\mathbb{R}$ compares the accuracy of the floating-point implementation to the robust accuracy of the real-valued relaxation of the network. The uncertainty measures indicated after the ± correspond to the standard deviation of the estimated accuracy, i.e., $\frac{\sqrt{p(1-p)}}{\sqrt{244}}$ where p is the observed accuracy.

Quantization level	Mean test error	Tightest bias upper bound
6 bits	4.46 $\pm$ 3.36	22
7 bits	4.14 $\pm$ 3.25	17
8 bits	4.37 $\pm$ 3.45	16
9 bits	4.38 $\pm$ 3.43	15
10 bits	4.59 $\pm$ 3.51	15

Table 2.6: Results for the formal analysis of the gender bias of a students' grade predictor. Mean and standard deviation of the models' errors on the test set. The maximum gender bias of the network monotonically decreases with increasing precision.

among all quantizations, showing that the change in precision did not affect the quality of the network in a way that was perceivable by standard measures. However, our formal analysis confirmed a gender bias in the network, producing twins with a 15 to 21 difference in predicted math score. The bias monotonically increased as the precision level in quantization lowered, indicating that quantization can play a role in determining the bias.

2.5 Conclusion

We introduced the first complete method for the verification of quantized neural networks which, by SMT solving over bit-vectors, accounts for their bit-precise semantics. We demonstrated, both theoretically and experimentally, that bit-precise reasoning is necessary to accurately ensure the robustness to adversarial attacks of a quantized network. We showed that robustness and non-robustness are non-monotonic in the number of bits for the numerical representation and that, consequently, the analysis of high-bits or real-numbered networks may derive false conclusions about their lower-bits quantizations. Experimentally, we confirmed that real-valued solvers produce many spurious results, especially on low-bit quantizations, and that also gradient descent may miss attacks. Additionally, we showed that quantization can affect not only robustness, but also other properties of neural networks, such as some notions of fairness. We also demonstrated that, using our balanced encoding, off-the-shelf SMT-solving can analyze networks with hundreds of neurons which, despite hitting the limits of current solvers, establishes an encouraging baseline for future research.

CHAPTER 3

Scalable Verification of Quantized Neural Networks

There are many efficient methods for verification of neural networks (e.g. [Katz et al., 2017, Tjeng et al., 2019, Bunel et al., 2018]), however most of them ignore rounding errors in computations. The few approaches that can handle the semantics of rounding operations are overapproximation-based methods, i.e., incomplete verification [Singh et al., 2018, 2019]. The imprecision introduced by quantization stands in stark contrast with the idealization made by verification methods for standard neural networks, which disregards rounding errors that appear due to the network's semantics. Consequently, verification methods developed for standard networks are not sound for and cannot be applied to quantized neural networks. Indeed, recently it has been shown that specifications that hold for a floating-point representation of a network need not necessarily hold after quantizing the network [Giacobbe et al., 2020]. As a result, specialized verification methods that take quantization into account need to be developed, due to more complex semantics of quantized neural networks. Groundwork on such methods demonstrated that special encodings of networks in terms of satisfiability modulo theories (SMT) [Clark and Cesare, 2018] with bit-vector [Giacobbe et al., 2020] or fixed-point [Baranowski et al., 2020] theories present a promising approach towards the verification of quantized networks. However, the size of networks that these tools can handle and runtimes of these approaches do not match the efficiency of advanced verification methods developed for standard networks like Reluplex [Katz et al., 2017] and Neurify [Wang et al., 2018a].

In this chapter, we provide first evidence that the verification problem for quantized neural networks is harder compared to verification of their idealized counterparts, thus explaining the scalability-gap between existing methods for standard and quantized network verification. In particular, we show that verifying quantized neural networks with bit-vector specifications is PSPACE-hard, despite the satisfiability problem of formulas in the given specification logic being in NP. As verification of neural networks without quantization is known to be NP-complete [Katz et al., 2017], this implies that the verification of quantized neural networks is a harder problem.

We then address the scalability limitation of SMT-based methods for verification of quantized neural networks, and propose three techniques for their more efficient SMT encoding. First, we introduce a technique for identifying those variables and constraints whose value can be determined in advance, thus decreasing the size of SMT-encodings of networks. Second, we show how to encode variables as bit-vectors of minimal necessary bit-width. This significantly reduces the size of bit-vector encoding of networks in Giacobbe et al. [2020]. Third, we propose a redundancy elimination heuristic which exploits bit-level redundancies occurring in the semantics of the network.

Finally, we propose a new method for the analysis of the quantized network's reachable value range, which is based on abstract interpretation and assists our new techniques for SMT-encoding of quantized networks. We evaluate our approach on two well-studied adversarial robustness verification benchmarks. Our evaluation demonstrates that the combined effect of our techniques is a speed-up of over three orders of magnitude compared to the existing tools.

The rest of this work is organized as follows: First, we provide background and discuss related works on the verification of neural networks and quantized neural networks. We then start with our contribution by showing that the verification problem for quantized neural networks with bit-vector specifications is PSPACE-hard. In the following section, we propose several improvements to the existing SMT-encodings of quantized neural networks. Finally, we present our experimental evaluation to assess the performance impacts of our techniques.

3.1 Background and Related work

A neural network is a function $f : \mathbb{R}^n \to \mathbb{R}^m$ that consists of several layers $f = l_1 \circ l_2 \circ \cdots \circ l_k$ that are sequentially composed, with each layer parameterized by learned weight values. Commonly found types of layers are linear

$$l(x) = Wx + b, W \in \mathbb{R}^{n_o \times n_i}, b \in \mathbb{R}^{n_o}, \tag{3.1}$$

ReLU $l(x) = \max\{x, 0\}$, and convolutional layers [LeCun et al., 1998]. A convolutional layer is a special form of a linear layer in Equation (3.1) where weight values in W are shared between some neurons within the layer and W are zeros except for a fixed local window that correspond to a spatial location in input representation of the layer.

In practice, the function f is implemented by floating-point arithmetic instead of real-valued computations. To distinguish a neural network from its approximation, we define an interpretation $[\![f]\!]$ as a map which assigns a new function to each network, i.e.

$$[\![]\!] : (\mathbb{R}^n \to \mathbb{R}^m) \to (\mathcal{D} \to \mathbb{R}^m), \tag{3.2}$$

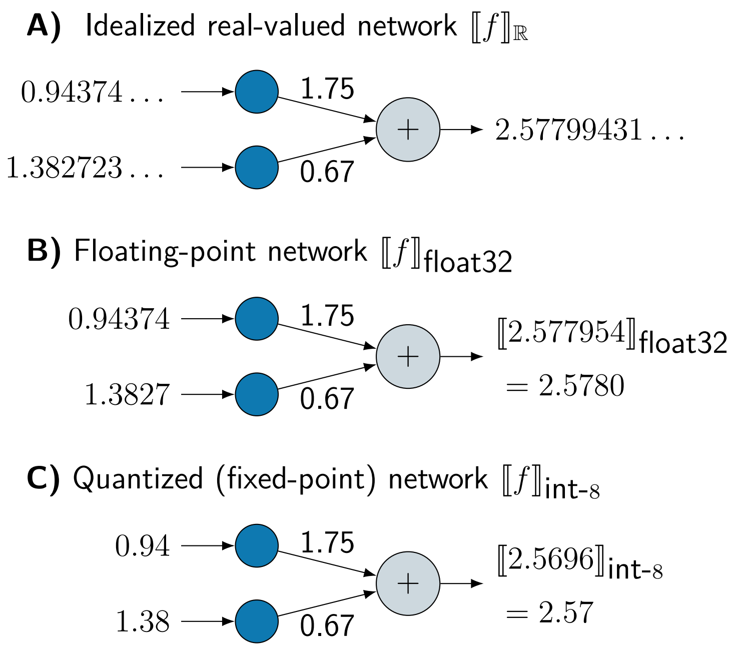
where $\mathcal{D} \subset \mathbb{R}^n$ is the admissible input domain. For instance, we denote by $[\![f]\!]_{\mathbb{R}} : f \mapsto f$ the idealized real-valued abstraction of a network f, whereas $[\![f]\!]_{\text{float32}}$ denotes its floating-point implementation, i.e. the realization of f using 32-bit IEEE floating-point [Kahan, 1996] instead of real arithmetic. Evaluating f, even under floating-point interpretation, can be costly in terms of computations and memory resources. In order to reduce these resource requirements, networks are usually quantized before being deployed to end devices [Jacob et al., 2018].

Formally, quantization is an interpretation $[\![f]\!]_{\text{int-}k}$ that evaluates a network f which uses k-bit fixed-point arithmetic [Smith et al., 1997], e.g. 4 to 8 bits. Let $[\mathbb{Z}]_k = \{0,1\}^k$ denote the set of all bit-vectors of bit-width k. For each layer $l : [\mathbb{Z}]_k^{n_i} \to [\mathbb{Z}]_k^{n_0}$ in $[\![f]\!]_{\text{int-}k}$, we define its semantics by defining $l(x_1, \dots, x_{n_i}) = (y_1, \dots, y_{n_0})$ as follows:

$$x_i' = \sum_{j=1}^{n_i} w_{ij} x_j + b_i, \tag{3.3}$$

$$x_i'' = \text{round}(x_i', k_i) = \lfloor x_i' \cdot 2^{-k_i} \rfloor, \qquad \text{and} \tag{3.4}$$

$$y_i = \max\{0, \min\{2^{N_i} - 1, x_i''\}\}, \tag{3.5}$$



A) Idealized real-valued network $[\![f]\!]_{\mathbb{R}}$
$0.94374\ldots$
$1.382723\ldots$
1.75
0.67
+
$2.57799431\ldots$
B) Floating-point network $[\![f]\!]_{\text{float32}}$
0.94374
1.3827
1.75
0.67
+
$[\![2.577954]\!]_{\text{float32}}$
$= 2.5780$
C) Quantized (fixed-point) network $[\![f]\!]_{\text{int-8}}$
0.94
1.38
1.75
0.67
+
$[\![2.5696]\!]_{\text{int-8}}$
$= 2.57$

Figure 3.1: Illustration of how different interpretations of the same network run with different numerical precision. **A)** $[\![f]\!]_{\mathbb{R}}$ assumes infinite precision. **B)** $[\![f]\!]_{\text{float32}}$ rounds the mantissa according on the IEEE 754 standard. **C)** $[\![f]\!]_{\text{int-8}}$ rounds to a fixed number of digits before and after the comma. (Note that this figure serves as a hypothetical example in decimal format, the actual computations run with the base-2 representation.)

Here, $w_{i,j}$ and b_i for each $1 \leq j \leq n_i$ and $1 \leq i \leq n_0$ denote the learned weights and biases of f, and k_i and N_i denote the bit-shift and the cut-off value associated to each variable y_i, respectively. Eq. (3.3) multiplies the inputs x_j with the weight values w_{ij} and adds the bias b_i, eq. (3.4) rounds the result to the nearest valid k-bit fixed-point value, and eq. (3.5) is a non-linear ReLU-N activation function [1].

An illustration of how the computations inside a network differ based on the used interpretation is shown in Fig. 3.1.

3.1.1 Verification of neural networks

The verification problem for a neural network and its given interpretation consists of verifying some input-output relation. More formally, given a neural network f, its interpretation $[\![f]\!]$ and two predicates φ and ψ over the input domain $\mathcal{D}$ and output domain $\mathbb{R}^m$ of $[\![f]\!]$ respectively, we want to check validity of the following formula (i.e. whether it holds for each $x \in \mathcal{D}$ and $y \in \mathbb{R}^m$)

$$\varphi(x) \wedge [\![f]\!](x) = y \Longrightarrow \psi(y). \tag{3.6}$$

Note that y refers to the outputs of the network f. We refer to the formula in eq. (3.6) as the formal specification that needs to be proved. In order to formally verify a neural network, it is insufficient to just specify the network without also providing a particular interpretation. A property that holds with respect to one interpretation need not necessarily remain true if we consider a different interpretation. For example, robustness of the real-valued abstraction does

[1]Note that for quanitzed neural networks, the double-side bounded ReLU-N activation is preferred over the standard ReLU activation function [Jacob et al., 2018]

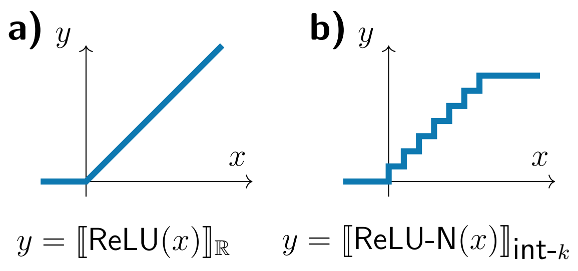


Figure 3.2: Illustration of **a)** the ReLU activation function under real-valued semantics, and **b)** ReLU-N activation under fixed-point semantics (right).

not imply robustness of the floating-point implementation of a network [Giacobbe et al., 2020, Jia and Rinard, 2021].

Ideally, we would like to verify neural networks under the exact semantics that are used for running networks on the end device, i.e., $[\![f]\!]_{\mathsf{float32}}$ most of the time. However, as verification methods for IEEE floating-point arithmetic are extremely inefficient, research has focused on verifying the idealized real-valued abstraction $[\![f]\!]_{\mathbb{R}}$ of f. In particular, efficient methods have been developed for a popular type or networks that only consist of linear and ReLU operations (Figure 3.2 a) [Katz et al., 2017, Ehlers, 2017, Tjeng et al., 2019, Bunel et al., 2018]. The piecewise linearity of such ReLU networks allows the use of Linear Programming (LP) techniques, which make the verification methods more efficient. The underlying verification problem of ReLU networks with linear inequality specifications was shown to be NP-complete in the number of ReLU operations [Katz et al., 2017], however advanced tools scale beyond toy networks.

Although these methods can handle networks of large size, they are building on the assumption that

$$[\![f]\!]_{\mathsf{float32}} \approx [\![f]\!]_{\mathbb{R}}, \tag{3.7}$$

i.e. that the rounding errors introduced by the IEEE floating-point arithmetic of both the network and the verification algorithm can be neglected. It has been recently shown that this need not always be true. For example, Jia and Rinard [Jia and Rinard, 2021] crafted adversarial counterexamples to the floating-point implementation of a neural network whose idealized interpretation was verified to be robust against such attacks, by exploiting subtle numerical differences between $[\![f]\!]_{\mathsf{float32}}$ and $[\![f]\!]_{\mathbb{R}}$.

3.1.2 Verification of quantized neural networks

The low numerical precision of few-bit fixed-point arithmetic implies that $[\![f]\!]_{\mathsf{int}\text{-}k} \neq [\![f]\!]_{\mathbb{R}}$. Indeed, Giacobbe et al. [2020] constructed a prototypical network that either satisfies or violates a formal specification, depending on the numerical precision used to evaluate the network. Moreover, they observed such discrepancy in networks found in practice. Thus, no formal guarantee on $[\![f]\!]_{\mathsf{int}\text{-}k}$ can be obtained by verifying $[\![f]\!]_{\mathbb{R}}$ or $[\![f]\!]_{\mathsf{float32}}$. In order to verify fixed-point implementations of (i.e. quantized) neural networks, new approaches are required.

Fig. 3.2 depicts the ReLU activation function for idealized real-valued ReLU networks and for quantized ReLU networks, respectively. The activation function under fixed-point semantics consists of an exponential number of piecewise constant intervals thus making the LP-based techniques, which otherwise work well for real-valued networks, extremely inefficient. So the approaches developed for idealized real-valued ReLU networks cannot be efficiently applied to quantized networks. Existing verification methods for quantized neural networks are based on bit-exact Boolean satisfiability (SAT) and SMT encodings. For 1-bit networks, i.e., binarized neural networks, Narodytska et al. [2018] and Cheng et al. [2018] proposed to encode the network semantics and the formal specification into an SAT formula, which is then checked by an off-the-shelf SAT solver. While their approach could handle networks of decent size, the use of SAT-solving is limited to binarized networks, which are not very common in practice.

Giacobbe et al. [2020] proposed to verify many-bit quantized neural network by encoding their semantics and specifications into quantifier-free bit-vector SMT (QF_BV) formulas. The authors showed that, by reordering linear summations inside the network, such monolithic bit-vector SMT encodings could scale to the verification of small but interestingly sized networks.

Baranowski et al. [2020] introduced an SMT theory for fixed-point arithmetic and showed that the semantics of quantized neural networks could be encoded in this theory very naturally. However, as the authors only proposed prototype solvers for reference purposes, the size of the verified networks was limited.

3.1.3 Limitations of neural network verification

The existing techniques for verification of idealized real-valued abstractions of neural networks have significantly increased the size of networks that can be verified [Ehlers, 2017, Katz et al., 2017, Bunel et al., 2018, Tjeng et al., 2019]. However, scalability remains the key challenge hindering formal verification of neural networks in practice. For instance, even the largest networks verified by the existing methods [Ruan et al., 2018] are tiny compared to the network architectures used for object detection and image classification [He et al., 2016].

Regarding the verification of quantized neural networks, no advanced techniques aiming at performance improvements have been studied so far. In this chapter, we address the scalability of quantized neural network verification methods that rely on SMT-solving.

3.2 Hardness of Verification of Quantized Neural Networks

The size of quantized neural networks that existing verification methods can handle is significantly smaller compared to the real arithmetic networks that can be verified by the state-of-the-art tools, i.e., compare Giacobbe et al. [2020] with Katz et al. [2017], Tjeng et al. [2019], Bunel et al. [2018]. Thus, a natural question is whether this gap in scalability is only because existing methods for quantized neural networks are less efficient, or if the verification problem for quantized neural networks is computationally harder.

In this section, we study the computational complexity of the verification problem for quantized neural networks. For idealized real arithmetic interpretation of neural networks, it was shown in Katz et al. [2017] that, if predicates on inputs and outputs are given as conjunctions of linear inequalities, then the problem is NP-complete. The fact that the problem is NP-hard is established by reduction from 3-SAT, and the same argument can be used to show that the verification problem for quantized neural networks is also NP-hard. In this work, we argue that the verification problem for quantized neural networks with bit-vector specifications is in fact PSPACE-hard, and thus potentially harder than verifying real arithmetic neural networks.

We also show that our result holds even for the special case when there are no constraints on the inputs of the network, i.e. when the predicate on inputs is assumed to be a tautology. The verification problem for a quantized neural network f that we consider consists of checking validity of a given input-output relation formula

$$[\![f]\!]_{\text{int-}k}(x) = y \Longrightarrow \psi(y).$$

Here, $[\![f]\!]_{\text{int-}k}$ is the k-bit fixed point arithmetic interpretation of f, and ψ is a predicate in some specification logic over the outputs of $[\![f]\!]_{\text{int-}k}$. Equivalently, we may also check satisfiability of the dual formula

$$[\![f]\!]_{\text{int-}k}(x) = y \wedge \neg\psi(y). \tag{3.8}$$

In order to study complexity of the verification problem, we also need to specify the specification logic to which formula ψ belongs. In this work, we study hardness with respect to the fragment $\mathrm{QF\_BV2}_{bw}$ of the fixed-size bit-vector logic $\mathrm{QF\_BV2}$ [Kovásznai et al., 2016]. The fragment $\mathrm{QF\_BV2}_{bw}$ allows bit-wise logical operations (such as bit-wise conjunction, disjunction and negation) and the equality operator. The index 2 in $\mathrm{QF\_BV2}_{bw}$ is used to denote that the constants and bit-widths are given in binary representation. It was shown in Kovásznai et al. [2016] that the satisfiability problem for formulas in $\mathrm{QF\_BV2}_{bw}$ is NP-complete.

Even though $\mathrm{QF\_BV2}_{bw}$ itself allows only bit-vector operations and not linear integer arithmetic, we show that by introducing dummy output variables in $[\![f]\!]_{\text{int-}k}$ we may still encode formal specifications on outputs that are boolean combinations of linear inequalities over network's outputs. Thus, this specification logic is sufficiently expressive to encode formal specifications most often seen in practice. Let $y_1, \dots, y_m$ denote output variables of $[\![f]\!]_{\text{int-}k}$. In order to encode an inequality of the form $a_1y_1 + \cdots + a_my_m + b \geq 0$ into the output specification, we do the following:

- Introduce an additional output neuron $\tilde{y}$ and a directed edge from each output neuron y_i to $\tilde{y}$. Let a_i be the weight of an edge from y_i to $\tilde{y}$, b be the bias term of $\tilde{y}$, $k-1$ be the bit-shift value of $\tilde{y}$, and $N = k$ be the number of bits defining the cut-off value of $\tilde{y}$. Then
$$\tilde{y} = \text{ReLU-N}(\text{round}(2^{-(k-1)}(a_1y_1 + \cdots + a_my_m + b))).$$
Thus, as we work with bit-vectors of bit-width k, $\tilde{y}$ is just the sign bit of $a_1y_1+\cdots+a_sy_s+b$ preceded by zeros.
- As $a_1y_1 + \cdots + a_sy_s + b \geq 0$ holds if and only if the sign bit of $a_1y_1 + \cdots + a_sy_s + b$ is 0, in order to encode the inequality into the output specification it suffices to encode that $\tilde{y} = \mathbf{0}$, which is a formula expressible in $\mathrm{QF\_BV2}_{bw}$.

By doing this for each linear inequality in the specification and since the logical operations are allowed by $\mathrm{QF\_BV2}_{bw}$, it follows that we may use $\mathrm{QF\_BV2}_{bw}$ to encode boolean combinations of linear inequalities over outputs as formal specifications that are to be verified.

Our main result in this section is that, if ψ in eq. (3.8) is assumed to be a formula in $\mathrm{QF\_BV2}_{bw}$, then the verification problem for quantized neural networks is PSPACE-hard. Since checking satisfiability of ψ can be done in non-deterministic polynomial time, this means that the additional hardness really comes from the quantized neural networks.

Theorem 1 (Complexity of verification of QNNs). *If the predicate on outputs is assumed to be a formula in* $\mathrm{QF\_BV2}_{bw}$*, the verification problem for quantized neural networks is* PSPACE*-hard.*

Proof. In order to prove that the verification problem for quantized neural networks is PSPACE-hard, we exhibit a reduction from TQBF which is known to be PSPACE-complete [Arora and Barak, 2009] to the QNN verification problem. TQBF is the problem of deciding whether a quantified boolean formula (QBF) in propositional logic of the form $Q_1x_1.\, Q_2x_2.\, \dots\, Q_nx_n.\, \phi(x_1, x_2, \dots, x_n)$ is true, where each $Q_i \in \{\exists, \forall\}$ and ϕ is a quantifier-free formula in propositional logic over the variables $x_1, \dots, x_n$.

TQBF. Given $\Phi = Q_1x_1.\, Q_2x_2.\, \dots\, Q_nx_n.\, \phi(x_1, x_2, \dots, x_n)$ a QBF formula, for each variable x_i let $u(i)$ be the number of universally quantified variables x_j with $j < i$. Then, Φ is true if it admits a truth table for each existentially quantified variable x_i, where the truth table for x_i specifies a value in $\{0, 1\}$ for each valuation of $u(i)$ universally quantified variables that x_i depends on. Hence, the size of the truth table for x_i is $2^{u(i)}$. In particular, if k is the total number of universally quantified variables, then to show that Φ is true it suffices to find $n - k$ truth tables with each of size at most 2^k.

Ordering of variable valuations. Let U denote the ordered set of all universally quantified variables in the QBF formula, where variables are ordered according to their indices. A *valuation* of U is an assignment in $\{0, 1\}^k$ of each variable in U. As a truth table for each existentially quantified variable x_i is defined with respect to all variable valuations of universally quantified variables on which x_i depends, it will be convenient to fix an ordering $\sqsubseteq$ of all 2^k valuations of U. For two valuations $(y_1, \dots, y_k)$ and $(y'_1, \dots, y'_k)$ in $\{0, 1\}^k$, we say that $(y_1, \dots, y_k) \sqsubseteq (y'_1, \dots, y'_k)$ if either they are equal or there exists an index $1 \leq i \leq k$ such that $y_i < y'_i$ and $y_j = y'_j$ for $j > i$. Equivalently, $(y_1, \dots, y_k) \sqsubseteq (y'_1, \dots, y'_k)$ if and only if

$$\sum_{i=1}^{k} y_i \cdot 2^i \leq \sum_{i=1}^{k} y'_i \cdot 2^i$$

Thus this is the lexicographic ordering on "reflected" valuations, i.e. the largest index having the highest priority. For brevity, we will still refer to it as the *lexicographic ordering*. For an existentially quantified variable x_i, its truth table can therefore also be defined by specifying a value in $\{0, 1\}$ for each of the first $2^{u(i)}$ valuations of U in the lexicographic ordering, since these are the orderings in which we consider all possible valuations of the first $u(i)$ universally quantified variables in U with setting the remaining universally quantified variables to equal 0.

Reduction. We now proceed to the construction of our reduction.
Given $\Phi = Q_1x_1.\, Q_2x_2.\, \dots\, Q_nx_n.\, \phi(x_1, x_2, \dots, x_n)$ a QBF formula, we need to construct

1. a quantized neural network f^Φ, and

2. a predicate ψ^Φ over the outputs of the neural network,

each of polynomial size in the size of Φ, such that Φ is true if and only if the neural networks admits inputs that satisfy the verification problem.

1. *Construction of the quantized neural network.* We construct f^Φ to consist of $n+1$ gadgets $f_1^\Phi, \dots, f_n^\Phi, g^\Phi$. Each gadget can be viewed as a sub-neural network, and all nodes in gadgets are of the bit-width 2^k each. Since bit-widths in quantized neural networks are given in binary representation, specifying bit-width is still of polynomial size. Each gadget f_i^Φ is associated to the variable x_i in the QBF formula. The purpose of each gadget is to produce the output of the following form:

   - If x_i is universally quantified in Φ, then f_i^Φ will return the single output neuron whose value is the constant bit-vector $c_i \in \{0,1\}^{2^k}$ whenever the predicate ψ^Φ is satisfied. For each $1 \leq j \leq 2^k$, the component $c_i[j]$ will be equal to 1 if and only if the value of x_i in the j-th valuation of U (w.r.t. the lexicographic ordering) is equal to 1. Thus, c_i will encode values of x_i in each valuation of U when ordered lexicographically.
   - If x_i is existentially quantified in Φ, then f_i^Φ will return a single output neuron which will encode a truth table for x_i. Recall, x_i depends only on the first $u(i)$ universally quantified variables in Φ, thus its truth table is of size $2^{u(i)}$. As the values of x_i should remain invariant if the values remaining universally quantified variables are changed, we will encode the truth table for x_i by first extracting the first $2^{u(i)}$ bits from the input neuron to encode the truth table itself, and then copying this block of bits $2^{k-u(i)}$ times in order to obtain an output bit-vector of bit-width 2^k.
   - The gadget g^Φ will return the constant bit-vector $\mathbf{1}$ consisting of all 1's (thus written in bold) whenever the predicate ψ^Φ is satisfied. Note, the constant bit-vector whose each bit is 1 is exponential in k and thus exponential in the size of the TQBF problem. Hence, as we will later need to encode $\mathbf{1}$ into the predicate ψ^Φ over outputs of the quantized neural network, we cannot do it directly but use the quantized neural network to construct $\mathbf{1}$.

   We now describe the architecture of the gadgets that can perform the tasks described above:

   - The gadget g^Φ consists only of the input and the output layer. The input layer consists of the single neuron x_g, and three output neurons y_g, y_g' and y_g''. Each edge between the layers has weight 1, bias 0 and the cut-off value 2^k. The bit-shifts of the edges from x_g to y_g, y_g' and y_g'' are 0, 2^k-1 and 1, respectively. The gadget g^Φ is accompanied by the predicate ψ_g^Φ which is satisfied if and only if $x_g = y_g = \mathbf{1}$. Formally, we define ψ_g^Φ via

     $$\psi_g^\Phi := ((y_g' = 1) \land (\neg(y_g'') \lor y_g)).$$

     To prove this, suppose first that ψ_g^Φ is true so we need to show that $x_g = y_g = \mathbf{1}$. Clearly, by our definition of the gadget we have $x_g = y_g$. Furthermore, $y_g' = 1$ implies that the first bit of $x_g = y_g$ is equal to 1. Finally, observe that

     $$\begin{aligned}(\neg(y_g'') \lor y_g) &\equiv (y_g'' \to y_g) \\ &\equiv (0 \to y_g[1]) \land \bigwedge_{j=1}^{2^k-1} (y_g[j] \to y_g[j+1]),\end{aligned}$$

     where the last inequality follows from the fact that y_g'' is obtained by shifting y_g by 1 bit. Here we use $y_g[j]$ to denote the j-th bit in y_g for each $1 \leq j \leq 2^k$, with

$y_g[1]$ denoting the most significant bit in y_g. Then, since we already showed that the first bit of $x_g = y_g$ is equal to 1, a simple induction on j shows that all bits of $x_g = y_g$ are equal to 1, i.e. $x_g = y_g = \mathbf{1}$.
Conversely, suppose that $x_g = y_g = \mathbf{1}$ so we need to show that ψ_g^Φ is true. The formula $\neg(y''_g) \vee y_g$ is trivially true as $y_g = \mathbf{1}$ and $y'_g = 1$ follows since y'_g is obtained by shifting $y_g = \mathbf{1}$ by $2^k - 1$ bits. This concludes the proof that ψ_g^Φ is true if and only if $x_g = y_g = \mathbf{1}$.

- For f_i^Φ corresponding to universally quantified variable x_i, let B_i be the block of bits starting with $2^{u(i)}$ zeros followed by $2^{u(i)}$ ones. The bit-vector c_i should then consist of $2^{k-u(i)-1}$ repetitions of the block B_i. The gadget f_i^Φ will thus consist of two sequentially composed parts. The first part $f_{i,1}^\Phi$ takes any bit-vector of bit-width 2^k as an input, and outputs a bit-vector of the same bit-width which starts with the block B_i followed by zeros. The second part $f_{i,2}^\Phi$ takes the output of the first part as an input, and outputs c_i.
$f_{i,1}^\Phi$ consists of 3 layers: the input layer L_0 with the single input neuron, layer L_1 with two neurons, and output layer L_2 with a single output neuron. The input neuron in L_0 is set to coincide with the output neuron of g^Φ and thus equals $\mathbf{1}$. The cut-off value of each neuron in $f_{i,1}^\Phi$ is $N = 2^k$. The weights of edges from the input neuron in L_0 to neurons in L_1 are set to $w'_{01} = w''_{01} = 1$, the biases $b'_1 = b''_1 = 0$ and the bit-shifts $F'_{01} = 2^{u(i)}$ and $F''_{01} = 2^{2u(i)}$. Hence, the output values of two neurons in L_1 will be the bit-vectors of bit-width 2^k that start with $2^{u(i)}$ (resp. $2^{2u(i)}$) zeros, followed by ones. Finally, the weights of edges from neurons in L_1 to the output neuron in L_2 are set to $w'_{12} = 1$ and $w''_{12} = -1$, the bias $b_2 = 0$ and the bit-shifts $F'_{12} = F''_{12} = 0$. The output value of the neuron in L_2 will thus be a bit-vector of bit-width 2^k which starts with the block of bits B_i and followed by zeros, as desired.
$f_{i,2}^\Phi$ consists of $2(k - u(i) - 1) + 1$ layers, where the input layer coincides with the output layer of $f_{i,1}^\Phi$. Then for each $1 \leq j \leq k - u(i) - 1$, the $2j$-th layer consists of two neurons and the $(2j+1)$-st layer consists of a single neuron. The cut-off value of each neuron is $N = 2^k$. The weights of each edge in $f_{i,2}^\Phi$ is 1 and the bias of each neuron is 0, thus we only need to specify the bit-shifts. For two edges from the neuron in the $(2j-1)$-st layer to neurons in the $(2j)$-th layer we set bit-shifts to be $F'_{2j-1,2j} = 0$ and $F''_{2j-1,2j} = u(i) + j$, respectively. For two edges from neurons in the $2j$-th layer to the neuron in the $(2j+1)$-st layer both bit-shits are set to 0. Given that the input $f_{i,2}^\Phi$ is a bit-vector of bit-width 2^k which starts with the block B_i of length $2^{2u(i)}$ followed by zeros, by simple induction one can show that the output of the neuron in the $(2j+1)$-st layer is a bit-vector of bit-width 2^k which starts with 2^j copies of B_i which are followed by zeros. Hence, the value of the output neuron of $f_{i,2}^\Phi$ will be c_i, as desired.
- For f_i^Φ corresponding to existentially quantified variable x_i, the neural network f_i^Φ will also consist of two sequentially composed parts. The first part $f_{i,1}^\Phi$ takes any bit-vector of bit-width 2^k as an input, and outputs a bit-vector of the same bit-width which starts with the same $2^{u(i)}$ bits as the input bit-vector but which are then followed by zeros. The second part $f_{i,2}^\Phi$ takes the output of the first part as an input, and outputs a bit-vector obtained by copying $2^{k-u(i)}$ times the block of the first $2^{u(i)}$ bits.
$f_{i,1}^\Phi$ consists only of the input and the output layer. The input layer consists of two neurons, one of which coincides with the input neuron x_g of the gadget g^Φ. We

denote the other input neuron by z_i. The output layer consists of two neurons h_i and h_g. The weights of edges from z_i to h_i and from x_g to h_g are set to 1, and weights of edges from z_i to h_g and from x_g to h_i are set to 0. Biases and cut-off values of all edges between the layers are set to 0 and 2^k, respectively. All bit-shifts are also set to 0, with the exception of the edge from x_g to h_g whose bit-shift is set to $2^{u(i)}$. These choices ensure that $h_i = z_i$ and that h_g is a bit-vector that starts with $2^{u(i)}$ 0-bits followed by $2^k - 2^{u(i)}$ 1-bits. The gadget $f^{\Phi}_{i,1}$ is accompanied by the predicate ψ^{Φ}_i defined via

$$\psi^{\Phi}_i := (h_i = (h_i \wedge \neg(h_g)).$$

This choice of ψ^{Φ}_i together with the design of $f^{\Phi}_{i,1}$ enforce that $\psi^{\Phi}_i \wedge \psi^{\Phi}_g$ is satisfied if and only if $x_i = z_i$ and the last $2^k - 2^{u(i)}$ bits of $x_i = z_i$ are equal to 0.

Since the goal of the second part is to just copy $2^{k-u(i)}$ times the block of the first $2^{u(i)}$ bits of the output h_i of $f^{\Phi}_{i,1}$, the second part $f^{\Phi}_{i,2}$ is constructed analogously as in the case of neural networks corresponding to universally quantified variables above.

Recall, constant bit-vectors, bit-widths of bit-vectors as well as the number of bits used for rounding (i.e. bit-shifts) are encoded in binary representation. Thus, each of the values used in the construction of gadgets f^{Φ}_i and g^{Φ} is encoded using at most k bits, and is polynomial in the size of Φ. On the other hand, from our construction one can check that each gadget consists of at most $2k+4$ neurons. Therefore, as there are $n+1$ gadgets the size of all networks combined is $O(k \cdot (2k+4) \cdot (n+1)) = O(n^3)$.

2. *Construction of the output predicate ψ^{Φ}.* Denote by $y_1, \dots, y_n, y_g$ the outputs of $f^{\Phi}_1, \dots, f^{\Phi}_n, g^{\Phi}$, respectively. We define ψ^{Φ} as

$$\psi^{\Phi} := (\phi_{bw}(y_1, \dots, y_n) = y_g) \wedge \psi^{\Phi}_{\text{auxiliary}}, \tag{3.9}$$

where ϕ_{bw} is the quantifier-free formula in $\mathrm{QF\_BV2}_{bw}$ identical to ϕ, with only difference being that the inputs of ϕ_{bw} are bit-vectors of bit-width 2^k instead of boolean variables and logical operations are also defined over bit-vectors. The formula $\psi^{\Phi}_{\text{auxiliary}}$ collects the auxiliary logical predicates that were introduced by our construction of each gadget above, i.e.

$$\psi^{\Phi}_{\text{auxiliary}} := \bigwedge_{i=1}^{n} \psi^{\Phi}_i \wedge \psi^{\Phi}_g.$$

As $y_g = \mathbf{1}$ if and only if Ψ^{Φ}_g is satisfied, the formula ψ^{Φ} is true if and only if the equality $\phi_{bw}(y_1, \dots, y_n) = y_g$ holds for each component. Intuitively, ψ^{Φ} performs bit-wise evaluation of the formula ϕ on each component of bit-vector inputs, and then checks if each output is equal to 1. The size of ψ^{Φ} is thus $O(|\phi_{bw}|) = O(|\phi| \cdot k + 3 \cdot n + 3) = O(|\phi| \cdot n)$, where the additional factor k comes from the fact that inputs of ϕ_{bw} are bit-vectors of bit-width 2^k, and bit-widths are encoded in binary representation.

Note that the above expression for ψ^{Φ} differs from that in the sketch proof of Theorem 1, where we omitted the formula $\psi^{\Phi}_{\text{auxiliary}}$ to simplify the presentation.

Hence, the size of the instance of the quantized neural network verification problem to which we reduced Φ is $O(n^3 + n \cdot |\phi|)$, which is polynomial in the size of Φ.

Correctness of reduction. It remains to prove correctness of our reduction, i.e. that Φ is true if and only if the corresponding quantized neural network verification problem is satisfiable.

Suppose first that Φ is true, i.e. that for each existentially quantified variable x_i in Φ there exists a truth table $\mathbf{t}_i$ of size $2^{u(i)}$, such that any valuation of universally quantified variables U together with the corresponding values of existentially quantified variables defined by truth tables form a satisfying assignment for the quantifier-free formula ϕ in Φ. Consider the following set of inputs $z_1, \dots, z_n, x_g$ to gadgets $f_1^\Phi, \dots, f_n^\Phi, g^\Phi$:

- If x_i is universally quantified, then $z_i = \mathbf{0}$.
- If x_i is existentially quantified, consider $\mathbf{t}_i$ as a bit-vector of bit-width $2^{u(i)}$ with elements ordered in such a way that corresponding valuations of universally quantified variables on which x_i depends in Φ are ordered lexicographically. Then z_i starts with a block of bits identical to $\mathbf{t}_i$, followed by zeros.
- $z_g = 0$.

From our construction of neural networks and the predicate ψ^Φ we know that:

- $g^\Phi(z_g) = \mathbf{1}$.
- If x_i is universally quantified, then $f_i^\Phi(z_i)$ is equal to the bit-vector c_i whose j-th component is equal to 1 if and only if the value of x_i in the j-th valuation of U in the lexicographic ordering is equal to 1, where $1 \leq j \leq 2^k$.
- If x_i is existentially quantified, then $f_i^\Phi(z_i)$ is the bit-vector obtained by copying the block $\mathbf{t}_i$ $2^{k-u(i)}$ times. Thus, the j-th component of $f_i^\Phi(z_i)$ is equal to the value in the truth table $\mathbf{t}_i$ corresponding to the j-th valuation of U in the lexicographic ordering, where $1 \leq j \leq 2^k$.

Finally, as ψ^Φ is obtained by considering a bit-vector version of formula ϕ and then checking if each component of the output is equal to 1, it follows that the output of ψ^Φ on inputs $f_1^\Phi(z_1), \dots, f_n^\Phi(z_n)$ is equal to 1, thus showing that the quantized neural network verification problem is satisfiable.

Conversely, suppose that $z_1, \dots, z_n, z_g$ is a set of satisfying inputs to the quantized neural network verification problems. Then for each existentially quantified variable x_i, we construct a truth table $\mathbf{t}_i$ as follows. Again, consider $\mathbf{t}_i$ as a bit-vector of bit-width $2^{u(i)}$, where elements are ordered in such a way that the corresponding valuations of universally quantified variables on which x_i depends are ordered lexicographically. Then we set $\mathbf{t}_i$ to be equal to the block of first $2^{u(i)}$ bits in z_i. From our construction of the quantized neural network and ψ^Φ, and the fact that $z_1, \dots, z_n, z_g$ is a satisfying inputs to the quantized neural network verification problem, it follows that for any valuation of U the corresponding values of existentially quantified variables defined by these turth tables yield a satisfying assignment for ϕ. Hence, the QBF formula Φ is true, as desired. □

Theorem 1 is to our best knowledge the first theoretical result which indicates that the verification problem for quantized neural networks is harder than verifying their idealized real arithmetic counterparts. It sheds some light on the scalability gap of existing SMT-based methods for their verification, and shows that this gap is not solely due to practical inefficiency of existing methods for quantized neural networks, but also due to the fact that the problem is computationally harder. While Theorem 1 gives a lower bound on the hardness of verifying quantized neural networks, it is easy to see that an upper bound on the complexity of this problem is NEXP since the inputs to the verification problem are of size that is exponential in the size of the problem. Closing the gap and identifying tight complexity bounds is an interesting direction of future work.

A potential blowup of the input is also the reason why the NP-completeness argument of the real-valued verification problem of Katz et al. [2017] does not apply to the QNN verification problem (assuming NP $\neq$ PSPACE). In particular, potential witnesses (x, y), i.e., assignments to the input and output variables that make the formula in Equation 3.8 evaluate to true, can be exponential in the size of the problem description. Consequently, there might not be an efficient, i.e., in polynomial time of the problem description, way to check the witness (again, assuming NP $\neq$ PSPACE).

Note though that the specification logic $\mathrm{QF\_BV2}_{bw}$ used to encode predicates over outputs is strictly more expressive than what we need to express boolean combinations of linear integer inequalities, which is the most common form of formal specifications seen in practice. This is because $\mathrm{QF\_BV2}_{bw}$ also allows logical operations over bit vectors, and not just over single bits. Nevertheless, our result presents the first step towards understanding computational hardness of the quantized neural network verification problem.

3.3 Improvements to Bit-vector SMT-encodings

In this section, we study efficient SMT-encodings of quantized neural networks that would improve scalability of verification methods for them. In particular, we propose three simplifications to the monolithic SMT encoding of eq. (3.3), (3.4), and (3.5) introduced in Giacobbe et al. [2020], which encodes quantized neural networks and formal specifications as formulas in the QF_BV2 logic : I) Remove dead branches of the If-Then-Else encoding of the activation function in eq. (3.5), i.e., branches that are guaranteed to never be taken; II) Allocate only the minimal number of bits for each bit-vector variable in the formula; and III) Eliminate sub-expressions from the summation in eq. (3.3). To obtain the information needed by the techniques I and II we further propose an abstract interpretation framework for quantized neural networks.

3.3.1 Abstract interpretation analysis

Abstract interpretation [Cousot and Cousot, 1977] is a technique for constructing over-approximations to the behavior of a system. Initially developed for software verification, the method has recently been adapted to robustness verification of neural networks and is used to over-approximate the output range of variables in the network. Instead of considering all possible subsets of real numbers, it only considers an abstract domain which consists of subsets of suitable form (e.g. intervals, boxes or polyhedra). This allows modeling each operation in the network in terms of operations over the elements of the abstract domain, thus over-approximating the semantics of the network. While it leads to some impreision, abstract interpretation allows more efficient output range analysis for variables. Due to its over-approximating nature, it remains sound for verifying neural networks.

Interval [Wang et al., 2018b, Tjeng et al., 2019], zonotope [Mirman et al., 2018, Singh et al., 2018], and convex polytope [Katz et al., 2017, Ehlers, 2017, Bunel et al., 2018, Wang et al., 2018a] abstractions have emerged in literature as efficient and yet precise choices for the abstract domains of real-valued neural networks. The obtained abstract domains have been used for output range analysis [Wang et al., 2018b], as well as removing decision points from the search process of complete verification algorithms [Tjeng et al., 2019, Katz et al., 2017]. One important difference between standard and quantized networks is the use of double-sided bounded activation functions in quantized neural networks, i.e., ReLU-N instead of ReLU [Jacob et al., 2018]. This additional non-linear transition, on one hand, renders linear abstractions less effective, while on the other hand it provides hard upper bounds to each neuron, which bounds the over-approximation error. Consequently, we adopt interval arithmetic (IA) on the quantized interpretation of a network to obtain reachability sets for each neuron in the network. As discussed in Tjeng et al. [2019], using a tighter abstract interpretation poses a tradeoff between verification and pre-processing complexity.

3.3.2 Dead branch removal

Suppose that through our abstract interpretation we obtained an interval $[lb, ub]$ for the input x of a ReLU-N operation $y = \text{ReLU-N}(x)$. Then, we can substitute the formulation of the ReLU-N by

$$y = \begin{cases} 0, & \text{if } ub \leq 0 \\ 2^N - 1, & \text{if } lb \geq 2^N - 1 \\ x, & \text{if } ub \geq 0 \text{ and } lb \leq 2^N - 1 \\ \max\{0, x\}, & \text{if } 0 < ub \leq 2^N - 1. \\ \min\{2^N - 1, x\}, & \text{if } 0 \leq lb < 2^N - 1. \\ \max\{0, \min\{2^N - 1, x\}\}, & \text{otherwise,} \end{cases}$$

which reduces the number of decision points in the SMT formula.

3.3.3 Minimum bit allocation

A k-bit quantized neural network represents each neuron and weight variable by a k-bit integer. However, when computing the values of certain types of layers, such as the linear layer in eq. (3.1), a wider register is necessary. The binary multiplication of a k-bit weight and a k-bit neuron value results in a number that is represented by $2k$-bits. Furthermore, summing up n such $2k$-bit integer requires

$$b_{\text{naive}} = 2k + \log_2(n) + 1 \tag{3.10}$$

bits to be safely represented without resulting in an overflow.

Thus, linear combinations are in practice usually computed on 32-bit integer registers. Application of fixed-point rounding and the activation function then reduces the neuron values back to a k-bit representation [Jacob et al., 2018].

QF_BV2 reasons over fixed-size bit-vectors, i.e. the bit width of each variable must be fixed in the formula regardless of the variable's value. Giacobbe et al. [2020] showed that the number of bits used for all weight and neuron variables in the formal affects the runtime of the SMT-solver significantly. For example, omitting the least significant bit of each variable cuts the runtime on average by half. However, the SMT encoding of Giacobbe et al. [2020] allocates b_{naive} bits according to eq. (3.10) for each accumulation variable of a linear layer.

Our approach uses the interval $[lb, ub]$ obtained for each variable by abstract interpretation to compute the minimal number of bits necessary to express any value in the interval. As the signed bit-vector variables are represented in the two's complement format, we can compute the bit width b of variable x with computed interval $[lb, ub]$ by

$$b_{\text{minimal}} = 1 + \log_2(\max\{|lb|, |ub|\} + 1). \tag{3.11}$$

Trivially, one can show that $b_{\text{minimal}} < b_{\text{naive}}$, as $|ub| \leq 2^{2k}n$ and $|lb| \leq 2^{2k}n$.

3.3.4 Redundant multiplication elimination

Another difference between quantized and standard neural networks is the rounding of the weight values to the nearest representable value of the employed fixed-point format. Consequently, there is a considerable chance that two connections outgoing from the same source neuron will have the same weight value. For instance, assuming an 8-bit network and a uniform weight distribution, the chance of two connections having the same weight value is around 0.4% compared to the much lower $4 \cdot 10^{-8}\%$ of the same scenario happening in a floating-point network.

Moreover, many weight values express some subtle form of redundancy on a bit-level. For instance, both multiplication by 2 and multiplication by 6 contain a shift operations by 1 digit in their binary representation. Thus, computations

$$y_1 = 3 \cdot x_1 \qquad\qquad y_2 = 6 \cdot x_1 \tag{3.12}$$

can be rewritten as

$$y_1 = 3 \cdot x_1 \qquad\qquad y_2 = y_1 << 1, \tag{3.13}$$

where $<<$ is a binary shift to the left by 1 digit. As a result, a multiplication by 6 is replaced by a much simpler shift operation. Based on this motivation, we propose a redundancy elimination heuristic to remove redundant and partially redundant multiplications from the SMT formula. Our heuristic first orders all outgoing weights of a neuron in ascending order and then sequentially applies a rule-matching for each weight value. The rules try to find a simpler way to compute the multiplication of the weight and the neuron value by using already performed multiplications. The algorithm aiming to remove bit-level redundancies is shown in Algorithm 3.1. The rules for matching a weight value to the set of existing computations V of a layer is Table 3.1.

Note that a similar idea was introduced by Cheng et al. [2018] in the form of a neuron factoring algorithm for the encoding of binarized (1-bit) neural networks into SAT formulas. However, the heuristic of Cheng et al. [2018] removes redundant additions, whereas we consider bit-level redundancies in multiplications. For many-bit quantization, the probability of two neurons having the same weight coming from more than one input neuron decreases exponentially with the number of bits (assuming the additionally representable quantized weight values by an increase in the number of bits are actually used). Consequently, the inter-neuron factoring of multiple inputs as proposed in Cheng et al. [2018] becomes less effective in many-bit quantized neural networks.

Algorithm 3.1: Multiplication redundancy elimination

```
Input: Outgoing weights $W = \{w_i | i = 1, \dots n\}$ of neuron $x$, with $n$ neurons in the
       next layers
Output: Outgoing values $w_i \cdot x$ of neuron $x$
1  Sort $W$ in ascending order by absolute value;
2  $V \leftarrow \{\}$, $Y \leftarrow \{\}$;
3  foreach $w_i \in W$ do
4      Find rule for $w_i$ according to Table 1 given $V$;
5      if rule found then
6          $Y \leftarrow Y \cup \{rule(w_i, V)\}$;
7      else
8          $y \leftarrow w_i \cdot x$;
9          $V \leftarrow V \cup \{w_i\}$, $Y \leftarrow Y \cup \{y\}$;
10     end
11 end
12 return $Y$;
```

Condition	Action
$w_i = 0$	$y_i = 0$
$w_i = 1$	$y_i = x$
$\exists w_j : w_i = w_j$	$y_i = y_j$
$\exists w_j : w_i = -w_j$	$y_i = -y_j$
$\exists w_j : w_i = w_j \cdot 2^k$	$y_i = y_j << k$

Table 3.1: Rules used for the multiplication redundancy elimination heuristic

3.4 Experimental Evaluation

We create an experimental setup to evaluate how much the proposed techniques affect the runtime and efficiency of the SMT-solver. Our reference baseline is the approach of Giacobbe et al. [2020], which consists of a monolithic and "balanced" bit-vector formulation for the Boolector SMT-solver. We implement our techniques on top of this baseline. We limited our evaluation to Boolector, as other SMT-solvers supporting bit-vector theories, such as Z3 [De Moura and Bjørner, 2008], CVC4 [Barrett et al., 2011], and Yices [Dutertre, 2014], performed much worse in the evaluation of Giacobbe et al. [2020].

Our evaluation comprises of two benchmarks. Our first evaluation considers the adversarial robustness verification of image classifier trained on the MNIST dataset [LeCun et al., 1998]. In particular, we check the l_∞ robustness of networks against adversarial attacks [Szegedy et al., 2013]. Other norms, such as l_1 and l_2, can be expressed in bit-vector SMT constraints as well, although with potentially negative effects on the solver runtime. In the second evaluation, we repeat the experiment on the slightly more complex Fashion-MNIST dataset [Xiao et al., 2017] .

All experiments are run on a 14-core Intel W-2175 CPU with 64GB of memory. We used the boolector [Niemetz et al., 2015] with the SAT-solvers Lingeling [Biere, 2017] (only for the baseline) and CaDiCal [Biere, 2019] (baseline + our improvements) as SAT-backend.

Attack radius	Checked instances	Baseline (+ Lingeling)	Baseline (+ CaDiCal)	Ours
$\varepsilon = 1$	99	63 ± 4.8 (63.6% ± 4.8)	92 ± 2.6 (92.9% ± 2.6)	**99** ± 0.0 (**100.0%** ± 0.0)
$\varepsilon = 2$	99	0 ± 0.0 (0.0% ± 0.0)	20 ± 4.0 (20.2% ± 4.0)	**94** ± 2.2 (**94.9%** ± 2.2)
$\varepsilon = 3$	96	0 ± 0.0 (0.0% ± 0.0)	2 ± 1.4 (2.1% ± 1.5)	**71** ± 4.3 (**74.0%** ± 4.5)
$\varepsilon = 4$	97	0 ± 0.0 (0.0% ± 0.0)	1 ± 1.0 (1.0% ± 1.0)	**54** ± 4.9 (**55.7%** ± 5.0)

Table 3.2: Number of solved instances of adversarial robustness verification on the MNIST dataset. Absolute numbers and in percentages of checked instances in parenthesis. Best method in bold. The number of checked instances correspond to the correctly classified samples out of the first 400 test samples. The uncertainty measures indicated after the ± correspond to the standard deviation of the estimated fraction of solved instances, i.e., $\frac{\sqrt{p(1-p)}}{\sqrt{n}}$ where p is the observed fraction of solved instances and n the number of checked instances.

Method	MNIST		Fashion-MNIST	
	Mean runtime	Median runtime	Mean runtime	Median runtime
Baseline (+Lingeling)	146.5 ± 20.0	146.7 (123.1, 173.5)	115.5 ± 2.6	115.5 (113.4, 117.5)
Baseline (+ CaDiCal)	65.5 ± 43.5	46.6 (29.2, 142.2)	57.9 ± 26.1	51.8 (31.8, 99.3)
Ours	**1.5** ± 9.7	**0.1** (0.1, 0.1)	**0.8** ± 5.7	**0.1** (0.1, 0.1)

Table 3.3: Mean and median runtime of adversarial robustness verification process per sample in minutes. The columns showing the mean also report the standard deviation after the ±. The columns showing the median runtimes report the 0.1 and 0.9 quantile in parenthesis. The reported values correspond to the non-timed-out samples out of 391 (for MNIST) and 264 (for Fashion-MNIST) checked instances.

Adversarial robustness specification can be expressed as

$$|x - x_i|_\infty \leq \varepsilon \wedge y = [\![f]\!]_{\text{int-}k}(x) \implies y = y_i, \tag{3.14}$$

where (x_i, y_i) is a human labeled test sample and ε is a fixed attack radius. As shown in eq. (3.14), the space of possible attacks increases with ε. Consequently, we evaluate with different attack radii ε and study the runtimes individually. In particular, for MNIST we check the first 100 test samples with an attack radius of $\varepsilon = 1$, the next 100 test samples with $\varepsilon = 2$, and the next 200 test samples with $\varepsilon = 3$ and $\varepsilon = 4$ respectively. For our Fashion-MNIST evaluation, we reduce the number of samples to 50 per attack radius value for $\varepsilon > 2$ due to time and compute limitations.

The network studied in our benchmark consists of four fully-connected layers (784,64,32,10), resulting in 52,650 parameters in total. It was trained using a quantization-aware training scheme with a 6-bit quantization.

The results for the MNIST evaluation in terms of solved instances and solver runtime are shown in Table 3.2 and Table 3.3 respectively. Table 3.4 and Table 3.3 show the results for the Fashion-MNIST benchmark.

3.4.1 Ablation analysis

We perform an ablation analysis where we re-run our robustness evaluation with one of our proposed techniques disabled. The objective of our ablation analysis is to understand how the

Attack radius	Checked instances	Baseline (+ Lingeling)	Baseline (+ CaDiCal)	Ours
$\varepsilon = 1$	87	2 ± 1.4 (2.3% ± 1.6)	44 ± 4.7 (50.6% ± 5.4)	**76** ± 3.1 (**87.4%** ± 3.6)
$\varepsilon = 2$	90	0 ± 0.0 (0.0% ± 0.0)	7 ± 2.5 (7.8% ± 2.8)	**73** ± 3.7 (**81.1%** ± 4.1)
$\varepsilon = 3$	43	0 ± 0.0 (0.0% ± 0.0)	1 ± 1.0 (2.3% ± 2.3)	**27** ± 3.2 (**62.8%** ± 7.4)
$\varepsilon = 4$	44	0 ± 0.0 (0.0% ± 0.0)	0 ± 0.0 (0.0% ± 0.0)	**18** ± 3.3 (**40.9%** ± 7.4)

Table 3.4: Number of solved instances of adversarial robustness verification on the Fashion-MNIST dataset. Absolute numbers and in percentages of checked instances in parenthesis. Best method in bold. The number of checked instances correspond to the correctly classified samples out of the first 300 test samples. The uncertainty measures indicated after the ± correspond to the standard deviation of the estimated fraction of solved instances, i.e., $\frac{\sqrt{p(1-p)}}{\sqrt{n}}$ where p is the observed fraction of solved instances and n the number of checked instances.

Method	Total solved instances	Mean runtime (minutes)	Cumulative runtime
No redundancy eliminiation	316 ± 7.8 (80.8% ± 2.0)	1.5 ± 13.0	7.7 h
No minimum bitwidth	315 ± 7.8 (80.6% ± 2.0)	**1.0** ± 9.8	**5.1 h**
No ReLU simplify	88 ± 8.3 (22.5% ± 2.1)	56.8 ± 29.9	83.2 h
No Abstract interpretation	107 ± 8.8 (27.4% ± 2.3)	70.7 ± 45.1	126.0 h
All enabled	**318** ± 7.7 (**81.3%** ± 2.0)	1.5 ± 9.7	7.9 h

Table 3.5: Results of our ablation analysis on the MNIST dataset ($n = 391$). The uncertainty measures indicated after the ± correspond to the standard deviation of the estimated quantities. The reported runtimes only account for non-timed-out samples.

individual techniques affect the observed efficiency gains. Due to time and computational limitations we focus our ablation experiments to MNIST exclusively.

The results in Table 3.5 show the highest number of solved instances were achieved when all our techniques were enabled. Nonetheless, Table 3.5 demonstrate these gains are not equally distributed across the three techniques. In particular, the ReLU simplification has a much higher contribution for explaining the gains compared to the redundancy elimination and minimum bitwidth methods. The limited benefits observed for these two techniques may be explain by the inner workings of the Boolector SMT-solver.

The Boolector SMT-solver [Niemetz et al., 2015] is based on a portfolio approach which sequentially applies several different heuristics to find a satisfying assignment of the input formula [Wintersteiger et al., 2009]. In particular, Boolector starts with fast but incomplete local search heuristics and falls back to slower but complete bit-blasting [Clark and Cesare, 2018] in case the incomplete search is unsuccessful [Niemetz et al., 2019]. Although our redundancy elimination and minimum bitwidth techniques simplify the bit-blasted representation of the encoding, it introduces additional dependencies between different bit-vector variables. As a result, we believe these extra dependencies make the local search heuristics of Boolector less effective and thus enabling only limited performance improvements.

3.5 Conclusion

We show that the problem of verifying quantized neural networks with bit-vector specifications on the inputs and outputs of the network is PSPACE-hard. We tackle this challenging problem by proposing three techniques to make the SMT-based verification of quantized networks more efficient. Our experiments show that our method outperforms existing tools by several orders of magnitude on adversarial robustness verification instances. Future work is necessary to explore quantized neural network verification's complexity with respect to different specification logics. On the practical side, our methods point to limitations of monolithic SMT-encodings for quantized neural network verification and suggest that future improvements may be obtained by integrating the encoding and the solver steps more tightly.

CHAPTER 4

Case Study: Adversarial Training for Robot Learning

In this chapter, we study the suitability of adversarial training methods as an ad-hoc replacement of standard empirical risk minimization (ERM) in robot learning applications. Adversarial training robustifies the learned model against visually imperceptible perturbations changing the model's decision. This process trades nominal performance gained by standard learning techniques, with worst-case performance under norm-bounded input perturbations [Kurakin et al., 2017, Madry et al., 2018, Xie et al., 2019].

Adversarial training has been mainly studied in static image classification [Biggio et al., 2013, Szegedy et al., 2013, Goodfellow et al., 2014b, Carlini and Wagner, 2017a, Stutz et al., 2019, Salman et al., 2020, Sietzen et al., 2021], which exclusively focused on how much it trades nominal for robust test accuracy. However, whether this trade is justified in practice and how much the gained robustness is reflected in real-world benefits remains unclear.

Here, we set out to investigate this tradeoff for robotic control tasks. These tasks are characterized by an open-loop supervised training, followed by a closed-loop deployment. Consequently, pure accuracy might not accurately reflect a robotic system's underlying performance. Moreover, as robots interact with real-world environments, there are additional requirements on the safety and robustness of the controller, e.g., stability of closed control loop. For example, Figure 4.1 shows an image perceived by a robot and the image overlayed by an adversarial attack mask of various magnitude. From a pure safety point of view, we may require image processing networks deployed in safety-critical applications to be robust regarding manipulating each pixel by up to 3%, i.e., an infinity-norm perturbation of 8 in Figure 4.1, as such a change would not impair the decision of a human controlling the system.

Another aspect missing in existing works on the robust-accuracy tradeoff is how the drop in nominal performance is distributed across the data domain. In particular, accuracy measures that a sample is misclassified but not how "bad" the sample is misclassified. Consider an image of a car correctly classified by a model trained with standard training but misclassified when learning the classifier with adversarial training. When using the classifier in a robot application, how the image is misclassified matters. For instance, it might be tolerable if the adversarially trained model classifies the image of the car by an adjacent class, e.g., a truck, but unacceptable if the image is classified as a pedestrian.

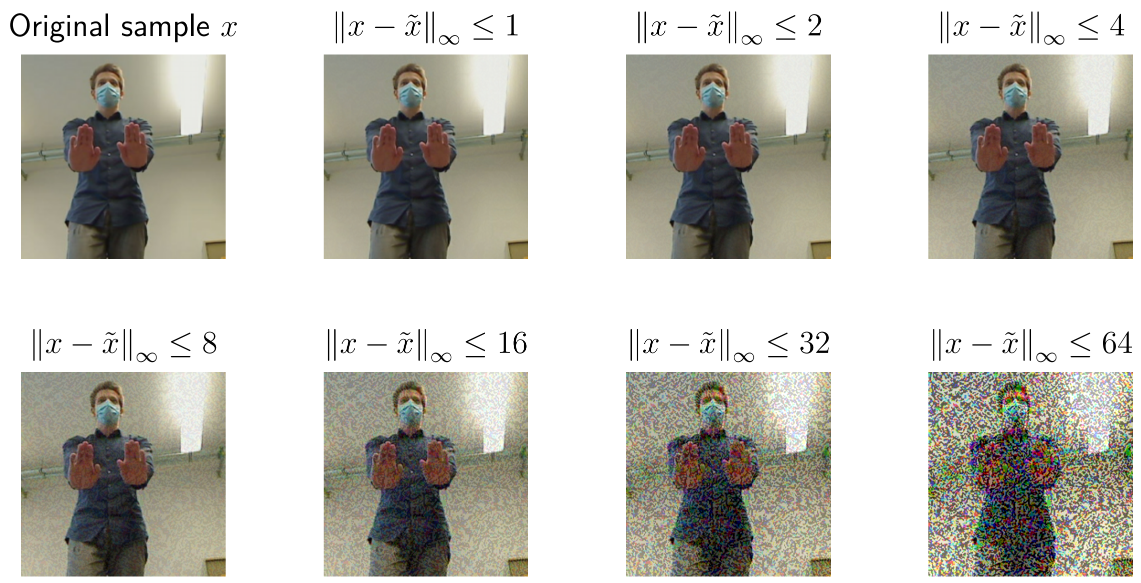

Original sample x
$\|x - \tilde{x}\|_\infty \leq 1$
$\|x - \tilde{x}\|_\infty \leq 2$
$\|x - \tilde{x}\|_\infty \leq 4$
$\|x - \tilde{x}\|_\infty \leq 8$
$\|x - \tilde{x}\|_\infty \leq 16$
$\|x - \tilde{x}\|_\infty \leq 32$
$\|x - \tilde{x}\|_\infty \leq 64$

Figure 4.1: Visualization of a sample image x and corresponding images $\tilde{x}$ attacked by FGSM [Goodfellow et al., 2014b] and different perturbation norms. Attacks with a norm less than or equal to 2 are visually indistinguishable from the original sample. Attacks with a norm of 4 and 8 appear as low magnitude noise. From a safety requirement perspective, if a human's decision is not affected by such low magnitude perturbations, why should we trust a network whose decision will change?

This chapter first proposes safety-domain training, a generalization of adversarial training, which allows us to incorporate more general forms of safety specifications as secondary training objectives. We then formalize a framework for characterizing error behaviors of learned controllers for robotic tasks. We then carry out a case study examining the error profiles introduced by adversarial and safety-domain training on three robot learning tasks. Finally, we test whether recent advances in robust training methods and theory can help overcome the robustness-accuracy tradeoff in practice.

Our results are negative in the sense that replacing standard ERM with adversarial ERM does not appear as a fair tradeoff but a net loss. More precisely, our experiments suggest that models trained by standard empirical risk minimization yield the best robotic performance in real-world scenarios. Counterintuitively, the best-performing agents are also vulnerable to making the robot crash under adversarial patterns. Conversely, while the models learned via our safety-domain training are guaranteed never to crash, they significantly perform worse in real-world scenarios. In particular, we observed that the strictness of the specifications enforced using adversarial or safety-domain training is a dominant factor in determining the expected real-world robotic performance. Finally, our results indicate that while advances in robust learning provide incremental relative improvements on the tradeoff, the negative side-effects caused by adversarial training still outweigh the improvements by an order of magnitude. Our empirical evaluations suggest that adversarial training of neural controllers requires rethinking before reliably using it as an ad-hoc replacement of standard ERM in robot learning schemes.

4.1 Background

A neural network is a function $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$ parameterized by θ. In supervised learning, the training objective is to fit the function to a given dataset in the form of $\{(x_1, y_1), \dots (x_n, y_n)\}$ assumed to be i.i.d. sampled from a probability distribution over $\mathcal{X} \times \mathcal{Y}$. This fitting process is done via empirical risk minimization (ERM) that minimizes

$$\frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(f_\theta(x_i), y_i) \tag{4.1}$$

via stochastic gradient descent. The differentiable loss function $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$ characterizes how well the network's prediction $f_\theta(x_i)$ matches the ground truth label y_i.

An adversarial attack is a sample (x_i, y_i) from the data distribution and a corresponding attack vector μ with $\|\mu\| \leq \varepsilon$ such that $f(x_i) \neq f(x_i + \mu)$ with ε being a thresholds. As illustrated in Figure 4.1, for image data small thresholds δ are usually not recognizable or appear as noise for human observers. Typical norms used in adversarial attacks are the ℓ_1, ℓ_2, and the ℓ_∞ norm. In this work, we focus on the ℓ_∞ norm. A network is robust on a given sample if no such attack μ exists within a neighborhood ε. The robust accuracy is the standard metric for measuring the robustness of a network aggregated over an entire dataset $\{(x_1, y_1), \dots (x_n, y_n)\}$ by counting the ratio of correctly classified samples that are also robust.

In practice, deciding whether a network is robust for a sample is an NP-hard problem [Katz et al., 2017] and, therefore, cannot be computed for typically sized networks in a reasonable time. Instead, the robustness of networks is often studied with respect to empirical gradient and black-box-based attack methods. The fast gradient sign method (FGSM) [Goodfellow et al., 2014b] computes an attack by

$$\mu = \varepsilon \ \text{sign}\Big(\frac{\partial \mathcal{L}(f_\theta(x_i), y_i)}{\partial x_i}\Big). \tag{4.2}$$

Despite its simplicity, adversarial training often uses the FGSM method due to its speed. The iterative fast gradient sign method (I-FGSM) [Kurakin et al., 2017] is a more sophisticated generalization of the FSGM. It computes an attack iteratively in k steps starting from $\mu_0 = \mathbf{0}$ and updating it by

$$\mu_i = \frac{\varepsilon}{k} \ \text{sign}\Big(\frac{\partial \mathcal{L}(f_\theta(x_i + \mu_{i-1}), y_i)}{\partial x_i}\Big). \tag{4.3}$$

DeepFool [Moosavi-Dezfooli et al., 2016], the C&W method [Carlini and Wagner, 2017b], and projected gradient descent [Madry et al., 2018] are other common iterative attack methods that are used for evaluating robustness but are too computationally expensive to incorporate in adversarial training. DeepFool [Moosavi-Dezfooli et al., 2016] linearizes the network in each iteration of updating μ_i. Projected gradient descent [Madry et al., 2018] applies unconstraint gradient descent but divides each μ_i by its norm and multiplies the results with ε to project it back into the given threshold. The C&W method [Carlini and Wagner, 2017b] avoids such projection by parametrizing the attack vector μ by another variable and a transformation that already normalizes the attack to stay within a given threshold. Any empirical attack method can be further enhanced by restarting the method several times from slightly perturbed samples via random noise. Experimentally results suggest that any network of non-trivial size is, at least in parts, vulnerable to such attacks [Madry et al., 2018].

Robust learning methods aim to train networks that are robust. One of the most common robust learning methods is adversarial training which changes the standard ERM objective to minimize

$$\frac{1}{n}\sum_{i=1}^{n}\max_{\mu:\|\mu\|\leq\varepsilon}\mathcal{L}(f_\theta(x_i+\mu),y_i), \tag{4.4}$$

where $\varepsilon > 0$ is some attack budget controlling how much each input can be perturbed. Due to the computation overhead by this training objective, fast attack generating methods are typically used for computing the $\max$ in Equation 4.4, e.g., the FGSM or I-FGSM.

Alternative approaches to adversarial training make minor modifications to the objective term in Equation 4.4. For instance, the TRADES algorithm [Zhang et al., 2019] replaces the label y_i in Equation 4.4 with the network's prediction of the original input, i.e., $f_\theta(x_i)$, and optimizes a joint objective of the standard ERM term and the robustness term. The approach of Shafahi et al. [2019a] removes the overhead imposed by the maximization step in Equation 4.4 by pre-computing μ in the previous gradient descent step. Although, such pre-computed μ can become inaccurate, i.e., stale, Shafahi et al. [2019a] showed that it improves robustness in practice.

The major limitation of adversarial training methods is that they negatively affect the network's standard accuracy (or other performance metrics). For example, medium-sized networks achieve an accuracy of 96% on the CIFAR-10 dataset when trained with standard ERM [Zagoruyko and Komodakis, 2016]. However, in Zhang et al. [2019] the best performing network trained with the TRADES algorithm could only achieve a standard accuracy of 89% on this dataset. This phenomenon of an antagonistic relation between accuracy and robustness was first studied in Tsipras et al. [2018] and is known as the accuracy-robustness tradeoff.

4.2 Related Works

Adversarial Training. Adversarial training has led to significant improvements of deep models' resiliency to imperceptible perturbations. This was shown both empirically [Madry et al., 2018, Miyato et al., 2018, Balaji et al., 2019, Zhang et al., 2019] and with certification [Lecuyer et al., 2019, Weng et al., 2018, Wong and Kolter, 2018, Raghunathan et al., 2018, Cohen et al., 2019, Salman et al., 2019, Yang et al., 2020]. An emerging line of work suggests that the representations learned by adversarially trained models resemble visual features as perceived by humans more accurately compared to standard networks [Ilyas et al., 2019, Engstrom et al., 2019, Santurkar et al., 2019, Allen-Zhu and Li, 2022, Kim et al., 2019, Kaur et al., 2019]. In contrast, a large body of work characterized the tradeoff between a model's robustness and accuracy when trained by adversarial training [Tsipras et al., 2018, Bubeck et al., 2019, Su et al., 2018, Raghunathan et al., 2019]. Some issues such as gradient obfuscation [Athalye et al., 2018, Uesato et al., 2018], during training, seemed to play a role in the mediocre performance of the models. Nevertheless, adversarially trained networks also showed to maintain their robustness properties [Shafahi et al., 2019b] as well as their accuracy [Utrera et al., 2021, Salman et al., 2020] in transfer learning settings.

This work shows that despite the vast success of adversarially trained models in obtaining robustness properties on image classification datasets, they can introduce novel error profiles in robot learning tasks. Our work aims to identify and report these profiles to guide future research directions on robust learning methods that provide robustness benefits in practical applications.

Robustness requires over-parametrization. Theoretical contributions to the robustness-accuracy tradeoff recently discovered that overparametrization is necessary for smoothly fitting the training data [Bubeck et al., 2021, Bubeck and Sellke, 2021]. While empirical results already suggested that the accuracy of larger models suffers less from adversarial training than for small models, the critical insight is that such large models are necessary. In particular, the authors proved that for a dataset of n samples with d-dimensional features, a model with n parameters can fit the training samples but cannot smoothly interpolate between them. Moreover, the authors show that a model needs at least nd parameters to fit the training data and interpolate them smoothly. The authors also demonstrated that contemporary models for standard datasets do not contain enough parameters with respect to their proven results. Consequently, the theory of Bubeck and Sellke [2021] suggests that adversarial training of a non-over-parametrized network must introduce errors somewhere in the data domain. Our work aims to study *where* these errors may occur.

Attention-based architectures might be more robust than convolutional networks. The vision transformer (ViT) [Dosovitskiy et al., 2020] is a powerful machine learning architecture that represents an image as a sequence of patches and processes this sequence using a self-attention mechanism [Vaswani et al., 2017]. Detailed experimental comparisons between vision transformer and convolutional neural networks suggest that ViTs are naturally more robust with respect to object occlusions and distributions shifts [Naseer et al., 2021]. Concurrent work on comparing ViTs to CNNs with respect to adversarial attacks has found that vision transformers seem to be naturally more robust to adversarial attacks as well [Bai et al., 2021].

Hyperparameters affecting robustness. Recent work suggests that the common ReLU activation function, i.e., $\max\{0, x\}$, is not well suited for adversarial training methods [Singla et al., 2021]. Instead, the authors observed that activation functions with smooth curvatures provide better robustness at roughly the same standard accuracy. Specifically, the sigmoid-weighted linear unit (SiLU) activation function [Elfwing et al., 2018], i.e., $x \cdot \frac{1}{1+\exp(-x)}$, was highlighted as having a smooth second derivative and observed to improve robustness compared to alternative activations. We note that the SiLU activation was concurrently proposed as swish activation function in Ramachandran et al. [2017].

The work of Pang et al. [2021] investigated how hyperparameters of the learning process affect adversarial training compared to standard ERM. For example, the authors experiment with learning rate schedules, early stopping, and batch size, among other settings. The authors observed that adversarial training benefits from a higher weight decay factor than standard training. Moreover, the authors confirmed that a smooth activation function improves robustness over the ReLU activation.

Adversarial Training for Safe Robot Learning. Related approaches can be grouped into three categories; i) adversarial learning as a data augmentation technique; ii) Hand-crafted perturbation distribution; and iii) Task-specific models.

(i) **Adversarial learning as a data augmentation technique** – A couple of recent works characterized generative adversarial networks (GAN) [Goodfellow et al., 2014a] as a data augmentation method to enhance neural controllers' transferability. For example, Chen et al. [2020] used GAN-based training for robotic visuomotor control, and Porav et al. [2018] explored GANs to determine robust metric localization by using appearance transfer (e.g., day to night transformation of input images). These methods fundamentally differ from safety-related adversarial training frameworks [Szegedy et al., 2013] that we explore in this chapter, as they refer to methods useful for data augmentation.

(ii) **Hand-crafted perturbation distribution** – invariant sets, i.e., hand-crafted changes in underlying data distribution such as change of a gripper's appearance and objects' color used in task-relevant adversarial imitation learning [Zolna et al., 2021]. These approaches require a simulator capable of generating domain-specific perturbations and are mainly designed for training performant agents. Our work discusses a more general setting where we do not require domain-specific attributes.

(iii) **Task-specific models** – Adversarial training in task-specific domains such as motion planning [Janson et al., 2018, Shi et al., 2020, Innes and Ramamoorthy, 2020] and localization [Yang and Huang, 2020] has been used for enhancing robustness. Moreover, in reinforcement learning (RL) environments, adversarial training benefited agents in competitive scenarios such as active perception [Shen and How, 2019], interaction-aware multi-agent tracking, and behavior prediction [Li et al., 2019] and identifying weaknesses of a learned policy [Pan et al., 2019, Kuutti et al., 2020]. These works do not evaluate existing general methods but propose tailored solutions for the specific task under-test. Our work focuses on the broad vision-based robot learning problems that use contemporary adversarial training for enhancing robustness.

4.3 Safety-domain Training

This section defines a generalization of adversarial training by relaxing the ε-neighborhood for arbitrary domains. We call this approach *Safety-Domain Training*. We then explain how to solve the inner optimization loop of safety-domain training, either by empirical or certified safety methods, and illustrate the resulting method in Algorithm 4.1.

We generalize adversarial training to a more generic safety-domain training. In particular, we replace the ε-neighborhoods of the training samples with arbitrary domains, i.e., labeled sets.

Definition 1 (Safety-domain training)**.** *Let f_θ be a neural network, $\{(x_i, y_i)|i = 1, 2, \dots n\}$ the training samples, $\mathcal{L}$ the loss function, and $\{(D_i, z_i)|i = 1, 2, \dots k\}$ the safety domains. Then safety-domain training optimizes the criterion*

$$\min_\theta \Big[\frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(y_i, f_\theta(x_i)) + \lambda \frac{1}{k}\sum_{i=1}^{k} \max_{\tilde{x}\in D_i} \mathcal{L}(z_i, f_\theta(\tilde{x}))\Big], \tag{4.5}$$

where the hyperparameter λ specifies the tradeoff between optimizing the empirical training risk and the worst-case risk on the safety-domains.

Safety-domain training generalizes adversarial training by defining $D_i := \{\tilde{x} : ||\tilde{x} - x_i|| < \varepsilon\}$ with $z_i := y_i$ and the training samples $\{\}$.

Empirical vs. certified safety In practice, we have two options for solving the inner maximization step of safety-domain training. The first option is to apply gradient descent-based optimization methods. While this approach is computationally efficient and straightforward to implement, it provides no true worst-case guarantees as SGD does not ensure convergence to the global optimum. In practice, empirical approaches are often used for adversarial training of classifiers to account for computational complexity.

A more rigorous approach, albeit expensive, is to compute an upper bound of the loss of each safety domain and minimize the upper bound via stochastic gradient descent [Lecuyer et al., 2019]. While computing an upper bound of a network's output is difficult and may overestimate the true maximum, it guarantees the worst-case loss. Abstract interpretation methods fall into the category in which the upper bound of the loss is computed by using abstract domains instead of individual points [Gowal et al., 2019, Huang et al., 2019]. The main difficulty of such certified methods is to scale the training to large networks. To obtain formal safety or robustness guarantees with this approach is to continue the training process until the upper bound of the loss over all safety domains is below a certain threshold as outlined in Algorithm 4.1.

4.4 Error Profiles in Robot Learning

Limited or stochastic training data, noise in the learning process, and inadequate causal modeling [Schölkopf, 2019] prevent the network from achieving a perfect mapping of the ground truth dependency between x and y. Consequently, these imperfections lead to errors during test time. Moreover, adversarial training methods have been shown to introduce additional errors to the model [Tsipras et al., 2018] in practice. Here, we propose to characterize these errors made by a neural controller by three categories: Systematic errors, transient errors, and conditional errors. Our objective is to analyze which error profiles occur when the neural controller is trained by safety-domain or adversarial training methods. We define these error types formally with respect to data that follows a functional relation, i.e., a single "perfect" decision. Particularly,

Assumption 1. *We assume the ground-truth data $\mathbb{D} \subset \mathcal{X} \times \mathcal{Y}$ has finite cardinality and follows a functional relation $\exists g\colon (x,y) \in \mathbb{D} \implies y = g(x)$.*

We note that our assumption of $\mathbb{D}$ being finite reflects sensor values and motor commands having finite precision.

First, we define transient errors as single points and corresponding neighborhoods with a higher loss than a broader neighborhood. Formally,

Algorithm 4.1: Safety-domain training with guarantees

```
Input: Training data $\{(x_i, y_i)|i = 1 \dots n\}$, Safety domains $\{(z_i, D_i)|i = 1 \dots k\}$
Parameters: safety threshold $\delta$, batch sizes $b_t, b_s$
    Learning rate $\alpha$, minimum training epochs $i_{\text{min}}$.
safety_bound = $\infty$
while $i < i_{\text{min}}$ and safety_bound $> \delta$ do
  i = i+1
  $(\tilde{x}, \tilde{y})$ = sample_batch($b_t, \{(x_i, y_i)|i = 1 \dots n\}$)
  $(\tilde{z}, \tilde{D})$ = sample_batch($b_s, \{(z_i, D_i)|i = 1 \dots k\}$)
  $\nabla = \frac{\partial}{\partial \theta} \frac{1}{b_t} \sum_{i=1}^{b_t} \mathcal{L}(\tilde{y}_i, f_\theta(\tilde{x}_i)$
  $\nabla = \nabla + \frac{\partial}{\partial \theta} \lambda \frac{1}{b_s} \sum_{i=1}^{b_s} \max_{x \in \tilde{D}_i} \mathcal{L}(\tilde{z}_i, f_\theta(x))$
  $\theta = \theta - \alpha \nabla$
  safety_bound = $\max_{i=1 \dots k} \max_{x \in D_i} \mathcal{L}(z_i, f_\theta(x))$
end while
return $\theta$
```

Definition 2 (Transient error). *Given a loss function $\mathcal{L}$, a neural network f_θ, a thresholds $\eta > 0$, two neighborhoods $\varepsilon_1 > \varepsilon_2 > 0$, and assume the , we call a point $(x', y') \in \mathbb{D}$ transient error if*

$$\mathcal{L}(y', f_\theta(x')) > \eta \text{ and } \mathcal{L}(\tilde{y}, f_\theta(\tilde{x})) < \eta, \tag{4.6}$$

for all $(\tilde{x}, \tilde{y}) \in \mathbb{D}$ with $\varepsilon_1 \geq ||\tilde{x} - x'|| \geq \varepsilon_2$.

This type of error characterizes sharp spikes in the loss landscape as illustrated in Figure 4.2 schematically for a single-dimensional regression problem. In terms of robot performance, if the network's output is aggregated or filtered over time, e.g., by electrical filters or dampening physical components, transient errors may be tolerable. Contrarily, if the network's decision triggers some downstream changes in the system, e.g., switching modes of a robot, such errors may dramatically decrease the robot's performance.

Our next error type is systematic errors, which characterize a loss uniformly distributed across the entire input domain. Formally, we define systematic error by the ratio of the average loss and the global worst case loss,

Definition 3 (Systematic error). *Given a continuous loss function $\mathcal{L}$, a neural network f_θ, then we say the systematic error η ratio of f_θ is defined as*

$$\eta = \frac{\frac{1}{|\mathbb{D}|} \sum_{(x,y)\in\mathbb{D}} \mathcal{L}(y, f_\theta(x))}{\max_{(x,y)\in\mathbb{D}} \mathcal{L}(y, f_\theta(x))}, \tag{4.7}$$

if $\max_{(x,y)\in\mathbb{D}} \mathcal{L}(y, f_\theta(x)) > 0$. Otherwise, we say f_θ is free of any error.

The systematic error η captures whether there are peaks in the loss surface. $\eta \approx 1$ indicates a uniformly distributed error, whereas $\eta \approx 0$ captures an error concentrated at a single point. Note that the systematic error defines a relative error, i.e., how the error is distributed, but discards its magnitude. In Figure 4.2 we visualize such error type in the bottom plot. Systematic errors may be preferable for robot tasks over other error types in applications where we want the robot to provide acceptable performance in all conditions.

Finally, we define conditional error to capture regions of the input domain with a higher average loss than the rest of the domain. Formally,

Definition 4 (Conditional error). *Given a loss function $\mathcal{L}$, a neural network f_θ and a threshold $\eta > 0$, then we call a set $\mathcal{D} \subset \mathbb{D}$ a conditional error if*

$$\frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} \mathcal{L}(y, f_\theta(x)) > \eta, \tag{4.8}$$

and

$$\frac{1}{|\mathcal{D} \setminus \mathbb{D}|} \sum_{(x,y)\in\mathcal{D}\setminus\mathbb{D}} \mathcal{L}(y, f_\theta(x)) < \eta. \tag{4.9}$$

Conditional errors, as illustrated in Figure 4.2, characterize certain conditions making a robot that otherwise performs well fail.

We note that the error type Definitions 2, 3, and 4 are subjective regarding the choice of the threshold and neighborhood values η, ε_1, and ε_2. Nonetheless, they allow us characterize and describe the distribution of errors in a straightforward way.

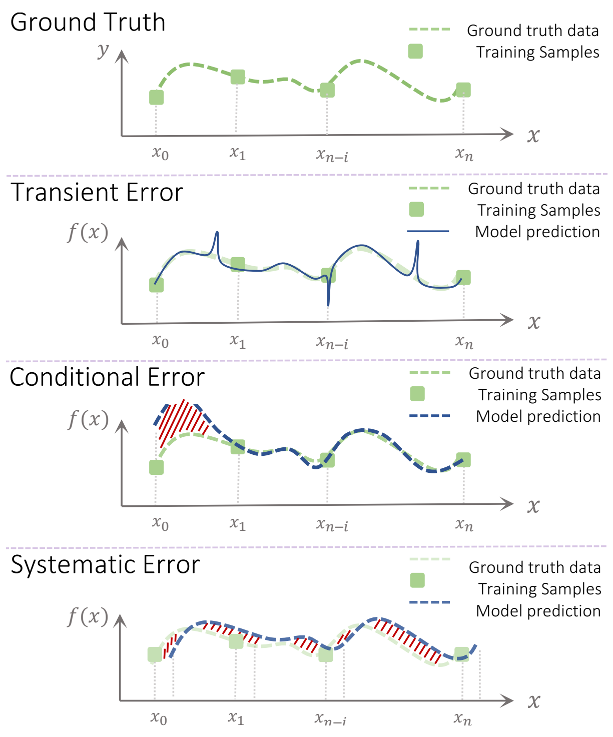


Figure 4.2: Different types of errors that can occur when fitting a network $f(x) = y$.

While the types of errors are independent of their contribution to the magnitude of the expected test loss, they can have a large effect on the closed-loop performance of the robot system. For example, transient errors may be filtered out over time if the controller runs with a high frequency, e.g., an autonomous vehicle that fails to detect another car in 1 out of 30 frames processed per second. Conversely, transient errors may have fatal consequences in applications where a single decision is enough to trigger downstream effects, e.g., a robot arm that releases a heavy object after erroneously classifying that space below the object is free. Similarly, to what degree conditional and systematic errors are tolerable or lead to catastrophic results depends on the specific application.

4.5 Case Study

In this section, we conduct a case study consisting of three robotic learning tasks. Our objective is to I) study the robustness-accuracy tradeoff in practical applications II) investigate what types of erroneous side-effects are introduced by an adversarial or safety-domain training scheme. We train neural networks on labeled datasets using open-loop data (supervised learning) for each of the three tasks. We then deploy the network in closed-loop scenarios and measure holistically if the robot can solve the given scenarios. Besides optimizing for high accuracy, we train with secondary robustness and safety specifications applied on the networks using adversarial training or our safety domain training algorithm. Moreover, we vary the strictness level of the specifications, e.g., adversarial attack radius, and study how the strictness affects the network's error characteristics incrementally.

Case Study task 1 - Visual gesture recognition

In the first task of our case study, we develop a controller for a mobile robot. A human operator enables and disables the mobile robot via visual gestures. Once activated, the robot navigates

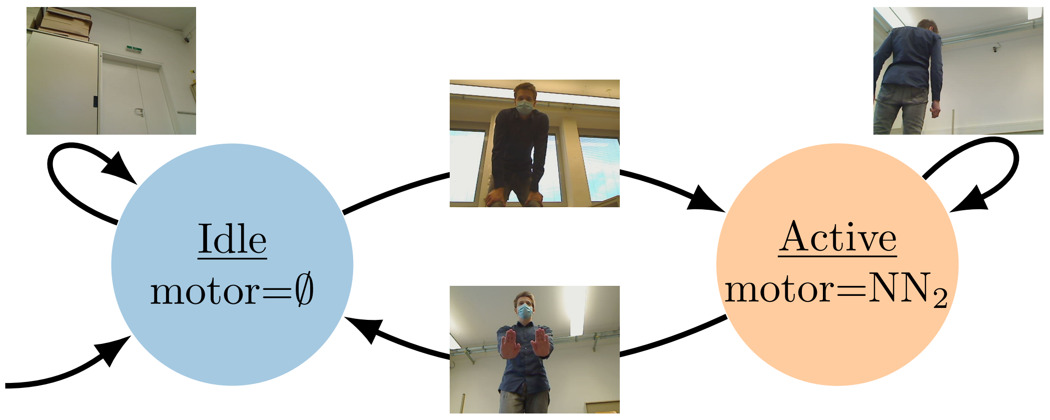


Figure 4.3: State machine of the high-level controller. Transitions between states are triggered by a ResNet50 image classifier. In the active state the second neural network translates the LiDAR inputs to motor outputs.

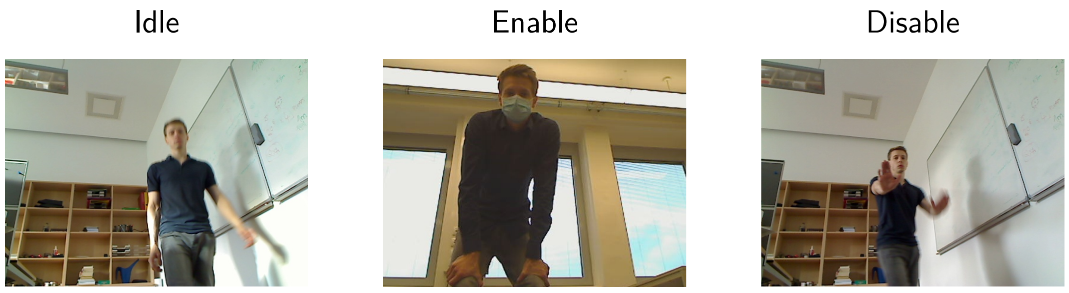


Figure 4.4: Example images corresponding to the three categories of the visual gesture recognition task.

to follow the operator such that it always faces the human operator at a distance of roughly one meter. The control software consists of two neural networks and a state machine with two states. State transitions are triggered by a neural network (the vision network) processing the camera inputs. Fig. 4.3 shows an illustration of the state-machine and its transition profiles. The robot's active behavior is realized via a second neural network (the follow network) that continuously translates a 2D-LiDAR scan of the environment into motor commands. The robot's behavior is entirely determined by the networks' decisions, making the controller well suited for our empirical robot learning study. Our physical robot is equipped with a Sick LMS1xx 2D-LiDAR rangefinder, a Logitech RGB camera, and a 4-wheeled differential drive.

In our first part, we study the vision network of our robot controller. The visual gesture recognition task concerns classifying images perceived from the robot's camera in three categories, i.e., idle, enable, and disable gestures, as illustrated in Fig. 4.4. We collect a total of 2029 sample images, i.e., idle (905 samples), enable (552 samples), and disable (572 samples), which we split into a training and a validation set with a 90%:10% ratio. We follow the standard practice for low data image classification problems of using a pre-trained network instead of learning one from scratch. In particular, we use a ResNet50 [He et al., 2016] pre-trained on ImageNet [Russakovsky et al., 2015] that maps input images to 2048 dimensional feature vectors. We then train a linear classifier that maps the feature vector to a three-dimensional softmax output corresponding to our three categories. After the linear top layer is trained, we fine-tune all layers for two epochs to further increase the accuracy of the network, i.e., we use a linear-probing then full fine-tuning protocol [Kumar et al., 2022].

Table 4.1: Evaluation of the image recognition networks.

#	Scenario description	Adversarial training radius		
		$\varepsilon = 0$	$\varepsilon = 1$	$\varepsilon = 2$
1	Forward-backward	1	-	5
2	With surgical mask	-	1	3
3	Against direct sunlight	-	-	1
4	Staying idle	1	-	4
5	Summon out of garage	-	1	1
6	Artificial lighting (idle)	-	-	3
7	Artificial lighting (follow)	-	-	1
Total		2	2	18

Note: Evaluation of the image recognition networks trained with and without adversarial training on seven test scenarios. Numbers indicate number of misclassified gestures that triggered a change in operation mode, i.e., errors without an effect are not counted. Dash represent zero misinterpretations.

Table 4.2: Training and validation accuracy of the vision network trained with different adversarial perturbation radii. Training accuracy represents adversarial accuracy and validation accuracy represents clean accuracy.

Level	**Training acc. (adversarial)**	**Validation acc. (clean)**
0	99.7 %	98.4%
1	52.0%	92.8%
2	32.5%	71.9%

For the training, we use the Adam optimizer [Kingma and Ba, 2015] with a learning rate of 0.0005 for the first phase and learning rate of 0.00001 for the second phase. The batch size is set to 64.

We train the vision network by adversarial training with the fast-gradient-sign method [Goodfellow et al., 2014b] and three different values for ϵ, i.e., l_∞ neighborhoods with $\varepsilon \in \{0, 1, 2\}$, see Fig. 4.7 for an example. Note that adversarial training with $\varepsilon = 0$ is equivalent to a standard empirical risk minimization training. The training and validation accuracy is reported in Table 4.2.

We evaluate each of the three vision networks in seven real-world benchmark scenarios with some situations not present in the training data. Each scenario consists of a sequence of enabling, disabling, idle, and following commands. The first scenario consists of a sequence of enabling, forward motion, disable, enabling, backward motion, and disabling commands by the human operator. The second scenario is the same as the first, except the human operator wears a surgical mask not present in the training data. In the third scenario, the robot faces direct sunlight. In the fourth scenario, the robot has to stay idle while the human operator moves in front of the robot without providing the robot with any enabling gesture command. The fifth scenario starts with the robot placed below a desk, which limits the robot's field of view. The sixth scenario repeats the first scenario, except that the room is illuminated by artificial lighting instead of sunlight coming from the windows. The final scenario is the same as the fourth scenario except for the lighting, which equals the sixth scenario.

Layer	Hyperparameters
Conv1D	f=32,k=5,s=1,relu
Conv1D	f=96,k=5,s=2,relu
Conv1D	f=96,k=5,s=2,relu
Conv1D	f=96,k=5,s=2,relu
Conv1D	f=96,k=5,s=2,relu
Conv1D	f=96,k=5,s=2,relu
Flatten	
Fully-connected	128 units, relu
Dropout	p=0.4
Fully-connected	7, softmax

Table 4.3: Architecture of the 1D convolutional follow network. f represents the number of filters, k the kernel size, and s the strides of the convolution layers. The network has a total of 361,127 trainable parameters.

We report the number of misinterpreted gestures for each tested scenario-network pair, i.e., enabling or disabling the controller without the operator's command. Consequently, some types of errors are masked out (e.g., "enabling when the controller is already enabled").

The results for the adversarially fine-tuned vision network are shown in Table 4.1. While the network trained with $\varepsilon = 1$ performed as well as the model trained by standard ERM, the performance significantly dropped when increasing the adversarial attack radius. Given that adversarial perturbations with $\varepsilon = 2$ are imperceptible for human observers, our results indicate that current training methods as ad-hoc replacement of standard ERM cannot enforce non-trivial adversarial robustness on an image classifier in a real-world robotic learning context. Moreover, the errors occurring sporadically across different scenarios suggest that adversarial training tends to cause transient errors.

Case Study task 2 - Following robot with safety guarantees

In the second task of our case study, we investigate the follow network of our robot controller from part 1 of our study. The task of the follow model is to map 541-dimensional laser range scans to 7 possible categories, i.e., stay, straight forward, left forward, right forward, straight backward, left backward, and right backward. An illustration of samples of this classification problem is shown in Fig. 4.6. We collected a total of 2705 training and 570 validation samples uniformly across the seven classes.

Our command-following network is a 1D convolutional neural network with the full architecture shown in Table 4.3.

As the follow network directly controls the motors, it potentially crashes into the human operator or an obstacle causing physical damage. To avoid such worst-case outcomes, we enforced safety specifications on the network. In particular, we want to avoid the forward movement of the robot in case an object is in front of it. In Table 4.4, we define four levels of safety domains that characterize our safety requirements with increasing strictness. For example, in safety level 1, we require that at least three consecutive rays give a laser scan reading of 20cm or closer while the remaining rays can measure an arbitrary distance up to 3 meters, then the model should never output any class corresponding to a forward

Table 4.4: Specification of the safety domains D_i for the different safety levels.

Level	**Description of safety domains D_i**
0	$D_i = \emptyset$
1	$D_i = \{x \mid 0 \leq x_j \leq 0.2$ for $j \in \{i-1, i, i+1\}$ and $0 \leq x_j \leq 3$ for $j \notin \{i-1, i, i+1\}\}$
2	$D_i = \{x \mid 0 \leq x_i \leq 0.2$ and $0 \leq x_j \leq 3$ for $j \neq i\}$
3	$D_i = \{x \mid 0 \leq x_i \leq 0.2$ and $0 \leq x_j \leq 4$ for $j \neq i\}$

Setup Details: For each level there are 240 domains, i.e., $i = 150 \ldots 390$. The corresponding labels z_i are defined as a any non-forward moving category, i.e., $z_i \in \{\text{stay}, \text{backward}, \text{left backward}, \text{right backward}\}$. The domains with increase safety level represent super-set of the lower safety level, e.g. the conditions considered at level 1 are a strict subset of the level 2 safety. Level 1 safety only considers cases where at least three consecutive LiDAR rays are less than 20 cm, whereas one ray is enough for level 2 and 3. Level 3 differs from level 2 in terms of the upper bounds on the other rays.

motion. Safety level 2 drops the three consecutive rays requirement for a single reading that gives a distance of 20cm or closer. Safety level 3 then relaxes the remaining rays' 0-3 meters requirement to the range 0-4 meters. Safety level 0 contains no safety domain and is equivalent to standard empirical risk minimization. A visualization of a safety domain from the level 1 specification is shown in Fig. 4.5 on the left.

We train the follow networks using our safety domain training in Algorithm 4.1. We use interval arithmetic to bound the inner maximization step of the training objective in Eq. 4.5, i.e., certified safety compared to the empirical safety of the vision network in part 1 of our case study. For the training, we use the Adam optimizer [Kingma and Ba, 2015] with a learning rate of 0.0001 and a batch size of 64. The safety level 0 model is trained for 20 epochs, while safety-domain training is applied for 2000 epochs.

The training and validation accuracies for the follow networks are shown in Table 4.5.

We evaluate each follow network in seven standardized scenarios in which the robot has to follow the human operator across a given path in an environment. The scenarios differ in complexity, e.g., operator path, obstacles (boxes or tables), environment. We report a holistic metric for each scenario depending on if the robot maneuvers correctly for the entire scenario.

The results are shown in Table 4.6. Only the network trained with standard ERM could successfully handle all scenarios. Interestingly, Fig. 4.5 (right image) shows that this network is vulnerable to adversarial misclassifications and would output a forward decision and crash

Table 4.5: Training and validation accuracy of the follow network trained when enforcing different safety-levels.

Level	**Training accuracy**	**Validation accuracy**
0	98.8%	84.7%
1	99.7%	76.8%
2	97.1%	73.4%
3	57.3%	53.2%

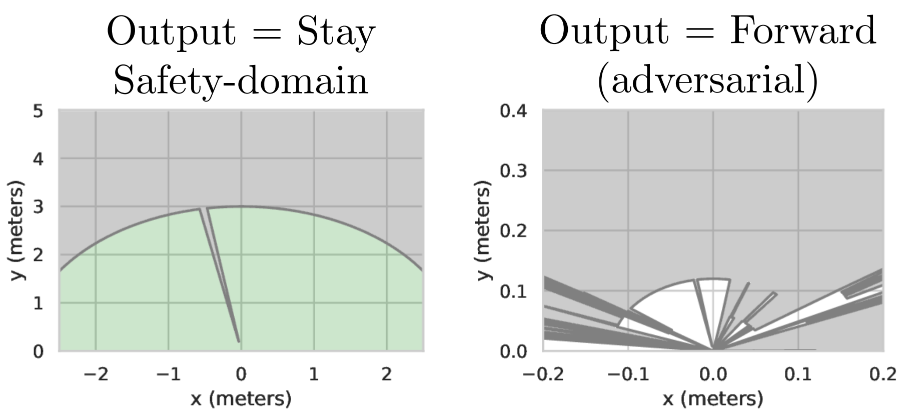


Figure 4.5: **Left:** Visualization of a safety-domain. No LiDAR signal in green area should be classified as a "forward" decision. **Right:** The network trained with standard ERM can be attacked to output a "forward" despite the LiDAR signal indicating a large object 10cm in front of the robot.

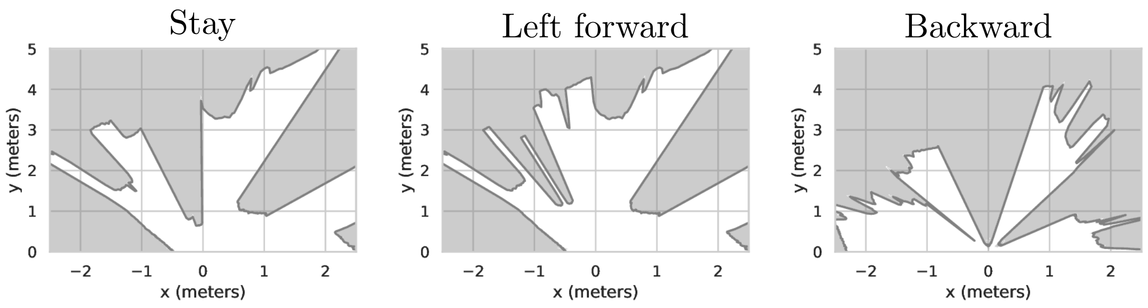


Figure 4.6: Three training samples of the follow network. The network has to detect the position of the human operator relative to the robot's pose in laser range scans. In total there are seven possible positions.

the robot if the large object is directly in front of the robot.

While the networks with safety-level one and above are immune to such attacks, they perform significantly worse on the seven test scenarios. With increasing specification levels, the performance monotonously decreases until the network trained with the most rigorous safety specification cannot handle any scenario at all. In contrast to part 1, where we observed transient errors, the defects made by the certified networks appear to be conditioned on specific scenarios. In particular, if a network with specification level 1 could not solve a scenario, then a network with 2 and 3 could not either, e.g., the "Around boxes" and "Narrow hallway" scenario. Moreover, the failure of the level 1 and level 2 networks happened only during forward locomotion, while no fault in a backward motion was observed. Our observation suggests that safety-domain training causes conditional errors in parts of the input space close to the safety domains.

Case Study task 3 - Autonomous driving from camera images

Finally, we study an autonomous vehicle on a lane-keeping task. In particular, a network is trained to predict the curvature of the road ahead of a car. The network is fed by images

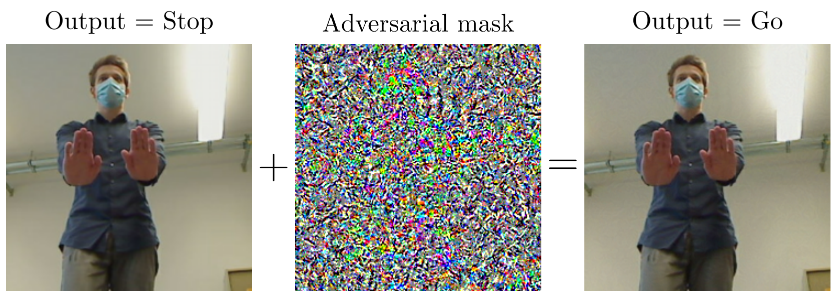


Figure 4.7: Visualization of an adversarial attack on our vision network. Adding an adversarial mask flips the decision from a stop command to an activation command.

Table 4.6: Evaluation of the LiDAR follow networks

#	Scenario Description	Standard training	Safety Level 1	Safety Level 2	Safety Level 3
1	Plain	✓	✓	✓	Fail
2	Around boxes	✓	Fail	Fail	Fail
3	Out of corner	✓	✓	✓	Fail
4	Through gate	✓	✓	Fail	Fail
5	Around table	✓	✓	✓	Fail
6	Garage parking	✓	✓	✓	Fail
7	Narrow hallway	✓	Fail	Fail	Fail
Total		7/7	5/7	4/7	0/7

Note: Evaluation of the LiDAR follow networks with various safety specification enforced on seven standardized test scenarios. Successful navigation of a scenario is marked by a ✓. Fail indicates unsuccessful tests.

Layer	Parameter
Conv2D	F=32, K=5, S=2, ReLU
Conv2D	F=64, K=5, S=1, ReLU
Conv2D	F=96, K=3, S=2, ReLU
Conv2D	F=128, K=3, S=1, ReLU
GlobalAveragePool2D	
Fully-connected	1000 units, ReLU
Fully-connected	100 units, ReLU
Fully-connected	1 unit

Table 4.7: Convolutional neural network baseline architecture for our autonomous driving case study task.

received at a camera on top of the vehicle as input [Amini et al., 2020a]. The predicted curvature can then be used for controlling the steering wheel of the car to keep the vehicle on the road. The training data is collected by a human driver who maneuvers the car around a test track. The networks are then trained on collected data using supervised learning. Finally, we deploy the networks in a closed-loop autonomous driving simulator. We use the VISTA simulation environment [Amini et al., 2021] for this purpose.

We evaluate a convolutional neural network with the architecture listed in Table 4.7. The

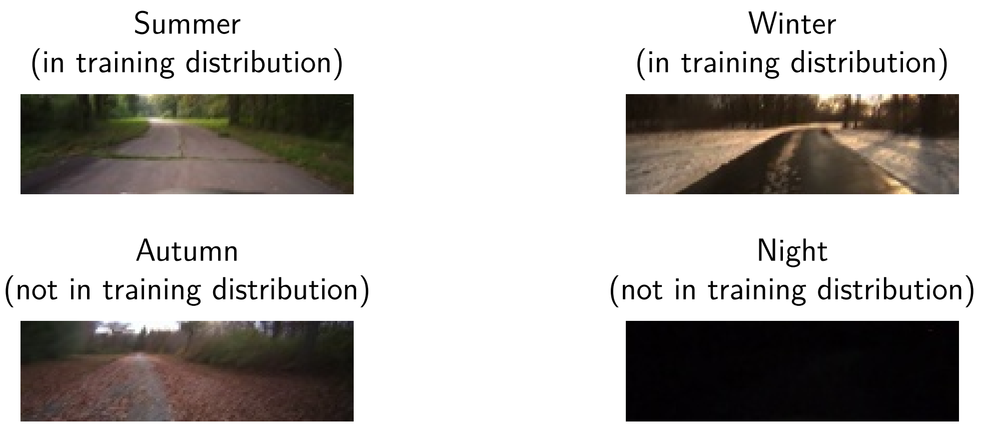

Summer
(in training distribution)
Winter
(in training distribution)
Autumn
(not in training distribution)
Night
(not in training distribution)

Figure 4.8: Test conditions of our closed-loop driving task using a data-driven simulation environment [Amini et al., 2021]. The training data are collected in summer and winter conditions (separated from the testing data).

inputs are 160-by-48 RGB images that are normalized per image such that the sample of all pixel values have zero mean and unit standard deviation. For the training, we use the Adam optimizer [Kingma and Ba, 2015] with a learning rate of 0.0003 and a batch size of 64. The weight decay is set to 10^{-5}, and training was performed for a total of 900,000 steps. We train all models with standard and adversarial training with increasing attack budget ($\varepsilon = 0, 1, \ldots 8$) and I-FGSM as attack methods.

For each model, we run a total of 400 simulations, split into 200 in-training-distribution and 200 not-in-training-distribution condition runs. The in-training distribution data were collected in summer and winter and were separated from the training data, i.e., there is no overlap between the training data and the evaluation data. The not-in-training-distribution data were collected in autumn and during the night, with no such condition present in the training data. The four conditions are visualized in Figure 4.8. As an evaluation metric, we report the number of crashes during the simulation, i.e., when the vehicle leaves the road.

The results in Table 4.8 shows that, while the network trained with a low attack budget provides acceptable performance in the in-training-distribution conditions (summer and winter), the number of crashes is substantially larger for the adversarially trained model on the not-in-training distribution runs (autumn and night). For example, the network trained with standard training failed only in 1 out of 100 runs in autumn conditions, whereas the crashes jump to 30 when the network is trained with adversarial ERM. A similar jump by doubling the number of crashes is observed in the night simulations. This observation suggests that the adversarial training process introduces conditional errors that affect regions with a shifted distribution than the training data. Similar to our observations in the previous tasks of our case study, for larger training attack budgets, the models yield very poor performance even on the in-training distribution simulation runs.

4.6 Experiments on the Case Study Datasets

In this section, we assess whether recent advances reported in the literature on the robustness of neural networks can overcome the limitations observed in the previous section. In particular, we test overparametrized models, vision transformers, and smooth curvature activation functions

Environment condition		Adversarial training budget ε 0	1	2	3	4	5	6	7	8
Summer	(in-training-distribution)	**0**	**0**	**0**	1	2	6	10	14	12
Winter	(in-training-distribution)	**0**	**0**	4	9	12	35	41	40	51
Autumn	(not-in-training-distribution)	**1**	30	40	61	61	64	91	89	85
Night	(not-in-training-distribution)	**36**	78	78	90	93	86	94	97	87

Table 4.8: Number of crashes out of 100 simulation runs per environment condition of our CNN trained with adversarial training under various attack budgets. Best values for each condition are highlighted in bold.

on the robot datasets from our case study in section 4.5. Due to the large number of models tested, we perform our experiments offline or in simulation instead of on the physical robots.

Visual gesture recognition dataset

Our first experiment investigates whether overparametrized networks improve the accuracy-robustness tradeoff on the visual gesture recognition dataset from case study task 1. The physical robot experiments suggest that a validation accuracy of above 90% is necessary for acceptable robot performance.

In this experiment, we resort to transfer learning of a pre-trained classifier using the big-transfer (BiT) fine-tuning protocol of initializing the output layer with all zeros and training all layers even from the first epoch on [Kolesnikov et al., 2020]. All pre-trained models are trained on the ImageNet dataset [Russakovsky et al., 2015]. We train networks of different sizes using adversarial training with increasing attack budget ($\varepsilon \in \{1, 2, 4, 8\}$) and report the clean and robust validation accuracy under I-FGSM attacks with various attack budgets. For increasing the size of the model, we test a ResNet50 (24M), ResNet101 (43M), and ResNet152 (58M) with the number of trainable parameters reported in parenthesis [He et al., 2016]. We also evaluate the vision transformer models ViT-Small (22M), ViT-Base (86M), and ViT-Large (304M) that process the images in the form of 16-by-16 pixel patches [Dosovitskiy et al., 2020]. For the training, we use the Adam optimizer [Kingma and Ba, 2015] with a learning rate of 0.00005 and a batch size of 64, except for the ResNet152, ViT-Base, and ViT-Large models where a batch size of 32, 32, and 16 respectively is used due to out-of-memory errors. We repeat each training run with 5 random seeds and report the mean and standard deviation.

As a proxy for the real-world test accuracy, we collect a test datasets comprising of 190 idle samples, 129 enable samples, and 140 disable samples. An example visualizing how the two data sources differ is shown in Figure 4.9. Our motivation for collecting this dataset is that the validation set might be temporally and spatially correlated with the training data, i.e., collected at the same time and location. Consequently, the validation set may not capture the actual real-world accuracy of our trained models due to robust overfitting [Rice et al., 2020]. We use the clean accuracy of the new test set as our test metric to estimate the real-world performance.

The results in Table 4.9 and Table 4.10 show that adversarial training with a small attack budget, e.g., $\varepsilon \in \{1, 2\}$ can have positive effects on the robustness while maintaining a good clean accuracy compared the networks trained with standard training. Interestingly, the models trained with such small attack budget express a non-trivial robust accuracy even for

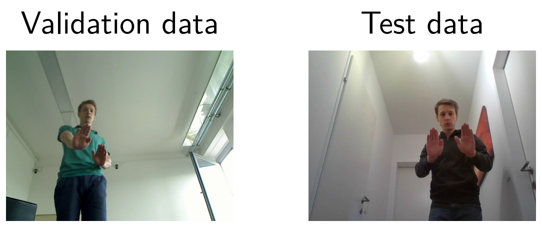


Figure 4.9: Comparison of sample images of validation and test set. The test data was collected with a different time and location than the original data, thus avoiding potential data leakage and making it capture real-world performance more accurately.

Model	Adversarial training budget	Validation accuracy	Validation accuracy under I-FGSM [Kurakin et al., 2017] attack $\varepsilon = 1$	$\varepsilon = 2$	$\varepsilon = 4$	$\varepsilon = 8$	Test accuracy
ResNet50	$\varepsilon = 0$	**99.5%** $\pm$ 0.0	85.3% $\pm$ 2.1	49.5% $\pm$ 6.6	8.7% $\pm$ 2.0	2.3% $\pm$ 1.9	**93.5%** $\pm$ 3.8
	$\varepsilon = 1$	**99.0%** $\pm$ 0.4	**98.1%** $\pm$ 0.4	**93.6%** $\pm$ 1.0	72.4% $\pm$ 3.4	18.9% $\pm$ 3.2	**86.1%** $\pm$ 3.1
	$\varepsilon = 2$	**98.7%** $\pm$ 0.6	**97.7%** $\pm$ 0.7	**96.0%** $\pm$ 0.5	89.5% $\pm$ 1.9	55.3% $\pm$ 3.1	76.0% $\pm$ 2.5
	$\varepsilon = 4$	**96.8%** $\pm$ 0.8	**95.4%** $\pm$ 1.3	**93.9%** $\pm$ 0.9	**90.7%** $\pm$ 2.0	77.0% $\pm$ 3.8	68.6% $\pm$ 3.2
	$\varepsilon = 8$	70.3% $\pm$ 5.1	68.9% $\pm$ 2.4	67.9% $\pm$ 4.2	66.3% $\pm$ 5.0	60.7% $\pm$ 2.7	51.8% $\pm$ 6.6
ResNet101	$\varepsilon = 0$	**99.2%** $\pm$ 0.4	**90.1%** $\pm$ 2.9	54.1% $\pm$ 7.6	10.9% $\pm$ 2.3	3.8% $\pm$ 1.3	**85.9%** $\pm$ 7.3
	$\varepsilon = 1$	**99.5%** $\pm$ 0.0	**97.8%** $\pm$ 0.5	**93.6%** $\pm$ 1.2	74.4% $\pm$ 4.0	20.7% $\pm$ 2.2	**82.9%** $\pm$ 4.4
	$\varepsilon = 2$	**98.7%** $\pm$ 1.1	**96.2%** $\pm$ 1.4	**94.7%** $\pm$ 1.0	87.3% $\pm$ 3.0	54.8% $\pm$ 1.4	76.1% $\pm$ 2.6
	$\varepsilon = 4$	44.2% $\pm$ 0.4	44.4% $\pm$ 0.5	43.9% $\pm$ 0.5	44.2% $\pm$ 0.7	44.4% $\pm$ 0.2	41.5% $\pm$ 1.2
	$\varepsilon = 8$	44.4% $\pm$ 0.5	44.1% $\pm$ 0.6	43.6% $\pm$ 0.5	43.9% $\pm$ 0.5	43.9% $\pm$ 0.5	41.7% $\pm$ 0.0
ResNet152	$\varepsilon = 0$	**98.8%** $\pm$ 0.8	86.8% $\pm$ 4.0	57.3% $\pm$ 4.8	14.8% $\pm$ 6.2	4.4% $\pm$ 4.4	**87.2%** $\pm$ 4.2
	$\varepsilon = 1$	**99.0%** $\pm$ 0.4	**97.6%** $\pm$ 0.4	**95.2%** $\pm$ 1.4	83.0% $\pm$ 1.4	29.6% $\pm$ 5.7	**82.1%** $\pm$ 7.0
	$\varepsilon = 2$	**98.1%** $\pm$ 0.8	**97.6%** $\pm$ 0.6	**95.5%** $\pm$ 0.7	**90.9%** $\pm$ 1.6	66.7% $\pm$ 4.2	75.8% $\pm$ 4.4
	$\varepsilon = 4$	60.5% $\pm$ 18.9	60.3% $\pm$ 18.4	60.0% $\pm$ 18.0	58.4% $\pm$ 16.4	52.6% $\pm$ 9.8	50.8% $\pm$ 12.4
	$\varepsilon = 8$	44.4% $\pm$ 0.4	44.3% $\pm$ 0.2	44.4% $\pm$ 0.4	43.9% $\pm$ 0.4	44.5% $\pm$ 0.4	41.7% $\pm$ 0.0

Table 4.9: Accuracies on the visual gesture recognition dataset of various pre-trained residual neural networks fine-tuned via standard and adversarial training. Validation accuracies greater than 90% and test accuracies greater than 80% are highlighted in bold. The different column show the standard accuracy and accuracy-under-attack with respect to the I-FGSM method and different attack budgets.

larger attack budgets under I-FGSM attacks. However, the performance drop significantly to an unacceptable performance when the networks are trained with a larger attack budget of $\varepsilon = 8$. A surprising observation is that our results contradict to some degree that larger models provide a better robustness in practice. In particular, out of all adversarially trained convolutional networks, the ResNet50 trained with $\varepsilon = 1$ provides the best test accuracy. Moreover, the ResNet50 also achieves the best test and validation accuracy under stronger training attack budgets, e.g. $\varepsilon \in \{4, 8\}$. In contrast, the vision vision transformer architecture seems to benefit from a larger model. The largest ViT provides the best test accuracy under adversarial training with an attack budget of $\varepsilon \in \{1, 2, \}$ out of all tested models. These results suggest that the type of learned representation, e.g. convolutional vs attention-based, plays a role in whether overparameterization improves robustness or not. In both convolutional and attention-based models, we observed a double descent phenomenon with respect to the model size [Nakkiran et al., 2019]. Particularly, both the mid-size ResNet101 and ViT-Base/16 performed worse than their smaller and larger counterparts.

Model	Adversarial	Validation	Validation accuracy under I-FGSM [Kurakin et al., 2017] attack				Test
	training budget	accuracy	$\varepsilon = 1$	$\varepsilon = 2$	$\varepsilon = 4$	$\varepsilon = 8$	accuracy
	$\varepsilon = 0$	**99.0%** $\pm$ 0.3	84.5% $\pm$ 4.7	50.4% $\pm$ 6.7	15.6% $\pm$ 8.6	0.8% $\pm$ 1.5	**94.1%** $\pm$ 2.1
	$\varepsilon = 1$	**99.5%** $\pm$ 0.0	**97.7%** $\pm$ 1.2	**91.8%** $\pm$ 2.0	70.3% $\pm$ 1.6	21.3% $\pm$ 5.6	**84.6%** $\pm$ 6.0
ViT-Small/16	$\varepsilon = 2$	**99.4%** $\pm$ 0.2	**98.9%** $\pm$ 0.2	**97.6%** $\pm$ 0.9	89.2% $\pm$ 2.3	48.0% $\pm$ 4.8	**83.3%** $\pm$ 4.7
	$\varepsilon = 4$	**99.3%** $\pm$ 0.5	**98.8%** $\pm$ 0.4	**97.0%** $\pm$ 0.8	**94.4%** $\pm$ 1.4	72.3% $\pm$ 5.0	75.2% $\pm$ 3.1
	$\varepsilon = 8$	44.1% $\pm$ 0.4	44.0% $\pm$ 0.4	43.6% $\pm$ 0.2	44.1% $\pm$ 0.4	43.9% $\pm$ 0.4	41.1% $\pm$ 1.9
	$\varepsilon = 0$	**98.4%** $\pm$ 1.1	89.4% $\pm$ 4.7	67.0% $\pm$ 6.2	28.7% $\pm$ 5.2	11.7% $\pm$ 4.5	**82.3%** $\pm$ 5.7
	$\varepsilon = 1$	**98.9%** $\pm$ 0.6	**99.2%** $\pm$ 0.5	**92.1%** $\pm$ 5.9	72.8% $\pm$ 11.8	37.9% $\pm$ 8.4	79.4% $\pm$ 1.7
ViT-Base/16	$\varepsilon = 2$	**99.5%** $\pm$ 0.0	**99.5%** $\pm$ 0.3	**97.9%** $\pm$ 0.9	**92.5%** $\pm$ 1.0	61.4% $\pm$ 3.8	**84.5%** $\pm$ 4.6
	$\varepsilon = 4$	**97.4%** $\pm$ 2.3	**94.5%** $\pm$ 4.8	**93.5%** $\pm$ 5.4	86.7% $\pm$ 7.5	67.1% $\pm$ 11.7	76.0% $\pm$ 1.9
	$\varepsilon = 8$	53.4% $\pm$ 7.3	53.2% $\pm$ 7.6	52.2% $\pm$ 7.2	50.0% $\pm$ 5.1	48.0% $\pm$ 3.7	54.6% $\pm$ 10.2
	$\varepsilon = 0$	**98.7%** $\pm$ 0.5	**92.1%** $\pm$ 1.6	74.7% $\pm$ 4.3	44.1% $\pm$ 11.0	20.8% $\pm$ 11.0	**85.9%** $\pm$ 2.5
	$\varepsilon = 1$	**99.3%** $\pm$ 0.2	**98.1%** $\pm$ 0.0	**95.0%** $\pm$ 0.7	77.6% $\pm$ 0.7	35.3% $\pm$ 1.2	**89.9%** $\pm$ 3.6
ViT-Large/16	$\varepsilon = 2$	**99.5%** $\pm$ 0.0	**98.6%** $\pm$ 0.5	**97.6%** $\pm$ 1.0	**92.3%** $\pm$ 2.4	67.1% $\pm$ 9.9	**89.6%** $\pm$ 3.2
	$\varepsilon = 4$	**98.6%** $\pm$ 1.0	**97.6%** $\pm$ 1.0	**96.0%** $\pm$ 1.6	**93.3%** $\pm$ 2.8	77.4% $\pm$ 6.6	71.0% $\pm$ 15.9
	$\varepsilon = 8$	64.6% $\pm$ 22.9	64.4% $\pm$ 22.3	63.8% $\pm$ 23.0	63.0% $\pm$ 21.3	58.0% $\pm$ 17.5	47.2% $\pm$ 9.8

Table 4.10: Accuracies on the visual gesture recognition task of various pre-trained vision transformer fine-tuned via standard and adversarial training. Validation accuracies greater than 90% and test accuracies greater than 80% are highlighted in bold. The different column show the standard accuracy and accuracy-under-attack with respect to the I-FGSM method and different attack budgets.

Following robot dataset

In this experiment, we study the safety-domain training procedure of the follow task of the case study in more detail. In particular, we test the overparametrization, increased weight decay (from 0 to 10^{-5}), and smooth activation function methods on this task. As a baseline, we use the 1D-CNN from the case study but define a widening factor w to modulate the size of the network as listed in Table 4.12. We use the exponential linear unit (ELU) activation function [Clevert et al., 2019] to represent a smooth activation due to the non-monotonically of SiLU being less compatible with the used interval abstract interpretation domains. We train all models with the Adam optimizer [Kingma and Ba, 2015] with a learning rate of 0.0001 and a batch size of 64. The safety level 0 models are trained for 20 epochs, while the networks are trained using safety-domain training for 2000 epochs.

We report the validation accuracy as an evaluation metric. Our case study on the physical robot suggests that a validation accuracy above 80% is necessary to achieve an acceptable real-world performance. Note that all models, except those trained with safety level 0, provide some form of formal safety guarantees. Therefore, this experiment studies how much validation accuracy is traded for the ensured safety. We repeat each training run with 5 random seeds and report the mean and standard deviation.

The result in Table 4.11 shows that safety-domain training benefits from an increased number of parameters (width). However, the improvement over the baseline is rather incremental and accounts only for a few percent. In contrast, the accuracy reduction caused by the safety-domain training is several times more significant, e.g., around 10%, and no network trained with safety-domain training exceeds the threshold of 80% accuracy. The networks with smooth activation function and increased weight decay performed worse than the baseline when using safety-domain training. This suggests that certified training methods such as safety-domain training may require different hyperparameters and learning settings than adversarial training.

Safety level		Validation accuracy Width 1	Width 2	Width 3	Width 4
0	Baseline	**83.2%** ± 0.8	**84.7%** ± 1.6	**83.9%** ± 1.9	**85.2%** ± 0.8
	ELU	73.3% ± 1.5	72.5% ± 3.3	73.3% ± 0.8	71.3% ± 1.3
	wd+	**82.5%** ± 2.0	**84.0%** ± 2.1	**85.7%** ± 1.3	**85.7**% ± 1.2
1	Baseline	75.1% ± 2.6	78.6% ± 3.7	77.4% ± 2.1	78.7% ± 3.4
	ELU	53.1% ± 0.6	53.5% ± 0.4	52.9% ± 0.6	52.3% ± 0.8
	wd+	74.2% ± 3.4	75.0% ± 1.8	65.9% ± 10.7	67.4% ± 12.0
2	Baseline	76.3% ± 3.1	76.8% ± 4.9	76.1% ± 2.8	78.5% ± 3.2
	ELU	53.6% ± 0.3	53.1% ± 0.3	53.2% ± 0.4	52.9% ± 0.6
	wd+	72.9% ± 3.3	75.5% ± 2.1	68.4% ± 8.6	70.7% ± 10.0
3	Baseline	51.8% ± 0.9	52.8% ± 0.5	53.3% ± 0.1	53.9% ± 0.3
	ELU	53.2% ± 0.8	53.8% ± 0.5	53.1% ± 0.1	53.2% ± 0.4
	wd+	51.4% ± 1.1	52.8% ± 0.7	52.8% ± 0.6	53.4% ± 0.4

Table 4.11: Validation accuracy on the robot follow task of 1D-convolutional NNs with various hyperparameters and trained with standard and safety-domain training. Values greater than 80% are highlighted in bold. Safety level 0 corresponds to standard training, while the network trained with safety level 1 and above provide formal safety guarantees of never crashing the robot into an obstacle. The columns show networks with different widening factor and consist of an increasing amount of learnable parameters, i.e., width 1 (360k), width 2 (1.4M), width 3 (3.2M), and width 4 (5.7M).

Layer	Parameter
Conv1D	F=w*32, K=5, S=1, ReLU
Conv1D	F=w*96, K=5, S=2, ReLU
Conv1D	F=w*96, K=5, S=2, ReLU
Conv1D	F=w*96, K=5, S=2, ReLU
Conv1D	F=w*96, K=5, S=2, ReLU
Conv1D	F=w*96, K=5, S=2, ReLU
Flatten	
Fully-connected	w*128 units, ReLU
Fully-connected 7	softmax

Table 4.12: Network architecture of the 1D-CNN trained with safety-domain training. (F= number of filters, K = kernel size, S = stride). w is the widening factor which controls the size and number of parameters of the network.

Autonomous driving dataset

Our final experiment considers an autonomous driving task.

We compare the performance of the baseline CNN used in the case study with four variations. First, we compare with an enlarged variant of the baseline CNN listed in Table 4.13 to validate the necessity of overparametrization for robustness empirically. Next, we equip the baseline with the smoother SiLU activation and increase the weight decay. Finally, we test a vision transformer model. The baseline model consists of 440k, the enlarged CNN of 7.7M, and the tested vision transformer of 2.0M trainable parameters. Our vision transforms splits the input image into non-overlapping patches of 16-by-12 pixels, uses a latent dimension of 256, with 4 attention heads, 384 feed-forward dimension, and 4 layers in total. For the training, we use the Adam optimizer [Kingma and Ba, 2015] with a learning rate of 0.0003 and a batch size of 64. The weight decay is set to 10^{-5}, except for the wd$+$ variant, which is trained with a decay factor of $5 \cdot 10^{-5}$. We train all networks for a total of 900,000 steps. We train all models with standard and adversarial training with increasing attack budget ($\varepsilon = 0, 1, \dots 8$) and I-FGSM as attack methods.

We deploy the trained networks in the VISTA simulator [Amini et al., 2021] using different data conditions, similar to our case study evaluation. We report the number of crashes during the simulation.

Table 4.14 and Table 4.15 show the crashes during the summer and winter simulations respectively. The best values are highlighted in bold. The results show that an overparametrized model and a vision transformer indeed provide better performance at a larger adversarial training budget. An increased weight decay improved the performance only at lower attack budget training, while the networks with SiLU activation performed worse in the closed-loop tests. No model could drive the car safely at larger training attack budgets, while most models learned by standard ERM could drive all 200 runs flawlessly.

The no-in-training-distribution simulation results for autumn and night conditions are shown in Table 4.16 and Table 4.17. We observe that adversarial training significantly hurt the no-in-training-distribution performance of all models, i.e., especially in the autumn data. In summary, the best driving performance across all four tested conditions was observed with networks trained with standard ERM.

4.7 Conclusion

In principle, adversarial training and its generalization, safety-domain training can learn robust and safe deep learning models. However, the benefits of these methods do not come for free but with a reduction of nominal performance, e.g., a tradeoff between robustness and accuracy. In this chapter, we provided empirical evidence that this tradeoff might not be a tradeoff but has overall net negative effects. In particular, we conducted a case study on three real-world robot learning tasks for which we trained the controller with and without adversarial and safety domain training. We then characterized the error profiles introduced by these training methods when deploying the networks in closed-loop robot controllers. Our results suggest that adversarial training methods require rethinking before being used as an ad-hoc replacement of standard training procedures in real-world robot learning tasks. Finally, we tested whether methods reported in the literature that enhance robustness can overcome this problem without sacrificing accuracy. We observed that while these methods provide relative gains in terms of both accuracy and robustness, the negative side-effects of adversarial training

Layer	Parameter
Conv2D	F=32, K=5, S=2
BatchNorm2D	ReLU (post BN)
Conv2D	F=128, K=5, S=1
BatchNorm2D	ReLU (post BN)
Conv2D	F=256, K=3, S=2
BatchNorm2D	ReLU (post BN)
Conv2D	F=512, K=3, S=1
BatchNorm2D	ReLU (post BN)
Conv2D	F=1024, K=3, S=1
GlobalAveragePool2D	
Fully-connected	1024 units, ReLU
Fully-connected	256 units, ReLU
Fully-connected	1 unit

Table 4.13: Enlarged neural network architecture for our autonomous driving experiment (7.7M parameters).

Model	Adversarial training budget ε								
	0	1	2	3	4	5	6	7	8
CNN	**0**	**0**	**0**	1	2	6	10	14	12
CNN-Large	**0**	**0**	**0**	**0**	**0**	1	2	4	19
ViT	**0**	**0**	**0**	**0**	**0**	1	9	4	4
CNN (SiLU)	**0**	**0**	2	16	25	42	46	71	77
CNN (wd+)	**0**	**0**	**0**	**0**	7	13	27	34	31

Table 4.14: Number of crashes out of 100 simulation runs using data collected in summer.

Model	Adversarial training budget ε								
	0	1	2	3	4	5	6	7	8
CNN	**0**	**0**	4	9	12	35	41	40	51
CNN-Large	**0**	**0**	**0**	2	2	5	9	13	20
ViT	**0**	2	1	**0**	4	2	6	5	5
CNN (SiLU)	3	5	19	38	42	59	57	78	82
CNN (wd+)	**0**	**0**	**0**	2	9	20	41	51	55

Table 4.15: Number of crashes out of 100 simulation runs using data collected in winter.

Model	Adversarial training budget ε								
	0	1	2	3	4	5	6	7	8
CNN	**1**	30	40	61	61	64	91	89	85
CNN-Large	**1**	14	42	55	67	83	81	80	86
ViT	**0**	43	65	67	74	91	69	84	78
CNN (SiLU)	**1**	45	60	86	87	91	80	89	91
CNN (wd+)	7	32	59	62	59	70	88	89	87

Table 4.16: Number of crashes out of 100 simulation runs using data collected in autumn.

Model	Adversarial training budget ε								
	0	1	2	3	4	5	6	7	8
CNN	**36**	78	78	90	93	86	94	97	87
CNN-Large	52	90	88	92	87	93	95	94	94
ViT	68	69	80	73	81	79	88	91	64
CNN (SiLU)	71	79	79	95	90	93	95	94	94
CNN (wd+)	62	60	68	69	91	95	93	92	95

Table 4.17: Number of crashes out of 100 simulation runs using data collected during the night.

still outweigh the improvements. Our results indicate that the type of learned representation, specifically transformer-based and pre-trained models, poses the most promising direction toward training robust models.

CHAPTER 5

Infinite Time Horizon Safety of Bayesian Neural Networks

Bayesian neural networks (BNNs) are a family of neural networks that place distributions over their weights [MacKay, 1992, Hinton and Van Camp, 1993, Barber and Bishop, 1998, MacKay, 1995, Neal, 2012]. This allows learning uncertainty in the data and the network's prediction, while preserving the strong modelling capabilities of neural networks. In particular, BNNs are flexible in the class of data distributions they can learn. This makes BNNs very appealing for robotic and medical applications [Herzog and Ostwald, 2013, McAllister et al., 2017, Amini et al., 2020b, Michelmore et al., 2020] where uncertainty is a central component of data.

Despite the large body of literature on verifying safety of neural networks, the formal safety verification of BNNs has received less attention. Notably, Cardelli et al. [2019], Wicker et al. [2020], Michelmore et al. [2020] have proposed sampling-based techniques for obtaining probabilistic guarantees about BNNs. Although these approaches provide some insight into BNN safety, they suffer from two key limitations. First, sampling provides only bounds on the probability of the BNN's safety which is insufficient for systems with critical safety implications. For instance, having an autonomous vehicle with a 99.9% safety guarantee is still insufficient for deployment if millions of vehicles are deployed. Second, samples can only simulate the system for a finite time, making it impossible to reason about the system's safety over an unbounded time horizon.

In this chapter, we study the safety verification problem for BNN policies in safety-critical systems over an infinite time horizon. Formally, we consider discrete-time closed-loop systems defined by a dynamical system and a BNN policy. Given a set of initial states and a set of unsafe (or bad) states, the goal of the safety verification problem is to verify that no system execution starting in an initial state can reach an unsafe state. Unlike existing literature which considers probability of safety, we verify *sure safety*, i.e. safety of every system execution of the system. In particular, we present a method for computing *safe weight sets* for which every system execution is safe as long as the BNN samples its weights from this set.

Our approach to restrict the support of the weight distribution is necessary as BNNs with Gaussian weight priors typically produce output posteriors with unbounded support. Consequently, there is a low but non-zero probability for the output variable to lie in an unsafe region, see Figure 5.1. This implies that BNNs are usually unsafe by default. We therefore consider the more general problem of computing safe weight sets. Verifying that a weight set is safe allows re-calibrating the BNN policy by rejecting unsafe weight samples in order to change the BNN's predictive distribution in a way that ensures safety.

A common choice for BNN architectures is to approximate the weight distribution via a parametrized class of simple distributions, e.g., typically independent Gaussian distributions. We, therefore, adopt weight sets in the form of products of intervals centered at the means of the BNN's independent Gaussian distributions. To verify safety of a weight set, we search for a safety certificate in the form of a *safe positive invariant* (also known as *safe inductive invariant*). A safe positive invariant is a set of system states that contains all initial states, is closed under the system dynamics and does not contain any unsafe state. The key advantage of using safe positive invariants is that their existence implies the *infinite time horizon safety*. We parametrize safe positive invariant candidates by (deterministic) neural networks that classify system states for determining set inclusion. Moreover, we phrase the search for an invariant as a learning problem. A separated verifier module then checks if a candidate is indeed a safe positive invariant by checking the required properties via constraint solving. In case the verifier finds a counterexample demonstrating that the candidate violates the safe positive invariant condition, we re-train the candidate on the found counterexample. We repeat this procedure until the verifier concludes that the candidate is a safe positive invariant ensuring that the system is safe. We note that the verification step requires the deterministic dynamics of the system to be given.

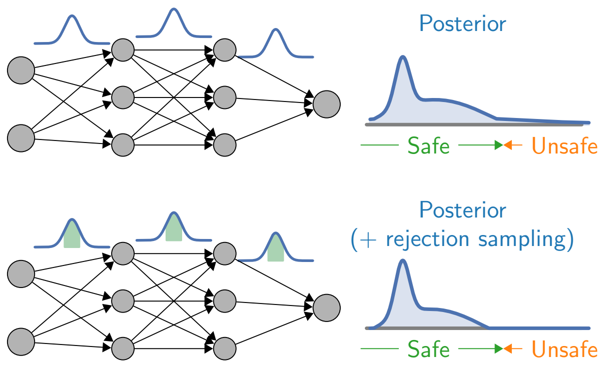


Figure 5.1: BNNs are typically unsafe by default. Top figure: The posterior of a typical BNN has unbounded support, resulting in a non-zero probability of producing an unsafe action. Bottom figure: Restricting the support of the weight distributions via rejection sampling ensures BNN safety.

The safe weight set obtained by our method can be used for safe exploration reinforcement learning. In particular, generating rollouts during learning by sampling from the safe weight set allows an exploration of the environment while ensuring safety. Moreover, projecting the (mean) weights onto the safe weight set after each gradient update further ensures that the improved policy stays safe.

Contributions Our contributions can be summarized as follows:

1. We define a safety verification problem for BNN policies which overcomes the unbounded posterior issue by computing and verifying safe weight sets. The problem generalizes the sure safety verification of BNNs and solving it allows re-calibrating BNN policies via rejection sampling to guarantee safety.

2. We introduce a method for computing safe weight sets in BNN policies in the form of products of intervals around the BNN weights' means. To verify safety of a weight set, our novel algorithm learns a safe positive invariant in the form of a deterministic neural network.

3. We evaluate our methodology on a series of benchmark applications, including non-linear systems and non-Lyapunovian safety specifications.

5.1 Related Works

Verification of feed-forward neural networks Verification of robustness and safety properties in feed-forward neural networks has received much attention but remains an active research topic [Katz et al., 2017, Henzinger et al., 2021, Gehr et al., 2018, Ruan et al., 2018, Bunel et al., 2018, Tjeng et al., 2019]. As the majority of verification techniques were designed for deterministic NNs, they cannot be readily applied to BNNs. The safety verification of feed-forward BNNs has been considered in Cardelli et al. [2019] by using samples to obtain statistical guarantees on the safety probability. The work of Wicker et al. [2020] also presents a sampling-based approach, however it provides certified lower bounds on the safety probability.

The literature discussed above considers neural networks in isolation, which can provide input-output guarantees on a NN but are unable to reason holistically about the safety of the system that the NN is applied in. Verification methods that concern the safety of NNs interlinked with a system require different approaches than standalone NN verification, which we will discuss in the rest of this section.

Finite time horizon safety of BNN policies The work in Michelmore et al. [2020] extends the method of Cardelli et al. [2019] to verifying safety in closed-loop systems with BNN policies. However, similar to the standalone setting of Cardelli et al. [2019], their method obtains only statistical guarantees on the safety probability and for the system's execution over a finite time horizon.

Safe RL Safe reinforcement learning (RL) has been primarily studied in the form of constrained Markov decision processes (CMDPs) [Altman, 1999, Geibel, 2006]. Compared to standard MDPs, an agent acting in a CMDP must satisfy an expected auxiliary cost term aggregated over an episode. The CMDP framework has been the base of several RL algorithms [Uchibe and Doya, 2007], notably the Constrained Policy Optimization (CPO) [Achiam et al., 2017]. Despite these algorithms providing a decent performance, the key limitation of CMDPs is that the constraint is satisfied in expectation, which makes violations unlikely but nonetheless possible. Consequently, the CMDP framework is unsuited for systems where constraint violations are critical.

Lyapunov-based stability Safety in the context of "stability", i.e. always returning to a ground state, can be proved by Lyapunov functions [Berkenkamp et al., 2017]. Lyapunov functions have originally been considered to study stability of dynamical systems [Khalil and Grizzle, 1996]. Intuitively, a Lyapunov function assigns a non-negative value to each state, and is required to decrease with respect to the system's dynamics at any state outside of the stable set. A Lyapunov-based method is proposed in Chow et al. [2018] to ensure safety in CMDPs during training. Recently, the work of Chang et al. [2019] presented a method for learning a policy as well as a neural network Lyapunov function which guarantees the stability of the policy. Similarly to our work, their learning procedure is counterexample-based. However, unlike Chang et al. [2019], our work considers BNN policies and safety definitions that do not require returning to a set of ground states.

Barrier functions for dynamical systems Barrier functions can be used to prove infinite time horizon safety in dynamical systems [Prajna and Jadbabaie, 2004, Prajna et al., 2007]. Recent works have considered learning neural network barrier functions [Zhao et al., 2020], and a counterexample-based learning procedure is presented in Peruffo et al. [2021].

Finite time horizon safety of NN policies Safety verification of continuous-time closed-loop systems with deterministic NN policies has been considered in Ivanov et al. [2019], Gruenbacher et al. [2020, 2021, 2022], which reduces safety verification to the reachability analysis in hybrid systems [Chen et al., 2013]. The work of Dutta et al. [2019] presents a method which computes a polynomial approximation of the NN policy to allow an efficient approximation of the reachable state set. Both works consider finite time horizon systems.

Our safety certificate most closely resembles inductive invariants for safety analysis in programs [Floyd, 1967] and positive invariants for dynamical systems [Blanchini and Miani, 2008].

5.2 Preliminaries and Problem Statement

We consider a discrete-time dynamical system

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t),\, \mathbf{x}_0 \in \mathcal{X}_0.$$

The dynamics are defined by the function $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ where $\mathcal{X} \subseteq \mathbb{R}^m$ is the state space and $\mathcal{U} \subseteq \mathbb{R}^n$ is the control action space, $\mathcal{X}_0 \subseteq \mathcal{X}$ is the set of initial states and $t \in \mathbb{N}_{\geq 0}$ denotes a discretized time. At each time step t, the action is defined by the (possibly probabilistic) positional policy $\pi : \mathcal{X} \to \mathcal{D}(\mathcal{U})$, which maps the current state $\mathbf{x}_t$ to a distribution $\pi(\mathbf{x}_t) \in \mathcal{D}(\mathcal{U})$ over the set of actions. We use $\mathcal{D}(\mathcal{U})$ to denote the set of all probability distributions over $\mathcal{U}$. The next action is then sampled according to $\mathbf{u}_t \sim \pi(\mathbf{x}_t)$, and together with the current state $\mathbf{x}_t$ of the system gives rise to the next state $\mathbf{x}_{t+1}$ of the system according to the dynamics f. Thus, the dynamics f together with the policy π form a closed-loop system (or a feedback loop system). The aim of the policy is to maximize the expected cumulative reward (possibly discounted) from each starting state, i.e., $\sum_{i=1}^{\infty} r_t \gamma^i$ where r_t is a reward value provided by the environment in step t and $0 < \gamma < 1$ a discount factor. Given a set of initial states $\mathcal{X}_0$ of the system, we say that a sequence of state-action pairs $(\mathbf{x}_t, \mathbf{u}_t)_{t=0}^{\infty}$ is a trajectory if $\mathbf{x}_0 \in \mathcal{X}_0$ and we have $\mathbf{u}_t \in$ supp$(\pi(\mathbf{x}_t))$ and $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$ for each $t \in \mathbb{N}_{\geq 0}$.

A neural network (NN) is a function $g : \mathbb{R}^m \to \mathbb{R}^n$ that consists of several sequentially composed layers $g = l_1 \circ \ldots \circ l_k$. A NN induces a policy by mapping each system state to a Dirac-delta distribution which picks a single action with probability 1. Each layer l_i for $i = 1, \ldots k$ is parametrized by learned weight values of the appropriate dimensions and an activation function a_i,

$$l_i(\mathbf{x}) = a_i(\mathbf{W}_i \mathbf{x} + \mathbf{b}_i), \mathbf{W}_i \in \mathbb{R}^{n_i \times n_{i-1}}, \mathbf{b}_i \in \mathbb{R}^{n_i},$$

where $\mathbf{W}_i$ and $\mathbf{b}_i$ denote the weights of layer i, n_0 and n_i correspond to the input dimension and the dimension of layer i respectively. In this work, we consider ReLU activation functions $a_i(\mathbf{x}) = \mathrm{ReLU}(\mathbf{x}) = \max\{\mathbf{x}, \mathbf{0}\}$, although other piecewise linear activation such as the leaky-ReLU [Jarrett et al., 2009] and PReLU [He et al., 2015] are applicable as well. Note that for the last layer k, no activation function is applied, i.e., $a_k(\mathbf{x}) = \mathbf{x}$.

In Bayesian neural networks (BNNs), weights are random variables and their values are sampled, each according to some distribution. Then each vector of sampled weights gives rise to a (deterministic) neural network. Given a training set $\mathcal{D}$, in order to train the BNN we assume a prior distribution $p(\mathbf{w}, \mathbf{b})$ over the weights. The learning then amounts to computing the posterior distribution $p(\mathbf{w}, \mathbf{b} \mid \mathcal{D})$ via the application of the Bayes rule. As analytical inference of the posterior is in general infeasible due to non-linearity introduced by the BNN architecture [MacKay, 1992, Hinton and Van Camp, 1993], practical training algorithms rely on approximate inference, e.g. Hamiltonian Monte Carlo [Neal, 2012], Langevin dynamics [Welling and Teh, 2011], variational inference [Blundell et al., 2015], dropout [Gal and Ghahramani, 2016], and stochastic weight averaging [Maddox et al., 2019]. In this work we consider BNN where the weight posterior $p(\mathbf{w}, \mathbf{b} \mid \mathcal{D})$ is approximated parametrically by independent Gaussians distributions, e.g., such as in Blundell et al. [2015] and Maddox et al. [2019]. Nonetheless, our approach of computing safe weight sets as subsets of the weight posteriors' support can be naturally extended to other forms of weight posteriors.

When the policy in a dynamical system is a BNN, the policy maps each system state $\mathbf{x}_t$ to a probability distribution $\pi(\mathbf{x}_t)$ over the action space. This distribution is defined implicitly by the following sampling process. First, BNN weights $\mathbf{w}$, $\mathbf{b}$ are sampled according to the posterior BNN weight distribution, and the sampled weights give rise to a deterministic NN policy $g_{\mathbf{w},\mathbf{b}}$. The action of the system is then defined as $\mathbf{u}_t = g_{\mathbf{w},\mathbf{b}}(\mathbf{x}_t)$.

Problem statement We now define the two safety problems that we consider in this work. The first problem considers feed-forward BNNs, and the second problem considers closed-loop systems with BNN policies. While our solution to the first problem will be a subprocedure in our solution to the second problem, the reason why we state it as a separate problem is that we believe that our solution to the first problem is also of independent interest for the safety analysis of feed-forward BNNs.

Let π be a BNN. Suppose that the vector $(\mathbf{w}, \mathbf{b})$ of BNN weights in π has dimension $p + q$, where p is the dimension of $\mathbf{w}$ and q is the dimension of $\mathbf{b}$. For each $1 \leq i \leq p$, let μ_i denote the mean of the random variable w_i. Similarly, for each $1 \leq i \leq q$, let μ_{p+i} denote the mean of the random variable b_i. Then, for each $\epsilon \in [0, \infty]$, we define the set W_ϵ^π of weight vectors via

$$W_\epsilon^\pi = \prod_{i=1}^{p+q} [\mu_i - \epsilon, \mu_i + \epsilon] \subseteq \mathbb{R}^{p+q}.$$

We now proceed to defining our safety problem for feed-forward BNNs. Suppose that we are given a feed-forward BNN π, a set $\mathcal{X}_0 \subseteq \mathbb{R}^m$ of input points and a set $\mathcal{X}_u \subseteq \mathbb{R}^n$ of unsafe (or bad) output points. For a concrete vector $(\mathbf{w}, \mathbf{b})$ of weight values, let $g_{\mathbf{w},\mathbf{b}}$ to be the (deterministic) NN defined by these weight values. We say that $g_{\mathbf{w},\mathbf{b}}$ is safe if for each $\mathbf{x} \in \mathcal{X}_0$ we have $g_{\mathbf{w},\mathbf{b}}(\mathbf{x}) \notin \mathcal{X}_u$, i.e. if evaluating $g_{\mathbf{w},\mathbf{b}}$ on all input points does not lead to an unsafe output.

Problem 1 (Feed-forward BNNs). *Let π be a feed-forward BNN with independent Gaussian weight distributions, $\mathcal{X}_0 \subseteq \mathbb{R}^m$ a set of input points and $\mathcal{X}_u \subseteq \mathbb{R}^n$ a set of unsafe output points. Let $\epsilon \in [0, \infty]$. Determine whether each deterministic NN in $\{g_{\mathbf{w},\mathbf{b}} \mid (\mathbf{w}, \mathbf{b}) \in W_\epsilon^\pi\}$ is safe.*

Next, we define our safety problem for closed-loop systems with BNN policies. Consider a closed-loop system defined by a dynamics function f, a BNN policy π and an initial set of states $\mathcal{X}_0$. Let $\mathcal{X}_u \subseteq \mathcal{X}$ be a set of unsafe (or bad) states. We say that a trajectory $(\mathbf{x}_t, \mathbf{u}_t)_{t=0}^\infty$ is safe if $\mathbf{x}_t \notin \mathcal{X}_u$ for all $t \in \mathbb{N}_0$, hence if it does not reach any unsafe states. Note that this definition implies infinite time horizon safety of the trajectory. Given $\epsilon \in [0, \infty]$, define the set $\mathsf{Traj}_\epsilon^{f,\pi}$ to be the set of all system trajectories in which each sampled weight vector belongs to W_ϵ^π.

Problem 2 (Closed-loop systems with BNN policies)**.** *Consider a closed-loop system defined by a dynamics function f, a BNN policy with independent Gaussian weight distributions π and a set of initial states $\mathcal{X}_0$. Let $\mathcal{X}_u$ be a set of unsafe states. Let $\epsilon \in [0, \infty]$. Determine whether each trajectory in* $\mathsf{Traj}_\epsilon^{f,\pi}$ *is safe.*

Note that the question of whether a BNN policy π with independent Gaussian weight distributions is safe (i.e. whether each trajectory of the system is safe) is a special case of the above problem which corresponds to $\epsilon = \infty$.

5.3 Main Results

In this section we present our method for solving the safety problems defined in the previous section, with Section 5.3.1 considering Problem 1 and Section 5.3.2 considering Problem 2. Both problems consider safety verification with respect to a given value of $\epsilon \in [0, \infty]$, so in Section 5.3.3 we present our method for computing the value of ϵ for which our solutions to Problem 1 and Problem 2 may be used to verify safety. We then show in Section 5.3.4 how our new methodology can be adapted to the safe exploration RL setting.

5.3.1 Safe weight sets for feed-forward BNNs

Consider a feed-forward BNN π with Gaussian weight distributions, a set $\mathcal{X}_0 \subseteq \mathbb{R}^m$ of inputs and a set $\mathcal{X}_u \subseteq \mathbb{R}^n$ of unsafe output of the BNN. Fix $\epsilon \in [0, \infty]$. To solve Problem 1, we show that the decision problem of whether each deterministic NN in $\{g_{\mathbf{w},\mathbf{b}} \mid (\mathbf{w}, \mathbf{b}) \in W_\epsilon^\pi\}$ is safe can be encoded as a system of constraints and reduced to constraint solving.

Let g be the neural network that underlines the BNN π. Suppose $g = l_1 \circ \ldots \circ l_k$ consists of k layers, with each $l_i(\mathbf{x}) = \mathrm{ReLU}(\mathbf{W}_i\mathbf{x} + \mathbf{b}_i)$. Denote by $\mathbf{M}_i$ the matrix of the same dimension as $\mathbf{W}_i$, with each entry equal to the mean of the corresponding random variable weight in $\mathbf{W}_i$. Similarly, define the vector $\mathbf{q}_i$ of means of random variables in $\mathbf{b}_i$. The real variables of our system of constraints are as follows, each of appropriate dimension:

1. $\mathbf{x}_0$ encodes the BNN inputs, $\mathbf{x}_l$ encodes the BNN outputs;
2. $\mathbf{x}_1^{\mathrm{in}}, \ldots, \mathbf{x}_{l-1}^{\mathrm{in}}$ encode vectors of input values of each neuron in the hidden layers;
3. $\mathbf{x}_1^{\mathrm{out}}, \ldots, \mathbf{x}_{l-1}^{\mathrm{out}}$ encode vectors of output values of each neuron in the hidden layers;
4. $\mathbf{x}_{0,\mathrm{pos}}$ and $\mathbf{x}_{0,\mathrm{neg}}$ are dummy variable vectors of the same dimension as $\mathbf{x}_0$ and which will be used to distinguish between positive and negative NN inputs in $\mathbf{x}_0$, respectively.

We use $\mathbf{1}$ to denote the vector/matrix whose all entries are equal to 1, of appropriate dimensions defined by the formula in which it appears. Our system of constraints is as follows:

$$\mathbf{x}_0 \in \mathcal{X}_0, \quad \mathbf{x}_l \in \mathcal{X}_u \qquad \text{(Input-output conditions)}$$

$$\mathbf{x}_i^{\mathrm{out}} = \mathrm{ReLU}(\mathbf{x}_i^{\mathrm{in}}), \text{ for each } 1 \leq i \leq l-1 \qquad \text{(ReLU encoding)}$$

$$\begin{aligned}(\mathbf{M}_i - \epsilon \cdot \mathbf{1})\mathbf{x}_i^{\mathrm{out}} + (\mathbf{q}_i - \epsilon \cdot \mathbf{1}) \leq \mathbf{x}_{i+1}^{\mathrm{in}}, \text{ for each } 1 \leq i \leq l-1 \\ \mathbf{x}_{i+1}^{\mathrm{in}} \leq (\mathbf{M}_i + \epsilon \cdot \mathbf{1})\mathbf{x}_i^{\mathrm{out}} + (\mathbf{q}_i + \epsilon \cdot \mathbf{1}), \text{ for each } 1 \leq i \leq l-1\end{aligned} \qquad \text{(BNN hidden layers)}$$

$$\begin{aligned}
&\mathbf{x}_{0,\text{pos}} = \text{ReLU}(\mathbf{x}_0), \quad \mathbf{x}_{0,\text{neg}} = -\text{ReLU}(-\mathbf{x}_0) \\
&(\mathbf{M}_0 - \epsilon \cdot \mathbf{1})\mathbf{x}_{0,\text{pos}} + (\mathbf{M}_0 + \epsilon \cdot \mathbf{1})\mathbf{x}_{0,\text{neg}} + (\mathbf{q}_0 - \epsilon \cdot \mathbf{1}) \leq \mathbf{x}_1^{\text{in}} \qquad \text{(BNN input layer)} \\
&\mathbf{x}_1^{\text{in}} \leq (\mathbf{M}_0 + \epsilon \cdot \mathbf{1})\mathbf{x}_0^{\text{out}} + (\mathbf{M}_0 - \epsilon \cdot \mathbf{1})\mathbf{x}_{0,\text{neg}} + (\mathbf{q}_0 + \epsilon \cdot \mathbf{1})
\end{aligned}$$

Denote by $\Phi(\pi, \mathcal{X}_0, \mathcal{X}_u, \epsilon)$ the system of constraints defined above. The proof of Theorem 2 shows that it encodes that $\mathbf{x}_0 \in \mathcal{X}_0$ is an input point for which the corresponding output point of $g_{\mathbf{w},\mathbf{b}}$ is unsafe, i.e. $\mathbf{x}_l = g_{\mathbf{w},\mathbf{b}}(\mathbf{x}_0) \in \mathcal{X}_u$. The first equation encodes the input and output conditions. The second equation encodes the ReLU input-output relation for each hidden layer neuron. The remaining equations encode the relation between neuron values in successive layers of the BNN as well as that the sampled BNN weight vector is in W_ϵ^π. For hidden layers, we know that the output value of each neuron is nonnegative, i.e. $\mathbf{x}_i^{\text{out}} \geq \mathbf{0}$ for the i-th hidden layer where $1 \leq i \leq l-1$, and so

$$(\mathbf{M}_i - \epsilon \cdot \mathbf{1})\mathbf{x}_i^{\text{out}} \leq (\mathbf{M}_i + \epsilon \cdot \mathbf{1})\mathbf{x}_i^{\text{out}}.$$

Hence, the BNN weight relation with neurons in the successive layer as well as the fact that the sampled weights are in W_ϵ^π is encoded as in equations (3)-(4) above. For the input layer, however, we do not know the signs of the input neuron values $\mathbf{x}_0$ and so we introduce dummy variables $\mathbf{x}_{0,\text{pos}} = \text{ReLU}(\mathbf{x}_0)$ and $\mathbf{x}_{0,\text{neg}} = -\text{ReLU}(-\mathbf{x}_0)$ in equation (5). This allows encoding the BNN weight relations between the input layer and the first hidden layer as well as the fact that the sampled weight vector is in W_ϵ^π, as in equations (6)-(7). Theorem 2 shows that Problem 1 is equivalent to solving the system of constraints $\Phi(\pi, \mathcal{X}_0, \mathcal{X}_u, \epsilon)$.

Theorem 2. *Let $\epsilon \in [0, \infty]$. Then each deterministic NN in $\{g_{\mathbf{w},\mathbf{b}} \mid (\mathbf{w}, \mathbf{b}) \in W_\epsilon^\pi\}$ is safe if and only if the system of constraints $\Phi(\pi, \mathcal{X}_0, \mathcal{X}_u, \epsilon)$ is not satisfiable.*

Proof. We prove the equivalent claim that there exists a weight vector $(\mathbf{w}, \mathbf{b}) \in W_\epsilon^\pi$ for which $g_{\mathbf{w},\mathbf{b}}$ is unsafe if and only if $\Phi(\pi, \mathcal{X}_0, \mathcal{X}_u, \epsilon)$ is satisfiable.

First, suppose that there exists a weight vector $(\mathbf{w}, \mathbf{b}) \in W_\epsilon^\pi$ for which $g_{\mathbf{w},\mathbf{b}}$ is unsafe and we want to show that $\Phi(\pi, \mathcal{X}_0, \mathcal{X}_u, \epsilon)$ is satisfiable. This direction of the proof is straightforward since values of the network's neurons on the unsafe input give rise to a solution of $\Phi(\pi, \mathcal{X}_0, \mathcal{X}_u, \epsilon)$. Indeed, by assumption there exists a vector of input neuron values $\mathbf{x}_0 \in \mathcal{X}_0$ for which the corresponding vector of output neuron values $\mathbf{x}_l = g_{\mathbf{w},\mathbf{b}}(\mathbf{x}_0)$ is unsafe, i.e. $\mathbf{x}_l \in \mathcal{X}_u$. By defining $\mathbf{x}_i^{\text{in}}$, $\mathbf{x}_i^{\text{out}}$ to be the vectors of the corresponding input and output neuron values for the i-th hidden layer for each $1 \leq i \leq l-1$ and by setting $\mathbf{x}_{0,\text{pos}} = \text{ReLU}(\mathbf{x_0})$ and $\mathbf{x}_{0,\text{neg}} = -\text{ReLU}(-\mathbf{x}_0)$, we easily see that these variable values satisfy the Input-output conditions, the ReLU encoding conditions and the BNN input and hidden layer conditions, therefore we get a solution to the system of constraints $\Phi(\pi, \mathcal{X}_0, \mathcal{X}_u, \epsilon)$.

We now proceed to the more involved direction of this proof and show that any solution to the system of constraints $\Phi(\pi, \mathcal{X}_0, \mathcal{X}_u, \epsilon)$ gives rise to weights $(\mathbf{w}, \mathbf{b}) \in W_\epsilon^\pi$ for which $g_{\mathbf{w},\mathbf{b}}$ is unsafe. Let $\mathbf{x}_0$, $\mathbf{x}_l$, $\mathbf{x}_{0,\text{pos}}$, $\mathbf{x}_{0,\text{neg}}$ and $\mathbf{x}_i^{\text{in}}$, $\mathbf{x}_i^{\text{out}}$ for $1 \leq i \leq l-1$, be real vectors that satisfy the system of constraints $\Phi(\pi, \mathcal{X}_0, \mathcal{X}_u, \epsilon)$. Fix $1 \leq i \leq l-1$. From the BNN hidden layers constraint for layer i, we have

$$(\mathbf{M}_i - \epsilon \cdot \mathbf{1})\mathbf{x}_i^{\text{out}} + (\mathbf{q}_i - \epsilon \cdot \mathbf{1}) \leq \mathbf{x}_{i+1}^{\text{in}} \leq (\mathbf{M}_i + \epsilon \cdot \mathbf{1})\mathbf{x}_i^{\text{out}} + (\mathbf{q}_i + \epsilon \cdot \mathbf{1}). \tag{5.1}$$

We show that there exist values $\mathbf{W}_i^*, \mathbf{b}_i^*$ of BNN weights between layers i and $i+1$ such that each weight value is at most ϵ apart from its mean, and such that $\mathbf{x}_{i+1}^{\text{in}} = \mathbf{W}_i^* \mathbf{x}_i^{\text{out}} + \mathbf{b}_i^*$. Indeed, to formally show this, we define $W_\epsilon^\pi[i]$ to be the set of all weight vectors between layers i and $i+1$ such that each weight value is distant from its mean by at most ϵ (hence, $W_\epsilon^\pi[i]$ is a projection of W_ϵ^π onto dimensions that correspond to the weights between layers i and $i+1$). We then consider a continuous function $h_i : W_\epsilon^\pi[i] \to \mathbb{R}$ defined via

$$h_i(\mathbf{W}_i, \mathbf{b}_i) = \mathbf{W}_i \mathbf{x}_i^{\text{out}} + \mathbf{b}_i.$$

Since $W_\epsilon^\pi[i] \subseteq \mathbb{R}^{m_i \times n_i} \times \mathbb{R}^{n_i}$ is a product of intervals and therefore a connected set w.r.t. the Euclidean metric and since h_i is continuous, the image of $W_\epsilon^\pi[i]$ under h_i is also connected in $\mathbb{R}$. But note that

$$h_i(\mathbf{M}_i - \epsilon \cdot \mathbf{1}, \mathbf{q}_i - \epsilon \cdot \mathbf{1}) = (\mathbf{M}_i - \epsilon \cdot \mathbf{1})\mathbf{x}_i^{\text{out}} + (\mathbf{q}_i - \epsilon \cdot \mathbf{1})$$

and

$$h_i(\mathbf{M}_i + \epsilon \cdot \mathbf{1}, \mathbf{q}_i + \epsilon \cdot \mathbf{1}) = (\mathbf{M}_i + \epsilon \cdot \mathbf{1})\mathbf{x}_i^{\text{out}} + (\mathbf{q}_i + \epsilon \cdot \mathbf{1}),$$

with $(\mathbf{M}_i - \epsilon \cdot \mathbf{1}, \mathbf{q}_i - \epsilon \cdot \mathbf{1}), (\mathbf{M}_i + \epsilon \cdot \mathbf{1}, \mathbf{q}_i + \epsilon \cdot \mathbf{1}) \in W_\epsilon^\pi[i]$. Thus, for the two points to be connected, the image set must also contain $\mathbf{x}_{i+1}^{\text{in}}$ which lies in between by eq. (5.1). Thus, there exists $(\mathbf{W}_i^*, \mathbf{b}_i^*) \in W_\epsilon^\pi[i]$ with $\mathbf{x}_{i+1}^{\text{in}} = \mathbf{W}_i^* \mathbf{x}_i^{\text{out}} + \mathbf{b}_i^*$, as desired.

For the input and the first hidden layer, from the BNN input layer constraint we know that

$$(\mathbf{M}_0 - \epsilon \cdot \mathbf{1})\mathbf{x}_{0,\text{pos}} + (\mathbf{M}_0 + \epsilon \cdot \mathbf{1})\mathbf{x}_{0,\text{neg}} + (\mathbf{q}_0 - \epsilon \cdot \mathbf{1}) \le \mathbf{x}_1^{\text{in}} \le (\mathbf{M}_0 + \epsilon \cdot \mathbf{1})\mathbf{x}_{0,\text{pos}} + (\mathbf{M}_0 - \epsilon \cdot \mathbf{1})\mathbf{x}_{0,\text{neg}} + (\mathbf{q}_0 + \epsilon \cdot \mathbf{1}).$$

Again, define $W_\epsilon^\pi[0]$ to be the set of all weight vectors between the input and the first hidden layer such that each weight value is distant from its mean by at most ϵ. Consider a continuous function $h_0 : W_\epsilon^\pi[0] \to \mathbb{R}$ defined via

$$h_0(\mathbf{W}_0, \mathbf{b}_0) = \mathbf{W}_0 \mathbf{x}_0 + \mathbf{b}_0.$$

Let $\mathrm{Msign}(\mathbf{x}_0)$ be a matrix of the same dimension as $\mathbf{M}_0$, with each column consisting of 1's if the corresponding component of $\mathbf{x}_0$ is nonnegative, and of -1's if it is negative. Then note that

$$h_0(\mathbf{M}_0 - \epsilon \cdot \mathrm{Msign}(\mathbf{x}_0), \mathbf{q}_0 - \epsilon \cdot \mathbf{1}) = (\mathbf{M}_0 - \epsilon \cdot \mathbf{1})\mathbf{x}_{0,\text{pos}} + (\mathbf{M}_0 + \epsilon \cdot \mathbf{1})\mathbf{x}_{0,\text{neg}} + (\mathbf{q}_0 - \epsilon \cdot \mathbf{1})$$

and

$$h_0(\mathbf{M}_0 + \epsilon \cdot \mathrm{Msign}(\mathbf{x}_0), \mathbf{q}_0 + \epsilon \cdot \mathbf{1}) = (\mathbf{M}_0 + \epsilon \cdot \mathbf{1})\mathbf{x}_{0,\text{pos}} + (\mathbf{M}_0 - \epsilon \cdot \mathbf{1})\mathbf{x}_{0,\text{neg}} + (\mathbf{q}_0 + \epsilon \cdot \mathbf{1}).$$

Since $(\mathbf{M}_0 - \epsilon \cdot \mathrm{Msign}(\mathbf{x}_0), \mathbf{q}_0 - \epsilon \cdot \mathbf{1}), (\mathbf{M}_0 + \epsilon \cdot \mathrm{Msign}(\mathbf{x}_0), \mathbf{q}_0 + \epsilon \cdot \mathbf{1}) \in W_\epsilon^\pi[0]$, analogous image connectedness argument as the one above shows that there exist values $\mathbf{W}_0^*, \mathbf{b}_0^*$ of BNN weights such tha $(\mathbf{W}_0^*, \mathbf{b}_0^*) \in W_\epsilon^\pi[0]$, and such that $\mathbf{x}_1^{\text{in}} = \mathbf{W}_0^* \mathbf{x}_0 + \mathbf{b}_0^*$.

But now, collecting $\mathbf{W}_0^*, \mathbf{b}_0^*$ and $\mathbf{W}_i^*, \mathbf{b}_i^*$ for $1 \le i \le l-1$ gives rise to a BNN weight vector $(\mathbf{W}^*, \mathbf{b}^*)$ which is contained in W_ϵ^π. Furthermore, combining what we showed above with the constraints in $\Phi(\pi, \mathcal{X}_0, \mathcal{X}_u, \epsilon)$, we get that:

1. $\mathbf{x}_0 \in \mathcal{X}_0$, $\mathbf{x}_l \in \mathcal{X}_u$, from the Input-output condition in $\Phi(\pi, \mathcal{X}_0, \mathcal{X}_u, \epsilon)$;
2. $\mathbf{x}_i^{\text{out}} = \mathrm{ReLU}(\mathbf{x}_i^{\text{in}})$ for each $1 \le i \le l-1$, from the ReLU-encoding;

3. $\mathbf{x}_1^{\text{in}} = \mathbf{W}_0^* \mathbf{x}_0 + \mathbf{b}_0^*$ and $\mathbf{x}_{i+1}^{\text{in}} = \mathbf{W}_i^* \mathbf{x}_i^{\text{out}} + \mathbf{b}_i^*$ for each $1 \leq i \leq l-1$, as shown above.

Hence, $\mathbf{x}_l \in \mathcal{X}_u$ is the vector of neuron output values of $\pi_{\mathbf{W}^*,\mathbf{b}^*}$ on the input neuron values $\mathbf{x}_0 \in \mathcal{X}_0$, so as $(\mathbf{W}^*, \mathbf{b}^*) \in W_\epsilon^\pi$ we conclude that there exists a deterministic NN in $\{g_{\mathbf{w},\mathbf{b}} \mid (\mathbf{w}, \mathbf{b}) \in W_\epsilon^\pi\}$ which is not safe. This concludes the proof. $\square$

Solving the constraints Observe that ϵ, $\mathbf{M}_i$ and $\mathbf{q}_i$, $1 \leq i \leq l-1$, are constant values that are known at the time of constraint encoding. Thus, in $\Phi(\pi, \mathcal{X}_0, \mathcal{X}_u, \epsilon)$, only the ReLU constraints and possibly the input-output conditions are not linear. Depending on the form of $\mathcal{X}_0$ and $\mathcal{X}_u$ and on how we encode the ReLU constraints, we may solve the system $\Phi(\pi, \mathcal{X}_0, \mathcal{X}_u, \epsilon)$ in several ways:

1. **MILP.** It is shown in Lomuscio and Maganti [2017], Dutta et al. [2018], Tjeng et al. [2019] that the ReLU relation between two real variables can be encoded via mixed-integer linear constraints (MILP) by introducing 0/1-integer variables to encode whether a given neuron is active or inactive. Hence, if $\mathcal{X}_0$ and $\mathcal{X}_u$ are given by linear constraints, we may solve $\Phi(\pi, \mathcal{X}_0, \mathcal{X}_u, \epsilon)$ by a MILP solver. The ReLU encoding requires that each neuron value is bounded, which is ensured if $\mathcal{X}_0$ is a bounded set and if $\epsilon < \infty$.

2. **Reluplex.** In order to allow unbounded $\mathcal{X}_0$ and $\epsilon = \infty$, we may use algorithms based on the Reluplex calculus [Katz et al., 2017, 2019] to solve $\Phi(\pi, \mathcal{X}_0, \mathcal{X}_u, \epsilon)$. Reluplex is an extension of the standard simplex algorithm for solving systems of linear constraints, designed to allow ReLU constraints as well. While Reluplex does not impose the boundedness condition, it is in general less scalable than MILP-solving.

3. **NRA-SMT.** Alternatively, if $\mathcal{X}_0$ or $\mathcal{X}_u$ are given by non-linear constraints we may solve them by using an NRA-SMT-solver (non-linear real arithmetic satisfiability modulo theory), e.g. dReal [Gao et al., 2012]. To use an NRA-SMT-solver, we can replace the integer 0/1-variables of the ReLU neuron relations encoding with real variables that satisfy the constraint $x(x-1) = 0$. While NRA-SMT is less scalable compared to MILP, we note that it has been used in previous works on RL stability verification [Chang et al., 2019].

Safety via rejection sampling As discussed above, once the safety of NNs in $\{g_{\mathbf{w},\mathbf{b}} \mid (\mathbf{w}, \mathbf{b}) \in W_\epsilon^\pi\}$ has been verified, we can "re-calibrate" the BNN to reject sampled weights which are not in W_ϵ^π. Hence, rejection sampling gives rise to a safe BNN.

5.3.2 Safe weight sets for closed-loop systems with BNN Policies

Now consider a closed-loop system with a dynamics function $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ with $\mathcal{X} \subseteq \mathbb{R}^m$ and $\mathcal{U} \subseteq \mathbb{R}^n$, a BNN policy π, an initial state set $\mathcal{X}_0 \subseteq \mathcal{X}$ and an unsafe state set $\mathcal{X}_u \subseteq \mathcal{X}$. Fix $\epsilon \in [0, \infty]$. In order to solve Problem 2 and verify safety of each trajectory contained in $\text{Traj}_\epsilon^{f,\pi}$, our method searches for a positive invariant-like safety certificate that we define below.

Positive invariants for safety A positive invariant in a dynamical system is a set of states which contains all initial states and which is closed under the system dynamics. These conditions ensure that states of all system trajectories are contained in the positive invariant. Hence, a positive invariant which does not contain any unsafe states can be used to certify safety of every trajectory over infinite time horizon. In this work, however, we are not trying to prove safety of every trajectory, but only of those trajectories contained in $\mathsf{Traj}_\epsilon^{f,\pi}$. To that end, we define W_ϵ^π-safe positive invariants. Intuitively, a W_ϵ^π-safe positive invariant is required to contain all initial states, to be closed under the dynamics f and the BNN policy π when the sampled weight vector is in W_ϵ^π, and not to contain any unsafe state.

Definition 5 (W_ϵ^π-safe positive invariants)**.** *A set* $\mathsf{Inv} \subseteq \mathcal{X}$ *is said to be a* W_ϵ^π-positive invariant *if* $\mathcal{X}_0 \subseteq \mathsf{Inv}$*, for each* $\mathbf{x} \in \mathsf{Inv}$ *and* $(\mathbf{w}, \mathbf{b}) \in W_\epsilon^\pi$ *we have that* $f(\mathbf{x}, g_{\mathbf{w},\mathbf{b}}(\mathbf{x})) \in \mathsf{Inv}$*, and* $\mathsf{Inv} \cap \mathcal{X}_u = \emptyset$*.*

Theorem 3 shows that W_ϵ^π-safe positive invariants can be used to verify safety of all trajectories in $\mathsf{Traj}_\epsilon^{f,\pi}$ in Problem 2.

Theorem 3. *If there exists a* W_ϵ^π*-safe positive invariant, then each trajectory in* $\mathsf{Traj}_\epsilon^{f,\pi}$ *is safe.*

Proof. Let Inv be a W_ϵ^π-safe positive invariant. Given a trajectory $(\mathbf{x}_t, \mathbf{u}_t)_{t=0}^\infty$ in $\mathsf{Traj}_\epsilon^{f,\pi}$, we need to show that $\mathbf{x}_t \notin \mathcal{X}_u$ for each $t \in \mathbb{N}_{\geq 0}$. Since $\mathsf{Inv} \cap \mathcal{X}_u = \emptyset$, it suffices to show that $\mathbf{x}_t \in \mathsf{Inv}$ for each $t \in \mathbb{N}_{\geq 0}$. We prove this by induction on t.

The base case $\mathbf{x}_0 \in \mathsf{Inv}$ follows since $\mathbf{x}_0 \in \mathcal{X}_0 \subseteq \mathsf{Inv}$. As an inductive hypothesis, suppose now that $\mathbf{x}_t \in \mathsf{Inv}$ for some $t \in \mathbb{N}_{\geq 0}$. We need to show that $\mathbf{x}_{t+1} \in \mathsf{Inv}$.

Since the trajectory is in $\mathsf{Traj}_\epsilon^{f,\pi}$, we know that the BNN weight vector $(\mathbf{w}_t, \mathbf{b}_t)$ sampled at the time-step t belongs to W_ϵ^π, i.e. $(\mathbf{w}_t, \mathbf{b}_t) \in W_\epsilon^\pi$. Thus, since $\mathbf{x}_t \in \mathsf{Inv}$ by the induction hypothesis and since Inv is closed under the system dynamics when the sampled weight vector is in W_ϵ^π, it follows that $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) = f(\mathbf{x}_t, g_{\mathbf{w}_t,\mathbf{b}_t}(\mathbf{x}_t)) \in \mathsf{Inv}$. This concludes the proof by induction. □

Learning positive invariants We now present a learning algorithm for a W_ϵ^π-safe positive invariant. It learns a neural network $g^{\mathsf{Inv}} : \mathbb{R}^m \to \mathbb{R}$, where the positive invariant is then defined as the set $\mathsf{Inv} = \{\mathbf{x} \in \mathcal{X} \mid g^{\mathsf{Inv}}(\mathbf{x}) \geq 0\}$. The pseudocode is given in Algorithm 5.1.

The algorithm first samples $\tilde{\mathcal{X}_0}$ from $\mathcal{X}_0$ and $\tilde{\mathcal{X}_u}$ from $\mathcal{X}_u$ and initializes the specification set D_{spec} to $\tilde{\mathcal{X}_u} \times \{0\} \cup \tilde{\mathcal{X}_0} \times \{1\}$ and the counterexample set D_{ce} to an empty set. Optionally, the algorithm also bootstraps the positive invariant network by initializing D_{spec} with random samples from the state space $\mathcal{X}$ labeled with Monte-Carlo estimates of reaching the unsafe states. The rest of the algorithm consists of two modules which are composed into a loop: the *learner* and the *verifier*. In each loop iteration, the learner first learns a W_ϵ^π-safe positive invariant candidate which takes the form of a neural network g^{Inv}. This is done by minimizing the loss function $\mathcal{L}$ that depends on D_{spec} and D_{ce}:

$$\mathcal{L}(g^{\mathsf{Inv}}) = \frac{1}{|D_{\text{spec}}|} \sum_{(\mathbf{x},y) \in D_{\text{spec}}} \mathcal{L}_{\text{cls}}(g^{\mathsf{Inv}}(\mathbf{x}), y) + \lambda \frac{1}{|D_{\text{ce}}|} \sum_{(\mathbf{x},\mathbf{x}') \in D_{\text{ce}}} \mathcal{L}_{\text{ce}}(g^{\mathsf{Inv}}(\mathbf{x}), g^{\mathsf{Inv}}(\mathbf{x}')), \quad (5.2)$$

where λ is a tuning parameter and $\mathcal{L}_{\text{cls}}$ a binary classification loss function, e.g. the 0/1-loss $\mathcal{L}_{0/1}(z, y) = \mathbb{1}[\mathbb{1}[z \geq 0] \neq y]$ or the logistic loss $\mathcal{L}_{\log}(z, y) = z - z \cdot y + \log(1 + \exp(-z))$ as its differentiable alternative. The term $\mathcal{L}_{\text{ce}}$ is the counterexample loss which we define via

$$\mathcal{L}_{\text{ce}}(z, z') = \mathbb{1}[z > 0]\mathbb{1}[z' < 0]\mathcal{L}_{\text{cls}}(z, 0)\mathcal{L}_{\text{cls}}(z', 1). \quad (5.3)$$

Intuitively, the first sum in eq. (5.2) forces g^{Inv} to be nonnegative at initial states and negative at unsafe states contained in D_{spec}, and the second term forces each counterexample in D_{ce} not to destroy the closedness of Inv under the system dynamics.

Algorithm 5.1: Learning algorithm for W_ϵ^π-safe positive invariants

```
Input Dynamics function $f$, BNN policy $\pi$, Initial state set $\mathcal{X}_0$, Unsafe state set $\mathcal{X}_u$, $\epsilon \in [0, \infty]$
$\tilde{\mathcal{X}}_0, \tilde{\mathcal{X}}_u \leftarrow$ random samples of $\mathcal{X}_0, \mathcal{X}_u$
$D_{\mathrm{spec}} \leftarrow \tilde{\mathcal{X}}_u \times \{0\} \cup \tilde{\mathcal{X}}_0 \times \{1\}$, $D_{\mathrm{ce}} \leftarrow \{\}$
Optional (bootstrapping): $S_{\mathrm{bootstrap}} \leftarrow$ sample finite trajectories with initial state sampled from $\mathcal{X}$

$D_{\mathrm{spec}} \leftarrow D_{\mathrm{spec}} \cup \{(\mathbf{x}, 0) | \exists s \in S_{\mathrm{bootstrap}}$ that starts in $\mathbf{x} \in \mathcal{X}$ and reaches $\mathcal{X}_u\}$

$\cup \{(\mathbf{x}, 1) | \exists s \in S_{\mathrm{bootstrap}}$ that starts in $\mathbf{x} \in \mathcal{X}$ and does not reach $\mathcal{X}_u\}$
Pre-train neural network $g^{\mathsf{Inv}}$ on datasets $D_{\mathrm{spec}}$ and $D_{\mathrm{ce}}$ with loss function $\mathcal{L}$
while timeout not reached do
  if $\exists(\mathbf{x}, \mathbf{x}', \mathbf{u}, \mathbf{w}, \mathbf{b})$ s.t. $g^{\mathsf{Inv}}(\mathbf{x}) \geq 0$, $g^{\mathsf{Inv}}(\mathbf{x}') < 0$, $(\mathbf{w}, \mathbf{b}) \in W_\epsilon^\pi$, $\mathbf{u} = g_{\mathbf{w},\mathbf{b}}(\mathbf{x})$, $\mathbf{x}' = f(\mathbf{x}, \mathbf{u})$ then
    $D_{\mathrm{ce}} \leftarrow D_{\mathrm{ce}} \cup \{(\mathbf{x}, \mathbf{x}')\}$
  else if $\exists(\mathbf{x})$ s.t. $\mathbf{x} \in \mathcal{X}_0$, $g^{\mathsf{Inv}}(\mathbf{x}) < 0$ then
    $D_{\mathrm{spec}} \leftarrow D_{\mathrm{spec}} \cup \{(\mathbf{x}, 1)\}$
  else if $\exists(\mathbf{x})$ s.t. $\mathbf{x} \in \mathcal{X}_u$, $g^{\mathsf{Inv}}(\mathbf{x}) \geq 0$ then
    $D_{\mathrm{spec}} \leftarrow D_{\mathrm{spec}} \cup \{(\mathbf{x}, 0)\}$
  else
    Return Safe
  end if
  Train neural network $g^{\mathsf{Inv}}$ on datasets $D_{\mathrm{spec}}$ and $D_{\mathrm{ce}}$ with loss function $\mathcal{L}$
end while
Return Unsafe
```

Once g^{Inv} is learned, the verifier checks whether Inv is indeed a W_ϵ^π-safe positive invariant. To do this, the verifier needs to check the three defining properties of W_ϵ^π-safe positive invariants:

1. *Closedness of* Inv *under system dynamics.* The verifier checks if there exist states $\mathbf{x} \in$ Inv, $\mathbf{x}' \notin$ Inv and a BNN weight vector $(\mathbf{w}, \mathbf{b}) \in W_\epsilon^\pi$ such that $f(\mathbf{x}, g_{\mathbf{w},\mathbf{b}}(\mathbf{x})) = \mathbf{x}'$. To do this, it introduces real variables $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^m$, $\mathbf{u} \in \mathbb{R}^n$ and $y, y' \in \mathbb{R}$, and solves:

$$\begin{aligned}
&\text{maximize } y - y' \text{ subject to} \\
&y \geq 0, y' < 0, y = g^{\mathsf{Inv}}(\mathbf{x}), y' = g^{\mathsf{Inv}}(\mathbf{x}') \\
&\mathbf{x}' = f(\mathbf{x}, \mathbf{u}) \\
&\mathbf{u} \text{ is an output of } g_{\mathbf{w},\mathbf{b}} \text{ on input } \mathbf{x} \text{ and weights } (\mathbf{w}, \mathbf{b}) \in W_\epsilon^\pi
\end{aligned}$$

The conditions $y = g^{\mathsf{Inv}}(\mathbf{x})$ and $y' = g^{\mathsf{Inv}}(\mathbf{x}')$ are encoded by using the existing techniques for encoding deterministic NNs as systems of MILP/Reluplex/NRA-SMT constraints. The condition in the third equation is encoded simply by plugging variable vectors $\mathbf{x}$ and $\mathbf{u}$ into the equation for f. Finally, for condition in the fourth equation we use our encoding from Section 5.3.1 where we only need to omit the Input-output condition. Note that the strict inequality $y' < 0$ may be slightly tightened as $y' \leq -\delta$ with some small slack $\delta > 0$ in case the constraint solving method does not support strict inequalities. The optimization objective is added in order to search for the "worst" counterexample. We note that MILP [Gurobi Optimization, 2021] and SMT [Gao et al., 2012] solvers allow optimizing linear objectives, and recently Reluplex algorithm [Katz et al., 2017] has also been extended to allow solving optimization problems [Strong et al., 2021]. If a counterexample $(\mathbf{x}, \mathbf{x}')$ is found, it is added to D_{ce} and the learner tries to learn a new candidate. If the system of constraints is unsatisfiable, the verifier proceeds to the second check.

2. *Non-negativity on $\mathcal{X}_0$.* The verifier checks if there exists $\mathbf{x} \in \mathcal{X}_0$ for which $g^{\mathsf{Inv}}(\mathbf{x}) < 0$. If such $\mathbf{x}$ is found, $(\mathbf{x}, 1)$ is added to D_{spec} and the learner then tries to learn a new candidate. If the system of constraints is unsatisfiable, the verifier proceeds to the third check.

3. *Negativity on $\mathcal{X}_u$.* The verifier checks if there exists $\mathbf{x} \in \mathcal{X}_u$ with $g^{\mathsf{Inv}}(\mathbf{x}) \geq 0$. If such $\mathbf{x}$ is found, $(\mathbf{x}, 0)$ is added to D_{spec} and the learner then tries to learn a new candidate. If the system of constraints is unsatisfiable, the veririfer concludes that Inv is a W_ϵ^π-positive invariant which does not contain any unsafe state and so each trajectory in $\mathsf{Traj}_\epsilon^{f,\pi}$ is safe.

Theorem 4 shows that neural networks f^{Inv} for which Inv is a W_ϵ^π-safe positive invariants are global minimizers of the loss function $\mathcal{L}$ with the 0/1-classification loss. Theorem 5 establishes the correctness of our algorithm.

Theorem 4. *The loss function $\mathcal{L}$ is nonnegative for any neural network g, i.e. $\mathcal{L}(g) \geq 0$. Moreover, if* Inv *is a W_ϵ^π-safe positive invariant and $\mathcal{L}_{cls}$ the 0/1-loss, then $\mathcal{L}(g^{\mathsf{Inv}}) = 0$. Hence, neural networks g^{Inv} for which* Inv *is a W_ϵ^π-safe positive invariant are global minimizers of the loss function $\mathcal{L}$ when $\mathcal{L}_{cls}$ is the 0/1-loss.*

Proof. Recall, the loss function $\mathcal{L}$ for a neural network g is defined via

$$\mathcal{L}(g) = \frac{1}{|D_{\text{spec}}|} \sum_{(\mathbf{x},y) \in D_{\text{spec}}} \mathcal{L}_{\text{cls}}(g(\mathbf{x}), y) + \lambda \frac{1}{|D_{\text{ce}}|} \sum_{(\mathbf{x},\mathbf{x}') \in D_{\text{ce}}} \mathcal{L}_{\text{ce}}(g(\mathbf{x}), g(\mathbf{x}')), \tag{5.4}$$

where λ is a tuning parameter and $\mathcal{L}_{\text{cls}}$ a binary classification loss function, e.g. the 0/1-loss $\mathcal{L}_{0/1}(z, y) = \mathbb{1}[\mathbb{1}[z \geq 0] \neq y]$ or the logistic loss $\mathcal{L}_{\log}(z, y) = z - z \cdot y + \log(1 + \exp(-z))$ as its differentiable alternative. The term $\mathcal{L}_{\text{ce}}$ is the counterexample loss which we define via

$$\mathcal{L}_{\text{ce}}(z, z') = \mathbb{1}[z > 0]\mathbb{1}[z' < 0]\mathcal{L}_{\text{cls}}(z, 0)\mathcal{L}_{\text{cls}}(z', 1). \tag{5.5}$$

The fact that $\mathcal{L}(g) \geq 0$ for each neural network g follows immediately from the fact that summands in the first sum in eq. (5.4) are loss functions which are nonnegative, and summands in the second sum are products of indicator and nonnegative loss functions and therefore also nonnegative.

We now show that, if $\mathcal{L}_{\text{cls}}$ is the 0/1-loss, $\mathcal{L}(g^{\mathsf{Inv}}) = 0$ whenever Inv is a W_ϵ^π-safe positive invariant, which implies the global minimization claim in the theorem. This follows from the following two items:

1. For each $(\mathbf{x}, y) \in D_{\text{spec}}$, we have $\mathcal{L}_{\text{cls}}(g^{\mathsf{Inv}}(\mathbf{x}), y) = 0$. Indeed, for $(\mathbf{x}, y)$ to be added to D_{spec} in Algorithm 1, we must have that either $\mathbf{x} \in \mathcal{X}_0$ and $y = 1$, or that $\mathbf{x} \in \mathcal{X}_u$ and $y = 0$. Thus, since Inv is assumed to be a W_ϵ^π-safe positive invariant, g^{Inv} correctly classifies $(\mathbf{x}, y)$ and the corresponding loss is 0.

2. For each $(\mathbf{x}, \mathbf{x}') \in D_{\text{ce}}$ we have $\mathcal{L}_{\text{ce}}(g^{\mathsf{Inv}}(\mathbf{x}), g^{\mathsf{Inv}}(\mathbf{x}')) = 0$. Indeed, since

$$\mathcal{L}_{\text{ce}}(g^{\mathsf{Inv}}(\mathbf{x}), g^{\mathsf{Inv}}(\mathbf{x}')) = \mathbb{1}[g^{\mathsf{Inv}}(\mathbf{x}) > 0]\mathbb{1}[g^{\mathsf{Inv}}(\mathbf{x}') < 0]\mathcal{L}_{\text{cls}}(g^{\mathsf{Inv}}(\mathbf{x}), 0)\mathcal{L}_{\text{cls}}(g^{\mathsf{Inv}}(\mathbf{x}'), 1),$$

   for the loss to be non-zero we must have that $g^{\mathsf{Inv}}(\mathbf{x}) \geq 0$ and $g^{\mathsf{Inv}}(\mathbf{x}) < 0$. But this is impossible since Inv is assumed to be a W_ϵ^π-safe positive invariant and $(\mathbf{x}, \mathbf{x}')$ was added by Algorithm 1 as a counterexample to D_{ce}, meaning that $\mathbf{x}'$ can be reached from $\mathbf{x}$ by following the dynamics function and sampling a BNN weight vector in W_ϵ^π. Therefore, by the closedness property of W_ϵ^π-safe positive invariants when the sampled weight vector is in W_ϵ^π, we cannot have both $g^{\mathsf{Inv}}(\mathbf{x}) \geq 0$ and $g^{\mathsf{Inv}}(\mathbf{x}) < 0$. Hence, the loss must be 0.

□

Theorem 5. *If the verifier in Algorithm 5.1 shows that constraints in three checks are unsatisfiable, then the computed* Inv *is indeed a W_ϵ^π-safe positive invariant. Hence, Algorithm 5.1 is correct.*

Proof. The fact that the first check in Algorithm 1 correctly checks whether there exist $\mathbf{x}, \mathbf{x}' \in \mathcal{X}_0$ and a weight vector $(\mathbf{w}, \mathbf{b}) \in W_\epsilon^\pi$ such that $\mathbf{x}' = f(\mathbf{x}, g_{\mathbf{w},\mathbf{b}}(\mathbf{x}))$ with $g^{\mathsf{Inv}}(\mathbf{x}) \geq 0$ and $g^{\mathsf{Inv}}(\mathbf{x}') < 0$ follows by the correctness of our encoding in Section 4.1, which was proved in Theorem 1. The fact that checks 2 and 3 correctly check whether for all $\mathbf{x} \in \mathcal{X}_0$ we have $g^{\mathsf{Inv}}(\mathbf{x}) \geq 0$ and for all $\mathbf{x} \in \mathcal{X}_u$ we have $g^{\mathsf{Inv}}(\mathbf{x}) < 0$, respectively, follows immediately from the conditions they encode.

Therefore, the three checks together verify that (1) Inv is closed under the system dynamics whenever the sampled weight vector is in W_ϵ^π, (2) Inv contains all initial states, and (3) Inv contains no unsafe states. As these are the 3 defining properties of W_ϵ^π-safe positive invariants, Algorithm 1 is correct and the theorem claim follows. □

Safety via rejection sampling As discussed above, once the safety of all trajectories in $\mathsf{Traj}_\epsilon^{f,\pi}$ has been verified, we can "re-calibrate" the BNN policy to reject sampled weights which are not in W_ϵ^π. Hence, rejection sampling gives rise to a safe BNN policy. Note that sampling and rejecting weight values jointly for all weights suffers from a rejection rate that increases exponentially with the number of weights. However, we can substantially speedup the rejection sampling process by exploiting the independence of the weight distribution and resampling only the subset of weights whose sample have actually landed outside the safe weight set.

5.3.3 Computation of safe weight sets and the value of ϵ

Problems 1 and 2 assume a given value of ϵ for which safety needs to be verified. In order to compute the largest value of ϵ for which our approach can verify safety, we start with a small value of ϵ and iteratively increase it until we reach a value that cannot be certified or until the timeout is reached, in order to compute as large safe weight set as possible. In particular, our Algorithm 5.1 starts with $\epsilon = 0$ and then increases ϵ according to a given schedule, e.g., $\epsilon \in \{0.1, 0.2, 0.5, 1, 1.5, 2, 3, 4, \dots\}$. The search for a largest ϵ terminates if the entire schedule has been successfully verified or a timeout signal is reached. In each iteration of the search for the largest ϵ, our Algorithm 5.1 does not start from scratch but is initialized with the g^{Inv} and D_{spec} from the previous successful iteration, i.e. attempting to

Algorithm 5.2: Safe Exploration Reinforcement Learning

```
Input Initial policy $\pi_0$, learning rate $\alpha$, number of iterations $N$
for $i \in 1, \dots N$ do
    $\epsilon \leftarrow$ find safe $\epsilon$ for $\pi_{i-1}$
    collect rollouts of $\pi_{i-1}$ with rejection sampling $\epsilon$
    compute $\nabla\mu_{i-1}$ for parameters $\mu_{i-1}$ of $\pi_{i-1}$ using DQN/policy gradient
    $\mu_i \leftarrow \mu_{i-1} - \alpha\nabla\mu_{i-1}$ (gradient descent)
    project $\mu_i$ back to interval $[\mu_{i-1} - \epsilon, \mu_{i-1} + \epsilon]$
end for
Return $\pi_N$
```

enlarge the current safe weight set. Our iterative process significantly speeds up the search process compared to naively restarting our algorithm in every iteration.

5.3.4 Safe exploration reinforcement learning

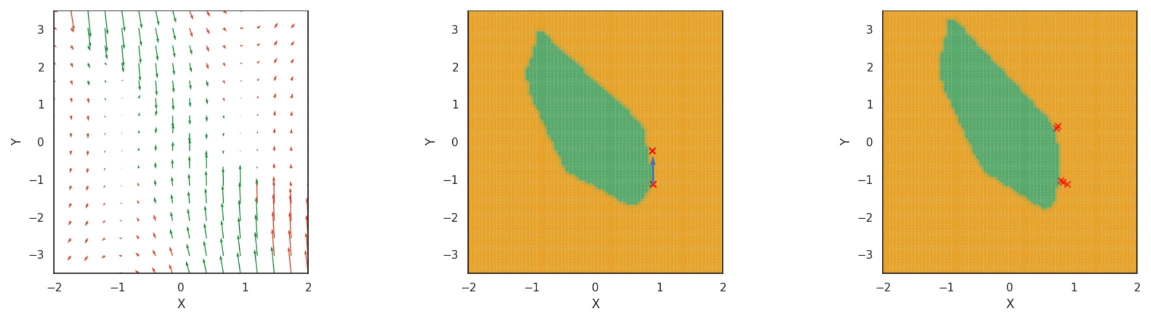
Given a safe but non-optimal initial policy π_0, safe exploration reinforcement learning concerns the problem of improving the expected return of π_0 while ensuring safety when collecting samples of the environment [Uchibe and Doya, 2007, Achiam et al., 2017, Nakka et al., 2020]. Our method from Section 5.3.2 for computing safe weight sets can be adapted to this setting with minimal effort. In particular, the safety bound ϵ for the intervals centered at the weight means can be used in combination with the rejection sampling to generate safe but randomized rollouts on the environment. Moreover, ϵ provides bounds on the gradient updates when optimizing the policy using Deep Q-learning or policy gradient methods, i.e., performing *projected gradient descent*.

Algorithm 5.2 shows our sketch of how standard RL algorithms, such as policy gradient methods and deep Q-learning, can be adapted to a safe exploration setup by using the safe weight sets computed by our method.

5.4 Experiments

We perform an experimental evaluation of our proposed method for learning positive invariant neural networks that prove infinite time horizon safety. Our evaluation consists of an ablation study where we disable different core components of Algorithm 5.1 and measure their effects on the obtained safety bounds and the algorithm's runtime. First, we run the algorithm without any re-training on the counterexamples. In the second step, we run Algorithm 5.1 by initializing D_{spec} with samples from $\mathcal{X}_0$ and $\mathcal{X}_u$ only. Finally, we bootstrap the positive invariant network by initializing D_{spec} with random samples from the state space labeled with Monte-Carlo estimates of reaching the unsafe states. We consider environments with a piecewise linear dynamic function, initial and unsafe state sets so that the verification steps of our algorithm can be reduced to MILP-solving using Gurobi [Gurobi Optimization, 2021].

We conduct our evaluation on three benchmark environments that differ in terms of complexity and safety specifications. We train two BNN policies for each benchmark-ablation pair, one with Bayesian weights from the second layer on (with $\mathcal{N}(0, 0.1)$ prior) and one with Bayesian weights in all layers (with $\mathcal{N}(0, 0.05)$ prior). Recall, in our BNN encoding in Section 5.3.1, we showed that encoding of the BNN input layer requires additional constraints and extra care,



(a) Vector field of the system when controlled by the posterior's mean. Green/red arrows indicate empirical safe/unsafe estimates.

(b) First guess of g^{Inv}. Green area shows $g^{\mathsf{Inv}} > 0$, orange area $g^{\mathsf{Inv}} < 0$. Red markers show the found counterexample.

(c) Final g^{Inv} proving the safety of the system. Previous counterexamples marked in red.

Figure 5.2: Invariant learning shown on the inverted pendulum benchmark.

Table 5.1: Results of our benchmark evaluation. The epsilon values are multiples of the weight's standard deviation σ. We evaluated several epsilon values, and the table shows the largest that could be proven safe. A dash "-" indicates an unsuccessful invariant search. Runtime in seconds.

Environment	No re-training		Init D_{spec} with $\mathcal{X}_0$ and $\mathcal{X}_u$		Bootstrapping D_{spec}	
	Verified	Runtime	Verified	Runtime	Verified	Runtime
Unstable LDS	-	3	1.5σ	569	2σ	760
Unstable LDS (all)	0.2σ	3	0.5σ	6	0.5σ	96
Pendulum	-	2	2σ	220	2σ	40
Pendulum (all)	-	2	0.2σ	1729	1.5σ	877
Collision avoid.	-	2	-	-	2σ	154
Collision avoid. (all)	-	2	-	-	1.5σ	225

since we do not know the signs of input neuron values. Hence, we consider two BNN policies in our evaluation in order to study how the encoding of the input layer affects the safe weight set computation.

Our first benchmark represents an unstable linear dynamical system of the form $x_{t+1} = Ax_t + Bu_t$. A BNN policy stabilizes the system towards the point $(0, 0)$. Consequently, the set of unsafe states is defined as $\{x \in \mathbb{R}^2 \mid |x|_\infty \geq 1.2\}$, and the initial states as $\{x \in \mathbb{R}^2 \mid |x|_\infty \leq 0.6\}$.

Our second benchmark is the inverted pendulum task, which is a classical non-linear control problem. The two state variables a and b represent the angle and angular velocity of a pendulum that must be controlled in an upward direction. The actions produced by the policy correspond to a torque applied to the anchor point. Our benchmark concerns a variant of the original problem where the non-linearity in f is expressed by piecewise linear functions. The resulting system, even with a trained policy, is highly unstable, as shown in Figure 5.2. The set of initial states corresponds to pendulum states in an almost upright position and with small angular velocity. The set of unsafe states represents the pendulum falling down. Figure 5.2 visualizes the system and the learned invariant's decision boundary for the inverted pendulum task.

Experiment	Input dimension	Hidden size	Output dimension
Linear dynamical system	2	16	1
Inverted pendulum	2	16	1
Collision avoidance	3	16	3

Table 5.2: Number of dimensions of the policy network for the three experiments.

While the previous two benchmarks concern stability specifications, we evaluate our method on a non-Lyapunovian safety specification in our third benchmark. In particular, our third benchmark is a collision avoidance task, where the system does not stabilize to the same set of terminal states in every execution. The system is described by three variables. The first variable specifies the agent's own vertical location, while the other two variables specify an intruder's vertical and horizontal position. The objective is to avoid colliding with the intruder who is moving toward the agent by lateral movement commands as the policy's actions. The initial states represent far-away intruders, and crashes with the intruder define the unsafe states.

Experimental details

Each policy is a ReLU network consisting of three layers. The first layer represents the input variables, the second one is a hidden layer with 16 neurons, and the last layer are the output variables. The size of the first and the last layer is task dependent and is shown in Table 5.2. The W_ϵ^π-safe positive invariant candidate network differs from the policy network in that its weights are deterministic, it has a different number of hidden units and a single output dimension. Particularly, the invariant networks for the linear dynamical system and the inverted pendulum have 12 hidden units, whereas the invariant network for the collision avoidance task has 32 neurons in its hidden layer. The policy networks are trained with a $\mathcal{N}(0, 0.1)$ (from second layer on) and $\mathcal{N}(0, 0.05)$ (all weights) prior for the Bayesian weights, respectively. MILP solving was performed by Gurobi 9.03 on a 4 vCPU with 32GB virtual machine.

Linear dynamical system The state of the linear dynamical system consists of two variables (x, y). The update function takes the current state (x_t, y_t) with the current action u_t and outputs the next states (x_{t+1}, y_{t+1}) governed by the equations

$$\begin{aligned} y_{t+1} &= y_t + 0.2 \cdot \text{clip}_{\pm 1}(u_t) \\ x_{t+1} &= x_t + 0.3 y_{t+1} + 0.05 \cdot \text{clip}_{\pm 1}(u_t), \end{aligned}$$

where the function clip$_{\pm 1}$ is defined by

$$\text{clip}_{\pm z}(x) = \begin{cases} -z & \text{if } x \leq -z \\ z & \text{if } x \geq z \\ x & \text{otherwise.} \end{cases}$$

The set of unsafe states is defined as $\{(x, y) \in \mathbb{R}^2 \mid |(x, y)|_\infty \geq 1.2\}$, and the initial states as $\{(x, y) \in \mathbb{R}^2 \mid |(x, y)|_\infty \leq 0.6\}$.

Inverted pendulum The state of the inverted pendulum consists of two variables $(\theta, \dot{\theta})$. The non-linear state transition is defined by

$$\begin{aligned}
\dot{\theta}_{t+1} &= \text{clip}_{\pm 8}(\dot{\theta}_t + \frac{-3g \cdot \text{angular}(\theta_t + \pi)}{2l} + \delta_t \frac{7.5\text{clip}_{\pm 1}(u_t)}{(m \cdot l^2)}) \\
\theta_{t+1} &= \theta_t + \dot{\theta}_{t+1} * \delta_t,
\end{aligned}$$

where $g = 9.81, l = 1, \delta_t = 0.05$ and $m = 0.8$ are constants. The function angular is defined using the piece-wise linear composition

$$\text{angular}(x) = \begin{cases} \text{angular}(x + 2\pi) & \text{if } x \leq \pi/2 \\ \text{angular}(x - 2\pi) & \text{if } x > 5\pi/2 \\ \frac{x - 2\pi}{\pi/2} & \text{if } 3\pi/2 < x \leq 5\pi/2 \\ 2 - \frac{x}{\pi/2} & \text{if } \pi/2 < x \leq 3\pi/2. \end{cases}$$

The set of initial states are defined by $\{(\theta, \dot{\theta}) \in \mathbb{R}^2 \mid |\theta| \leq \pi/6 \text{ and } |\dot{\theta}| \leq 0.2\}$. The set of unsafe states are defined by $\{(\theta, \dot{\theta}) \in \mathbb{R}^2 \mid |\theta| \geq 0.9 \text{ or } |\dot{\theta}| \geq 2\}$.

Collision avoidance The state of the collision avoidance environment consists of three variables (p_x, a_x, a_y), representing the agent's vertical position and the vertical and the horizontal position of an intruder. The intruder moves toward the agent, while the agent's vertical position must be controlled to avoid colliding with the intruder. The particular state transition is given by

$$\begin{aligned}
p_{x,t+1} &= p_{x,t} + u_t \\
a_{x,t+1} &= a_{x,t} \\
a_{y,t+1} &= a_{y,t} - 1.
\end{aligned}$$

Admissible actions are defined by $u_t \in \{-1, 0, 1\}$. The set of initial states are defined as $\{(p_x, a_x, a_y) \in \mathbb{Z}^3 \mid |p_x| \leq 2 \text{ and } |a_x| \leq 2 \text{ and } a_y = 5\}$. Likewise, the set of unsafe states are given by $\{(p_x, a_x, a_y) \in \mathbb{Z}^3 \mid |p_x - a_x| \leq 1 \text{ and } a_y = 5\}$.

Results

Table 5.1 shows the results of our evaluation. Our results demonstrate that re-training with the counterexamples is the key component that determines our algorithm's success. In all cases, except for the linear dynamical system, the initial guess of the invariant candidate violates the invariant condition. Moreover, boostrapping D_{spec} with random points labeled by empirical estimates of reaching the unsafe states improves the search process significantly.

5.5 Conclusion

In this chapter we formulated the safety verification problem for BNN policies in infinite time horizon systems, that asks to compute safe BNN weight sets for which every system execution is safe as long as the BNN samples its weights from this set. Solving this problem allows re-calibrating the BNN policy to reject unsafe weight samples in order to guarantee system safety. We then introduced a methodology for computing safe weight sets in BNN policies in the form of products of intervals around the BNN weight's means, and a method for verifying their safety by learning a positive invariant-like safety certificate. We believe that our results present an important first step in guaranteeing safety of BNNs for deployment in safety-critical scenarios. While adopting products of intervals around the BNN's weight means is a natural choice given that BNN priors are typically unimodal distributions, this is still a somewhat restrictive shape for safe weight sets. Thus, an interesting direction of future work would be to study more general forms of safe weight sets that could be used for re-calibration of BNN posteriors and their safety verification. Another interesting problem would be to design an approach for refuting a weight set as unsafe which would complement our method, or to consider closed-loop systems with stochastic environment dynamics. Extending our approach to other types of uncertainty learning models, such as Amini et al. [2020b], is yet another promising future work direction.

Any verification method for neural networks, even more so for neural networks in feedback loops, suffers from scalability limitations due to the underlying complexity class [Katz et al., 2017, Ivanov et al., 2019]. Promising research directions on improving the scalability of our approach by potentially speeding up the constraint solving step are gradient based optimization techniques [Henriksen and Lomuscio, 2020] and to incorporate the constraint solving step already in the training procedure [Zhang et al., 2020].

CHAPTER 6

Conclusion

This thesis studied the problem of learning neural networks with formal guarantees on their functional specification beyond test accuracies such as adversarial robustness or safety. In particular, we considered a number of different types of networks and the specification, including static feedforward networks and closed-loop systems with neural network control policies. Moreover, we investigated how including formal guarantees in the training process can negatively impact the original task.

- In Chapter 2, we investigated how quantization affects neural networks' robustness to adversarial attacks. We showed that neither robustness nor adversarial vulnerability is monotonic with respect to the number of bits of its representation and that neither is preserved by quantization of a real-numbered network. We introduced the first verification method for quantized neural networks, which accounts for their exact, bit-precise semantics using SMT solving over bit-vectors. We demonstrate that, compared to our approach, existing methods for analyzing real-numbered networks often derive false conclusions about the robustness of their quantized representation in practice.
- In Chapter 3, we proposed more efficient bit-vector SMT encodings of quantized neural networks. In particular, our method prunes variables and constraints from the SMT encoding for which their numeric or truth value can already be determined by a fast abstract interpretation reachability analysis of internal variables of the quantized network. We experimentally demonstrated that our improved encodings speed up the verification time up to three orders of magnitude over our naive encodings from the previous chapter. Additionally, we prove that verifying the bit-exact implementation of quantized neural networks with bit-vector specifications is PSPACE-hard, even though verifying idealized real-valued networks and satisfiability of bit-vector specifications alone are each in NP.
- In Chapter 4, we empirically studied the robustness-vs-accuracy tradeoff that arises when learning a robust network using adversarial training methods. We specifically focused on the tradeoff in robot learning settings, where the robot's holistic performance depends on accuracy and robustness. We introduced several error profiles to better characterize the difference of a network's mispredictions. Our results indicated that standard training yields better-performing robots than adversarial training.
- In Chapter 5, we considered the safety of closed-loop control systems with Bayesian neural networks as control policies. We proposed computing safe weight sets for recalibrating

the Bayesian neural network's posterior to form a distribution over safe decisions only. For determining which decision is safe and which is not, we learned positive invariant networks that classify system states depending on safety and invariances, i.e., closed under the system's dynamics. We proved that our learned invariant network certifies the system's safety and experimentally demonstrated that our method works in practice.

6.1 Future Directions

In this section, we highlight promising future research directions based on the results and observations of this thesis.

Real arithmetic based verification of quantized neural networks The work of Baranowski et al. [2020] has shown that the fixed-point arithmetic for running quantized neural networks can be expressed as real arithmetic. Using such real arithmetic representation for verifying quantized neural networks potentially allows using more efficient linear programming-based solving methods. However, as discussed in Baranowski et al. [2020], the rounding involved in fixed-point arithmetic requires non-linear operations. Similarly, Zohar et al. [2022] have observed a promising performance increase when reducing certain bit-vector formulas to integer SMT instead of handling them with traditional bit-vector SMT solver approaches. Thus, future work might consider efficiently expressing the rounding steps inside a quantized neural network as real or integer arithmetic SMT or mixed-integer linear programming constraints.

Robustness and quantization-aware training Quantized neural networks are typically trained with quantization-aware training, which models a reduced precision already during the forward pass of a training iteration [Jacob et al., 2018]. Similarly, interval bound propagation [Gowal et al., 2019] directly optimizes a network's abstract interpretation during the training phase for learning robust networks. Future work may combine these techniques in a quantization-aware interval bound propagation training algorithm that learns robust quantized networks.

Closing the robustness-vs-accuracy tradeoff gap Our experiments in Chapter 4 empirically confirmed that some approaches from the literature to enhance both robustness and accuracy provide improvements on robot learning tasks. Hence, research on advancing the robustness of network architectures, adversarial training settings, and theoretical insights might further close the robustness-vs-accuracy tradeoff gap.

New metrics for the robustness-vs-accuracy tradeoff For making progress on learning robust networks despite the accuracy vs robustness tradeoff, we suggest the adoption of new metrics that measure both values in a meaningful manner. For instance, robustness at $\geq$90% accuracy could capture advances in robust training methods while ensuring the learned networks have relevant standard accuracy.

Probabilistic closed-loop safety guarantees Neural network supermartingales have been recently introduced for verifying stability in stochastic closed-loop systems with deterministic neural network control policies [Lechner et al., 2022b]. Future work might consider such supermartingales for verifying stability with Bayesian neural networks and other types of specifications, including probabilistic safety, persistence, and recurrence.

GPU- and ASIC-based verification methods The enormous size of modern neural networks results in a large amount of floating-point operations required to train them. Consequently, training such networks in a reasonable time is only feasible using graphics processing units (GPUs) or application-specific integrated circuits (ASICs), e.g., Google's Tensor processing units (TPUs), to parallelize and accelerate the computations. Contrarily, most existing toolchains for verifying neural networks do not use such accelerators but run entirely on the central processing unit (CPU). The SMT and MILP solvers used in this thesis are examples of such purely CPU-based tools. Adapting SMT and MILP solvers to custom neural network hardware is non-trivial as these accelerators are based on an inherently parallel programming model. As a result, naive ports of existing algorithms may suffer from poor hardware utilization.

To scale neural networks verification methods, new algorithms based on this parallel programming model are required to make full use of the available hardware. For example, Gowal et al. [2019] proposed an abstract interpretation method for verifying the robustness of feed-forward networks that runs on GPUs and TPUs. Similarly, Lechner et al. [2022b] introduced an inherently parallel algorithm for verifying the stability of closed-loop systems with neural network policies. We believe that the most impactful future verification approaches will be the ones that use hardware accelerators such as GPUs and TPUs.

6.2 Reflections and Broader Outlook

Although our work and existing literature provide thorough insights and algorithms toward learning neural networks with formal guarantees, the adoption of these methods in practice remains sparse.

In particular, the two major challenges preventing a more widespread use are the limited scalability of verification methods and the problem of learning networks that jointly satisfy multiple specifications, e.g., accuracy and robustness, as we outlined in Chapter 4. Without addressing these two issues, we predict that the formal verification of learned components will remain a pure research area without much practical significance.

Although researchers often claim that improvements in computing power will eventually make verification methods scale to practical applications, they neglect that the complexity of real-world systems also increases. For example, while the hardware for running neural network verification methods has become faster in the past years, the size of networks deployed in practice has also increased significantly [Brown et al., 2020]. As a result, we argue that *algorithmic advances* in the scalability of neural network verification methods are necessary to adopt these methods more broadly in practice.

One characteristic specific to the research area of neural network verification is that the research community is relatively fragmented. For instance, advances in verification methods for neural networks have been published in the proceedings of conferences from the programming language [Singh et al., 2019], formal methods [Giacobbe et al., 2020], robotics [Lechner et al., 2021a], control theory [Gruenbacher et al., 2020], artificial intelligence [Henzinger et al., 2021], computer vision [Gowal et al., 2019], and the machine learning [Lechner et al., 2021b] research domains. In contrast, advances in deep learning methods usually rely on the same notation [Goodfellow et al., 2016] and are published in a handful of conferences and journals. The resulting differences in terminology, research values, and preferences of publications on neural network verification from these different research communities creates friction. Moreover, the distributed character of the field makes it difficult to grasp its current state.

To overcome this friction and address the mentioned challenges, we suggest the development of universal benchmarks and competitions. For instance, the ImageNet Large Scale Visual Recognition Challenge [Russakovsky et al., 2015], better known as ImageNet, has sparked a revolution in computer vision and fueled many recent advances in machine learning. Similarly, in the formal methods domain, Boolean satisfiability (SAT solving) competitions [Froleyks et al., 2021] enabled significant progress in the scalability of SAT solvers. The International Verification of Neural Networks Competition (VNN-COMP) [Bak et al., 2021], although in an early stage and limited in exposure, provides the first step toward this paradigm.

Another factor that hinders progress in the field is the weighting of the novelty of machine learning models and algorithms, which is often confused as scientific significance by the machine learning research community [Sambasivan et al., 2021]. We believe the high weighting of novelty is misaligned from the real-world relevance of research results. This imbalance is manifested by algorithms and neural network architectures that are considered novel, having an easier time passing the peer-review process and gaining more citations. Research funding is often tied to the number of accepted conference papers and citation counts, so researchers are incentivized to introduce new methods instead of adding breadth and depth to existing ones. This leads to an excess of new methods and models with limited experimental and shallow mathematical contributions and a shortcoming of follow-up studies that reproduce proposed algorithms and study them in more detail. Many researchers have said that machine learning and computer vision suffer from a reproducibility crisis.

We believe the research community has to reevaluate the importance of algorithmic novelty and emphasize on rigorous results. One possible way to achieve this balance is by creating separate submission tracks for journals and conferences. For instance, a scientific conference could have a separate submission track to reproduce existing methods and algorithms. As a result, the peer-reviewers can judge a submission by its significance to the values of the specific submission track. In particular, the 2022 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL 2022) introduced such a reproducibility track for precisely this reason. Similarly, the thirty-fifth Conference on Neural Information Processing Systems (NeurIPS 2021) introduced a dataset and benchmark track separated from the main track to avoid dataset and benchmark submissions being marked as *"not novel"*. We believe that these two examples are the first steps in the right direction.

Bibliography

Alessandro Abate, Daniele Ahmed, Mirco Giacobbe, and Andrea Peruffo. Formal synthesis of Lyapunov neural networks. *IEEE Control Systems Letters (L-CSS)*, 2021.

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International Conference on Machine Learning (ICML)*, 2017.

Zeyuan Allen-Zhu and Yuanzhi Li. Feature purification: How adversarial training performs robust deep learning. In *IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, 2022.

Eitan Altman. *Constrained Markov Decision Processes*, volume 7. CRC Press, 1999.

Aaron D Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control barrier functions: Theory and applications. In *European Control Conference (ECC)*, 2019.

Alexander Amini, Igor Gilitschenski, Jacob Phillips, Julia Moseyko, Rohan Banerjee, Sertac Karaman, and Daniela Rus. Learning robust control policies for end-to-end autonomous driving from data-driven simulation. *IEEE Robotics and Automation Letters (RAL)*, 2020a.

Alexander Amini, Wilko Schwarting, Ava Soleimany, and Daniela Rus. Deep evidential regression. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020b.

Alexander Amini, Tsun-Hsuan Wang, Igor Gilitschenski, Wilko Schwarting, Zhijian Liu, Song Han, Sertac Karaman, and Daniela Rus. VISTA 2.0: An open, data-driven simulator for multimodal sensing and policy learning for autonomous vehicles. *arXiv preprint arXiv:2111.12083*, 2021.

Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*, 2019.

Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International Conference on Machine Learning (ICML)*, 2018.

Christopher G Atkeson and Stefan Schaal. Robot learning from demonstration. In *International Conference on Machine Learning (ICML)*, 1997.

Zahra Babaiee, Ramin M. Hasani, Mathias Lechner, Daniela Rus, and Radu Grosu. On-off center-surround receptive fields for accurate and robust image classification. In *International Conference on Machine Learning (ICML)*, 2021.

Yutong Bai, Jieru Mei, Alan L Yuille, and Cihang Xie. Are transformers more robust than CNNs? In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

Stanley Bak, Changliu Liu, and Taylor Johnson. The second international verification of neural networks competition (VNN-COMP 2021): Summary and results. *arXiv preprint arXiv:2109.00498*, 2021.

Yogesh Balaji, Tom Goldstein, and Judy Hoffman. Instance adaptive adversarial training: Improved accuracy tradeoffs in neural nets. *arXiv preprint arXiv:1910.08051*, 2019.

Marek S. Baranowski, Shaobo He, Mathias Lechner, Thanh Son Nguyen, and Zvonimir Rakamaric. An SMT theory of fixed-point arithmetic. In *International Joint Conference on Automated Reasoning (IJCAR)*, 2020.

David Barber and Christopher M Bishop. Ensemble learning in Bayesian neural networks. In *Neural Networks and Machine Learning*. 1998.

Solon Barocas, Moritz Hardt, and Arvind Narayanan. Fairness in machine learning. *NeurIPS tutorial*, 2017.

Clark Barrett, Christopher L Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *International Conference on Computer Aided Verification (CAV)*, 2011.

Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa, 2017. Available at `www.SMT-LIB.org`.

Felix Berkenkamp, Matteo Turchetta, Angela P. Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

Armin Biere. CaDiCaL, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2017. In *Proceedings of SAT Competition 2017: Solver and Benchmark Descriptions*, 2017.

Armin Biere. CaDiCaL at the SAT Race 2019. In *Proceedings of SAT Race 2019: Solver and Benchmark Descriptions*, 2019.

Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, 2013.

Franco Blanchini and Stefano Miani. *Set-Theoretic Methods in Control*. Springer, 2008.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *International Conference on Machine Learning (ICML)*, 2015.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

Sébastien Bubeck and Mark Sellke. A universal law of robustness via isoperimetry. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

Sébastien Bubeck, Yin Tat Lee, Eric Price, and Ilya Razenshteyn. Adversarial examples from computational constraints. In *International Conference on Machine Learning (ICML)*, 2019.

Sébastien Bubeck, Yuanzhi Li, and Dheeraj M Nagaraj. A law of robustness for two-layers neural networks. In *Conference on Learning Theory (COLT)*, 2021.

Rudy R Bunel, Ilker Turkaslan, Philip Torr, Pushmeet Kohli, and Pawan K Mudigonda. A unified view of piecewise linear neural network verification. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

Luca Cardelli, Marta Kwiatkowska, Luca Laurenti, Nicola Paoletti, Andrea Patane, and Matthew Wicker. Statistical guarantees for the robustness of Bayesian neural networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Workshop on Artificial Intelligence and Security (AISec)*, 2017a.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy (S&P)*, 2017b.

Ya-Chien Chang, Nima Roohi, and Sicun Gao. Neural Lyapunov control. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

Xi Chen, Xiaolin Hu, Hucheng Zhou, and Ningyi Xu. FxpNet: Training a deep convolutional neural network in fixed-point representation. In *International Joint Conference on Neural Networks (IJCNN)*, 2017.

Xi Chen, Ali Ghadirzadeh, Mårten Björkman, and Patric Jensfelt. Adversarial feature training for generalizable robotic visuomotor control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification (CAV)*, 2013.

Chih-Hong Cheng, Georg Nührenberg, Chung-Hao Huang, and Harald Ruess. Verification of binarized neural networks via inter-neuron factoring. In *Working Conference on Verified Software: Theories, Tools, and Experiments (VSTTE)*, 2018.

Yinlam Chow, Ofir Nachum, Edgar A. Duéñez-Guzmán, and Mohammad Ghavamzadeh. A Lyapunov-based approach to safe reinforcement learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

Barrett Clark and Tinelli Cesare. Satisfiability modulo theories. In Edmund M Clarke, Thomas A Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*. Springer, 2018.

Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *International Conference on Computer Aided Verification (CAV)*, 2000.

Edmund M Clarke, Thomas A Henzinger, Helmut Veith, and Roderick Bloem. *Handbook of Model Checking*, volume 10. Springer, 2018.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). In *International Conference on Learning Representations (ICLR)*, 2019.

Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning (ICML)*, 2019.

Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Symposium on Principles of Programming Languages (POPL)*, 1977.

George Dantzig. *Linear programming and extensions*. Princeton university press, 2016.

Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2008.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2020.

Bruno Dutertre. Yices 2.2. In *International Conference on Computer Aided Verification (CAV)*, 2014.

Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output range analysis for deep neural networks. In *NASA Formal Methods Symposium (NFM)*, 2018.

Souradeep Dutta, Xin Chen, and Sriram Sankaranarayanan. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2019.

Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018.

Rüdiger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2017.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018.

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Brandon Tran, and Aleksander Madry. Adversarial robustness as a prior for learned representations. *arXiv preprint arXiv:1906.00945*, 2019.

Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning models. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Robert W. Floyd. Assigning meanings to programs. *Proceedings of Symposium on Applied Mathematics*, 1967.

Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda. SAT competition 2020. *Artificial Intelligence*, 2021.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning (ICML)*, 2016.

Sicun Gao, Jeremy Avigad, and Edmund M Clarke. δ-complete decision procedures for satisfiability over the reals. In *International Joint Conference on Automated Reasoning (IJCAR)*, 2012.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev. AI2: safety and robustness certification of neural networks with abstract interpretation. In *IEEE Symposium on Security and Privacy (S&P)*, 2018.

Peter Geibel. Reinforcement learning for MDPs with constraints. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *European Conference on Machine Learning (ECML)*, 2006.

Mirco Giacobbe, Thomas A. Henzinger, and Mathias Lechner. How many bits does it take to quantize your neural network? In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2020.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2014a.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014b.

Sven Gowal, Krishnamurthy Dj Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. Scalable verified training for provably robust image classification. In *IEEE International Conference on Computer Vision (ICCV)*, 2019.

Sophie Gruenbacher, Jacek Cyranka, Mathias Lechner, Md. Ariful Islam, Scott A. Smolka, and Radu Grosu. Lagrangian reachtubes: The next generation. In *IEEE Conference on Decision and Control (CDC)*, 2020.

Sophie Gruenbacher, Ramin M. Hasani, Mathias Lechner, Jacek Cyranka, Scott A. Smolka, and Radu Grosu. On the verification of neural ODEs with stochastic guarantees. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

Sophie Gruenbacher, Mathias Lechner, Ramin Hasani, Daniela Rus, Thomas A Henzinger, Scott Smolka, and Radu Grosu. Gotube: Scalable stochastic verification of continuous-depth models. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2022.

LLC Gurobi Optimization. Gurobi optimizer reference manual, 2021. URL `http://www.gurobi.com`.

Ramin M. Hasani, Alexander Amini, Mathias Lechner, Felix Naser, Radu Grosu, and Daniela Rus. Response characterization for auditing cell dynamics in long short-term memory networks. In *International Joint Conference on Neural Networks (IJCNN)*, 2019.

Ramin M. Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. A natural lottery ticket winner: Reinforcement learning with ordinary neural circuits. In *International Conference on Machine Learning (ICML)*, 2020.

Ramin M. Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. Liquid time-constant networks. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Patrick Henriksen and Alessio R. Lomuscio. Efficient neural network verification via adaptive refinement and adversarial search. In *European Conference on Artificial Intelligence (ECAI)*, 2020.

Thomas A. Henzinger, Mathias Lechner, and Đorđe Žikelić. Scalable verification of quantized neural networks. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

Stefan Herzog and Dirk Ostwald. Sometimes Bayesian statistics are better. *Nature*, 494(7435), 2013.

Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Conference on Computational Learning Theory (COLT)*, 1993.

Mark Horowitz. Computing's energy problem (and what we can do about it). In *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014.

Po-Sen Huang, Robert Stanforth, Johannes Welbl, Chris Dyer, Dani Yogatama, Sven Gowal, Krishnamurthy Dvijotham, and Pushmeet Kohli. Achieving verified robustness to symbol substitutions via interval bound propagation. *arXiv preprint arXiv:1909.01492*, 2019.

Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification (CAV)*, 2017.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2016.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research (JMLR)*, 2017.

Peter J. Huber. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 1(35), 1964.

Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

Craig Innes and Ram Ramamoorthy. Elaborating on learned demonstrations with temporal logic specifications. In *Robotics: Science and Systems (RSS)*, 2020.

Radoslav Ivanov, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2019.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Lucas Janson, Tommy Hu, and Marco Pavone. Safe motion planning in unknown environments: Optimality benchmarks and tractable policies. In *Robotics: Science and Systems (RSS)*, 2018.

Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *IEEE International Conference on Computer Vision (ICCV)*, 2009.

Kai Jia and Martin Rinard. Exploiting verified neural networks via floating point numerical error. In *International Static Analysis Symposium (SAS)*, 2021.

Norman P. Jouppi, Cliff Young, Nishant Patil, David A. Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. In *International Symposium on Computer Architecture (ISCA)*. ACM, 2017.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 2021.

kaggle.com. Students performance in exams. `https://www.kaggle.com/spscientist/students-performance-in-exams`, (accessed December 1, 2021).

William Kahan. IEEE standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE*, 1996.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification (CAV)*, 2017.

Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljic, David L. Dill, Mykel J. Kochenderfer, and Clark W. Barrett. The marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification (CAV)*, 2019.

Simran Kaur, Jeremy Cohen, and Zachary C Lipton. Are perceptually-aligned gradients a general property of robust classifiers? *arXiv preprint arXiv:1910.08640*, 2019.

Hassan K Khalil and Jessy W Grizzle. *Nonlinear Systems*, volume 3. Prentice Hall, 1996.

Beomsu Kim, Junghoon Seo, and Taegyun Jeon. Bridging adversarial robustness and gradient interpretability. *arXiv preprint arXiv:1903.11626*, 2019.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (BiT): General visual representation learning. In *European Conference on Computer Vision (ECCV)*, 2020.

Nikola Konstantinov and Christoph Lampert. Robust learning from untrusted sources. In *International Conference on Machine Learning (ICML)*, 2019.

Gergely Kovásznai, Andreas Fröhlich, and Armin Biere. Complexity of fixed-size bit-vector logics. *Theory of Computing Systems (TOCS)*, 2016.

Alex Krizhevsky and Geoff Hinton. Convolutional deep belief networks on CIFAR-10. `https://www.cs.toronto.edu/~kriz/conv-cifar10-aug2010.pdf`, 2010.

Daniel Kroening and Ofer Strichman. *Decision Procedures - An Algorithmic Point of View*. Springer, second edition, 2016.

Ananya Kumar, Aditi Raghunathan, Robbie Matthew Jones, Tengyu Ma, and Percy Liang. Fine-tuning can distort pretrained features and underperform out-of-distribution. In *International Conference on Learning Representations (ICLR)*, 2022.

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In *International Conference on Learning Representations (ICLR)*, 2017.

Sampo Kuutti, Saber Fallah, and Richard Bowden. Training adversarial agents to exploit weaknesses in deep control policies. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

Mathias Lechner. Learning representations for binary-classification without backpropagation. In *International Conference on Learning Representations (ICLR)*, 2020.

Mathias Lechner, Radu Grosu, and Ramin M Hasani. Worm-level control through search-based reinforcement learning. *arXiv preprint arXiv:1711.03467*, 2017.

Mathias Lechner, Ramin M. Hasani, Manuel Zimmer, Thomas A. Henzinger, and Radu Grosu. Designing worm-inspired neural networks for interpretable robotic control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

Mathias Lechner, Ramin Hasani, Alexander Amini, Thomas A Henzinger, Daniela Rus, and Radu Grosu. Neural circuit policies enabling auditable autonomy. *Nature Machine Intelligence*, 2020a.

Mathias Lechner, Ramin M. Hasani, Daniela Rus, and Radu Grosu. Gershgorin loss stabilizes the recurrent neural network compartment of an end-to-end robot learning scheme. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020b.

Mathias Lechner, Ramin M. Hasani, Radu Grosu, Daniela Rus, and Thomas A. Henzinger. Adversarial training is not ready for robot learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021a.

Mathias Lechner, Dorde Zikelic, Krishnendu Chatterjee, and Thomas A. Henzinger. Infinite time horizon safety of Bayesian neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021b.

Mathias Lechner, Alexander Amini, Daniela Rus, and Thomas A. Henzinger. Revisiting the adversarial robustness-accuracy tradeoff in robot learning. *arXiv preprint arXiv:2204.07373*, 2022a.

Mathias Lechner, Đorđe Žikelić, Krishnendu Chatterjee, and Thomas A Henzinger. Stability verification in stochastic control systems via neural network supermartingales. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2022b.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradientbased learning applied to document recognition. *Proceedings of the IEEE*, 1998.

Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. In *IEEE Symposium on Security and Privacy (S&P)*, 2019.

Hao Li, Soham De, Zheng Xu, Christoph Studer, Hanan Samet, and Tom Goldstein. Training quantized nets: A deeper understanding. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

Jiachen Li, Hengbo Ma, and Masayoshi Tomizuka. Interaction-aware multi-agent tracking and probabilistic behavior prediction via adversarial learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward ReLU neural networks. *arXiv preprint arXiv:1706.07351*, 2017.

David J. C. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 1992.

David JC MacKay. Probable networks and plausible predictions-a review of practical Bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6(3), 1995.

Wesley J Maddox, Pavel Izmailov, Timur Garipov, Dmitry P Vetrov, and Andrew Gordon Wilson. A simple baseline for Bayesian uncertainty in deep learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations (ICLR)*, 2018.

Rowan McAllister, Yarin Gal, Alex Kendall, Mark Van Der Wilk, Amar Shah, Roberto Cipolla, and Adrian Weller. Concrete problems for autonomous vehicle safety: Advantages of Bayesian deep learning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

Rhiannon Michelmore, Matthew Wicker, Luca Laurenti, Luca Cardelli, Yarin Gal, and Marta Kwiatkowska. Uncertainty quantification with statistical guarantees in end-to-end autonomous driving control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning (ICML)*, 2018.

Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: A regularization method for supervised and semi-supervised learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2018.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: A simple and accurate method to fool deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning (ICML)*, 2010.

Yashwanth Kumar Nakka, Anqi Liu, Guanya Shi, Anima Anandkumar, Yisong Yue, and Soon-Jo Chung. Chance-constrained trajectory optimization for safe exploration and learning of nonlinear systems. *IEEE Robotics and Automation Letters (RAL)*, 2020.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. In *International Conference on Learning Representations (ICLR)*, 2019.

Nina Narodytska, Shiva Kasiviswanathan, Leonid Ryzhyk, Mooly Sagiv, and Toby Walsh. Verifying properties of binarized deep neural networks. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

Muhammad Muzammal Naseer, Kanchana Ranasinghe, Salman H Khan, Munawar Hayat, Fahad Shahbaz Khan, and Ming-Hsuan Yang. Intriguing properties of vision transformers. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

Radford M Neal. *Bayesian Learning for Neural Networks*. Springer Science & Business Media, 2012.

Aina Niemetz, Mathias Preiner, and Armin Biere. Boolector 2.0 system description. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 2015.

Aina Niemetz, Mathias Preiner, and Armin Biere. Boolector at the SMT competition 2019. Technical report, Stanford University and JKU Linz, 2019.

Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 2018.

Xinlei Pan, Daniel Seita, Yang Gao, and John Canny. Risk averse robust adversarial reinforcement learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

Tianyu Pang, Xiao Yang, Yinpeng Dong, Hang Su, and Jun Zhu. Bag of tricks for adversarial training. In *International Conference on Learning Representations (ICLR)*, 2021.

Andrea Peruffo, Daniele Ahmed, and Alessandro Abate. Automated and formal synthesis of neural barrier certificates for dynamical models. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2021.

Horia Porav, Will Maddern, and Paul Newman. Adversarial training for adverse conditions: Robust metric localisation using appearance transfer. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

Stephen Prajna and Ali Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2004.

Stephen Prajna, Ali Jadbabaie, and George J. Pappas. A framework for worst-case and stochastic safety verification using barrier certificates. *IEEE Transactions on Automatic Control (TACON)*, 2007.

Luca Pulina and Armando Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *International Conference on Computer Aided Verification (CAV)*, 2010.

Luca Pulina and Armando Tacchella. Challenging SMT solvers to verify neural networks. *AI Communications*, 2012.

Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2018.

Aditi Raghunathan, Sang Michael Xie, Fanny Yang, John C Duchi, and Percy Liang. Adversarial training can hurt generalization. *arXiv preprint arXiv:1906.06032*, 2019.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Leslie Rice, Eric Wong, and Zico Kolter. Overfitting in adversarially robust deep learning. In *International Conference on Machine Learning (ICML)*, 2020.

Spencer M Richards, Felix Berkenkamp, and Andreas Krause. The Lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems. In *Conference on Robot Learning (CoRL)*, 2018.

Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 1986.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 2015.

Hadi Salman, Jerry Li, Ilya Razenshteyn, Pengchuan Zhang, Huan Zhang, Sebastien Bubeck, and Greg Yang. Provably robust deep learning via adversarially trained smoothed classifiers. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

Hadi Salman, Andrew Ilyas, Logan Engstrom, Ashish Kapoor, and Aleksander Madry. Do adversarially robust ImageNet models transfer better? *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

Nithya Sambasivan, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Paritosh, and Lora M Aroyo. "Everyone wants to do the model work, not the data work": Data cascades in high-stakes AI. In *Conference on Human Factors in Computing Systems (CHI)*, 2021.

Shibani Santurkar, Andrew Ilyas, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Image synthesis with a single (robust) classifier. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

Bernhard Schölkopf. Causality for machine learning. *arXiv preprint arXiv:1911.10500*, 2019.

Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019a.

Ali Shafahi, Parsa Saadatpanah, Chen Zhu, Amin Ghiasi, Christoph Studer, David Jacobs, and Tom Goldstein. Adversarially robust transfer learning. In *International Conference on Learning Representations (ICLR)*, 2019b.

Macheng Shen and Jonathan P How. Active perception in adversarial scenarios using maximum entropy deep reinforcement learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

Guangyao Shi, Lifeng Zhou, and Pratap Tokekar. Robust multiple-path orienteering problem: Securing against adversarial attacks. In *Robotics: Science and Systems (RSS)*, 2020.

Stefan Sietzen, Mathias Lechner, Judy Borowski, Ramin Hasani, and Manuela Waldner. Interactive analysis of CNN robustness. In *Computer Graphics Forum*, 2021.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.

Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. In *Symposium on Principles of Programming Languages (POPL)*, 2019.

Vasu Singla, Sahil Singla, Soheil Feizi, and David Jacobs. Low curvature activations reduce overfitting in adversarial training. In *IEEE International Conference on Computer Vision (ICCV)*, 2021.

Steven W Smith et al. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Pub. San Diego, 1997.

Chuanbiao Song, Kun He, Liwei Wang, and John E Hopcroft. Improving the generalization of adversarial training with domain adaptation. In *International Conference on Learning Representations (ICLR)*, 2019.

Christopher A Strong, Haoze Wu, Aleksandar Zeljić, Kyle D Julian, Guy Katz, Clark Barrett, and Mykel J Kochenderfer. Global optimization of objective functions represented by ReLU networks. *Machine Learning*, 2021.

David Stutz, Matthias Hein, and Bernt Schiele. Disentangling adversarial robustness and generalization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Dong Su, Huan Zhang, Hongge Chen, Jinfeng Yi, Pin-Yu Chen, and Yupeng Gao. Is robustness the cost of accuracy? – A comprehensive study on the robustness of 18 deep image classification models. In *European Conference on Computer Vision (ECCV)*, 2018.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Vincent Tjeng, Kai Y Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations (ICLR)*, 2019.

Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. In *International Conference on Learning Representations (ICLR)*, 2018.

Eiji Uchibe and Kenji Doya. Constrained reinforcement learning from intrinsic and extrinsic rewards. In *IEEE International Conference on Development and Learning (ICDL)*, 2007.

Jonathan Uesato, Brendan O'Donoghue, Aaron van den Oord, and Pushmeet Kohli. Adversarial risk and the dangers of evaluating against weak attacks. In *International Conference on Machine Learning (ICML)*, 2018.

Francisco Utrera, Evan Kravitz, N. Benjamin Erichson, Rajiv Khanna, and Michael W. Mahoney. Adversarially-trained deep nets transfer better: Illustration on image classification. In *International Conference on Learning Representations (ICLR)*, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

Charles Vorbach, Ramin Hasani, Alexander Amini, Mathias Lechner, and Daniela Rus. Causal navigation by continuous-time neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

Abraham Wald. Statistical decision functions which minimize the maximum risk. *Annals of Mathematics*, pages 265–280, 1945.

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2018a.

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *USENIX Conference on Security Symposium (SEC)*, 2018b.

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *International Conference on Machine Learning (ICML)*, 2011.

Lily Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane Boning, and Inderjit Dhillon. Towards fast computation of certified robustness for ReLU networks. In *International Conference on Machine Learning (ICML)*, 2018.

Matthew Wicker, Luca Laurenti, Andrea Patane, and Marta Kwiatkowska. Probabilistic safety for Bayesian neural networks. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2020.

Christoph M Wintersteiger, Youssef Hamadi, and Leonardo De Moura. A concurrent portfolio approach to SMT solving. In *International Conference on Computer Aided Verification (CAV)*, 2009.

Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning (ICML)*, 2018.

Weiming Xiang, Hoang-Dung Tran, and Taylor T Johnson. Output reachable set estimation and verification for multi-layer neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2018.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Cihang Xie, Yuxin Wu, Laurens van der Maaten, Alan L Yuille, and Kaiming He. Feature denoising for improving adversarial robustness. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Huan Xu, Constantine Caramanis, and Shie Mannor. Robustness and regularization of support vector machines. *The Journal of Machine Learning Research (JMLR)*, 10, 2009.

Greg Yang, Tony Duan, J Edward Hu, Hadi Salman, Ilya Razenshteyn, and Jerry Li. Randomized smoothing of all shapes and sizes. In *International Conference on Machine Learning (ICML)*, 2020.

Yulin Yang and Guoquan Huang. Map-based localization under adversarial attacks. In *Robotics Research*. Springer, 2020.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference (BMCV)*, 2016.

Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric Xing, Laurent El Ghaoui, and Michael Jordan. Theoretically principled trade-off between robustness and accuracy. In *International Conference on Machine Learning (ICML)*, 2019.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

Huan Zhang, Hongge Chen, Chaowei Xiao, Sven Gowal, Robert Stanforth, Bo Li, Duane S. Boning, and Cho-Jui Hsieh. Towards stable and efficient training of verifiably robust neural networks. In *International Conference on Learning Representations (ICLR)*, 2020.

Hengjun Zhao, Xia Zeng, Taolue Chen, and Zhiming Liu. Synthesizing barrier certificates using neural networks. In *International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2020.

Yiren Zhao, Ilia Shumailov, Robert Mullins, and Ross Anderson. To compress or not to compress: Understanding the interactions between adversarial attacks and neural network compression. In *Conference on Machine Learning and Systems (MLSys)*, 2019.

Yoni Zohar, Ahmed Irfan, Makai Mann, Aina Niemetz, Andres Nötzli, Mathias Preiner, Andrew Reynolds, Clark Barrett, and Cesare Tinelli. Bit-precise reasoning via int-blasting. In *International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, 2022.

Konrad Zolna, Scott Reed, Alexander Novikov, Sergio Gómez Colmenarejo, David Budden, Serkan Cabi, Misha Denil, Nando de Freitas, and Ziyu Wang. Task-relevant adversarial imitation learning. In *Conference on Robot Learning (CoRL)*, 2021.

,