

IMPERIAL COLLEGE LONDON
DEPARTMENT OF COMPUTING

PHD THESIS

**Towards Safer Deep Learning:
Verification, Robustness, and
Explainability**

Author:

Mr. Benjamin James BATTEN

Supervisor:

Prof. Alessio LOMUSCIO

External Assessor:

Prof. Xiaowei HUANG

Internal Assessor:

Prof. William KNOTTENBELT

IMPERIAL

Abstract

Deep neural networks have achieved remarkable success across diverse applications, from autonomous vehicles to medical diagnostics. However, their deployment in safety-critical systems is hindered by vulnerabilities to adversarial attacks, lack of robustness guarantees, and opaque decision-making. This work addresses these fundamental challenges through contributions towards verification, robustness, and explainability of neural networks in both deterministic and probabilistic settings. We first tackle the problem of verifying neural networks against geometric perturbations, developing a novel piecewise-linear approximation method that approximates non-convex spatial transformations more precisely than existing approaches. This technique produces verification bounds up to 30% tighter than the benchmark methods, enabling formal safety guarantees for 32% more cases on vision-based baselines. Secondly, we explore the use adversarial training in improving out-of-distribution generalisation in LiDAR-based 3D object detection. By combining a physics-inspired initialisation scheme with adversarial training, we achieve up to 6% improvement in detection accuracy under adverse weather conditions compared to simulation-based methods. Further, we extend verification techniques to probabilistic settings by developing novel algorithms for Bayesian Neural Networks (BNNs). Our Pure Iterative Expansion and Gradient-guided Iterative Expansion meth-

ods dynamically adapt verification regions based on the parameter posterior space, yielding probabilistic robustness certificates up to 40% tighter than previous approaches, while also eliminating the need for extensive hyperparameter tuning. Finally, we introduce the first formal framework for generating counterfactual explanations on BNNs, leveraging inherent uncertainty quantification to produce explanations with superior plausibility and robustness. Our evaluation demonstrates that BNN-based counterfactuals consistently outperform deterministic and ensemble-based alternatives across multiple metrics and datasets. These contributions collectively advance the state of the art in neural network safety, providing both theoretical foundations and practical tools for building more dependable AI systems.

Acknowledgements

Throughout my PhD I have been fortunate to receive the generous support of both my supervisor, Professor Alessio Lomuscio, as well as from excellent colleagues in Mattia Villani, Benedikt Brückner, Maria Stoica, Edward Stevinson, and many more. I am also grateful to the CDT in Safe and Trusted Artificial Intelligence, whose excellent students and academic staff have continually cultivated my knowledge and interest in areas beyond that of my own work. Additionally, I thank Rachmaninoff, Scriabin, and Puccini whose music has spent countless hours accompanying me in my pursuits. Integral to the completion of this work are my parents, for whom I cannot exhaustively express my gratitude. Their patience, empathy, and enduring belief in me is of the highest order and has shaped me as a researcher and a man.

Statement of Originality

I declare that the work in this thesis is my own work, except where otherwise stated. The writing is original and my own, generative AI was used in limited instances for synonym proposals and high-level planning.

Copyright Declaration

The copyright of this thesis rests with the author. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution-Non Commercial 4.0 International Licence (CC BY-NC). Under this licence, you may copy and redistribute the material in any medium or format. You may also create and distribute modified versions of the work. This is on the condition that: you credit the author and do not use it, or any derivative works, for a commercial purpose. When reusing or sharing this work, ensure you make the licence terms clear to others by naming the licence and linking to the licence text. Where a work has been adapted, you should indicate that the work has been changed and describe those changes. Please seek permission from the copyright holder for uses of this work that are not included in this licence or permitted under UK Copyright Law.

Contents

Abstract	1
Acknowledgements	3
Statement of Originality	4
Copyright Declaration	5
1 Introduction	16
1.1 Vulnerabilities of Neural Networks	17
1.2 Verification and Robustness	18
1.3 Interpretability and Explainability	19
1.4 Objectives	20
1.4.1 Contributions	20
1.4.2 Publications	22
1.5 Dissertation Roadmap	22
2 Background	24
2.1 Neural Networks	24
2.1.1 Activation Functions	26
2.1.2 Layer Operations	28

2.1.3	Training Neural Networks	30
2.2	Bayesian Neural Networks	38
2.2.1	Training Bayesian Neural Networks	40
2.3	Adversaries and Robustness	44
2.4	Verification of Neural Networks	49
2.4.1	Incomplete Verifiers	51
2.4.2	Complete Verifiers	60
2.5	Verification on Geometric Perturbations	61
2.6	Verification on Non-deterministic Models	64
2.7	Counterfactual Explanations	67
3	Verification of Geometric Robustness Specifications in Vision	
	Tasks	71
3.1	Geometric Robustness Verification	72
3.2	Piecewise Linear Formulation	74
3.2.1	Piecewise Linear Bounds	75
3.2.2	Linear Optimisation Based on Sampling Points	76
3.2.3	Lipschitz Optimisation for Obtaining Sound Piecewise Linear Bounds	80
3.3	Algorithmic Details	85
3.3.1	Linear Optimisation over Sub-domains	85
3.3.2	Branch-and-bound Procedure	87
3.4	Experimental Evaluation	88
3.4.1	Experimental Setup	89
3.4.2	Experimental Results	89
3.5	Conclusions	95

4	Improving Weather-based OOD Generalisation in Lidar-based Perception Models via Adversarial Training	96
4.1	Introduction	96
4.2	Lidar-based Fog Simulation	98
4.3	Adversarial Training for Lidar-based Vision Systems	101
4.3.1	Simulation-driven Initialisation	102
4.3.2	Adversarial Training on Point Clouds	105
4.4	Evaluation	106
4.4.1	Model Training	108
4.4.2	Adversarial Training for Adverse Weather	109
4.4.3	Ablations	112
4.5	Conclusions	114
5	Probabilistic Verification of Bayesian Neural Networks	115
5.1	Introduction	115
5.2	Computing the Probabilistic Robustness for BNNs	116
5.2.1	Pure Dynamic Scaling	118
5.2.2	Gradient-guided Dynamic Scaling	120
5.2.3	Theoretical Comparison	123
5.3	Evaluation	124
5.3.1	Comparison of Pure Sampling, PIE, and GIE Approaches	126
5.3.2	Ablation Study on the Use of Gradient-Based Dynamic Scaling Factors	127
5.3.3	Ablation Study on the Number of Iterations	128
5.4	Evaluation of Timing for PIE Algorithm	129
5.5	Training Details	130
5.6	Comparison Against Empirical Probabilistic Robustness	131
5.7	Conclusions	131

6	Probabilistic Counterfactual Explanations on Bayesian Neural Networks	134
6.1	Introduction	134
6.2	Counterfactual Explanations for BNNs	135
6.3	Evaluation	139
6.3.1	Visual Interpretation	142
6.3.2	Metrics	143
6.3.3	Numeric Results	144
6.3.4	Comparison with Ensembles	147
6.4	Conclusions	148
7	Conclusions	151
7.1	Summary of Contributions	151
7.2	Broader Impact and Limitations	153
7.3	Future Work	154
7.3.1	Scaling Verification to Modern Architectures	154
7.3.2	Unified Framework for Natural Perturbations	154
7.3.3	Verification-Aware Training	155
7.3.4	Richer Explanation Frameworks	155
7.3.5	Cross-Domain Safety Metrics	156
7.4	Closing Remarks	156

List of Figures

1.1	An adversarial attack on a correctly-classified input image (a), with noise (b) produced by PGD and added to the input, producing an attacked input (c). The attacks are performed on a ResNet18 model trained on ImageNet.	17
2.1	Visual representation of a feed-forward neural network with two hidden layers of size 8 and 6 respectively, an input dimension of 11, and an output dimension of 2.	25
2.2	An FFNN with one hidden layer and ReLU activations applied. Weight values are displayed adjacent to their corresponding edges and post-activation IBP bounds are shown in square brackets adjacent to nodes. The bias values for the network are set to 0.	53
2.3	Linear bounds on the output, x , of a ReLU node in terms of the input, $\tilde{x}$ for a) inactive nodes, b) active nodes, and c) unstable nodes. In each case, the pink region represents the feasible output values for a given input value. In each case, l and u represent concrete interval bounds on the input $\tilde{x}$	56

2.4	An FFNN with one hidden layer and ReLU activations applied. Weight values are displayed adjacent to their corresponding edges and post-activation LBP bounds are shown adjacent to the nodes. The bias values for the network are set to 0.	57
2.5	Illustration of a stop sign with rotation and shear transformations applied.	62
3.1	Comparison of sound and unsound piecewise (PW) linear domains (our work), sound linear domain (gray area) [Singh et al., 2019a], and interval bounds (dashed line) [Singh et al., 2019a]. The true pixel value function (the green curve) is marked for a rotation of 18°	74
3.2	A comparison of area captured by piecewise linear and linear bounds as a function of transformation parameter. Relative bound area is defined as $1 - (V_{\text{PWL}}/V_L)$	90
4.1	Relative received powers from an erroneous ‘soft’ target and the true ‘hard’ target. In this example, backscattering from the fog produces a stronger received signal and so the lidar would interpret R' as the true distance to an object.	101
4.2	Comparison of a clean, clear-weather point cloud, against a point cloud modified by the fog simulation presented in Hahner et al. [2021]. The lidar scanner is marked by the blue sphere in the centre.	103
4.3	Mean Average Precision (mAP) computed using R40 scores for each class plotted against position budget ϵ_p . Results are for 50 epochs of adversarial training.	112

5.1	A comparison between the area of the set of safe weights $\mathcal{J}$ in Example 4 that is covered by the pure sampling (5.1a), PIE after two iterations (5.1b), and GIE after two iterations (5.1c) for the same initial weights $\mathbf{w}_1, \dots, \mathbf{w}_4 \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$. See Section 5.3 for detailed comparisons, including comparison with equal compute.	120
5.2	A 2-dimensional cross-section of the safe region $\mathcal{C}$ covered by pure sampling (black), PIE (blue), and GIE (Red) on the same image in CIFAR dataset.	125
5.3	Probabilistic certification as a function of ρ , the gradient-based scaling factor in Algorithm 6.	128
5.4	The number of expansion iterations before PIE reaches max probabilistic certifications in Table 5.1.	129
5.5	Comparison of the pure sampling, PIE, and GIE approaches against the empirical probabilistic robustness, obtained by sampling parameters from the BNN posterior and using IBP to verify whether they result in correct classification. 50 samples were used for calculating each empirical accuracy.	132
6.1	Generation of a counterfactual for an MLP, an Ensemble, and a BNN (top to bottom, respectively). a) shows the original image after which a counterfactual with target ‘6’ is progressively generated. b)-d) show snapshots of the image as number of iterations increases.	141
6.2	Figure 6.2a shows low sensitivity (high robustness), where the distance between $\mathbf{x}_c$ and $\mathbf{x}'_c$ is smaller compared to the high sensitivity (low robustness) scenario in Figure 6.2b.	144

6.3 Visual comparison of two counterfactuals produced under the same setting as in [Schut et al., 2021] for (b) an Ensemble model, and (c) a BNN. Original inputs are shown in (a). The top counterfactual is for an original input of 5, with a target of 6 and the lower counterfactual is for an original input of 9 with target class 4.149

List of Tables

3.1	Comparison of verification results for piecewise linear constraints and linear constraints.	91
3.2	Comparison of L and PWL using VENUS, with verification results taken from DeepG [Balunović et al., 2019].	91
4.1	Numerical results showing R40 mAP scores across each class and difficulty. <i>Clean</i> indicates scores obtained on clear-weather data only. <i>Fog</i> indicates scores obtained on the foggy test set from STF [Shi et al., 2019]. The intersection over union (IoU) thresholds are 0.5, 0.25, and 0.25 for the car, cyclist, and pedestrian classes respectively.	107
4.2	Numeric results showing the relative performance of adversarial training (AT), simulated data [Hahner et al., 2021], and the random initialisation scheme outlined in Section 4.3. Columns report mAP40 average results for the same thresholds presented in Table 4.1.	111

5.1	Probabilistic robustness results for the pure sampling method from [Wicker et al., 2020] and ours. Adversarial accuracies were obtained using an $\epsilon = 0.05$ bounded PGD attack for all networks. Probabilistic certification denotes the mean p_{safe} by each approach. Also, see Section 5.6 for a comparison of the probabilistic certification results obtained by each approach with the empirical probabilistic robustness values for each network.	127
5.2	Impact of computing gradients for PIE algorithm on runtime. The data are presented as a percentage increase or decrease in runtime as compared to the [Wicker et al., 2020] approach on the same networks.	130
6.1	Numeric results for counterfactuals produced on the MNIST, credit, diabetes, news, and spambase datasets, and the MLP, Ensemble, and BNN model types. For each dataset/model pair we report three metrics covering the plausibility and robustness of the counterfactuals. We also report the clean accuracy, percentage of <i>valid</i> counterfactuals found for each pair, and the mean l_2 cost. The arrows indicate for each metric whether high is better ($\uparrow$) or lower is better ($\downarrow$).	145
6.2	Best performance results by metric across all hyperparameter tuning runs. Arrows indicate for each metric whether high is better ($\uparrow$) or lower is better ($\downarrow$).	147
6.3	Numeric results for MNIST, credit, diabetes, news, and spam-base datasets on the [Schut et al., 2021] MLP Ensembles and our BNNs. For each metric arrows indicate whether higher is better($\uparrow$) or lower is better ($\downarrow$).	148

Chapter 1

Introduction

Since the term ‘machine learning’ was coined to describe algorithms that improve performance through experience, the field has experienced fluctuating periods of prominence and practical application. Today, machine learning is experiencing unprecedented growth. Modern computing hardware and widespread internet connectivity have created an optimal combination of vast training datasets and computational resources. Consequently, machine learning techniques have been adopted across virtually every commercial sector.

Deep learning is a subfield of machine learning that encompasses models characterised by high complexity and over-parametrisation [LeCun et al., 2015]. Models in this domain are commonly referred to as neural networks (NNs) or deep neural networks (DNNs), terminology that reflects their inspiration from biological neural systems.

Neural networks eliminate the need for explicitly defined functional forms, instead relying on a large number of parameters combined with simple operations to approximate highly complex functions during a process called training. This learning methodology requires substantial amounts of data to overcome

the high model variance introduced by extensive parametrisation. Where sufficient data is available, neural networks demonstrate exceptional performance on complex tasks and have become fundamental to fields such as autonomous vehicles [Geiger et al., 2012, Grigorescu et al., 2020], computational diagnostics [Litjens et al., 2017, Miotto et al., 2018], and natural language processing [Young et al., 2018, Touvron et al., 2023, Vaswani et al., 2017].

1.1 Vulnerabilities of Neural Networks

Despite their remarkable flexibility and performance, neural networks have been shown to be vulnerable to targeted input perturbations called *adversarial attacks* [Goodfellow et al., 2014]. Moreover, the high complexity of neural networks creates an opaque decision process, making it difficult to identify and remedy vulnerabilities.

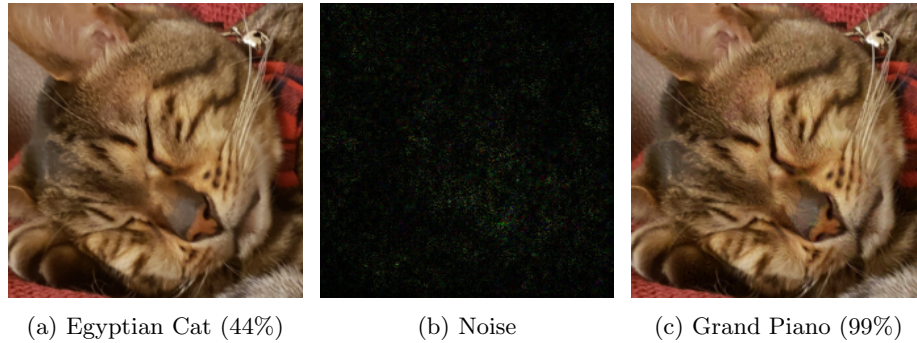


Figure 1.1: An adversarial attack on a correctly-classified input image (a), with noise (b) produced by PGD and added to the input, producing an attacked input (c). The attacks are performed on a ResNet18 model trained on ImageNet.

Figure 1.1 displays an example adversarial attack on an image classification model. Figure 1.1a shows an unperturbed input which is correctly classified by the model as an Egyptian cat. By adding the small amount of noise shown in Figure 1.1b we obtain the attacked input shown in Figure 1.1c. When passed

to the model Figure 1.1c is incorrectly classified as a grand piano with 99% confidence. Though the model has made this prediction, the images in Figure 1.1a and Figure 1.1c are indistinguishable to a human.

In addition to the threat of adversarial attack, neural networks also suffer from prediction failures that result from their mode of training. Concretely, neural networks are trained on large, diverse swathes of data relevant to the task the model aims to accomplish. As we have mentioned, this is highly effective in many cases, however, in most cases it is not practically feasible to collect data covering all the possible input scenarios a model could, in theory, come across. The result is that models tend to perform much better on data the same, or similar to, that which it sees in training. Such behaviour is termed *overfitting*, because the model is over-fit to the training set. In Chapter 2 we discuss some methods to mitigate this. Regardless, there will always exist inputs on which a model is less familiar, and more prone to making mistakes. Broadly, these inputs are called out-of-distribution (OOD) inputs, as they lie outside of the training distribution. In Chapter 2 we also discuss some approaches to improving performance on OOD inputs, and in Chapter 4 we introduce a novel approach for doing this in the space of driverless cars.

1.2 Verification and Robustness

In response to these identified vulnerabilities, research has focused on developing methods to improve neural network stability, particularly for safety-critical applications. One line of research modifies training algorithms and dataset compositions to improve a network’s natural resistance to adversarial attack, a property often referred to as the *robustness* of a model [Tramèr et al., 2018, Shafahi et al., 2019]. Differently, *verification* is a line of work aiming to provide formal *guarantees* on the output of a model [Tran et al., 2020, Ehlers, 2017,

Wang et al., 2021, Botoeva et al., 2020b].

A relatively new field, verification has grown significantly since its introduction in 2017 [Ehlers, 2017, Katz et al., 2017]. Because verification aims to provide formal assurances about a model’s resilience to attacks, it establishes a highly restrictive metric that provides a lower bound on model robustness. Consequently, verification is computationally expensive - far more so than adversarial attacks themselves. As such, a major challenge to researchers is the scalability of verification procedures, which have advanced from verifying networks of the order $\approx 10^2$ parameters to $\approx 10^7$ in the last 8 years [Brix et al., 2023, 2024].

Despite significant progress in methods for facilitating safety and providing assurances on neural networks, the literature continues to fall short in industry-scale problem settings. Current work has focused predominantly on a generalised class of model threats restricted by ℓ_p -bounded noise, such as the attack shown in Figure 1.1. A counterargument to this approach is that the adversarial attacks modelled are implausible in real-world scenarios and would require a malicious agent with access to model parameters to be realised. While verification is an important tool for ensuring safety in machine learning systems, it remains limited in both scalability and the formulation of realistic safety criteria.

1.3 Interpretability and Explainability

In addition to the input-based vulnerabilities discussed above, neural networks also suffer from limitations in model *interpretability*. We define interpretability to mean a practitioner’s ability to directly understand the internal workings of a model. Closely related to interpretability is *explainability*, which we define as the ability to understand *why* a model made a particular decision, without necessarily understanding the entire process. We refer to these techniques

collectively as methods for model *transparency*. Beyond contributing to the vulnerabilities outlined above, a lack of model transparency also prohibits use in domains where a concrete understanding of model decisions is essential. For example, in finance or medicine where decisions based on faulty logic can result in costly mistakes.

To address model interpretability concerns, a line of research on *explaining* model decisions proposes techniques to understand the reasoning behind a model output. Such tools provide practitioners with abstractions of the model’s reasoning, which can either reassure users when explanations align with the semantic logic of a problem or alert users when they reveal logical inconsistencies.

1.4 Objectives

1.4.1 Contributions

In this work, we explore the main methods for improving the safety of neural network-based systems. Primarily, the goal is to examine current approaches used to provide safety assurances in AI and explore their deficiencies namely, their over-reliance on either simplistic noise-based threat models, or model architectures. This work posits that by expanding on currently-existing methodologies, many safety techniques are more-widely applicable to real-world threat models and common-but-unsupported model designs. Specifically, we target four research questions to support the thesis:

1. Can more complex convex relaxations used for encoding network operations in formal verification be repurposed to encode non-convex input specifications?
2. Can we modify verification procedures designed for deterministic paradigms to handle popular model architectures based in probabilistic paradigms?

3. Can the robustification technique of adversarial training be used to improve generalisation in out-of-distribution settings?
4. Can counter-causal explainability techniques be modified to apply under probabilistic model assumptions?

Below we summarise the contributions made towards addressing the challenges outlined above.

- We develop a state-of-the-art piecewise-linear encoding for geometric input perturbations to verification queries. The improved tightness resulting from our formulation enables us to resolve up to 32% more verification queries than the previous state of the art. We make the procedure and code publicly available. This work explores question 1.
- We propose a novel method for improving the robustness of lidar-based vision models in unseen on-the-road environments. By leveraging adversarial training methods we are able to improve the precision of 3D perception models in our benchmark suite by up to 6% under natural adversarial attacks when compared with defensive training algorithms based on simulation. This addresses question 3.
- We develop a state-of-the-art algorithm for probabilistic verification on Bayesian neural networks. Building on a McCormick envelope-based framework, we improve verification performance by proposing an adaptive, gradient-driven search procedure over the space of the parameter posterior. We observe verification bounds which are up to 40% tighter than the previous state of the art. This contribution tackles question 2.
- We introduce a framework for producing probabilistic counterfactual explanations on Bayesian neural networks. Hypothesising that explanations produced in Bayesian settings better capture the data distribution, we observe that our

method is able to produce explanations with increased robustness and plausibility compared with an equivalently sized ensemble. This work directly addresses question 4.

1.4.2 Publications

Here we list the published works supporting the contributions outlined above.

- B. Batten, P. Kouvaros, A. Lomuscio, and Y. Zheng. Efficient neural network verification via layer-based semidefinite relaxations and linear cuts. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI21)*, pages 2184–2190, 2021.
- B. Batten, M. Hosseini, and A. Lomuscio. Tight Verification of Probabilistic Robustness in Bayesian Neural Networks. In *Proceedings of the 27th International Conference on Artificial Intelligence and Statistics (AISTATS 2024)*, pages 4906–4914, PMLR, 2024. This work supports the contributions outlined in Chapter 3.
- B. Batten, Y. Zheng, A. De Palma, P. Kouvaros, A. Lomuscio. Verification of Geometric Robustness of Neural Networks via Piecewise Linear Approximation and Lipschitz Optimisation. In *ECAI 2024*, pages 2362–2369, IOS Press, 2024. This work is expanded on in Chapter 5.
- B. Batten, A. Lomuscio. Improving Weather-based OOD Generalisation in Lidar-based Object Detection Models via Adversarial Training. In *CVPRW 2025*, pages 4321–4329, 2025. The content of Chapter 4 builds on this work.

1.5 Dissertation Roadmap

The remainder of the thesis is organised as follows: In Chapter 2, we define the necessary notation and cover the technical preliminaries for the subsequent

chapters, as well as discussing the relevant literature. In Chapter 3, we detail our work on verification of geometric perturbations and discuss the results of this work. In Chapter 4, we propose our adversarial training-based approach to ‘robustifying’ lidar-based vision models against naturally occurring attacks. In Chapter 5, we extend our definition of verification to include probabilistic settings and outline our state-of-the-art procedure for probabilistic verification of Bayesian neural networks. In Chapter 6, we continue to examine safety properties in probabilistic settings, introducing an algorithm for producing counterfactual-based explanations on Bayesian neural networks and comparing the safety properties of these explanations with those produced in deterministic settings. Finally, in Chapter 7, we conclude the thesis by summarising and reflecting on the results of this work and proposing future work.

Chapter 2

Background

2.1 Neural Networks

Neural Networks (NNs), or *artificial neural networks*, are a class of machine learning model whose architecture is inspired by biological neural structures [McCulloch and Pitts, 1943]. Broadly, neural networks are characterised by a sequence of layers containing *nodes* and interconnected by *weights*. Within each layer, a node takes a weighted sum of inputs and adds a node-wise bias value before applying a non-linear *activation* function to give the final node output. The concatenation of these outputs across all nodes in a layer constitutes the layer's output.

We focus on a specific subclass of NNs called *feed-forward neural networks* (FFNNs); when discussed, a neural network may be assumed to be a feed-forward neural network, unless otherwise stated. FFNNs are characterised by consisting exclusively of layers which take, as input, the output of *previous* layers in the network [Rumelhart et al., 1986]. In Figure 2.1 we show a graphical representation of a feed-forward neural network. The flow of operations is left

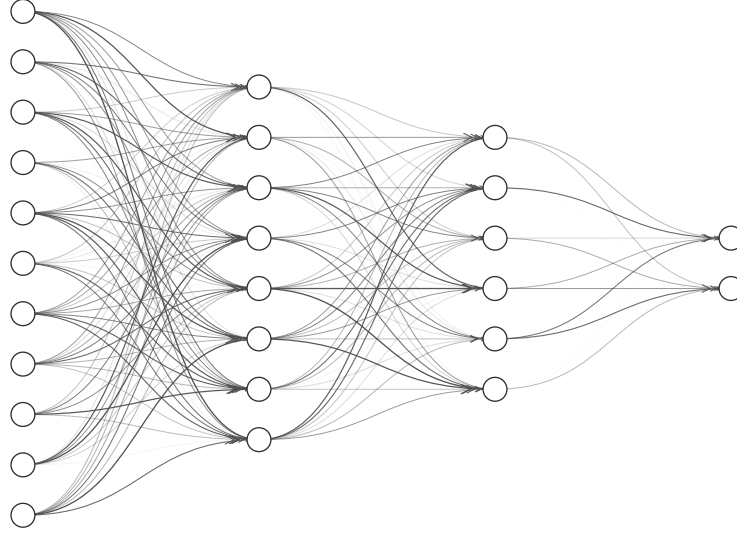


Figure 2.1: Visual representation of a feed-forward neural network with two hidden layers of size 8 and 6 respectively, an input dimension of 11, and an output dimension of 2.

to right, where the edges represent opacity-weighted parameters, and the hollow circles represent the nodes of the network.

Definition 1 (Feed-forward Neural Network (FFNN)). We define a feed-forward neural network, N , as a set, $\mathcal{T} = \{L^{(l)} \mid l = 1, \dots, K\}$, of K layers. Each layer, $L^{(l)} = (h_{P^{(l)}}(\cdot), g(\cdot))$ is a tuple consisting of a parameterised linear operation, $h_{P^{(l)}}(\mathbf{x}^{(l-1)})$, and a non-linear, element-wise activation function, such that $L^{(l)}(\mathbf{x}^{(l-1)}) = \mathbf{x}^{(l)} = g(h_{P^{(l)}}(\mathbf{x}^{(l-1)}))$. Here, $P^{(l)}$ is a set of learned parameters for layer l . We denote the pre-activation output $\tilde{\mathbf{x}}^{(l)} = h_{P^{(l)}}(\mathbf{x}^{(l-1)})$. The network output is the composition of the layer-wise operations in order, $N(\mathbf{x}^{(0)}) = L^{(K)}(L^{(K-1)}(\dots L^{(1)}(\mathbf{x}^{(0)})))$ where, in general, we write $\mathbf{x}^{(0)} = \mathbf{x}$.

Unless otherwise specified, the non-linear activation function $g(\cdot)$ is assumed

to be the Rectified Linear Unit (ReLU) operation¹,

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0, \end{cases} \quad (2.1)$$

though a discussion on activation functions is given in Section 2.1.1.

Neural Networks are *universal approximators* [Hornik et al., 1989], meaning that, under certain conditions on their architecture, they can approximate any function to an arbitrary precision. Moreover, the functional representation they develop emerges from the data on which they are trained (see Section 2.1.3). This makes them highly adaptable across domains, and are appropriate choices where specific parametric forms are not applicable or obvious. Improvements in parallel computing hardware have enabled efficient training with increasingly large datasets [Jouppi et al., 2017]; a discussion on training approaches is given in Section 2.1.3.

2.1.1 Activation Functions

Although we focus on the ReLU layer-wise activation function, it is worth discussing the range of activation functions used across the field. We focus on the most common and relevant to this work, for a more comprehensive discussion see [Dubey et al., 2022].

- **ReLU:** The ReLU activation function dominates the deep learning landscape, in large part due to its computational simplicity. Specifically, its gradient is piecewise-constant. There are variations on the ReLU function, such as Leaky ReLU, designed to mitigate the ‘dying neuron’ problem, and Parametric ReLU, which can be used to improve model performance by introducing

¹We focus on ReLU as this is the most widely-used activation function. We discuss other activation functions in Section 2.1.1.

degrees of freedom in the form of a learnable gradient parameter.

- **Hyperbolic Tangent:** The hyperbolic tangent activation,

$$\text{Tanh}(\mathbf{x}) = \frac{e^{2x} - 1}{e^{2x} + 1}, \quad (2.2)$$

is sometimes used as an alternative to ReLU activations in the hidden layers of a neural network. Differently to ReLU, this activation is smooth everywhere.

- **Sigmoid:** The sigmoid function,

$$\text{Sigmoid}(\mathbf{x}) = \frac{1}{1 + e^{-x}}, \quad (2.3)$$

compresses any input to an output between 0 and 1. For this reason it is frequently implemented in models where outputs that resemble probabilities are desired; a frequent use case is binary classification. This function is typically only applied on the output layer.

- **Softmax:** The softmax function,

$$\text{Softmax}_i(\mathbf{x}) = \frac{e^{x_i}}{\sum_{j=1}^N e^{x_j}}, \quad (2.4)$$

is the multi-class parallel of Sigmoid. It normalises outputs such that their sum is 1. In this sense, the output meets the criteria of a probability mass function, and Softmax is frequently deployed during the training of multi-class classification models. Though, due to its monotonicity, it is not necessary at inference and is often omitted for computational efficiency. As with Sigmoid, this function is usually only applied to the output layer.

2.1.2 Layer Operations

In Definition 1 we use $h_{P_i}(\cdot)$ to denote a generalised network operation. We now examine some of the choices for $h_{P_i}(\cdot)$, focussing on the operations most necessary for the work outlined in subsequent chapters.

- **Linear weighted sum (Fully Connected):** The fully connected linear operation is the most pervasive operation; and is used to some degree in almost all architectures. It involves taking a weighted sum of *all* the previous layer's nodes and adding an offset (bias) term. Formally, it processes a layer $\mathbb{R}^n \rightarrow \mathbb{R}^m$, where

$$\text{FC}(\mathbf{x})_{i \in m} = \sum_{j \in n} W_{i,j} \mathbf{x}_j + \mathbf{b}_i, \quad (2.5)$$

where W and $\mathbf{b}$ are the weight matrix and bias vector respectively. The fully-connected nature of the operation affords it great flexibility, as the conditional relationship between nodes in subsequent layers is left entirely up to the network during training. For this reason it is not limited to only certain domains and appears in almost every application of deep learning. For example, in modern natural language processing (NLP) transformer-based models currently dominate the landscape due to their highly parallelisable operations and powerful ability to generate meaningful representations of written language. With this said, the fully connected layer remains a key part of their architecture and is responsible for decoding embeddings into relevant, task-specific outputs.

- **Convolution:** The convolution operation [LeCun et al., 1998], or layer, is less universal than fully connected layers. However, it is widely deployed in computer vision and has also been proposed for time-series tasks; we will focus on its computer vision applications. The 2D convolution operation for an input, $\mathbf{x} \in \mathbb{R}^{m \times n \times c}$, parameterised by a kernel, $K \in \mathbb{R}^{h \times w \times c}$, and bias

vector, $\mathbf{b} \in \mathbb{R}^c$ can be expressed as

$$\text{CONV}(\mathbf{x}) = \mathbf{x} * K + \mathbf{b}, \quad (2.6)$$

where $*$ is a linear convolution between the input and the kernel. In general, $h, w < m, n$, and the kernel is ‘slid’ across $m \times n$ 2D slices of the input to compute an output $\text{CONV}(\mathbf{x}) \in \mathbb{R}^{m-h \times n-w}$. The intuitive principal behind the convolution operation is to extract higher level representations of features which are geometrically correlated. For this reason it is widely used in image processing where local pixels correlate. Further, there are special kernels such as Gaussian blur, for image blurring or smoothing, and ridge detection, for identifying sharp changes in pixel values, which may be used as fixed (unlearned) kernels in a neural network, or directly for image preprocessing.

- Residual Connection²: Differently to the previous operations, residual connections [He et al., 2016] do not interact exclusively with the adjacent layer. Instead, they permit the transmission of information through a network by ‘skipping’ layers, this is widely referred to as a *skip connection*. The group of layers traversed by the skip connection are jointly referred to as the *residual block*. The number of layers skipped is an architecture choice, for a residual block with a skip connection across two fully-connected layers we can write,

$$\mathbf{x}^{(l+3)} = \text{FC}^{(l+2)}(\text{FC}^{(l+1)}(\mathbf{x}^{(l)})) + \mathbf{x}^{(l)}. \quad (2.7)$$

An evident constraint of residual blocks is that the output of $\text{FC}^{(l+2)}$ must be of equal dimension to the input of $\text{FC}^{(l+1)}$. Or more generally, the input to the residual block must have dimension equal to the residual block out-

²We note that our definition for feed-forward neural networks as set out in Definition 1 does not account for skip connections. Although architectures leveraging residual connections are considered feed-forward, for ease of exposition, we will be explicit when referring to residual layers and assume standard linear connections otherwise.

put. Residual blocks were popularised by a model architecture sharing their namesake, ResNet, and are favoured for their ability to tackle the vanishing gradient problem in large networks, as well as empirical evidence of better generalisation compared with networks consisting only of fully-connected layers.

2.1.3 Training Neural Networks

Before we can use a neural network to make helpful predictions (called *inference*), we must first learn parameter values such that the model encodes the functional form of the problem we wish to solve. We call this parameter update step training, or learning. Neural network training algorithms can broadly be separated into those based on labelled data, known as *supervised* learning [LeCun et al., 2015], algorithms in the absence of labelled data, known as *unsupervised* [Hinton and Salakhutdinov, 2006], and reinforcement learning-based algorithms. Within each of these settings, there are two further broad categories of learning tasks³: *regression*, and *classification*. In the former, we are concerned with numeric outputs in a continuous space; in the latter, we infer discrete outputs from the continuous numeric network outputs.

Training a neural network is driven by the *loss function*, which encodes the objective we want a network to learn for a given task. In this work, we focus mainly on supervised learning tasks for classification, where access to correctly labelled data is assumed. That notwithstanding, below we outline some common loss functions used in each setting:

³Increasingly, Generative AI may be considered a further category; however, it can usually be viewed under one of the three frameworks outlined here. For example, text generation may be considered a special case of supervised classification over token space, and image generation a special case of unsupervised regression on pixel values. Since we do not discuss Generative AI in this thesis, we omit further discussion from the text.

- Least Squares: Least squares loss,

$$\mathcal{L}_{\text{Least Squares}}(\mathcal{D}) = \sum_{\mathbf{x}, y \in \mathcal{D}} (y - f(\mathbf{x}))^2, \quad (2.8)$$

where we have a dataset $\mathcal{D}$ of pairs of data $\mathbf{x}$ and y , and a generic model function $f(\cdot)$. Least squares loss is used extremely frequently for its agreeable statistical properties: in the setting where model error across data points is independent, minimising least squares is provably equivalent to maximising likelihood [Hastie, 2009], further, it produces unbiased estimates of the conditional expectation, $\mathbb{E}[y|\mathbf{x}]$. As least squares addresses minimising the square error of a model, it is used for regression tasks.

- Absolute Loss: Absolute loss parallels least squares loss but takes the sum of absolute error rather than square error,

$$\mathcal{L}_{\text{Absolute}}(\mathcal{D}) = \sum_{\mathbf{x}, y \in \mathcal{D}} |(y - f(\mathbf{x}))|. \quad (2.9)$$

It differs from least squares as it is an estimate of the conditional median⁴ of the output, rather than conditional mean. This may be desirable in some cases where the underlying sampling distribution is poorly behaved or non-convergent. Further, least squares is highly susceptible to outliers due to the square term exploding, absolute loss limits this effect by taking the absolute value instead. Like square loss, it is used in regression tasks.

- Pinball (quantile) Loss: Pinball loss estimates conditional quantiles of the

⁴‘Conditional’ here means that we consider the statistical properties of the distribution over y given a value of $\mathbf{x}$. In general, when considering statistical properties of predictive models we follow the nomenclature of [Hastie, 2009].

output value. Its functional form for a single training pair,

$$\mathcal{L}_{\tau, \text{Pinball}}(\mathbf{x}, y) = \begin{cases} (y - f(\mathbf{x}))\tau & \text{if } y \geq f(\mathbf{x}) \\ (f(\mathbf{x}) - y)(1 - \tau) & \text{if } f(\mathbf{x}) > y, \end{cases} \quad (2.10)$$

is parameterised by $0 < \tau < 1$, which controls the conditional quantile represented by the loss. For example, with $\tau = 0.5$, a model will learn to estimate the conditional median, and the loss function becomes proportional to Absolute Loss. Pinball loss is useful as it permits flexibility over the statistical quantities a model is trained to estimate. Moreover, because it enables us to learn functions for different quantiles, we can use an ensemble of models trained with pinball losses of varying τ to estimate a full conditional distribution with a non-parametric form. This is extremely useful in tasks where conditional probabilities need calculating, or where the variance of the label data, y , is not independent of x , this is called heteroscedasticity and many of the statistical properties of square loss breakdown under its presence. Like the previous loss functions, pinball loss is used for regression tasks.

- **Log Loss (logistic regression):** Log loss is used in logistic regression which, despite the name, is a technique used in binary classification (a special case of classification where there are two classes represented by a single output node). The loss function applies the sigmoid activation (see Section 2.1.1) to the output logit of a network,

$$\mathcal{L}_{\text{Log}}(\mathbf{x}, y) = \begin{cases} -\ln(\text{Sigmoid}(f(\mathbf{x}))) & \text{if } y = 1 \\ -\ln(1 - \text{Sigmoid}(f(\mathbf{x}))) & \text{if } y = 0, \end{cases} \quad (2.11)$$

where the setting, binary classification, ensures $y \in \{0, 1\}$. The intuitive interpretation is that $\text{Sigmoid}(f(\mathbf{x}))$ represents the probability (or likelihood)

of class ‘1’ given $\mathbf{x}$. By taking the log of the estimated probability, we obtain a quantity proportional to the information entropy of the prediction [Cover, 1999]. Minimising the entropy in this case can be shown as equivalent to maximising the likelihood of a prediction. This is a common approach to reasoning about classification loss functions. Moreover, log loss is mathematically equivalent to Binary cross entropy loss.

- **Categorical Cross Entropy (CCE):** Categorical cross entropy is an extension of log loss/binary cross entropy loss which is applicable to multi-class scenarios. Accordingly, its functional form mirrors that of log loss,

$$\mathcal{L}_{\text{CCE}}(\mathbf{x}, y) = - \sum_{c \in \mathcal{C}} y_c \ln(\text{Softmax}(f(\mathbf{x}))_c), \quad (2.12)$$

where $\mathcal{C}$ is the set of categorical classes. In classification, labels are usually ‘one-hot’ encoded, such that $y_c = 0$ where $c \neq k$ and $y_k = 1$, where k is the correct class. Where this holds we can simplify Equation (2.12) to $\mathcal{L}_{\text{CCE}}(\mathbf{x}, y) = -\ln(\text{Softmax}(f(\mathbf{x}))_k)$, where the similarity with log loss becomes clear. Categorical cross entropy is used extensively across multi-class classification in deep learning. It will reappear frequently in the remainder of this thesis and so it is worth outlining some key properties. Firstly, minimising CCE is equivalent to maximising the likelihood of predictions under the observed data and model parameters (this result draws on the assumption that the samples in our dataset are independent and identically distributed (iid)), secondly, it is smooth and differentiable everywhere, and thirdly, it is lower bounded at 0. It is also worth noting that after training, it is not necessary to apply the sigmoid function to the output logits during inference. Given the monotonicity of the sigmoid function, the maximal logit is the same with or without sigmoid. For this reason it is usually left out during inference

for computational efficiency, and during training it is usually computed in combination with the loss function itself to improve numeric stability.

When we train a neural network we are fitting a highly parametrised model to a sample of data points. It is key to recall that the sample of data to which we have access does not represent the full data distribution in which we are interested. Due to this, neural networks are prone to *overfit* where the model performs well on the training data, but poorly on unseen data, not used during training [Goodfellow, 2016]. It is, therefore, common to modify the loss functions outlined above with terms designed to limit the effects of overfit and encourage better performance on unseen data. We call a network’s ability to perform well on unseen data ‘generalisability’ or ‘generalisation’. The terms we add to loss functions to encourage better generalisation by introducing model bias are called *regularisation* terms [Bishop and Nasrabadi, 2006].

Below we outline a few of the most common regularisation techniques used in training neural networks:

- Lasso: Lasso, sometimes called ‘L1’ regularisation [Tibshirani, 1996], adds to the loss a penalty proportional to the magnitude of the network parameters,

$$\mathcal{L}_{\text{Lasso}}(x, y) = \mathcal{L}(x, y) + \lambda \sum_{w \in W} |w|, \quad (2.13)$$

where $\mathcal{L}$ is any generic loss function, λ , is a term to weight the effect of the regularisation, and W represents the set of parameters in the model.

- Weight Decay (Ridge): Weight decay, often called Ridge in classical machine learning or L2 more recently [Hoerl and Kennard, 1970]⁵, is the most frequently used regularisation scheme and is almost always applied during

⁵The ways in which ridge regularisation and weight decay are applied is slightly different. However, in the context of neural network training by stochastic gradient descent, they are mathematically equivalent. For more information, see [Goodfellow, 2016].

training of neural networks. Functionally, it is similar to Lasso, however the penalty term is proportional to the squared magnitude of parameters,

$$\mathcal{L}_{\text{Weight Decay}}(x, y) = \mathcal{L}(x, y) + \lambda \sum_{w \in W} w^2. \quad (2.14)$$

- Dropout: Differently to the previous regularisation schemes, dropout is a stochastic process in which, during training, a random selection of node values are temporarily set to zero [Srivastava et al., 2014]. The proportion of nodes ‘dropped out’ is parameterised by α , where $\alpha = 0.1$ will randomly set 10% of the nodes in each layer to zero. After training, dropout is not applied during inference.

With the network operations and loss functions defined, the final component of training a neural network is updating the model parameters. Specifically, we want to update the parameters such that they reduce the value of the loss function. In deep learning, an efficient algorithm called ‘backpropagation’⁶ is used to compute the gradient of the loss with respect to the model parameters. Backpropagation recursively applies the chain rule backwards through the network layers to obtain gradients in terms of the parameters in each layer [Rumelhart et al., 1986]. Formally,

$$\frac{\partial \mathcal{L}}{\partial W^{(l)}} = \frac{\partial \mathcal{L}}{\partial L^{(l)}(x^{(l-1)})} \times \frac{\partial L^{(l)}(x^{(l-1)})}{\partial W^{(l)}}, \quad (2.15)$$

where $L^{(l)}(\cdot)$ is the layer-wise operation defined in Definition 1 and $W^{(l)}$ represents the set of parameters (weights and bias) in layer l . Recursively applying Equation (2.15) to previous layers gives us a value for the gradient of the loss with respect to all model parameters. In essence, we have a $1 \times K$ Jacobian

⁶There exist efficient processes for computing gradients called *automatic differentiation*. In particular, when computing gradients on neural networks, an algorithm called reverse-mode automatic differentiation is usually used [Griewank and Walther, 2008].

$\nabla_W \mathcal{L}(\mathbf{x}, y) = \frac{\partial \mathcal{L}}{\partial \mathbf{W}} = [\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}}, l = 1, \dots, K]^\top$, where K is the number of layers.

With the gradients computed, updating the parameters is expressed by

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_W \mathcal{L}(\mathbf{x}, y), \quad (2.16)$$

where η is the step size hyperparameter. In Equation (2.16) we have represented the data involved in a single training step as a single input-output pair, $(\mathbf{x}, y)$. In practice, this is not often the case, instead training is performed in batches, which may, in the case where the dataset is not memory-intensive, include the entire training set. More often, training steps are computed over ‘mini-batches’, the gradients are computed for all inputs in the batch, as in Equation (2.15), and then averaged for the update step in Equation (2.16) [Goodfellow, 2016]. Modern computer hardware enables highly efficient parallel computation of mini-batch gradients. A simplified algorithmic exposition of training is given in Algorithm 1.

Algorithm 1: Training Neural Network**Input:** Training set $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$, neural network, N , step size η , number of epochs E , loss function, $\mathcal{L}$, and batch size, S **Output:** Learned model parameters W

```

1 Initialize parameters  $W$ ;
2 for epoch  $\leftarrow 1$  to  $E$  do
3   for each mini-batch  $B \subset \{1, \dots, N\}$  do
4     Loss  $\leftarrow 0$ ;
5     for  $(\mathbf{x}^{(i)}, y^{(i)})$  in  $B$  do
6        $\hat{y}^{(i)} \leftarrow N(\mathbf{x}^{(i)})$ ;
7       Loss  $\leftarrow$  Loss +  $\mathcal{L}(\hat{y}^{(i)}, y^{(i)})$ ;
8     end
9     Loss  $\leftarrow$  Loss/ $S$ ;
10     $\nabla_W \leftarrow$  Backprop(Loss,  $W$ );
11     $W \leftarrow W - \eta \nabla_W$ ;
12  end
13 end

```

In practice, the step size η , is adjusted dynamically during training, decreasing as training progresses [Bengio, 2012]. This improves convergence and stability of the training algorithm when approaching local optima. All hyperparameters of the model - step size, epochs, batch size, network architecture, and any loss-specific parameters - are refined iteratively in a process called hyperparameter fine-tuning [Bergstra and Bengio, 2012]. A given set of hyperparameters coupled with the training algorithm is often called a *training scheme*. Though intelligent initialisation schemes have been proposed, there is currently no task-invariant training scheme with universally good performance; however, parameter initialisations have been shown to influence post-training performance and

convergence times. An initialisation scheme called ‘He initialisation’ [He et al., 2015] is frequently applied, including as the default in `PyTorch`, a widely-used package for deep learning.

In this section, we have introduced neural networks and formally defined their common operations, activations, and outlined the procedures and mathematical details involved in training them. In the next section we extend the notion of neural networks to cover probabilistic methods.

2.2 Bayesian Neural Networks

Thus far, the discussion has been limited to models designed to learn point estimates in the parameter space. The resulting models represent deterministic mappings from input to output. Though this training methodology has been shown to be effective in many tasks, its lack of probabilistic expressivity makes it challenging to quantify model uncertainty.

In general, we can divide the uncertainty associated with a model’s prediction into two contributors: *epistemic* uncertainty and *aleatoric* uncertainty [Hüllermeier and Waegeman, 2021]. Aleatoric uncertainty arises from inherent randomness in the data generation and/or collection processes. Due to this, aleatoric uncertainty is sometimes referred to as the *irreducible* error. Conversely, epistemic uncertainty, or model uncertainty, results from a model’s inability to perfectly capture the true, underlying data distribution. The primary contributor to epistemic uncertainty is the fact that training sets do not represent a complete data manifold. Accordingly, in areas of the input space under-represented in the training set, we would expect a model’s confidence to be lower and uncertainty to be higher.

In the models discussed in Section 2.1, we can obtain an estimate for the prediction uncertainty in classification tasks by examining the Sigmoid or Soft-

max of the model output. However, probability estimates produced in this way are known to be poorly calibrated [Guo et al., 2017]. Moreover, examining the probabilistic confidence tells us little about the origin of the uncertainty.

Bayesian approaches aim to specifically model epistemic uncertainty by considering the uncertainty in the parameter estimates. Research includes approaches which attempt to indirectly approximate Bayesian uncertainty using stochastic processes such as dropout [Gal and Ghahramani, 2016]. BNNs, in contrast, directly model uncertainty as distributions on the parameters [Neal, 2012a].

Definition 2 (Bayesian Neural Network (BNN)). A Bayesian neural network B , is a set $\mathcal{T} = \{L^{(l)} \mid l = 1, \dots, K\}$, of K layers. Each layer is a tuple $L^{(l)} = (h_{\mathbf{w}^{(l)}}(\cdot), g(\cdot))$ consisting of an element-wise activation function, $g(\cdot)$, and a parameterised linear function, $h_{\mathbf{w}^{(l)}}$. The parameters of the linear function are drawn from an inferred posterior distribution, $\mathbf{w}^{(l)} \sim P(\mathbf{w}^{(l)}|\mathcal{D}) = P_{\mathbf{w}^{(l)}}$, with training dataset $\mathcal{D}$. The operation of a single layer of a BNN is defined as $\mathbf{x}^{(l)} = L^{(l)}(\mathbf{x}^{(l-1)}) = g(h_{\mathbf{w}^{(l)}}(\mathbf{x}^{(l-1)}))$. The output of the model is the recursive output through the layers,

$$B(\mathbf{x}^{(0)}) = \int_{\mathbf{W}} P(L^{(l)}(L^{(l-1)}(\dots L^{(1)}(\mathbf{x}^{(0)})))|\mathbf{x}^{(0)}, \mathbf{w})P(\mathbf{w}|\mathcal{D}) \, d\mathbf{w}. \quad (2.17)$$

For the linear and activation functions $h_{\mathbf{w}^{(l)}}(\cdot)$ and $g(\cdot)$ we do not require any different from those defined in Section 2.1. The significance of a BNN's parameters being defined by a sampling distribution is twofold: firstly, it gives us a tool to reason directly about the uncertainty of a model as indicated by the parameter distributions; secondly, it changes our training paradigm from that of maximum likelihood estimation to a maximum a posteriori estimation via Bayesian inference [Bishop and Nasrabadi, 2006]. Discussion on BNN training is deferred to Section 2.2.1.

The integral shown in Equation (2.17) and, accordingly, the output of the BNN, represent the marginal likelihood of the output. Concretely, it represents the probability of an output, weighted by the posterior probability density of the network parameters. In practice, exactly computing the marginal likelihood as written in Equation (2.17) is computationally infeasible. Instead, practitioners approximate the marginal likelihood through Monte Carlo sampling [Doersch, 2016],

$$B(\mathbf{x}^{(0)}) \approx \tilde{B}(\mathbf{x}^{(0)}) = \frac{1}{N} \sum_{s=1}^N P(L^{(l)}(L^{(l-1)}(\dots L^{(1)}(\mathbf{x}^{(0)}))) | \mathbf{x}^{(0)}, \mathbf{w}_s), \quad (2.18)$$

where $\mathbf{w}_s$ represents a single sample of all parameters from the posterior, $\mathbf{w}_s \sim P(\mathbf{w}|\mathcal{D})$. Hereafter, whenever referring to the output of a BNN, we will take the output to be that described by Equation (2.18).

2.2.1 Training Bayesian Neural Networks

As mentioned above, ‘training’ a BNN entails learning the posterior distribution of the parameters, $P(\mathbf{w}|\mathcal{D})$, this learning procedure is termed *maximum a posteriori* (MAP) estimation. MAP is semantically different from MLE: instead of maximising the probability of observed data given some parameters, we maximise the probability that a set of parameters explains the observed data [Bishop and Nasrabadi, 2006]. Learning a posterior distribution in this way is called *inference*, and in the case of a BNN, it is specifically *Bayesian inference*.

Bayesian statistics is a paradigm in which we assume some knowledge of a random variable, and then update our *belief* in the presence of new data [Hastie, 2009]. Accordingly, Bayesian inference requires both a prior distribution over the parameters, and a dataset with which to update our belief about the pa-

rameters. Formally, we compute a Bayesian posterior as

$$P(\mathbf{w}|\mathcal{D}) = \frac{P(\mathcal{D}|\mathbf{w}) P(\mathbf{w})}{P(\mathcal{D})}, \quad (2.19)$$

where $P(\mathbf{w})$ is the prior over the parameters and $P(\mathcal{D})$ is called the marginal. Since the marginal, $P(\mathcal{D})$, is not a function of the network parameters and is constant for a fixed training set, maximising the posterior is equivalent to maximising the numerator in Equation (2.19); for this reason, we can ignore the marginal $P(\mathcal{D})$. The prior, $P(\mathbf{w})$, forms part of the design choice of a BNN, and there is significant research output on selecting *informed* priors for various tasks. In general, Gaussian priors are accepted as the default starting point for most tasks [Neal, 2012a]. Unless otherwise stated, in this work we use exclusively standard normal (Gaussian) priors over the parameters.

It is worth stopping to note the connection between Bayesian MAP and MLE. Selection of the prior is key to Bayesian statistics as it reflects an initial guess of where we expect the true parameters to lie. For example, if we believe we have no information about where the true parameters lie, we could choose to use a uniform distribution over the support of our parameters as a prior distribution. In this circumstance, the prior in Equation (2.19) is constant for all values of $\mathbf{w}$, this means maximising the posterior $P(\mathbf{w}|\mathcal{D})$ is equivalent to maximising the likelihood $P(\mathcal{D}|\mathbf{w})$, with this prior, the MAP estimate of $\mathbf{w}$ is equivalent to the MLE estimate.

To parallel the narrative of training an FFNN, ‘training’⁷ a BNN consists of obtaining samples from the posterior to be applied in Equation (2.18). In general, the true posterior distribution, $P(\mathbf{w}|\mathcal{D})$, does not have a well-defined functional form. This makes it extremely difficult to sample from. Accordingly,

⁷We use the term training cautiously, as technically we are performing statistical inference. Hereafter we will use ‘training’ to refer to the parameter updates outlined in Section 2.1, and ‘inference’ when discussing either obtaining samples from a posterior distribution, or approximating parameters of a posterior.

there are a number of approaches to obtaining posterior samples, w_i . Here we outline a few:

- Variational Inference (VI): Variational inference is a class of inexact algorithms which assume a functional form on the posterior distribution [Blei et al., 2017, Bishop and Nasrabadi, 2006]. Specifically, they define an approximate posterior distribution, $Q_\theta(\mathbf{w}|\mathcal{D})$, and update the distribution parameters to approximate the true posterior $P(\mathbf{w}|\mathcal{D})$ as closely as possible. Concretely, we assume a multivariate Gaussian prior, $P(\mathbf{w}) = N(\mathbf{0}, I)$ with $\mathbf{0} \in \mathbb{R}^{|W|}$, and I the $|W| \times |W|$ identity matrix. Then we estimate the parameters of the posterior distribution, $\boldsymbol{\mu} \in \mathbb{R}^{|W|}$ and $\Sigma \in \mathbb{R}^{|W| \times |W|}$, corresponding to the mean vector and covariance matrix of a multivariate Gaussian. To estimate the parameters VI algorithms minimise the KL-divergence between the approximate posterior and the true posterior,

$$\hat{\theta} = \arg \min_{\theta} KL[Q_\theta(\mathbf{w}|\mathcal{D})||P(\mathbf{w}|\mathcal{D})], \quad (2.20)$$

where $KL[\cdot]$ is the Kullback-Leibler statistical distance and θ represent the parameters of the approximate posterior [Kullback and Leibler, 1951]. It is important to note that these are *not* the same as the parameters of the model over which we are trying to obtain a posterior distribution. The advantage of VI is that the functional form of the posterior enables runtime inference of parameter values w_i , removing the need to permanently store many sampled values for use at runtime. Its drawback is obvious, the posterior obtained is an approximation and so the samples w_i incur statistical bias based on the functional form used for the approximate posterior. VI is an architecture-agnostic approach for approximating Bayesian posteriors. A seminal work on inference algorithms for BNNs introduced an efficient approach called Bayes-by-backprop [Blundell et al., 2015]; to this day, it is a popular approach in

BNN posterior inference.

- Markov Chain Monte Carlo (MCMC): Unlike VI-based algorithms, MCMC algorithms obtain samples through simulation of the posterior [Brooks et al., 2011]. These approaches do *not* require a user to assume a parametric form of the posterior, and are therefore considered to be a more representative approach to obtaining posterior samples. We will focus on an MCMC algorithm called Hamiltonian Monte Carlo (HMC) as it is known to produce highly accurate samples from the posterior immediately (that is, without *warm-up* samples, required for accurate inference in some other MCMC algorithms) [Neal, 2012b]. HMC models the posterior (of which we do not have a functional form) as a pseudo-physical system governed by Hamiltonian dynamics. By beginning at a randomly initialised value for the parameters, the algorithm simulates traversing the posterior, taking periodic samples. For a given sampling rate, the algorithm is guaranteed to return samples distributed according to the true, unknown posterior. Since we obtain samples directly, it is necessary to permanently store a set of samples to be used in Equation (2.18) at inference. A sample size of 100 is commonly accepted as a suitable trade-off between memory requirements and statistical significance; the storage requirement for such a model is 100x that of a deterministic model of the same architecture.

In this section we extended our definition of neural networks from Section 2.1 to cover their probabilistic counterpart, Bayesian Neural Networks. Further, we have outlined some of the most prominent approaches for carrying out Bayesian inference. We now have the building blocks necessary to begin considering neural network safety. We begin by introducing the notion of network adversaries and robustness, before moving on to look at formal safety guarantees in later sections.

2.3 Adversaries and Robustness

Though deep learning has demonstrated remarkable performance across numerous domains, a landmark paper [Goodfellow et al., 2014] revealed neural networks to be highly susceptible to making incorrect predictions when exposed to small, often imperceptible⁸ perturbations. The authors termed such an input an *adversarial attack*. Since then, numerous methods for constructing adversarial inputs have been proposed. We can broadly categorise them into *white-box* attacks and *black-box* attacks [Papernot et al., 2016]. The former relies on full knowledge of the model architecture and parameter values, while the latter assumes access only to the model inputs and outputs. Intuitively, white-box attacks are typically more powerful. In this work, we focus on the white-box case, and define adversarial attacks under the white-box assumption.

Definition 3 (Adversarial Attack). Given a neural network, N , an unperturbed input, $\mathbf{x}$, a set of input constraints, $\mathcal{C}_\epsilon$, parameterised by ϵ , a label, y , and the model output, $N(\mathbf{x}) = \hat{y}$, where $y = \hat{y}$. We define an adversarial attack as any input, $\mathbf{x}'$, such that

$$N(\mathbf{x}') = \hat{y}' \neq y \quad \text{s.t.} \quad \mathbf{x}, \mathbf{x}' \vdash \mathcal{C}_\epsilon, \quad (2.21)$$

where $\mathbf{x}, \mathbf{x}' \vdash \mathcal{C}_\epsilon$ means that the unperturbed and perturbed inputs jointly satisfy the input constraints.

In Definition 3 we present an adversarial attack in the task-agnostic case. In regression tasks, we might consider $\hat{y}' \neq y$ to mean an output that is sufficiently numerically far from the correct or original output. In classification tasks, we would consider a successful attack to be one in which the output class is changed;

⁸Imperceptible meaning either visually undetectable to a human in the case of vision tasks, or semantically insignificant in the case of numeric data.

that is, the maximum output logit is changed under the perturbed input, $\mathbf{x}'$. Moreover, in Definition 3 we do not explicitly define the set of input constraints, $\mathcal{C}_\epsilon$. Though we will explore different options for $\mathcal{C}_\epsilon$ in Chapter 3, the most common set of constraints are given by ℓ_p -norm balls centred on $\mathbf{x}$,

$$\mathcal{C}_\epsilon = \{\|\mathbf{x} - \mathbf{x}'\|_p \leq \epsilon\}. \quad (2.22)$$

In [Goodfellow et al., 2014], the authors use attacks bounded by the ℓ_∞ -norm, though subsequent works have investigated attacks based on a variety of other attack budgets. Notably, in Chapter 3 we discuss an attack budget built on geometric transformations in the computer vision domain. When discussing adversarial attacks hereafter, we will assume an ℓ_∞ -norm budget on the input, unless otherwise stated.

We can compute an adversarial attack using the same computational framework outlined in Section 2.1. Formally,

$$\mathbf{x}' = \arg \max_{\hat{\mathbf{x}}} \mathcal{L}(\hat{\mathbf{x}}, y) \quad \text{s.t.} \quad \mathcal{C}_\epsilon, \quad (2.23)$$

where $\mathcal{L}(\cdot)$ can be any of the loss functions outlined in Section 2.1 and we update $\mathbf{x}$ via backpropagation, taking derivatives with respect to the input rather than the parameters of the network. We commonly use *projected gradient descent* (PGD) [Madry et al., 2018] to solve Equation (2.23); an outline of the PGD algorithm is given in Algorithm 2. It is important to note that we cannot *guarantee* finding an adversarial attack by Equation (2.23), nor can we guarantee finding the true point at which loss is maximised, this is due to the non-convexity of the loss space, and the restrictions imposed by the constraints [Carlini and Wagner, 2017a]. In general, we may not want to find the point maximising the loss as outlined in Equation (2.23), instead favouring the adversarial point

which minimises the distance to the unperturbed input. It is trivial to achieve this by adding an early-stopping criterion to Equation (2.23) when $\hat{y}' \neq y$.

Algorithm 2: Projected Gradient Descent

Input: Input-output pair $(\mathbf{x}, y)$, neural network, N , step size η , loss function, $\mathcal{L}$, step count, S , and attack constraints $\mathcal{C}_\epsilon$.

Output: Adversarially perturbed input $\mathbf{x}'$.

```

1 Initialize parameters  $W$ ;
2  $\mathbf{x}' \leftarrow \mathbf{x}$ ;
3 for each step  $s = 1, \dots, S$  do
4      $\hat{y} \leftarrow N(\mathbf{x}')$ ;
5     Loss  $\leftarrow \mathcal{L}(\hat{y}, y)$ ;
6      $\nabla_{\mathbf{x}'} \leftarrow \text{Backprop}(\text{Loss}, \mathbf{x}')$ ;
7      $\mathbf{x}' \leftarrow \text{Project}(\mathbf{x}' + \eta \nabla_{\mathbf{x}'}, \mathcal{C}_\epsilon)$ ;
8 end
```

Robustness is closely related to adversarial attacks, and can best be described as a network’s natural resistance to adversarial attacks. Formally, we can measure the empirical robustness, $\tilde{R}(N)$, of a neural network N as,

$$\tilde{R}_{\mathcal{C}_\epsilon}(N) = \frac{1}{N} \sum_{i=1}^N \mathbb{1}(N(\mathbf{x}') = y), \quad (2.24)$$

where $\mathbb{1}(\cdot)$ is the indicator function and $\mathbf{x}'$ is an adversarial input found via Equation (2.23). The empirical robustness is upper bounded by the accuracy of the model on unperturbed inputs. Intuitively, we would tend to favour a model with higher empirical robustness and, to this end, training methods have been developed with the goal of improving model robustness [Goodfellow et al., 2014].

Adversarial training is a training scheme designed to improve the empirical robustness of models [Madry et al., 2018]. The premise is to utilise adversarial examples during training in a data augmentation process. Specifically, the

attacks are produced dynamically during training to avoid over-fitting to any specific adversaries. We can define the objective function for adversarial training on a single input,

$$\min_W \max_{\mathbf{x} \in \mathcal{C}_\epsilon} \mathcal{L}(N_W(\mathbf{x}), y). \quad (2.25)$$

Whilst solving Equation (2.25) directly is intractable, we can approximate a solution by sequentially performing the inner maximisation using Algorithm 2, and the outer minimisation using Algorithm 1. In practice, we do not attack every point in the training set using PGD when performing adversarial training; instead, we use a combination of unperturbed and perturbed inputs. Adding attacked points to the dataset in this way is a special case of data augmentation. It is clear to see that the inner maximisation step adds a computational overhead on top of that for standard training: for every training step, we must perform S additional forward passes and back-propagations. For this reason, a small value for S is often used in combination with a large step size, η . This approach favours computational efficiency over finding accurate solutions to the inner maximisation of Equation (2.25). A similar approach introduced by [Shafahi et al., 2019] foregoes recomputing gradients during the attack phase, instead recycling gradients computed for the parameter update on the unperturbed input. Using these gradients produces an even coarser estimate of the adversary in favour of further efficiency.

Though we have thus far discussed well-defined, targeted attacks, it is important to note that models are vulnerable to other, naturally occurring threats. As mentioned in Chapter 1, model performance is unpredictable on data which differs meaningfully from that seen in training, we call such input data out-of-distribution (OOD) data. Examples of OOD data are extremely wide ranging - common examples are images of objects or animals not present in the training data. For example, a cat/dog classifier which has only seen Pugs in the dog

class my struggle to correctly classify a German Shepherd, and may behaviour unpredictably in such a scenario. Examples of OOD can also be more subtle, such as different data compression methods being applied during training and inference, rare market conditions unseen in training unsettling an algorithmic trading model, or background noise in a medical scan from inputs produced by a new model of scanner. Many of the methods we discuss in subsequent chapters pertain to mitigating the effects of OOD data, or guaranteeing resilience to it altogether. To aid in these discussions, we now we outline some of the previous work on generalised defences to OOD data. In an early work, [Hendrycks and Gimpel, 2016] demonstrated that the maximum softmax output, a frequent measure of confidence, could be used to identify OOD inputs to the model. Identifying OOD inputs can be a powerful defence, as it enables practitioners to be cautious with the associated predictions, or commit more resources to such samples, such as deploying human validators. Detecting potentially-problematic inputs such as this remains an active field and has come to be known as ‘monitoring’ [Liu et al., 2020, Devries and Taylor, 2018, Hendrycks et al., 2019]. In [Ding and Pang, 2024], the authors note that OOD detection models can be susceptible to the same attacks we have discussed above, and propose a training scheme which combines adversarial attacks with OOD data to improve detection robustness. Further work has explored the link between OOD robustness and robustness to adversarial attacks [Zou and Liu, 2023, Yi et al., 2021], with [Yi et al., 2021] demonstrating the effectiveness of adversarial training in directly improving the robustness to OOD inputs. It is this insight that we build upon in Chapter 4.

2.4 Verification of Neural Networks

In Section 2.3 we outlined neural network’s vulnerability to adversarial attacks and discussed the notion of robustness, as well as how we can use adversarial training to improve the empirical robustness. We also introduced a simple safety metric in empirical robustness, and delineated its computation. In this section, we discuss a stronger safety metric called *verification*.

Verification, in the context of deep learning, seeks to apply formal methods to the network function and obtain *guarantees* on the set of the output [Katz et al., 2017, Pulina and Tacchella, 2010, Ehlers, 2017, Lomuscio and Maganti, 2017].

Definition 4 (Verification Query). Given a network, N , an input, $\mathbf{x}$, a parameterised set of input constraints, $\mathcal{C}_\epsilon$, ϵ , and a correct output, y , where $N(\mathbf{x}) = \hat{y} = y$. A verification query, $\mathcal{Q}$, seeks to prove

$$\mathcal{Q} := \nexists \mathbf{x}' \vdash \mathcal{C}_\epsilon \quad \text{s.t.} \quad N(\mathbf{x}') \neq y, \quad (2.26)$$

where $\mathbf{x}' \vdash \mathcal{C}_\epsilon$ means an input $\mathbf{x}'$ satisfying the attack budget constraints as discussed in Section 2.3.

Typically, we can frame Equation (2.26) as an optimisation problem,

$$\begin{aligned} \gamma &= \min_{\mathbf{x}'} \mathbf{a}^\top N(\mathbf{x}'), \\ &\text{s.t. } \mathbf{x}' \vdash \mathcal{C}_\epsilon, \end{aligned} \quad (2.27)$$

constrained by the input constraints in $\mathcal{C}_\epsilon$. In Equation (2.27) we use a vector, $\mathbf{a}$, composed with the network’s output layer, to encode the desired output constraints. For example, in a classification setting we may have $a_t = -1$, $a_c = 1$ and $a_i = 0, \forall i \neq c, t$. In this formulation, an objective value greater than

0, $\gamma > 0$, ensures that the logit associated with the correct class, c , is always greater than that of some incorrect target class, t . Repeating this for all output nodes $i \neq c$, would provide guarantees against all classes. We can produce constraints on the output for regression tasks similarly by re-characterising $\mathbf{a}$. We will denote a valid lower bound on γ as $\underline{\gamma}$, and the optimal lower bound under a given approach as $\underline{\gamma}^*$. It is important to emphasise that in order for a verification procedure to be *sound*⁹, we require that the optimisation problem given in Equation (2.27) is convex.

As with robustness, we can define a verification metric function, $\tilde{V}(\cdot)$,

$$\tilde{V}(N) = \frac{1}{N} \sum_{i=1}^N \mathbb{1}(N(\mathbf{x}') = y \quad \forall \mathbf{x}' \vdash \mathcal{C}_\epsilon). \quad (2.28)$$

Given that $\tilde{V}(\cdot)$ computes the average number of inputs in a sample set with *no* adversarial attacks, and $\tilde{R}(\cdot)$ computes the average number of inputs in a sample set with no empirically observed adversarial attacks, it follows that $\tilde{V}(\cdot)$ is a lower bound on $\tilde{R}(\cdot)$ for a given attack budget ϵ . The numeric difference between the metrics arises from the fact that when solving Equation (2.23) we are not guaranteed to find $\mathbf{x}^*$, the input satisfying ϵ which globally maximises $\mathcal{L}(\cdot)$. This is due to the non-convexity of the loss space and is directly related to the risk of local minima when training a network.

Solving Equation (2.28), then, is a difficult task, and equates to finding a valid lower bound on $\mathbf{x}^*$, $\underline{\mathbf{x}^*}$, and ensuring that $N(\underline{\mathbf{x}^*}) = y$. If we can show that $N(\underline{\mathbf{x}^*}) = y$ holds, then we say our verification query is *satisfied*, $\mathcal{Q} = \text{SAFE}$; if, otherwise, we can provably falsify $\mathcal{Q}$ by finding an input $\mathbf{x}' \vdash \mathcal{C}_\epsilon$ such that $N(\mathbf{x}') \neq y$, then we say the query is *unsatisfied*, $\mathcal{Q} = \text{UNSAFE}$; and finally, if we cannot provably show either, then we say the query is *undecided*,

⁹Meaning that if we obtain a positive verification result, we have guarantees that this result is valid.

$\mathcal{Q} = \text{UNDECIDED}$. Verification approaches can be broadly divided into two categories called, *complete* and *incomplete* verification.

Definition 5 (Complete Verification). A verification algorithm is complete if it can prove firstly that,

$$\mathcal{Q} \in \{\text{SAFE}, \text{UNSAFE}\}, \quad (2.29)$$

and secondly, that it terminates in finite time.

Definition 6 (Incomplete Verification). A verification algorithm is incomplete if it can find a valid lower bound, $\underline{\mathbf{x}}^*$, in finite time; that is,

$$\mathcal{Q} \in \{\text{SAFE}, \text{UNSAFE}, \text{UNDECIDED}\}. \quad (2.30)$$

We now explore some common approaches to incomplete and complete verification and outline their applicability. We will constrain the exposition to the high-level technical details and refer to [Liu et al., 2021] for a more in-depth discussion.

2.4.1 Incomplete Verifiers

As a reminder, in verification we are concerned only with algorithms which produce sound bounds on the output. Within this category, there are two approaches: complete and incomplete. Incomplete verifiers use convex over-approximations of the model operations in obtaining bounds on the output set. For this reason, they may not always result in a conclusive answer to a verification query (termed UNDECIDED above). Nonetheless, the over-approximate arithmetic they rely on is typically much faster than exact methods used in complete verifiers, and so they are still used extensively in modern verification.

Interval Bound Propagation

Interval bound propagation (IBP) is a forward propagation technique based on interval arithmetic [Zhang et al., 2018, Wang et al., 2018a]. To further discuss IBP, we must generalise our notation of neural networks to consider bounds on latent representations. To this end, based on a set of attack constraints parameterised by ϵ and as discussed in Section 2.3, we can define the IBP bounds on the input vector, $\underline{\mathbf{x}}_{\text{IBP}} = \min_{\mathbf{x}'} \mathbf{x}' \vdash \mathcal{C}_\epsilon$, where the minimisation is applied element-wise; the upper bound is computed accordingly via maximisation. After each layer of an FFNN, we can define the IBP bounds in terms of the bounds from the previous layer, and the parameters of the layer. Concretely,

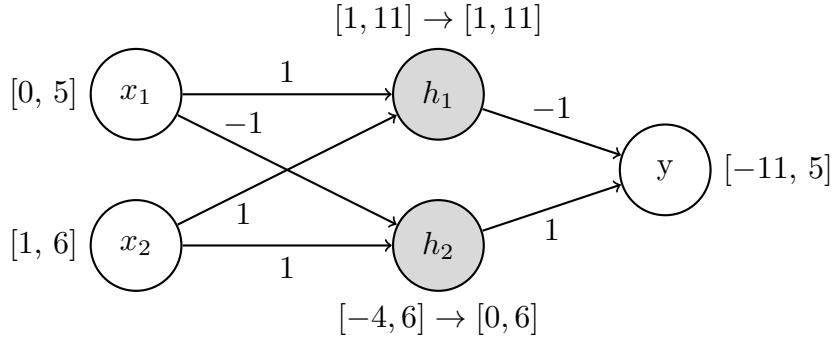
$$\begin{aligned} \underline{x}_{\text{IBP},i}^{(l)} &= g \left(\sum_{j \in J^+} [W_{i,j}^{(l)} \underline{x}_{\text{IBP},j}^{(l-1)}] + \sum_{j \in J^-} [W_{i,j}^{(l)} \bar{x}_{\text{IBP},j}^{(l-1)}] + b_i^{(l)} \right), \\ \bar{x}_{\text{IBP},i}^{(l)} &= g \left(\sum_{j \in J^+} [W_{i,j}^{(l)} \bar{x}_{\text{IBP},j}^{(l-1)}] + \sum_{j \in J^-} [W_{i,j}^{(l)} \underline{x}_{\text{IBP},j}^{(l-1)}] + b_i^{(l)} \right), \\ J^+ &= \{j | W_{i,j} \geq 0\} \quad J^- = \{j | W_{i,j} < 0\}. \end{aligned} \tag{2.31}$$

In this case, we consider the parameters, W and $\mathbf{b}$, to be fixed values. In most cases, this is a fair assumption. However, in Chapter 5 we will examine how we must shift our definition for BNNs where the parameters are not fixed, but rather sampled from distributions.

The arithmetic approach of IBP allows us to observe bounds on concrete values at each layer of the network and is extremely fast to compute. Specifically, the computational complexity is equivalent to two forward passes of the network itself: once for the lower bound and once for the upper bound. However, the approach is limited due to the coarseness of interval arithmetic over a neural network. To further illustrate this point, we refer to Example 1.

Example 1 (IBP). *We begin with an FFNN as defined in Definition 1 with one*

Figure 2.2: An FFNN with one hidden layer and ReLU activations applied. Weight values are displayed adjacent to their corresponding edges and post-activation IBP bounds are shown in square brackets adjacent to nodes. The bias values for the network are set to 0.



hidden layer. We can, without loss of generality, set all bias values to 0 for ease of exposition. The network is displayed in Figure 2.2 where the weights and IBP bounds are also given. The input to this network is bounded by $0 \leq x_1 \leq 3$ and $0 \leq x_2 \leq 2$. This corresponds to an ℓ_∞ -bounded input vector, $\mathbf{x} = (2.5, 3.5)^\top$ with budget $\epsilon = 2.5$. To propagate the input bounds, which by definition of the threat model are defined in terms of interval bounds, through the hidden layer we apply Equation (2.31). The pre-activation bounds for h_2 are $[-4, 6]$; however, we can truncate the bound to that shown in the figure by applying ReLU. It is informative to look at the input values corresponding to the bounds at the hidden layer. For h_1 we have $x_1 = 0, x_2 = 1 \rightarrow \underline{h_1}$ and $x_1 = 5, x_2 = 6 \rightarrow \overline{h_1}$. Moreover, for h_2 , $x_1 = 5, x_2 = 1 \rightarrow \underline{h_2}$ and $x_1 = 0, x_2 = 6 \rightarrow \overline{h_2}$. It is important to note that the inputs associated with $\underline{h_1}, \overline{h_2}$ are different. The same holds for the pair $\overline{h_1}, \underline{h_2}$. This is not an issue at the hidden layer; in fact, it can be shown that the bounds are exact with respect to the input bounds. Proceeding, however, to the output bound by again applying Equation (2.31) we have that $h_1 = \overline{h_1}, h_2 = \underline{h_2} \rightarrow \underline{y}$. Here there is a contradiction; we have shown that $\overline{h_1}$ and $\underline{h_2}$ are not reachable for a fixed input $\mathbf{x}$. Concretely, the events that $h_1 = \overline{h_1}$ and

$h_2 = \underline{h_2}$ are mutually exclusive. Specifically, for the output $y = \underline{y}$ we require, for example, that $x_2 = 6$ and $x_2 = 1$, since the value for x_2 cannot assume two values simultaneously, this event is impossible. What we have observed is over-approximation of the bounds on the network. The reason for this is that we ignore the relationship between the bounds on h_1 and h_2 that is implicit in their mutual dependence on $\mathbf{x}$. This over-approximation is an inherent feature of IBP, and leads to loose bounds on the output of the network.

We have seen that IBP is a highly computationally efficient method for verification that incurs substantial error in large networks. In practice, we find that IBP is insufficient to solve verification queries on modern models, frequently returning UNDECIDED. However, as we will see, it is a key component in tighter verification algorithms such as linear bound propagation.

Linear Bound Propagation

Linear bound propagation (LBP) is the first example of symbolic interval propagation (SIP) we discuss, wherein we propagate algebraic intervals through the model's layers as opposed to concrete intervals as in IBP [Singh et al., 2018]. By propagating algebraic expressions, we are able to retain the dependencies on the network's input bounds. Specifically, for LBP we express the bounds in terms of the linear expressions of the network as defined in Definition 1. Concretely, for any layer $L^{(l)}$ we have

$$L^{(l)} = \mathbf{x}^{(l)} = g(W^{(l)}(g(W^{(l-1)}(\dots g(W^{(1)}x^{(0)} + b^{(1)}) \dots) + b^{(l-1)}) + b^{(l)})). \quad (2.32)$$

When we have $g(\cdot)$ as a linear function, we can directly solve this via a linear solver. However, neural networks frequently apply non-linear activations (see Section 2.1.1 for an outline of commonly used functions).

To apply LBP to networks with non-linear activation functions, such as

ReLU, we modify the bound propagation to take linear over-approximations of the non-linear function. Specifically, suppose we are interested in bounding the non-linear ReLU function as in Equation (2.1). Given that ReLU is applied element-wise, we will simplify the notation to consider a scalar input and output, $x = \text{ReLU}(\tilde{x})$. We assume access to concrete lower and upper bounds, l and u , on the input to the activation function, i.e., $l \leq \tilde{x} \leq u$. With these values fixed, we can be in one of three states: $l, u < 0$, $0 \leq l, u$, or $l < 0 \leq u$. We refer to these states as being, *inactive*, *active*, and *unstable* respectively. In the case where a ReLU node is inactive or active (we refer to a node being in either of these states as being *stable*) the ReLU function is guaranteed to be linear over the range $[l, u]$. Since we are concerned with finding linear representations of the output, we can stop here and simply propagate one of $x = 0$ or $x = \tilde{x}$, depending on the state of the node. However, in the case of unstable nodes, there is no way to exactly represent the output, x , in terms of $\tilde{x}$ in a linear form. To navigate this, we introduce linear upper and lower bounds on x defined in terms of the given interval bound $[l, u]$,

$$\begin{aligned}\bar{x} &= \frac{u}{u-l}\tilde{x} + \frac{ul}{l-u} \\ \underline{x} &= \frac{u}{u-l}\tilde{x}.\end{aligned}\tag{2.33}$$

To better understand the form of these bounds, we refer to Figure 2.3, which illustrates the linearly-bounded output set for inactive, active, and unstable nodes. As previously indicated, we observe that for inactive and active nodes the output values are described exactly. Equivalently, the upper and lower linear bounds on stable nodes are identical linear functions of the input. However, in the unstable case the bounds are not identical and form a region enclosing the feasible domain and range of the ReLU operation. Moreover, for a given value of $\tilde{x} = v$, the output of the function as described by the linear bounds is not a

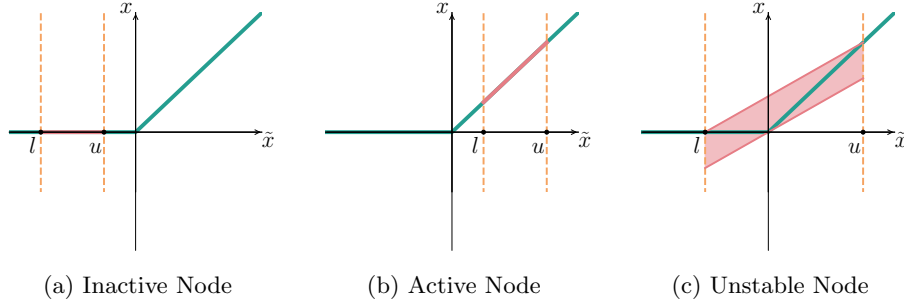


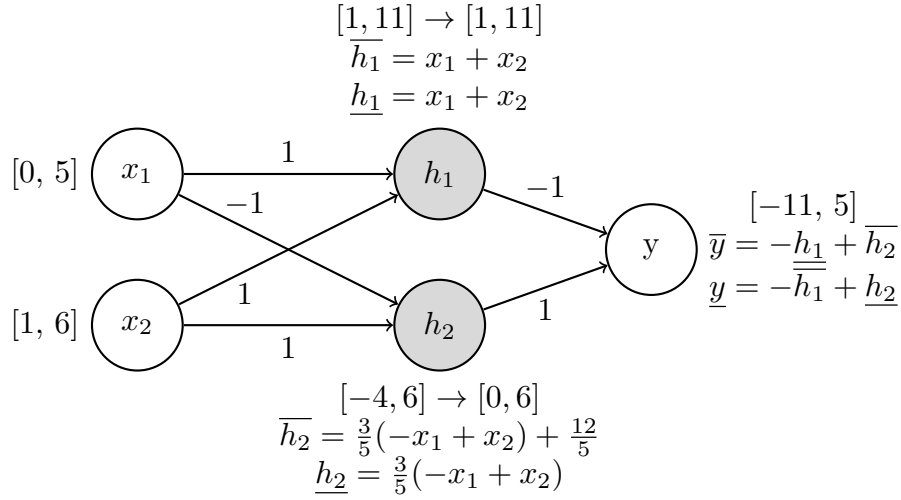
Figure 2.3: Linear bounds on the output, x , of a ReLU node in terms of the input, $\tilde{x}$ for a) inactive nodes, b) active nodes, and c) unstable nodes. In each case, the pink region represents the feasible output values for a given input value. In each case, l and u represent concrete interval bounds on the input $\tilde{x}$.

single value but a range. This mirrors the inexactness observed in Section 2.4.1 for IBP. However, we have reduced the over-approximation error substantially. Thus far, we have assumed access to values of l and u . In practice, these are obtained via IBP as a necessary precursor to running LBP. Given that we know the IBP bounds to be coarse estimates, some works recursively regenerate l and u using the bounds obtained from LBP. Finally, we have shown here one example of a set of linear bounds for unstable ReLU outputs. Many other formulations exist, most of which involve variations on the lower bound gradient [Wang et al., 2018a, Singh et al., 2018, Henriksen and Lomuscio, 2020, Singh et al., 2019b, Zhang et al., 2018, Henriksen and Lomuscio, 2021].

Having examined the bounding procedure for a single ReLU unit, we now take a step back and examine the LBP algorithm over a full network. For this, we refer to Example 2.

Example 2 (LBP). *We study a network with one hidden layer, an input, $\mathbf{x} \in \mathbb{R}^2$, scalar output, and ReLU activations; the network is illustrated in Example 2. The input bounds are the same as in Figure 2.2 and the pre-/post-activation bounds are shown in brackets adjacent to the nodes. Additionally, the bounds based on linear functions of the previous nodes are shown for the non-*

Figure 2.4: An FFNN with one hidden layer and ReLU activations applied. Weight values are displayed adjacent to their corresponding edges and post-activation LBP bounds are shown adjacent to the nodes. The bias values for the network are set to 0.



input nodes. For h_1 , we have that $\bar{h}_1 = \underline{h}_1$; this is because the pre-activation bounds on h_1 are all positive, making this a stable, active node, illustrated by Figure 2.3a. However, node h_2 has pre-activation bounds spanning zero and is an unstable node. Its bounds, $\bar{h}_2$ and $\underline{h}_2$, therefore, are derived according to Equation (2.33). We can write the bounds on y in terms of the input, $\mathbf{x}$ by substituting in expressions for the bounds on h_1 and h_2 ,

$$\begin{aligned}
 \bar{y} &= -x_1 - x_2 + \frac{3}{5}(-x_1 + x_2) + \frac{12}{5}, \\
 &= -\frac{8}{5}x_1 - \frac{2}{5}x_2 + \frac{12}{5}, \\
 \underline{y} &= -x_1 - x_2 + \frac{3}{5}(-x_1 + x_2), \\
 &= -\frac{8}{5}x_1 - \frac{2}{5}x_2.
 \end{aligned} \tag{2.34}$$

With these expressions, we can then minimise the lower bound and maximise the upper bound to find concrete bounds on the output y . As these expressions

are linear, solving for their bounds is trivial. In this case, we get $-10.4 \leq y \leq 2$. This is a range of 12.4, which is 22.5% smaller than the bound range obtained by IBP - a gain achieved by using linear propagation across a single hidden layer.

We have introduced one implementation of LBP and worked through a concrete example to demonstrate its advantage over IBP. Verification by LBP is appealing because it is highly computationally efficient, scaling with $O(NK)$, where N is the input dimension and K is the complexity of a forward pass on the network.

Other Incomplete Methods

LBP + Input Splitting: In Section 2.4.1 we discuss in detail the premise for linear bound propagation. A simple, but important extension to LBP is *input splitting* [Wang et al., 2018b] - a recursive approach to LBP which involves solving sub-problems of the verification query based on splits to the input domain. The details of this algorithm are not essential to the remainder of this work; a high-level understanding, however, is certainly worth the time. In LBP we can represent the input-output relationship of a node exactly when the node is stable; over-approximation errors are introduced only for linear relaxations on unstable nodes. The motivation for input splitting is to reduce the number of unstable nodes, and thereby, the proportional over-approximation errors in the sub-problems. Concretely, given a verification query, $\mathcal{Q}$, with constraints on the input $0 \leq x_i \leq 10$, $i = 1, \dots, |\mathbf{x}|$, input splitting first selects a dimension on which to split, s , yielding two sub-problems, $\mathcal{Q}_1$ and $\mathcal{Q}_2$, with constraints $0 \leq x_i \leq 10$, $\forall i \neq s$; $0 \leq x_s \leq 5$ and $0 \leq x_i \leq 10$, $\forall i \neq s$; $5 < x_s \leq 10$ respectively. In this example we split dimension s in half, though in practice we can split at any point in the range. When splitting on multiple dimensions, we must solve 2^n sub-problems where n is the number of splits. Despite the

exponential complexity with split number, LBP + Input splitting is an effective verification algorithm, largely due to the speed of each single LBP query.

Semidefinite Programming (SDP): Semidefinite programming-based verifiers for ReLU networks leverage the fact that the ReLU function can be encoded exactly by a set of linear and quadratic constraints [Raghunathan et al., 2018],

$$\begin{aligned} \mathbf{x}^{(i)} &\geq 0, \\ \mathbf{x}^{(i)} &\geq W^{(i)}\mathbf{x}^{(i-1)} + b^{(i)}, \\ \mathbf{x}^{(i)} \odot (\mathbf{x}^{(i)} - W^{(i)}\mathbf{x}^{(i-1)} - b^{(i)}) &= 0, \end{aligned} \tag{2.35}$$

where $\odot$ denotes the Hadamard product. Encoding the network layers with Equation (2.35) and solving for the maximum and minimum values on the output is a non-convex problem. However, we can relax the constraints in Equation (2.35) to a convex positive semidefinite (PSD) constraint and solve the verification query by SDP methods. The details of the relaxation and resulting optimisation problem are beyond the necessary material for this work; however, we refer the reader to [Brown et al., 2022] for further details on SDP-based verifiers.

Linear Programming: We have introduced an LBP-based bounding algorithm; however, using propagation-based methods is inherently constrained by our ability to propagate bounds. Specifically, we can only propagate a single upper and lower bound. The convex hull of a ReLU function given input bounds $[l, u]$ is given by,

$$\begin{aligned} \mathbf{x}^{(i)} &\geq 0, \\ \mathbf{x}^{(i)} &\geq \tilde{\mathbf{x}}^{(i)}, \\ \mathbf{x}^{(i)} &\leq \frac{u}{u-l}\tilde{\mathbf{x}}^{(i)} + \frac{ul}{l-u}. \end{aligned} \tag{2.36}$$

This set of constraints is well known and called the *triangle relaxation* [Singh et al., 2019b]. The third constraint is familiar as we introduced it as the upper bound in the LBP algorithm outlined in Section 2.4.1. The lower bound, how-

ever, is now given by two linear constraints; although the problem is tighter, we cannot propagate single bounds through the layers to obtain an expression for the output node. Instead, we must apply linear programming techniques to solve the joint system of linear equations. Though, as mentioned above, for large networks, even with highly-optimised linear solvers and powerful hardware, we find that propagation-based methods combined with tightening techniques such as input splitting provide a better balance between tightness and computational complexity.

2.4.2 Complete Verifiers

Mixed-integer Linear Programming

As stated in Definition 5, complete verification requires a guaranteed definite answer to the verification query in finite time. To achieve this, we must compose an optimisation problem which captures a network’s operations exactly. So far, we have explored strictly linear constraints used to *bound* non-linear operations. Mixed-integer linear programming (MILP) introduces integer constraints to the optimisation problem which, in the case of ReLU activations, enable an exact encoding [Bunel et al., 2020a]. As for LBP, we compute pre-activation bounds on the nodes using IBP. For any stable node, we need only consider the linear expressions $x = \tilde{x}$ or $x = 0$. For unstable nodes, we encode the operation as a union of integer and linear constraints:

$$\mathcal{C}_{\text{MILP}} = x \geq \tilde{x} \cup x \geq 0 \cup x \leq \tilde{x} - l(1 - \delta) \cup x \leq u\delta \cup \delta \in \{0, 1\}, \quad (2.37)$$

where l and u are the upper and lower concrete bounds obtained by IBP respectively, and δ is a binary variable. By inspection of Equation (2.37), we can see that $\delta = 0 \rightarrow x = 0$, and $\delta = 1 \rightarrow x = \tilde{x}$. With this encoding, we frame a

verification query as solving the following optimisation:

$$\begin{aligned}
& \min_{x'} c^\top x^{(l)} \\
& \text{s.t. } \mathcal{C}_{\text{MILP}} \\
& \quad x' \vdash \mathcal{C}_\epsilon,
\end{aligned} \tag{2.38}$$

where c is a constant vector reflecting the output constraints of a verification query. We can use off-the-shelf solvers such as Gurobi to solve Equation (2.38). Similar to input splitting, solving MILP problems suffers from exponential time complexity, $\mathcal{O}(2^n)$, where n is the number of binary variables δ , in the constraints. For this reason, MILP is unsuitable for verification queries on large networks and is reserved for verification queries requiring exact encodings.

2.5 Verification on Geometric Perturbations

Thus far, we have introduced the notion of verification for a generalised setting. Specifically, we explored examples for verification procedures against noise-based threat models. Though noise-based input sets have dominated the adversarial attack, robustness, and verification literature, there is growing interest in developing verification procedures against more specific classes of perturbation. In this section, we explore verification of *geometric* perturbations, covering previous literature and the necessary background.

To emphasise the need for a wider class of threat models, we discuss a concrete example in the image classification domain.

Example 3 (Geometrically-constrained Perturbations). *We begin with the simple case of correctly classifying an image of a stop sign. In this setting, there are many threat models we might want to consider: the stop sign might be dirty, in the shade, illuminated by headlights, or discoloured by wear and tear. All of these*

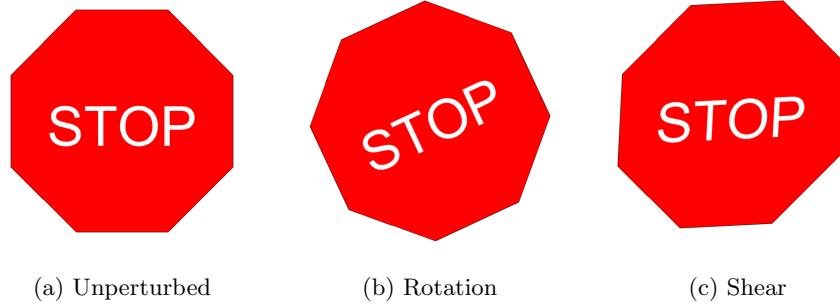


Figure 2.5: Illustration of a stop sign with rotation and shear transformations applied.

effects can be captured by noise-based input perturbations. However, we might also want to check against geometrically-constrained perturbations, such as rotation, shearing, or scaling. These perturbations could represent a stop sign being askew, perhaps due to a collision or vandalism, or at an angle to a driverless car’s camera system. Capturing these effects through noise-based perturbation and verifying a network against them would not be possible, as the change to individual pixel values would be extremely large. To illustrate this point, we refer to Figure 2.5 which shows a stop sign in three positions: Figure 2.5a shows the stop sign in its unperturbed state, Figure 2.5b is the sign after applying a 25° rotation, and Figure 2.5c is the sign after applying a shear in the x and y axes of 0.05. To achieve the transition from Figure 2.5a to Figure 2.5b using a noise-constrained perturbation would require an excessive budget, as the pixels near the edge of the sign go from pure red to pure white, meaning the green and blue channel budgets must span the entire domain of the pixel values. Under such a set of constraints, any image imaginable would be reachable, and it is clear we cannot verify against such an input. Instead, a geometrically-constrained input perturbation has constraints that are directly in terms of the perturbation parameter - for example, the rotation in degrees.

The rich body of work verifying ℓ_p -bounded inputs via MILP [Botoeva et al.,

2020a, Tjeng et al., 2017], branch-and-bound [Bunel et al., 2020a,b, De Palma et al., 2021, Xu et al., 2021, Wang et al., 2021, Henriksen and Lomuscio, 2021, Ferrari et al., 2022, Zhang et al., 2022], and abstract interpretation [Gehr et al., 2018, Singh et al., 2018, 2019b] cannot be used to verify geometric robustness out of the box, as ℓ_p -balls are unable to accurately represent geometric transformations [Kouvaros and Lomuscio, 2018, Singh et al., 2019a]. One of the earliest works on formal methods for geometric verification [Pei et al., 2017] discretises the transformation domains, enabling robustness verification through the evaluation of the model at a finite number of discretised transformations.

To reason about geometrically-bounded input constraints, we need to extend our notion of the constraint set $\mathcal{C}_\epsilon$. Formally, a geometric transformation of an image is a composite function, consisting of a spatial transformation $\mathcal{T}_\mu$, a bilinear interpolation $\mathcal{I}(u, v)$, which handles pixels that are mapped to non-integer coordinates, and changes in brightness and contrast $\mathcal{P}_{\alpha, \beta}$. The spatial transformation $\mathcal{T}_\mu$ can be a composition of rotation, translation, shearing, and scaling; see *e.g.*, [Balunović et al., 2019] for detailed descriptions. The pixel value $\hat{p}_{u,v}$ at position (u, v) of the transformed image is obtained as follows: (1) the pre-image, $\mathcal{T}_\mu^{-1}$, of (u, v) is calculated under $\mathcal{T}_\mu$; (2) the resulting coordinate is interpolated via $\mathcal{I}$ to obtain a value ξ ; (3) $\mathcal{P}_{\alpha, \beta}(\xi) = \alpha\xi + \beta$ is applied to compute the final pixel value $\hat{p}_{u,v}$. In other words, we have that $\hat{p}_{u,v} = \mathcal{G}_{u,v}(\alpha, \beta, \mu)$, where:

$$\mathcal{G}_{u,v}(\alpha, \beta, \mu) := \mathcal{P}_{\alpha, \beta} \circ \mathcal{I} \circ \mathcal{T}_\mu^{-1}(u, v). \quad (2.39)$$

With this formulation, we can consider the input space as the set of reachable pixel values under interval input constraints on α , β , and μ . $\mathcal{G}_{u,v}(\alpha, \beta, \mu)$ represents a non-linear, non-convex mapping between the pixel value at (u, v) before and after a transformation. As discussed in Section 2.4, obtaining sound verification bounds via Equation (2.27) requires a convex set of input constraints. Given

that we cannot precisely capture the non-linear function described in Equation (2.39) as a set of convex constraints, we must over-approximate the set of reachable values. This problem mirrors the convex ReLU approximations discussed in Section 2.4, however, in this setting we cannot assume anything about the form of the underlying pixel value function, nor can we assume consistency of the function’s behaviour across pixels and images.

We will refer to $\mathcal{G}_{u,v}(\alpha, \beta, \mu)$ as the *pixel value function*, as it outputs a given pixel’s value as a function of the transformation parameters. To verify a model against a geometric perturbation, we must express the range of the pixel value function as a convex set. A trivial solution is to define concrete pixel-wise input bounds,

$$\min_{\alpha, \beta, \mu} (\mathcal{G}_{u,v}(\alpha, \beta, \mu)) \leq x_{u,v} \leq \max_{\alpha, \beta, \mu} (\mathcal{G}_{u,v}(\alpha, \beta, \mu)). \quad (2.40)$$

We can proceed to verify a network against this input set using the same methods as for an ℓ_∞ -norm ball.

In [Balunović et al., 2019], the authors propose a linear bound formulation to capture the pixel value as a function of the transformation parameters. Although linear bounds are a substantially tighter approximation than interval bounds, the highly non-linear form of the pixel value function results in large over-approximate bounding sets when using linear bounds. In Chapter 3 we build on the work in [Balunović et al., 2019] to produce a tighter, piecewise-based bounding procedure.

2.6 Verification on Non-deterministic Models

Thus far, we have introduced the relevant concepts and notation for robustness and verification in a deterministic setting - that is, we have considered mod-

els where the parameters are point estimates, fixed after training is complete. In Section 2.2, we introduced Bayesian neural networks and outlined their differences from standard, deterministic neural networks, along with some benefits to this model class. Here, we consider the literature addressing the challenge of verifying Bayesian neural networks, as well as introducing the necessary mathematical notation.

Work towards robustness and verification of Bayesian neural networks builds on the vast, pre-existing robustness and verification literature for deterministic models. For example, in [Cardelli et al., 2019b] the authors compute upper bounds on the robustness of BNNs. Closely related to this, [Cardelli et al., 2019a] consider the problem of computing the probability of whether, for a given input to a BNN, there exists another input within a bounded set around the input such that the network’s prediction differs for the two points, and provide an approach to approximate this probability while providing statistical guarantees. In the same line of research, [Carbone et al., 2020] show that the vulnerability of BNNs to gradient-based attacks arises as a result of the data lying on a lower-dimensional submanifold of the ambient space. [Wicker et al., 2020] use a sampling-based approach to bound the probabilistic robustness of BNNs. Specifically, they sample *orthotopes* from the posterior distribution and use modified IBP and LBP to determine whether a given orthotope is safe. By repeating this process, they obtain a robustness certificate and a lower bound for the probabilistic robustness of the BNN. [Berrada et al., 2021] consider the adversarial robustness of BNNs. They use the notion of Lagrangian duality, which replaces standard Lagrangian multipliers with functional multipliers. For an optimal choice of functional multipliers, this approach leads to exact verification - i.e., the exact computation of probabilistic robustness. Nevertheless, this may not be practical. The authors show that for specific classes of multipliers,

however, one can obtain lower bounds for the probabilistic robustness. [Adams et al., 2023] introduced an algorithm to verify the adversarial robustness of BNNs. They use dynamic programming alongside bound propagation and convex relaxations to compute lower and upper bounds on BNN predictions for a given input set. [Wicker et al., 2023] build on the algorithm first presented in [Wicker et al., 2020] to provide upper bounds on the probabilistic robustness of BNNs in addition to lower bounds. Moreover, they extend the algorithmic formulation in [Wicker et al., 2020] to handle the specification proposed in [Berrada et al., 2021] for posterior distributions with unbounded support. [Lechner et al., 2021] use MILP to obtain robustness guarantees for infinite time-horizon safety and robustness of BNNs. However, this approach can only be used for small networks. [Wijk et al., 2022] obtain robustness guarantees for credal BNNs via constraint relaxation over probabilistic circuits.

We now formally define the probabilistic robustness problem for BNNs.

Definition 7 (Probabilistic Robustness Problem). Given a BNN $B_{\mathbf{w}} : \mathbb{R}^m \rightarrow \mathbb{R}^n$, a *robustness specification* for $B_{\mathbf{w}}$, is a tuple $\varphi = \langle \mathcal{X}, \mathcal{S} \rangle$, where $\mathcal{X} \subseteq \mathbb{R}^m$ is bounded by a set of ϵ -parameterised constraints, $\mathcal{C}_\epsilon$, and $\mathcal{S} \subseteq \mathbb{R}^n$ is a half-space

$$\mathcal{S} = \{\mathbf{y} : \mathbf{a}^\top \cdot \mathbf{y} \geq b\}, \quad (2.41)$$

for some $\mathbf{a} \in \mathbb{R}^n$ and $b \in \mathbb{R}$ where $\mathbf{y} = B(\mathbf{x})$. The *probabilistic robustness problem* concerns determining the probability that for all inputs in $\mathcal{X}$ the output of $B_{\mathbf{w}}$ is in $\mathcal{S}$, i.e.,

$$P_{\text{safe}}(\mathcal{X}, \mathcal{S}) = P_{\mathbf{w} \sim w}(\forall \mathbf{x} \in \mathcal{X}, B_{\mathbf{w}}(\mathbf{x}) \in \mathcal{S}).$$

The probabilistic robustness problem is sometimes referred to as the *probabilistic safety problem* [Cardelli et al., 2019b] or the *adversarial robustness*

problem [Berrada et al., 2021] for BNNs; correspondingly, φ is also referred to as the *safety specification*.

Definition 8 (Set of Safe Weights). The *set of safe weights* for a BNN with respect to a specification $\varphi = \langle \mathcal{X}, \mathcal{S} \rangle$, is defined as

$$\mathcal{J} = \{\mathbf{w} \in \mathbb{R}^{n_w} : \forall \mathbf{x} \in \mathcal{X}, B_{\mathbf{w}}(\mathbf{x}) \in \mathcal{S}\}. \quad (2.42)$$

Proposition 1. Given a BNN $B_{\mathbf{w}}$, such that $\mathbf{w} \sim q(\mathbf{w})$ for a probability density function q , and a safety specification $\langle \mathcal{X}, \mathcal{S} \rangle$, we have that

$$P_{\text{safe}}(\mathcal{X}, \mathcal{S}) = \int_{\mathcal{J}} q(\mathbf{w}) d\mathbf{w}.$$

Since solving the robustness problem for deterministic networks is NP-complete [Katz et al., 2017, Sinha et al., 2017], verifying whether a given $\mathbf{w} \in \mathcal{J}$ is also NP-complete. Therefore, it is infeasible to calculate P_{safe} exactly even for moderately sized networks. In Chapter 5 we introduce two algorithms for effectively computing lower-bound approximations of P_{safe} in a provably tighter way than previous approaches.

2.7 Counterfactual Explanations

So far, we have discussed rigorous AI safety methods such as robustness and verification. In addition to these guarantee-based assurances, there exist safety methods which leverage logical interpretation of a model’s mechanics and outputs, enabling practitioners to check the semantic logic of the model by hand. These methods are categorised under AI *interpretability* or *explainable AI* (xAI). *Counterfactual Explanations* (CFXs) have emerged as a powerful tool for interpreting the decision-making processes of ML models, offering actionable insights

into how the input to an ML model needs to be changed for the model to produce a different, and often desirable, outcome (see [Guidotti, 2024] for a recent survey). Insights provided by CFXs are valuable for safety as they enable users to perform sanity checks on the decision criteria for specified inputs. For example, if we find that we can change a model’s output by changing only a small subset of inputs, perhaps to values that misalign with the logic we wish our model to encode, then a CFX will highlight this. Moreover, CFXs can be used to find sufficient criteria for decision changes; checking that these criteria align with what we expect to see under operating conditions is essential in, for example, control systems. Counterfactual explanations are particularly useful in the context of algorithmic recourse [Karimi et al., 2023], where CFXs are used to generate recourse recommendations for users who have been negatively affected by the decisions of an ML model. CFXs are particularly suited for this task given their intelligibility [Byrne, 2019], appeal to users [Barocas et al., 2020], information capacity [Kenny and Keane, 2021] and alignment with human reasoning [Miller, 2019].

To see what makes CFXs useful in this context, consider a fictional loan application where a customer applies for credit with a bank which uses an ML model to process applications and predict whether a customer will be able to repay the loan or not. For illustration, assume the application is modelled by an input x with features: 32 years of *age*, \$10,000 *loan amount*, and \$25,000 *salary*. Assume that the application is initially rejected based on the model’s prediction that the customer will not be able to repay the loan back. A possible CFX for this rejection could be an altered input x_c , where a salary of \$30,000, with the other features unchanged, would result in the loan being accepted, thus pointing the applicant to what they would need to change to ensure a successful application in the future.

Despite their potential, current approaches to generating CFXs often fall short in terms of satisfying two key properties: *plausibility* [Laugel et al., 2019] and *robustness* [Jiang et al., 2024]. The former requires that CFXs adhere as much as possible to the data manifold, to avoid suggesting unrealistic input changes. The latter requires that similar CFXs be generated for similar inputs [Artelt et al., 2021], to ensure fairness in applications such as algorithmic recourse [Slack et al., 2021]. These properties are more than just metrics characterising the utility of CFXs; they are core desiderata without which CFXs may erode trust in the model they are trying to explain.

Mainstream approaches to compute CFXs characterise these explanations in terms of the solutions to an optimisation problem [Wachter et al., 2017, Mohammadi et al., 2021], which we present now for a binary classification setting without loss of generality. Let N be a machine learning model mapping an input $\mathbf{x} \in \mathcal{X}$ to label $y \in \{0, 1\}$. For ease of exposition, we refer to $N(\mathbf{x}) = 0$ as the *negative outcome* and to $N(\mathbf{x}) = 1$ as the *positive outcome*. Assuming N initially produces a negative outcome for an input $\mathbf{x}$, a CFX, $\mathbf{x}_c$, for this decision can be obtained by,

$$\arg \min_{\mathbf{x}_c \in \mathcal{X}} d(\mathbf{x}, \mathbf{x}_c) \text{ s.t. } N(\mathbf{x}_c) = 1, \quad (2.43)$$

where $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}^+$ is a distance metric defined over the input space from which $\mathbf{x}$ and $\mathbf{x}_c$ are drawn. It is worth noting that this formulation mirrors that given in Equation (2.23) for adversarial attacks; here, however, we are not constrained in our changes to the input. Also, as we will discuss, CFX algorithms have their own set of desirable properties for which it helps to develop a dedicated notation. Since computing an exact solution for the problem presented in Equation (2.43) may be viable only for certain types of machine learning models, the following relaxation is typically considered for more general classes

of differentiable models:

$$\arg \min_{\mathbf{x}_c \in \mathcal{X}} \mathcal{L}(N(\mathbf{x}_c), 1) + \lambda \cdot d(\mathbf{x}, \mathbf{x}_c) \quad (2.44)$$

where $\mathcal{L}$ is a differentiable loss function that guides the search towards an input $\mathbf{x}_c$ for which N yields a positive outcome with high confidence, and λ is a parameter controlling the trade-off between the first term and a distance loss d defined as in Equation (2.43).

Several metrics have been proposed to assess the quality of CFXs [Karimi et al., 2023]. For example, *validity* captures the basic requirement that a CFX should change the output of a model, turning a negative outcome into a positive one. Validity is typically considered in tandem with *proximity* [Wachter et al., 2017], which gives a higher preference to CFXs that are closer to the original input. Additionally, CFXs are typically required to be *actionable* [Ustun et al., 2019] and only alter features that can be realistically modified by the user - e.g., users cannot modify their age, but they can act on their credit score. *Sparsity* [Wachter et al., 2017] is also deemed important in many cases, whereby CFXs requiring changes to fewer features are to be preferred to avoid overloading users with too much information. Another important requirement is *plausibility* [Dhurandhar et al., 2018, Altmeyer et al., 2024], which requires that counterfactual explanations adhere as much as possible to the data manifold to avoid causing unrealistic changes to input features. Finally, robustness [Artelt et al., 2021, Slack et al., 2021, Leofante and Potyka, 2024] advocates for the generation of similar CFXs for similar inputs to ensure CFXs are not perceived as potentially malicious or discriminatory.

In Chapter 6 we explore CFX under a probabilistic setting, and highlight the positive advantages that CFXs exhibit with this model type in terms of their plausibility, cost, and robustness.

Chapter 3

Verification of Geometric Robustness Specifications in Vision Tasks

As discussed in Section 2.5, an important topic in the field of verifying neural networks is the development of task-specific perturbations, beyond the standard ℓ_p -norm balls. Specifically, we outlined a class of perturbations built on geometric transformations, and discussed the complexity introduced by the intrinsic non-linearity and non-convexity of such transformations - a necessary condition for producing sound bounds on Equation (2.27).

Furthermore, in Section 2.5 we cover the relevant literature on the problem of verification under geometric perturbations. Though this is an actively-researched topic, current methods are based on loose approximations of the pixel value function, resulting in similarly loose verification bounds. Producing tight verification bounds is key to determining the safety of networks; accordingly,

tight, convex approximations of the pixel value function are essential.

In this chapter, we present a contribution towards tighter convex approximations of the pixel value function under geometric transformations. The motivation and intuition for this work is best motivated by reviewing the example in Figure 2.5, where we examine a perfectly-legible stop sign in multiple orientations. To the human eye, the geometric perturbations do little to obscure the identifiability of the sign. However, models are known to be susceptible to such perturbations. It is clearly important that a stop sign be recognised by a driverless car system from any angle or, moreover, that an aircraft be able to recognise runways, other aircraft, or ground vehicles, regardless of the orientation and scale. This chapter expands on work within the geometric verification field which provides guarantees that models are robust to geometrically-bound perturbations.

The rest of the chapter is organised as follows: in Section 3.1 we develop the problem of verifying neural networks against geometric robustness properties, and cover specific technical prerequisites for our method. In Section 3.2 we present our novel piecewise-linear approximation strategy via sampling, optimisation and shifting. In Section 3.3 we cover the algorithmic details of the approach and in Section 3.4 we discuss the experimental results obtained and compare them against a state-of-the-art approach on a suite of benchmark networks. The content of this chapter is based on [Batten et al., 2024].

3.1 Geometric Robustness Verification

Having introduced the technical landscape for geometric verification in Section 2.5, we now concretely define the important functions essential to this work. For the interpolation component of $\mathcal{G}_{u,v}(\alpha, \beta, \mu)$ from Equation (2.39),

we consider the following standard bilinear interpolation:

$$\mathcal{I}(u, v) = \sum_{\delta_i, \delta_j \in \{0,1\}} p_{i+\delta_i, j+\delta_j} (1 - |i + \delta_i - u|)(1 - |j + \delta_j - v|),$$

where (i, j) denotes the lower-left corner of the interpolation region $[i, i + 1] \times [j, j + 1]$ that contains pixel (u, v) , and the matrix p denotes the pixel values of the original image. Note that the interpolation function $\mathcal{I}$ is continuous on $\mathbb{R}^2$ but can be non-smooth on the boundaries of interpolation regions. Thus, $\mathcal{G}_{u,v}(\alpha, \beta, \mu)$ is in general *non-smooth* with respect to the spatial parameter μ , e.g., rotation.

For simplicity, in the following we will denote the transformation parameter as $\kappa = (\alpha, \beta, \mu)$. The geometric attack model assumes interval constraints on (α, β, μ) , denoted by $\mathcal{B} \subset \mathbb{R}^d$, where d is the dimension of κ . The attack space, $\Omega_\epsilon(\bar{x})$, is then defined as the set of all images resulting from the application of $\mathcal{G}_{u,v}(\kappa)$ on each pixel (u, v) of $\bar{x}$, for all $\kappa \in \mathcal{B}$. More formally, given $\text{im} : \mathbb{R}^n \rightarrow \mathbb{R}^{h \times w}$, a mapping that rearranges images into their spatial dimensions, $\Omega_\epsilon(\bar{x}) = \{x' \in \mathbb{R}^n \mid \text{im}(x')[u, v] \in \Omega_\epsilon(\bar{x})[u, v]\}$, with $\Omega_\epsilon(\bar{x})[u, v] = \{\mathcal{G}_{u,v}(\kappa) \mid \forall \kappa \in \mathcal{B}\}$.

The geometric attack model $\Omega_\epsilon(\bar{x})$ defines a highly non-convex constraint on the admissible image inputs, which is not readily supported by bounding techniques designed for ℓ_p perturbations. As a result, previous work replaces it with over-approximations [Balunović et al., 2019, Singh et al., 2019a], which allow verification through ℓ_p -based neural network verifiers. Nevertheless, as described in Section 2.5, their over-approximations are imprecise, resulting in loose lower bounds $\underline{\gamma}$ (see Equation (2.27)). In this chapter, we derive a tighter convex relaxation of the geometric attack model $\Omega_\epsilon(\bar{x})$ based on piecewise linear constraints. By relying on network verifiers with support for these constraints, we then show that our approach leads to more effective verification bounds for Equation (2.27).

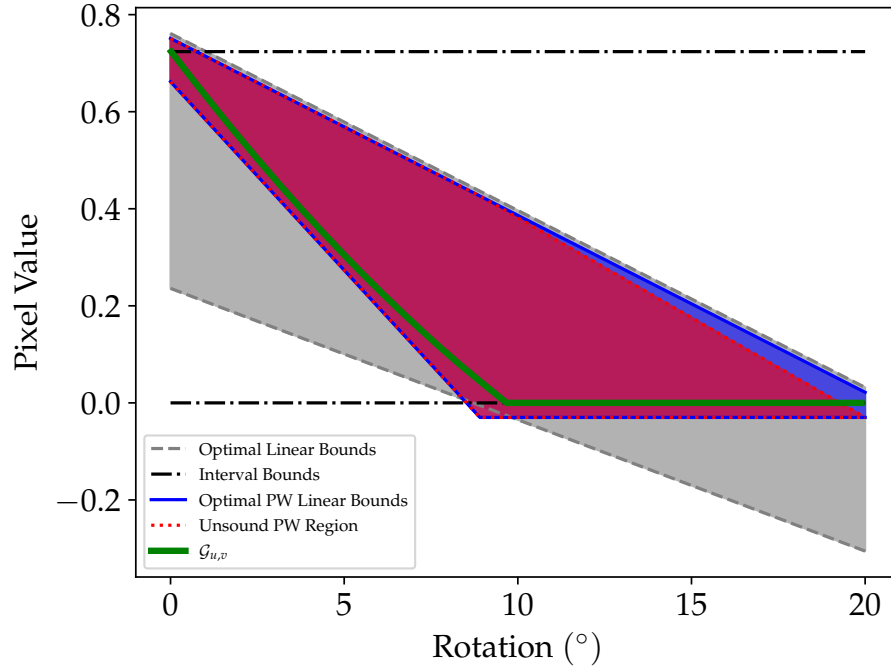


Figure 3.1: Comparison of sound and unsound piecewise (PW) linear domains (our work), sound linear domain (gray area) [Singh et al., 2019a], and interval bounds (dashed line) [Singh et al., 2019a]. The true pixel value function (the green curve) is marked for a rotation of 18° .

3.2 Piecewise Linear Formulation

As mentioned above, the pixel value function $\mathcal{G}_{u,v}(\kappa)$ at location (u, v) is generally *non-linear* and *non-smooth* with respect to the transformation parameters κ . This is one source of difficulty for solving the verification problem in Equation (2.27). In this section, we introduce a new convex relaxation method to derive tight over-approximations of $\mathcal{G}_{u,v}(\kappa)$.

3.2.1 Piecewise Linear Bounds

Deriving an interval bound for each pixel (u, v) , *i.e.*, $L_{u,v} \leq \mathcal{G}_{u,v}(\kappa) \leq U_{u,v}$, for all $\kappa \in \mathcal{B}$ and lower and upper bounds $L_{u,v}, U_{u,v} \in \mathbb{R}$, is arguably the simplest way to obtain a convex relaxation [Singh et al., 2019a, Kouvaros and Lomuscio, 2018]. However, even a small geometric transformation can lead to a large interval bound, making this approach too loose for effective verification.

This naive interval bound approach has been extended in [Balunović et al., 2019], where linear lower and upper bounds were used for each pixel value, *i.e.*,

$$\underline{w}^\top \kappa + \underline{b} \leq \mathcal{G}_{u,v}(\kappa) \leq \bar{w}^\top \kappa + \bar{b}, \quad \forall \kappa \in \mathcal{B}. \quad (3.1)$$

The linear bounds in Equation (3.1), however, may still be too loose to approximate the non-linear function $\mathcal{G}_{u,v}(\kappa)$ (see Figure 3.1 for illustration). Here, we present a novel piecewise linear bounding approach to approximate the pixel values:

$$\max_{j=1,\dots,q} \{\underline{w}_j^\top \kappa + \underline{b}_j\} \leq \mathcal{G}_{u,v}(\kappa) \leq \min_{j=1,\dots,q} \{\bar{w}_j^\top \kappa + \bar{b}_j\}, \quad (3.2)$$

$\forall \kappa \in \mathcal{B}$, where q is the number of piecewise segments, $\underline{w}_j \in \mathbb{R}^d, \underline{b}_j \in \mathbb{R}, j = 1, \dots, q$ define the piecewise linear lower bound, and $\bar{w}_j \in \mathbb{R}^d, \bar{b}_j \in \mathbb{R}, j = 1, \dots, q$ define the piecewise linear upper bound. We remark that the pixel values constrained by Equation (3.2) form a convex set. Furthermore, our approach can include the strategies in [Singh et al., 2019a, Balunović et al., 2019] as special cases. Employing the relational constraints among the piecewise segments will result in a tighter set.

For each pixel value, we would like to derive optimal and sound piecewise linear bounds by minimising the approximation error. Specifically, we aim to

compute the lower bound via

$$\begin{aligned} \min_{\underline{w}_j, \underline{b}_j, j=1, \dots, q} \quad & \int_{\mathcal{B}} \left(\mathcal{G}_{u,v}(\kappa) - \left(\max_{j=1, \dots, q} \{ \underline{w}_j^\top \kappa + \underline{b}_j \} \right) \right) d\kappa \\ \text{s.t.} \quad & \max_{j=1, \dots, q} \{ \underline{w}_j^\top \kappa + \underline{b}_j \} \leq \mathcal{G}_{u,v}(\kappa), \quad \forall \kappa \in \mathcal{B}. \end{aligned} \quad (3.3)$$

Computing the upper bound for Equation (3.2) is similar. The optimisation problem in Equation (3.3) is non-trivial to solve since *the integral cost function is difficult to evaluate owing to the non-linearity of $\mathcal{G}_{u,v}(\kappa)$* . Motivated by [Balunović et al., 2019], we first take samples, κ_i , of the transformation parameter from within $\mathcal{B}$ and, by applying $\mathcal{G}_{u,v}(\kappa)$, obtain the sampled pixel values $\mathcal{G}_{u,v}(\kappa_i)$. We then solve a sampled version of Equation (3.3). The resulting piecewise bound is guaranteed to be sound on the sampling points $\kappa_i \in \mathcal{B}$ but could be unsound over non-sampled regions. To derive the final sound, piecewise bounds for $\mathcal{G}_{u,v}(\kappa)$, we bound the maximum violation over the entire $\mathcal{B}$ using a branch-and-bound Lipschitz optimisation procedure.

3.2.2 Linear Optimisation Based on Sampling Points

Here, we first randomly sample N transformation parameters $\kappa_i \in \mathcal{B}$, $i = 1, \dots, N$, to obtain a sampled version of Equation (3.3) as follows:

$$\begin{aligned} \min_{\underline{w}_j, \underline{b}_j, j=1, \dots, q} \quad & \frac{1}{N} \sum_{i=1}^N \left(\mathcal{G}_{u,v}(\kappa_i) - \left(\max_{j=1, \dots, q} \{ \underline{w}_j^\top \kappa_i + \underline{b}_j \} \right) \right) \\ \text{subject to} \quad & \max_{j=1, \dots, q} \{ \underline{w}_j^\top \kappa_i + \underline{b}_j \} \leq \mathcal{G}_{u,v}(\kappa_i), i = 1, \dots, N. \end{aligned} \quad (3.4)$$

We denote the optimal cost value of Equation (3.4) as β^* . In Equation (3.4), the number of piecewise linear segments q is fixed *a priori*. However, Equation (3.4) is intractable to solve jointly for all piecewise segments $\underline{w}_j, \underline{b}_j, j = 1, \dots, q$ unless $q = 1$ (in which case Equation (3.4) is reduced to a single linear program). One

difficulty is determining the effective domain of each piecewise linear segment.

To alleviate this, we propose to split the whole domain $\mathcal{B}$ into q sub-domains $\mathcal{B}_1, \dots, \mathcal{B}_q$, and then optimise each piecewise linear segment over $\mathcal{B}_j, j = 1, \dots, q$, individually. We then use the following q independent linear programs to approximate the solution to Equation (3.4):

$$\begin{aligned} \beta_j^* &:= \min_{\underline{w}_j, \underline{b}_j} \frac{1}{N} \sum_{\kappa_i \in \mathcal{B}_j} (\mathcal{G}_{u,v}(\kappa_i) - (\underline{w}_j^\top \kappa_i + \underline{b}_j)) \\ &\text{subject to } \underline{w}_j^\top \kappa_i + \underline{b}_j \leq \mathcal{G}_{u,v}(\kappa_i), \quad i = 1, \dots, N, \end{aligned} \quad (3.5)$$

for $j = 1, \dots, q$. Note that in Equation (3.5), we minimise the approximation error over only the sample points within a given domain $\mathcal{B}_j$; however, we force each segment to satisfy the constraints at every sample point $\kappa_i \in \mathcal{B}$ over the whole domain.

We have the following result for the quality of the solution from Equation (3.5).

Proposition 2. *Given any subdomains $\mathcal{B}_j, j = 1, \dots, q$, the optimal solutions $\underline{w}_j, \underline{b}_j, j = 1, \dots, q$, to Equation (3.5) are suboptimal to Equation (3.4), i.e., $\sum_{j=1}^q \beta_j^* \geq \beta^*$. There exists a set of subdomains $\mathcal{B}_j, j = 1, \dots, q$, such that the optimal solutions to Equation (3.4) and Equation (3.5) are identical, i.e., $\sum_{j=1}^q \beta_j^* = \beta^*$.*

Proof. Consider the piecewise linear function in the objective function Equation (3.4). Let $\mathcal{B}_j, j = 1, \dots, q$ be the effective piecewise domain of the j th segment, i.e.,

$$\max_{j=1, \dots, q} \{\underline{w}_j^\top \kappa_i + \underline{b}_j\} = \begin{cases} \underline{w}_1^\top \kappa_i + \underline{b}_1, & \text{if } \kappa_i \in \mathcal{B}_1 \\ \vdots \\ \underline{w}_q^\top \kappa_i + \underline{b}_q, & \text{if } \kappa_i \in \mathcal{B}_q. \end{cases} \quad (3.6)$$

Then, the objective function in Equation (3.4) can be equivalently written as

$$\begin{aligned} & \frac{1}{N} \sum_{i=1}^N \left(\mathcal{G}_{u,v}(\kappa_i) - \left(\max_{j=1,\dots,q} \{ \underline{w}_j^\top \kappa_i + \underline{b}_j \} \right) \right) \\ &= \frac{1}{N} \sum_{j=1}^q \left(\sum_{\kappa_i \in \mathcal{B}_j} (\mathcal{G}_{u,v}(\kappa_i) - (\underline{w}_j^\top \kappa_i + \underline{b}_j)) \right) \end{aligned}$$

Therefore, Equation (3.4) is equivalent to

$$\begin{aligned} & \min_{\underline{w}_j, \underline{b}_j, \mathcal{B}_j, j=1,\dots,q} \\ & \sum_{j=1}^q \left(\frac{1}{N} \sum_{\kappa_i \in \mathcal{B}_j} (\mathcal{G}_{u,v}(\kappa_i) - (\underline{w}_j^\top \kappa_i + \underline{b}_j)) \right) \\ & \text{s.t.} \quad \underline{w}_j^\top \kappa_i + \underline{b}_j \leq \mathcal{G}_{u,v}(\kappa_i), \quad i = 1, \dots, N, j = 1, \dots, q. \end{aligned} \quad (3.7)$$

Note that the piecewise domains $\mathcal{B}_j$ are determined by the linear segments $\underline{w}_j, \underline{b}_j, j = 1, \dots, q$ implicitly in Equation (3.6). We need to simultaneously optimise the choices of $\mathcal{B}_j$ in Equation (3.8), making it computationally challenging to solve.

A suboptimal solution for Equation (3.8) is to *a priori* fix the effective domain $\mathcal{B}_j$ and optimize over $\underline{w}_j, \underline{b}_j, j = 1, \dots, q$ only, i.e.,

$$\begin{aligned} & \hat{\beta} := \\ & \min_{\underline{w}_j, \underline{b}_j, j=1,\dots,q} \sum_{j=1}^q \left(\frac{1}{N} \sum_{\kappa_i \in \mathcal{B}_j} (\mathcal{G}_{u,v}(\kappa_i) - (\underline{w}_j^\top \kappa_i + \underline{b}_j)) \right) \\ & \text{s.t.} \quad \underline{w}_j^\top \kappa_i + \underline{b}_j \leq \mathcal{G}_{u,v}(\kappa_i), \quad i = 1, \dots, N, j = 1, \dots, q, \end{aligned} \quad (3.8)$$

which is decoupled into q individual linear programs, $j = 1, \dots, q$

$$\begin{aligned} \beta_j^* &:= \min_{\underline{w}_j, \underline{b}_j} \frac{1}{N} \sum_{\kappa_i \in \mathcal{B}_j} (\mathcal{G}_{u,v}(\kappa_i) - (\underline{w}_j^\top \kappa_i + \underline{b}_j)) \\ \text{subject to } & \underline{w}_j^\top \kappa_i + \underline{b}_j \leq \mathcal{G}_{u,v}(\kappa_i), \quad i = 1, \dots, N. \end{aligned} \quad (3.9)$$

Therefore, it is clear that $\hat{\beta} = \sum_{j=1}^q \beta_j^* \geq \beta^*$. On the other hand, suppose the optimal solution to Equation (3.4) leads to the optimal effective domains $\mathcal{B}_j, j = 1, \dots, q$ in Equation (3.6). Then, using this set $\mathcal{B}_j, j = 1, \dots, q$, the decoupled linear programs Equation (3.9) are equivalent to Equation (3.8) and Equation (3.4). $\square$

To obtain a good solution in Equation (3.4), choosing effective subdomains $\mathcal{B}_j$ becomes essential. A uniform grid partition is one, naive choice. Another is to partition the subdomains based on the distribution of the sampled pixel values, $\mathcal{G}_{u,v}(\kappa_i)$. We discuss the details of the splitting procedure further in Section 3.3.

Remark 1. (Explicit input splitting vs. piecewise linear constraints) We note that one can perform explicit input splitting $\mathcal{B}_j, j = 1, \dots, q$, and verify each of them by solving Equation (2.27) separately in order to certify the original large domain $\mathcal{B}$. The primary drawback of explicit input splitting is that it requires to call a verifier for each subdomain $\mathcal{B}_j$, which can be computationally intense and not scalable. conversely, it requires only to solve multiple small linear programs, Equation (3.5), to derive the piecewise linear constraints. The piecewise constraints can then be analysed by the solver in a single call to validate whether Equation (2.27) is satisfied. For tight verifiers, such as those mentioned in Section 2.5, this process is much more efficient than explicit input splitting.

3.2.3 Lipschitz Optimisation for Obtaining Sound Piecewise Linear Bounds

The piecewise linear constraints from Equation (3.5) are valid for the sampling points $\kappa_i \in \mathcal{B}, i = 1, \dots, N$. To make the constraints sound over all $\kappa \in \mathcal{B}$, we must shift them such that all points on the pixel value function, $\mathcal{G}_{u,v}(\kappa)$, satisfy the constraints in Equation (3.2). For this, we define a new function that tracks the violation of a piecewise bound over the entire domain $\mathcal{B}$:

$$\xi_{u,v}^* := \max_{\kappa \in \mathcal{B}} \underline{f}_{u,v}(\kappa), \quad (3.10)$$

where $\underline{f}_{u,v}(\kappa) = \max_{j=1,\dots,q} \{\underline{w}_j^\top \kappa + \underline{b}_j\} - \mathcal{G}_{u,v}(\kappa)$. Then, we naturally have a sound piecewise linear lower bound as

$$\max_{j=1,\dots,q} \{\underline{w}_j^\top \kappa + \underline{b}_j\} - \xi_{u,v}^* \leq \mathcal{G}_{u,v}(\kappa), \quad \forall \kappa \in \mathcal{B}.$$

However, computing the exact maximum ξ^* is computationally hard due to the *non-convexity, non-linearity, and non-smoothness* of $\underline{f}_{u,v}(\kappa)$. Instead, given any $\epsilon > 0$, we can use a branch-and-bound Lipschitz optimisation procedure to find $\underline{\xi}^* \in \mathbb{R}$ satisfying $\underline{\xi}^* \leq \xi_{u,v}^* \leq \underline{\xi}^* + \epsilon$.

To establish the branch-and-bound Lipschitz optimisation procedure, we need to characterise the properties of the violation function $\underline{f}_{u,v}(\kappa)$.

Proposition 3. *The violation function $\underline{f}_{u,v}(\kappa) := \max_{j=1,\dots,q} \{\underline{w}_j^\top \kappa + \underline{b}_j\} - \mathcal{G}_{u,v}(\kappa)$ is non-convex, non-smooth, and Lipschitz continuous over $\mathcal{B} \subset \mathbb{R}^d$. Furthermore, there exist $L_m > 0, m = 1, \dots, d$, such that $\forall \kappa_1, \kappa_2 \in \mathcal{B}$*

$$|\underline{f}_{u,v}(\kappa_1) - \underline{f}_{u,v}(\kappa_2)| \leq \sum_{m=1}^d L_m |\kappa_1(m) - \kappa_2(m)|. \quad (3.11)$$

Proof. The pixel value function is given by $\mathcal{G}_{u,v}(\kappa) := \mathcal{P}_{\alpha,\beta} \circ \mathcal{I} \circ \mathcal{T}_\mu^{-1}(u,v)$. We know that the spatial transformation $\mathcal{T}_\mu(u,v)$ and $\mathcal{P}_{\alpha,\beta}$ are continuous and differentiable everywhere. The interpolation function $\mathcal{I}(u,v)$ is continuous everywhere, but it is only differentiable within each interpolation region and can be non-smooth on the boundary. Also, $\mathcal{T}_\mu(u,v)$ and $\mathcal{I}(u,v)$ are generally non-convex.

In addition, the piecewise linear function $\max_{j=1,\dots,q} \{\underline{w}_j^\top \kappa + \underline{b}_j\}$ is continuous but not differentiable everywhere. Therefore, the violation function $\underline{f}_{u,v}(\kappa)$ is non-convex and non-smooth in general. Finally, all the functions $\mathcal{T}_\mu(u,v)$, $\mathcal{P}_{\alpha,\beta}$, $\mathcal{I}(u,v)$, and $\max_{j=1,\dots,q} \{\underline{w}_j^\top \kappa + \underline{b}_j\}$ are Lipschitz continuous; therefore, so is the violation function $\underline{f}_{u,v}(\kappa)$. Thus, there exist $L_m > 0$, $m = 1, \dots, d$, such that Equation (3.11) holds. $\square$

The properties of the violation function $\underline{f}_{u,v}(\kappa)$ in Proposition 3 are directly inherited from the non-convexity and non-smoothness of the interpolation function $\mathcal{I}(u,v)$. The Lipschitz continuity is also from the interpolation function and the piecewise linear function.

With the information about L_m in Equation (3.11), we are ready to obtain a lower and an upper bound for ξ^* upon evaluating the function at any point $\kappa_0 \in \mathcal{B}$:

$$\begin{aligned} \underline{f}_{u,v}(\kappa_0) &\leq \xi^* \\ &= \max_{\kappa \in \mathcal{B}} \underline{f}_{u,v}(\kappa) \\ &\leq \max_{\kappa \in \mathcal{B}} \underline{f}_{u,v}(\kappa_0) + \sum_{m=1}^d L_m |\kappa(m) - \kappa_0(m)| \\ &\leq \underline{f}_{u,v}(\kappa_0) + \sum_{m=1}^d L_m h_m, \end{aligned} \tag{3.12}$$

where $h_m > 0$ denotes the difference between the lower and upper bounds in each box constraint of $\mathcal{B}$. These lower and upper bounds in Equation (3.12) are

useful in the branch-and-bound procedure.

However, we need to estimate the Lipschitz constant L_m in Equation (3.11). We now examine how to estimate the constant L_m based on the gradient of $\underline{f}_{u,v}(\kappa)$ whenever it is differentiable (note that $\underline{f}_{u,v}(\kappa)$ is not differentiable everywhere).

Proposition 4. *Let $\text{Diff}(\mathcal{B})$ be the subset of $\mathcal{B}$ where $\underline{f}_{u,v}(\kappa)$ is differentiable. Then, the Lipschitz constants in Equation (3.11) can be chosen as*

$$L_m = \sup_{\kappa \in \text{Diff}(\mathcal{B})} |\nabla \underline{f}_{u,v}^\top e_m|, \text{ where } e_m \in \mathbb{R}^d \text{ is a basis vector with only the } m\text{-th element being one and the rest being zero.}$$

Proof. This proof is motivated by [Jordan and Dimakis, 2020]. In order to prove Proposition 4, we first state a useful result from [Jordan and Dimakis, 2020, Lemma 3]. Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be Lipschitz continuous over an open set $\Omega \subset \mathbb{R}^n$. We denote $\text{Diff}(\Omega)$ as the subset of Ω where $f(x)$ is differentiable. We also let $\mathcal{D}$ be the set of $(x, v) \in \mathbb{R}^{2n}$ for which the directional derivative $\nabla_v f(x)$ exists and $x \in \Omega$. Finally, we let $\mathcal{D}_v$ be the set $\mathcal{D}_v = \{x \in \mathbb{R}^n \mid (x, v) \in \mathcal{D}\}$. Then, we have the following inequality:

$$\sup_{x \in \mathcal{D}_v} |\nabla_v f(x)| \leq \sup_{x \in \text{Diff}(\Omega)} |\nabla f(x)^\top v|. \quad (3.13)$$

We now proceed to prove Proposition 4. Fix any $\kappa_1, \kappa_2 \in \mathcal{B}$, and we define a function $h : \mathbb{R} \rightarrow \mathbb{R}$ as $h(t) = \underline{f}_{u,v}(\kappa_1 + t(\kappa_2 - \kappa_1))$. Since $\underline{f}_{u,v}(\kappa)$ is Lipschitz continuous in $\mathcal{B}$, it is clear that $h(t)$ is Lipschitz continuous on the interval $[0, 1]$. Thus, by Rademacher's Theorem Evans and Gariepy [2015], $h(t)$ is differentiable everywhere except for a set of measure zero.

We can further define a Lebesgue integrable function $g(t)$ that equal to $h'(t)$

almost everywhere as follows:

$$g(t) = \begin{cases} h'(t), & \text{if } h'(t) \text{ exists} \\ \sup_{s \in [0,1]} |h'(s)|, & \text{otherwise} \end{cases}.$$

Note that if $\underline{f}_{u,v}(\kappa)$ is differentiable at some point, we have

$$h'(t) = \nabla \underline{f}_{u,v}(\kappa_1 + t(\kappa_2 - \kappa_1))^T (\kappa_2 - \kappa_1).$$

Then we have the following inequalities:

$$\begin{aligned} |\underline{f}_{u,v}(\kappa_1) - \underline{f}_{u,v}(\kappa_2)| &= |h(1) - h(0)| = \left| \int_0^1 g(t) dt \right| \\ &\leq \int_0^1 |g(t)| dt \\ &\leq \int_0^1 \sup_{s \in [0,1]} |h'(s)| dt = \sup_{s \in [0,1]} |h'(s)| \\ &\leq \sup_{\kappa \in \mathcal{D}_{\kappa_2 - \kappa_1}} |\nabla_{\kappa_2 - \kappa_1} \underline{f}_{u,v}(\kappa)|. \end{aligned}$$

Furthermore, considering the inequality in Equation (3.13), we have

$$\begin{aligned} |\underline{f}_{u,v}(\kappa_1) - \underline{f}_{u,v}(\kappa_2)| &\leq \sup_{\kappa \in \text{Diff}(\mathcal{B})} |\nabla \underline{f}_{u,v}(\kappa)^T (\kappa_2 - \kappa_1)| \\ &\leq \sum_{m=1}^d \sup_{\kappa \in \text{Diff}(\mathcal{B})} |\nabla \underline{f}_{u,v}(\kappa)^T e_m| |\kappa_2(m) - \kappa_1(m)| \end{aligned}$$

where $e_m \in \mathbb{R}^d$ is a basis vector with only the m -th element being one and the rest being zero. Therefore, the Lipschitz constants in Equation (3.11) can be chosen as $L_m = \sup_{\kappa \in \text{Diff}(\mathcal{B})} |\nabla \underline{f}_{u,v}(\kappa)^T e_m|$. $\square$

Maximum directional gradient. To bound the maximum violation $\xi_{u,v}^*$ in Equation (3.10) using Equation (3.12), we need to estimate the constant L_m ,

and Proposition 4 requires us to calculate the maximum directional gradient $|\nabla \underline{f}_{u,v}^\top e_m|$. Each component of $\nabla \underline{f}_{u,v}$ varies independently with respect to any constituent of the transformation composition, $\mathcal{T}_\mu(\kappa_m)$, $m = 1, \dots, d$. Each L_m depends only on a transformation, $\mathcal{T}_\mu$, and interpolation, $\mathcal{I}_{u,v}$. The only component that is not differentiable everywhere in the parameter space $\kappa \in \mathcal{B}$, is the interpolation $\mathcal{I}_{u,v}(x, y)$ due to its being disjoint across interpolation regions. We overcome this by calculating the interpolation gradient $\nabla_{x,y} \mathcal{I}_{u,v}$ separately in each interpolation region, and taking the maximum interval of gradients from the union $[\nabla_{x,y} I_{min}, \nabla_{x,y} I_{max}] = [\min(\cup_{k=1,\dots,n} \nabla_{x,y} I_k), \max(\cup_{k=1,\dots,n} \nabla_{x,y} I_k)]$, where I_k are the relevant interpolation regions and $\cup_{k=1,\dots,n} I_k = \mathcal{R} \subset \mathcal{B}$. Computing a bound on L_m this way mirrors the IBP-based procedure outlined in [Balunović et al., 2019]. With this, we can calculate an upper bound on L_m to be applied in the Lipschitz algorithm.

Branch-and-bound Lipschitz optimisation procedure. Similar to [Balunović et al., 2019], we use a branch-and-bound procedure where $\underline{f}_{u,v}$ and $\mathcal{B}$ are given as inputs alongside the Lipschitz error, ϵ , and samples per subdomain, n . The procedure first samples the violation function $\underline{f}_{u,v}$, obtaining maximum value candidates; this is placed in a list of 3-tuples with the upper bound $\underline{f}_{\text{bound},i}$ and corresponding domain $\mathcal{B}_i$. The key upper bound operation $\text{bound}(\cdot)$ is obtained using Equation (3.12). We then check that each 3-tuple in our list meets the termination criteria, as parameterised by ϵ . If the requirement is satisfied for all elements, then we terminate and return $\underline{\xi}^*$. Until the requirement is met for every list element, we iteratively split unsatisfied subdomains. This process is repeated until a satisfactory maximum candidate is found, splitting κ in each iteration. We can ignore any sub-domain κ_1^n of κ_1 where the function bound $\underline{f}_{\text{bound}}$ in κ_1^n is smaller than a maximum value candidate $\underline{f}_{\text{max}}$ in any other sub-domain. Deciding how to split subdomains becomes significant for higher

dimensional parameter spaces. In the case $\kappa \in \mathbb{R}^1$, we need only decide where to split on a single axis, for which we select the domain midpoint. The crux of our algorithm is approximating the gradient of $\underline{f}_{u,v}(\kappa)$ when it is differentiable, as stated in Proposition 4. For bounding the violation of piecewise linear bounds, we can consider the piecewise bound itself to be made of q linear sub-regions with each one bounded by the intersection with the neighbouring linear piece, or the lower and upper bounds on the transformation parameters. We can then bound the Lipschitz constant in the same way as for a single linear bound, instead starting with q sub-domains. Solving the Lipschitz bounding procedure for each linear segment over only its local domain in this way enables us to bound the Lipschitz constant of a piecewise linear bound in the same time that a linear bound takes.

In the following section, we will discuss the concrete algorithmic details of the implementation.

3.3 Algorithmic Details

In this section, we begin by discussing the practical details of implementing the linear optimisation step, before detailing the Lipschitz branch-and-bound procedure.

3.3.1 Linear Optimisation over Sub-domains

For all experiments, we fix $q = 2$ in Equation (3.4). With this constraint, we can find suboptimal piecewise bounds by solving two independent linear optimisation problems, where each problem is applied over a subset of the piecewise domain, divided at a given sample point, n . We name the parameter sub-spaces divided by n , κ_1 , and κ_2 where $\kappa_1, \kappa_2 \subset \mathcal{B}$. Expressing Equation (3.4) in this

way gives

$$\min_{\underline{w}_1, \underline{b}_1} \frac{1}{N} \sum_{i=1}^n (\mathcal{G}_{u,v}(\kappa_i) - \{\underline{w}_1^\top \kappa_i + \underline{b}_1\}) \quad (3.14a)$$

$$\text{subject to } \{\underline{w}_1^\top \kappa_i + \underline{b}_1\} \leq \mathcal{G}_{u,v}(\kappa_i), \quad i = 1, \dots, N,$$

$$\min_{\underline{w}_2, \underline{b}_2} \frac{1}{N} \sum_{i=n}^N (\mathcal{G}_{u,v}(\kappa_i) - \{\underline{w}_2^\top \kappa_i + \underline{b}_2\}) \quad (3.14b)$$

$$\text{subject to } \{\underline{w}_2^\top \kappa_i + \underline{b}_2\} \leq \mathcal{G}_{u,v}(\kappa_i), \quad i = 1, \dots, N.$$

In Equations (3.14a) and (3.14b) we optimise the area over only the sample points within a piece's domain, κ_1 , or κ_2 ; however, we enforce the constraints at every sample point. By doing this, we guarantee convexity of our piecewise constraints. We develop a heuristic to determine the sample point, n , at which we split $\mathcal{B}$ based on the error between sampled points and optimal *linear* bounds:

$$\underline{n} = \max_{i=1, \dots, N} (\{\bar{w}^\top \kappa_i + \bar{b}\} - \mathcal{G}_{u,v}(\kappa_i)), \quad (3.15)$$

where $\underline{n}$ is the splitting point for the lower bound. We calculate $\bar{n}$ correspondingly using the lower linear bound. There exists a splitting point n which produces optimal piecewise bounds; however, identifying n is infeasible. In practice, we first compute single, non-piecewise linear upper and lower bounds and use them to propose a splitting point. Specifically, we use Equation (3.15) as a heuristic for selecting the location of the piecewise splits. Once this point is fixed, we then independently find linear bounds for the domain on either side of the splitting point. These segments form the piecewise elements of the final bound. We compute the bounds in this way for two reasons: firstly, it enables us to apply our splitting heuristic in Equation (3.15), and secondly, it is computationally efficient in our experimental setting where we require the linear bounds, for which we can use the original pre-split bound, for comparison.

3.3.2 Branch-and-bound Procedure

With our unsound constraints, our method closely follows that of [Balunović et al., 2019], with the important exception that we treat our single piecewise bound as two separate linear bounds with domains κ_1 and κ_2 . We first define a function $\underline{f}_{u,v}$ to track the violation of a bound by the pixel value function $\mathcal{G}_{u,v}$. In the case that the lower bound is piecewise, we will maximise $\underline{f}_{u,v}$ twice over κ_1 and κ_2 , and $\bar{f}_{u,v}$ once over $\mathcal{B}$. Maximisation of $\underline{f}_{u,v}$ is carried out via the branch-and-bound Lipschitz procedure outlined in Algorithm 3. For each instance of $\underline{f}_{u,v}(\kappa)$ where $\kappa \in \kappa_1$, we first approximate the Lipschitz constant L_i and use it to bound $\underline{f}_{u,v}$,

$$\underline{f}_{\text{bound}} = L_i \frac{\kappa_1}{2} + \left(\underline{f}_{u,v}(\kappa_i) \right), \quad (3.16)$$

where κ_i is the midpoint of κ_1 . We find upper bound candidates by sampling the violation function $\underline{f}_{u,v}(\kappa)$ at four, evenly spaced points in κ_1 ; the largest value obtained becomes the maximum value candidate $\underline{f}_{\text{max}}$. We aim to find a maximum value-bound pair that satisfies $\underline{f}_{\text{bound}} - \underline{f}_{\text{max}} < \epsilon$, with ϵ given. This process is repeated until a satisfactory maximum candidate is found, splitting κ in each iteration. We can ignore any sub-domain, κ_1^n , of κ_1 where the function bound $\underline{f}_{\text{bound}}$ in κ_1^n is smaller than a maximum value candidate $\underline{f}_{\text{max}}$ in any other sub-domain. This is because in this case we can guarantee that the maximum value is not in the κ_1^n sub-domain.

Algorithm 3: Branch-and-bound Lipschitz Optimisation Procedure**Input:** $\underline{f}_{u,v}, \mathcal{B}, \epsilon, n, N$ **Output:** $\underline{\xi}^*$

```

1  $\underline{f}_{\max} \leftarrow \max_{l=1,\dots,n} \underline{f}_{u,v}(\kappa_l)$ , where  $\kappa_l \in \mathcal{B}$ ;
2  $\underline{f}_{\text{bound}} \leftarrow \text{bound}(\underline{f}_{u,v}, \nabla \underline{f}_{u,v}, \mathcal{B})$ ;
3  $\mathcal{G} \leftarrow [(\underline{f}_{\max}, \underline{f}_{\text{bound}}, \mathcal{B})]$ ;
4 while  $\underline{f}_{\text{bound},i=1,\dots,N} - \underline{f}_{\max,i=1,\dots,N} > \epsilon$  do
5   for  $i \leftarrow 1$  to  $N$  do
6     if  $\underline{f}_{\text{bound},i} - \underline{f}_{\max,i} > \epsilon$  then
7        $\mathcal{B}_{i,i+N} \leftarrow \text{split}(\mathcal{B}_i)$ ;
8        $\underline{f}_{\max,i} \leftarrow \max_{l=1,\dots,n} \underline{f}_{u,v}(\kappa_l)$ , where  $\kappa_l \in \mathcal{B}_i$ ;
9        $\underline{f}_{\text{bound},i} \leftarrow \text{bound}(\underline{f}_{u,v}, \nabla \underline{f}_{u,v}, \mathcal{B}_i)$ ;
10       $\mathcal{G}_i \leftarrow [(\underline{f}_{\max,i}, \underline{f}_{\text{bound},i}, \mathcal{B}_i)]$ ;
11    end
12  end
13 end
14 return  $\underline{\xi}^* = \max_{i=1,\dots,N} \underline{f}_{\max,i}$ ;

```

3.4 Experimental Evaluation

In this section, we present three sets of results: (i) a quantitative study directly comparing the model-agnostic bounds produced by our piecewise linear approach against the state-of-the-art linear bounds [Balunović et al., 2019], (ii) an empirical evaluation of verification results obtained using linear and piecewise linear bounds, without input splitting and using the same neural network verifier [Bottoeva et al., 2020a], and (iii) a comparison of our results against the current state-of-the-art method [Balunović et al., 2019].

3.4.1 Experimental Setup

We consider the MNIST image recognition dataset [LeCun et al., 2010] and CIFAR10 [Krizhevsky, 2012]. In line with [Balunović et al., 2019], we use two fully-connected ReLU networks, MLP2 and MLP6, and one convolutional ReLU network, CONV, from the first competition for neural network verification (VNN-COMP) [VNN-COMP, 2020]. The fully-connected networks comprise 2 and 6 layers respectively. Each layer of each of the networks has 256 ReLU nodes. The convolutional network comprises two layers. The first layer has 32 filters of size 5×5 , a padding of 2, and strides of 2. The second layer has 64 filters of size 4×4 , a padding of 2, and strides of 1. Additionally, we employ a larger convolutional ReLU network from [Balunović et al., 2019], composed of three layers: a convolutional layer with 32 filters of size 4×4 and strides of 2, a convolutional layer with 64 filters of size 4×4 and strides of 2, and a fully connected layer with 200 nodes. All experiments were carried out on an Intel Core i9-10940X (3.30GHz, 28 cores) equipped with 256GB RAM and running Linux kernel 5.12.

Once a convex over-approximation of the attack space $\Omega_\epsilon(\bar{x})$ is computed, (cf. Section 3.1) a neural network verifier is required to provide a lower bound on problem Equation (2.27). Unless stated otherwise, the verification results reported in this section are obtained using VENUS, a complete MILP-based verification toolkit for feed-forward neural networks [Botoeva et al., 2020a].

3.4.2 Experimental Results

In the following, we will use “L” to denote the linear relaxation from equation (3.1), and “PWL” to denote the piecewise linear relaxation from equation (3.2).

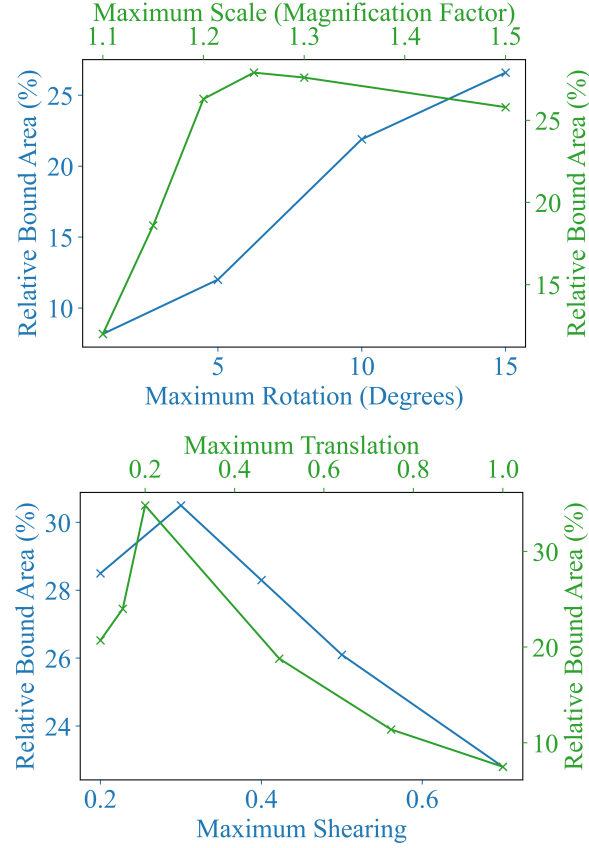


Figure 3.2: A comparison of area captured by piecewise linear and linear bounds as a function of transformation parameter. Relative bound area is defined as $1 - (V_{\text{PWL}}/V_{\text{L}})$.

PWL vs L: Comparing Areas. Figure 3.2 is a direct comparison of bound *tightness* between the piecewise linear bounds presented here and the current state-of-the-art linear bounds [Balunović et al., 2019]. For each image, linear and piecewise linear bounds are generated, each one capturing the reachable pixel values for a given transformation. We always use two piecewise segments ($q = 2$) and use a Lipschitz error of 0.01 to compute bounds. The area enclosed by each set of bounds is then calculated and averaged for every pixel over all images. In each case, the piecewise linear bounds are guaranteed to be tighter (enclose a

Table 3.1: Comparison of verification results for piecewise linear constraints and linear constraints.

Model	Attack	Verified		Falsified		Time (s)	
		L	PWL	L	PWL	L	PWL
MLP2	R(5)	26	28	74	72	0.7	3.7
	Sh(0.2)	20	26	80	74	1.2	12
	Sc(1.1)	24	24	76	76	1.1	51
	T(0.1)	16	16	84	84	11	54
MLP6	R(15)	0	2	12	32	1602	1253
	Sh(0.5)	0	0	16	68	1591	778
	Sc(1.3)	0	0	24	78	1404	727
	T(0.2)	0	2	26	74	1397	648
CONV	R(10)	20	48	2	0	1447	1044
	Sh(0.2)	18	50	0	0	1548	1044
	Sc(1.3)	0	10	4	4	1750	1663
	T(0.15)	0	32	2	0	1767	1397

Table 3.2: Comparison of L and PWL using VENUS, with verification results taken from DeepG [Balunović et al., 2019].

Dataset	Transformation	Accuracy (%)	DeepG	Linear (Ours)		PWL (Ours)	
			Certified (%)	Certified (%)	Time (s)	Certified (%)	Time (s)
MNIST	R(30)	99.1	87.8	90.8	37.9	92.9	28.3
CIFAR	R(2)Sh(2)	68.5	54.2	65.0	239.5	66.0	204.9

smaller volume in pixel value-transformation parameter space) than the linear bounds, as shown in Section 3.2. Figure 3.2 shows the relative area (specifically, $1 - (V_{\text{PWL}}/V_{\text{L}})$ with V_{PWL} and V_{L} the volume enclosed by the piecewise linear and linear bounds respectively) of the two bound types. In Figure 3.2, there is an initial increase in relative tightness for all transformations - this is a result of linear bounds being unable to efficiently capture the increasing nonlinearity in the pixel value curve, $\mathcal{G}_{u,v}(\kappa)$. After an initial increase, the behaviour for different transformations diverges. For rotation, the relative advantage of the piecewise bounds continues to increase up to 15 degrees. For scaling, however, there is a peak at $1.25\times$ magnification, followed by a decrease in the relative tightness. This result is explained by a corresponding increase in the complexity of the pixel value curve. Notably, the piecewise bounds are best suited to

non-monotonic pixel value curves with a single, sharp vertex. For curves with many vertices and large fluctuations, piecewise linear bounds become increasingly linear (the gradient of the pieces converge) to maintain convexity. Though this is the case for $q = 2$, as we study here, for larger numbers of piecewise segments the advantage over linear bounds will continue to hold, as the piecewise bounds approximate the convex hull of the pixel values for $q \rightarrow \infty$. The plots for shearing and translation show a similar pattern to scaling. Although the relative tightness may decrease for larger transformations, the total bounded area increases, making any proportional reduction in area more significant.

PWL vs L: Verification Results. Table 3.1 reports the experimental results obtained for verification queries using VENUS, on the VNN-COMP networks. For each type of input bound – piecewise linear and linear – the table shows the percentage of certified images (Verified column), the percentage of images for which a valid counter example was found (Falsified column), and the average verification time. We verify the robustness of each of the networks with respect to one of four transformations - rotation, scaling, shearing, or translation - on 50 randomly selected images from the MNIST test set. For each verification query, we use a timeout of 30 minutes. We observe a considerable performance advantage using piecewise linear bounds for the convolution network, in every case, at least doubling the count of verifiable images. For the 6-layer MLP network, many of the transformations tried could not be verified, leading to numerous counter examples and time-outs. However, for every transformation the piecewise linear bounds were able to find more counter examples than linear bounds – this is a result of the improved tightness of piecewise linear bounds. For the 2-layer MLP, results across the bound types are very similar, in some cases they are equal. This is due to two factors, both of which stem from the network’s small size. Firstly, the 2-layer network is the least robust of

all three. Accordingly, our results are for very small transformations for which the pixel value curve is approximately linear. In these cases, linear bounds can capture the input set as well as piecewise linear bounds. Secondly, the advantage of piecewise linear bounds' tightness is compounded over each layer of a network - the 2-layer MLP is so small that this effect is minimal, further aligning the performance of the approaches. Finally, the use of piecewise linear constraints results in a reduction of average verification times on both the 6-layer MLP and the convolutional network: this is due to the fact that their relative tightness compensates for the additional cost of their encoding, leading the employed MILP-based verifier to positive lower bounds on the verification problem Equation (2.27) in less time.

Comparison with Literature Results. In Table 3.2 we provide a comparison of verification results obtained using VENUS with both linear and piecewise linear constraints, with the DeepG [Balunović et al., 2019] results, obtained using linear constraints and the DeepPoly [Singh et al., 2019a] verifier, which relies on a relatively loose LP relaxation of Equation (2.27). Further, we use a MILP-based verifier which enabled us to add the pixel domain constraints in addition to our transformation-based bounds. This, coupled with the tighter verifier, enables our linear bounds to outperform those from DeepG. We consider MNIST and CIFAR10 benchmarks presented in Balunović et al. [2019]. The MNIST example consists of verifying a 30-degree rotation transformation by way of 10, 3-degree sub-problems. This is in contrast to Table 3.1, where each perturbation is represented by a single set of bounds and a single verifier call per image. Table 3.2 shows that, even in the small-perturbation setting, the use of tighter verification algorithms (L versus DeepG) increases the number of verified properties. Furthermore, we show that the method proposed here, PWL, leads to the tightest certification results. The CIFAR10 example com-

prises a composition of rotation and shearing of 2 degrees and 2% respectively. This query is solved via 4 sub-problems (with each transformation domain split in half). The results show a 12% improvement for the PWL bounds over the DeepG result. However, much of this gain comes from the verifier itself. The gap between the linear bounds and their piecewise counterpart is 1%. We attribute this smaller gap to the relatively small domain over which each sub-problem runs. Nevertheless, we point out that verifying perturbations through a series of sub-problems is extremely expensive, as it requires repeated calls to both neural network verifiers, and to the constraint-generation procedure (including the branch-and-bound-based Lipschitz optimisation). For this reason, we focus on verification in the setting without transformation splitting, and aim to maximise certifications through the use of tight verifiers and over-approximations of the geometric transforms.

The results herein reveal two key takeaways: firstly, by increasing the linear elements we use to bound the reachable pixel space of geometric perturbations, we achieve substantially tighter verification queries; by up to 32% on the convolutional network. Secondly, the effect diminishes with perturbation size, as we see from Figure 3.2. This is an effect of the increasing multimodality of the pixel value function with increasing perturbation range; an order-two piecewise linear function can only tightly fit unimodal pixel value functions. Interestingly, increasing the number of piecewise segments further would not mitigate this effect due to the constraint of convexity on the bounded region. Though we have demonstrated the effectiveness and efficiency of using piecewise linear bounds for geometric verification, it remains necessary to verify queries with a large range using multiple independent, smaller queries - such queries would nonetheless benefit from the improved tightness and efficacy of the piecewise constraints introduced here.

3.5 Conclusions

In this chapter, we have introduced a new piecewise-linear approximation method for geometric robustness verification. The approach can generate provably tighter convex relaxations for images obtained by geometric transformations than the state-of-the-art methods [Singh et al., 2019a, Balunović et al., 2019]. We have shown experimentally that the proposed method improves verification precision in certifying robustness against geometric transformations over prior work [Balunović et al., 2019], while being more computationally efficient.

Despite the positive results brought by our piecewise linear approximation method, further topics deserve exploration. Firstly, it remains challenging to obtain the optimal piecewise linear constraints via Equation (3.4). To obtain a good set of piecewise linear constraints, our current method Equation (3.5) requires obtaining an effective heuristic partition of the domain $\mathcal{B}_1, \dots, \mathcal{B}_q$. It will be interesting to further investigate and quantify the sub-optimality of the solution from Equation (3.5). Secondly, the number of piecewise linear segments q is a hyperparameter in our framework. A larger value q leads to a better approximation of the pixel value function in theory; however, this also results in more linear constraints for the verification problem in practice. Future work will investigate how to choose a good value of q based on the curvature of the pixel value function. We discuss the outcomes and goals for future work in the broader context of safety in deep neural networks in Chapter 7.

Chapter 4

Improving Weather-based OOD Generalisation in Lidar-based Perception Models via Adversarial Training

4.1 Introduction

In Chapter 2 we defined the notion of adversarial attacks. Furthermore, we discuss their potential utility as a means to help defend networks against run-time attacks by way of *adversarial training*, a data augmentation-based approach to improving post-training model robustness. In addition to promoting model

robustness, in [Yi et al., 2021] the authors demonstrate that adversarial training can be used to improve the post-training performance on unseen, out-of-distribution (OOD) data in a computer vision setting. In this chapter, we posit that adversarial training can similarly be leveraged to improve the OOD performance in more complex computer vision settings. Specifically, we investigate LiDAR-based object detection for on-the-road environments in the presence of fog - a particularly important problem for self-driving cars. We focus on fog as it is often under-represented in training datasets due to its relative rarity in real-world weather patterns. As such, naively-trained models perform poorly when exposed to fog-affected test samples. Although we specifically focus on fog, there are numerous weather patterns which disrupt model performance for the same reasons.

Currently, the challenge of OOD generalisation for lidar-based object detection is addressed using costly simulation-based techniques. In particular, in [Bijelic et al., 2019, Hahner et al., 2021, 2022, Kilic et al., 2021] the authors develop adverse weather simulations which aim to approximate the effects of adverse weather on lidar sensors. An accurate simulation enables the authors to augment training datasets with partially-synthetic data resembling adverse weather by applying a simulation to more readily-available clear weather data. Specifically, in [Hahner et al., 2021] the authors model the attenuation of lidar pulses in foggy environments as an exponential decay. Then, given a clear-weather point cloud, they can apply the simulation to modify points affected by the simulated fog. In [Kilic et al., 2021], the authors develop a simulation for rainy environments by modelling raindrops as perfectly spherical Mie scatterers [Mie, 1908]. In [Hahner et al., 2022], the authors model snowfall by extending the scattering-based approach from [Kilic et al., 2021] to include backscattering.

The work presented in this chapter is motivated in two ways; firstly, as an

exploration of the applicability of adversarial training in addressing the challenge of OOD inference in the 3D object detection space, and secondly, as the proposed method does not require any direct simulation of point clouds, instead using cheap initialisations in combination with adversarial training, it offers a cheaper alternative to the simulation-based techniques introduced above. Further, as its methodology is not tied to the efficacy of a specific simulation approach, it does not suffer from the same brittleness in simulation accuracy.

The rest of this chapter is organised as follows: In Section 4.2 we introduce the necessary background on simulation of adverse weather effects as well as extending the mathematical framework presented in Section 2.3 to include adversarial training: in Section 4.3 we outline our proposed method and highlight the design choices made. In Section 4.4 we discuss experimental results on unseen, adverse weather and benchmark against a state-of-the-art simulation-based approach [Hahner et al., 2021] before concluding in Section 4.5. The content of this chapter is based on work published in [Batten and Lomuscio, 2025].

4.2 Lidar-based Fog Simulation

As mentioned in Section 4.1, state-of-the-art approaches to improving OOD performance on adverse weather scenarios leverage simulation techniques to emulate various weather effects on the lidar point cloud. We now outline the approach introduced in [Hahner et al., 2021] for simulating the effects of fog.

A lidar system operates by emitting pulses of laser light in a given direction and timing the response power and time. With this information, the scanner can compute the distance and direction to the object responsible for the reflected signal. In practice, a lidar scanner will rotate while continuously transmitting columns of pulses, enabling it to build a 360° scan. We term the power of a

pulse transmitted by the scanner as the *transmitted signal power*, P_T , and the return signal as the *received signal power*, P_R . The authors in [Hahner et al., 2021] adopt a linear model, introduced by [Rasshofer et al., 2011], of the received signal power:

$$P_R(R) = C_A \int_0^{2R/c} P_T(t) H\left(R - \frac{ct}{2}\right) dt, \quad (4.1)$$

with c the speed of light, R the distance, and $H(\cdot)$ a function modelling the impulse response given by an object at R . The transmitted signal power is given by a time-based sinusoid which factors in the non-instantaneous transmission time [Hahner et al., 2021, Eq. 2]. We can factorise $H(R)$ into a contribution from the target, and a contribution from optical effects:

$$H(R) = H_C(R) H_T(R). \quad (4.2)$$

We term the contributions the optical (H_C) and target (H_T) channels. [Hahner et al., 2021] show that, in a foggy weather setting, we can further decompose $H_T(R)$ into two sources: one from the true, ‘*hard*’, target, and one from fog-induced backscattering, or the, ‘*soft*’, target:

$$\begin{aligned} H_T(R) &= H_T^{\text{soft}}(R) + H_T^{\text{hard}}(R) \\ &= \beta U(R_0 - R) + \beta_0 \delta(R - R_0), \end{aligned} \quad (4.3)$$

where β is a fog-dependent backscattering constant, β_0 is a constant capturing the reflectivity of the hard target, R_0 is the distance to the hard target, $U(\cdot)$ is the Heaviside step function, and $\delta(\cdot)$ is the Dirac delta function. The delta function ensures we only consider the hard signal at the distance at which it exists, and the Heaviside function ensures we do not model fog effects from behind the hard target. For a detailed derivation, see [Hahner et al., 2021, Eqs. (9-14)]. Furthermore, we can also decompose the received signal power into an

erroneous contribution from fog, and a contribution from the hard target:

$$P_{R,\text{fog}}(R) = P_{R,\text{fog}}^{\text{hard}}(R) + P_{R,\text{fog}}^{\text{soft}}(R). \quad (4.4)$$

With $P_{R,\text{fog}}^{\text{hard}}(R)$ an exponential decay with R ,

$$P_{R,\text{fog}}^{\text{hard}}(R) = \exp(-2\alpha R_0) P_{R,\text{clear}}(R), \quad (4.5)$$

with α a decay constant related to fog density. Moreover, we can express the return signal power for the soft target (the fog) as

$$P_{R,\text{fog}}^{\text{soft}}(R) = C_A P_0 \beta \int_0^{2\tau_H} \sin^2\left(\frac{\pi}{2\tau_H} t\right) \cdot \frac{\exp(-2\alpha(R - \frac{ct}{2}))}{(R - \frac{ct}{2})^2} \xi(R - \frac{ct}{2}) U(R_0 - R + \frac{ct}{2}) dt, \quad (4.6)$$

where P_0 is the peak power of the laser pulse, τ_H a constant describing the time-width of the pulse, and $\xi(\cdot)$ a function modelling the beam crossover of the non-coaxial pulse transmitter and receiver. We cannot solve Equation (4.6) analytically and so the authors perform numerical integration on each point in the clear-weather point cloud to obtain $P_{R,\text{fog}}^{\text{soft}}$. Computing $P_{R,\text{fog}}^{\text{hard}}$ is trivial. The final step of the simulation is to check $\max_R P_{R,\text{fog}}^{\text{soft}}(R) < P_{R,\text{fog}}^{\text{hard}}(R_0)$; if this holds, then the position of the point is unchanged and its intensity is updated to be $P_{R,\text{fog}}^{\text{hard}}(R_0)$. If the condition does not hold, then the point is moved to a distance $R' = \arg \max_R P_{R,\text{fog}}^{\text{soft}}(R)$ and the intensity is set to $P_{R,\text{fog}}^{\text{soft}}(R')$. Computing this for every point in the original point cloud concludes the simulation. Here, we have outlined the simulation at a high level and we relate it to our contribution in Section 4.3. For a detailed exposition of the simulation, see [Hahner et al., 2021].

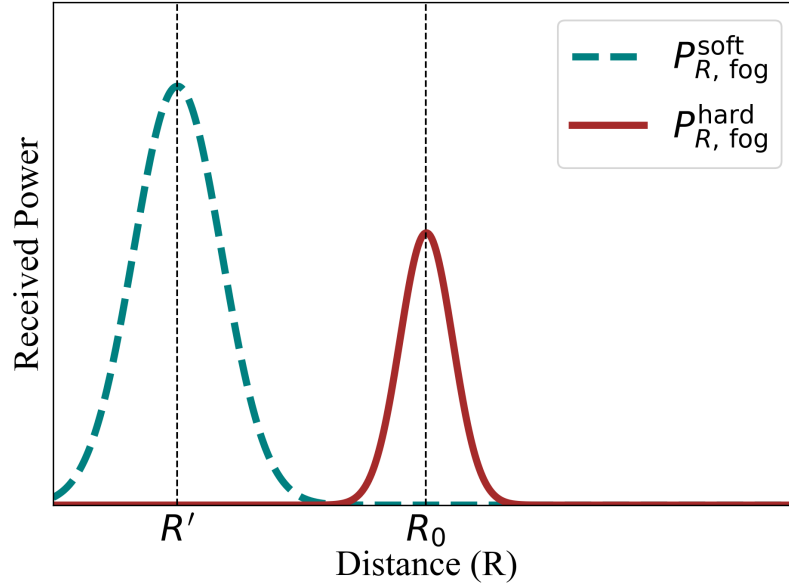


Figure 4.1: Relative received powers from an erroneous ‘soft’ target and the true ‘hard’ target. In this example, backscattering from the fog produces a stronger received signal and so the lidar would interpret R' as the true distance to an object.

4.3 Adversarial Training for Lidar-based Vision Systems

Our approach to improving weather-based OOD generalisation on lidar-based object detection models consists of two steps. First, we perform a cheap input initialisation step, and second, we carry out adversarial training. In this section, we outline both, beginning with a discussion of the training sample initialisation, after which, we detail the training algorithm and the design choices made before discussing experimental results in Section 4.4.

4.3.1 Simulation-driven Initialisation

Drawing on key insights from the fog simulation introduced in [Hahner et al., 2021], we propose to initialise training samples prior to adversarial training such that they echo features of the simulation. Specifically, we seek to cheaply mimic how a subset of points from the clear-weather point cloud are erroneously shifted closer to the lidar scanner when fog is simulated. To understand this effect, we refer to Figure 4.1 which shows the relative $P_{R,\text{fog}}(R)$ for the soft and hard targets. In this example, the soft target, i.e., the fog, produces a stronger received power signal, and, accordingly, R' would be interpreted as the distance to an object. It is important here to make two observations: firstly, that whenever the response from $P_{R,\text{fog}}^{\text{hard}}$ is dominant, the distance will be unchanged at R_0 , the same as the original distance in the clear-weather point cloud. However, when, as in Figure 4.1, the signal from $P_{R,\text{fog}}^{\text{soft}}$ is dominant, the point will be shifted to the distance, R' . To further illustrate this, in Figure 4.2 we show a comparison of a clear-weather point cloud and the same cloud after running the fog simulation from [Hahner et al., 2021]. The ring of points around the lidar scanner present in Figure 4.2b but lacking in Figure 4.2a illustrates the effect described above. Importantly, we note that the simulated points are densely packed in a ring around the scanner. This behaviour can be understood by examining Equation (4.6), where we note that only the Heaviside step function $U(R_0 - R + ct/2)$ depends on R_0 ; the Heaviside function by definition will always be evaluated to 1 for any simulated point. Since no other term depends on the original clear-weather received power $P_{R,\text{clear}}$, or the distance R_0 , we can say that for every point in the cloud, the value of R which maximises $P_{R,\text{fog}}^{\text{soft}}$ is a constant, $R' = \arg \max_R (P_{R,\text{fog}}^{\text{soft}})$. The simulated points in Figure 4.2b do not sit perfectly on the fixed radius R' as a result of noise independently added to R' for each point. The value of R' is determined by the assumed density of fog

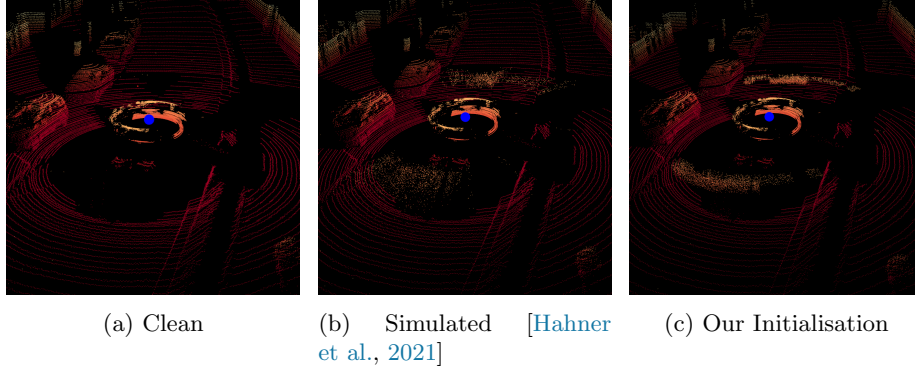


Figure 4.2: Comparison of a clean, clear-weather point cloud, against a point cloud modified by the fog simulation presented in [Hahner et al. \[2021\]](#). The lidar scanner is marked by the blue sphere in the centre.

being simulated, controlled by α , and by the optical properties of the lidar scanner used, handled in $\xi(\cdot)$. Specifically, the axial separation of the transmitter and receiver.

This ‘ring’ of simulated points is the key insight we leverage in initialising the adversarial training procedure. Specifically, prior to adversarial training we compute scores, $S(\mathbf{p})$, for each point, $\mathbf{p}_i = (I_i, [p_x, p_y, p_z]_i^T) = (I, \mathbf{p}_{i,xyz})$, in the cloud:

$$S(\mathbf{p}) = \frac{I}{\exp\left(\sqrt{p_x^2 + p_y^2 + p_z^2}\right)} = \frac{I}{\exp(\|\mathbf{p}_{xyz}\|)}, \quad (4.7)$$

where I is the intensity of a point and is linear with P_R . With the scores computed, we select n points with the lowest scores to be repositioned under the initialisation, where s_{thresh} is the threshold score. We refer to the set of point indices to be repositioned as $\mathcal{P} = \{i | I_i \leq s_{\text{thresh}}, \forall i\}$. We constrain the new locations of the points to be on the straight line segment joining the original location and the lidar scanner. To determine where on the segment to place the points, we choose $R'_{\text{ours}} = \arg \max_R (\xi(R))$, which is the distance at which the transmission beam cone and receiver cone first completely overlap. In [\[Rasshofer et al., 2011\]](#) the authors discuss in detail non-coaxial beam configurations (also

called bistatic beam configurations) and define the distance R'_{ours} , referred to as R_2 in [Rasshofer et al., 2011]. Once we obtain a value for R'_{ours} for a given lidar scanner, we are ready to reposition the n points. Echoing the approach in [Hahner et al., 2021], we add noise independently to R'_{ours} when setting the final position of each $\mathbf{p}_i$. We handle intensity by setting the adjusted point intensity to the arithmetic mean of *all* points in the cloud. Concretely, the resulting modified points are given by:

$$\begin{aligned} \mathbf{p}'_i &= (\bar{I}, \mathbf{p}'_{i,xyz}) \\ \mathbf{p}'_{i,xyz} &= R'_{\text{ours}} \cdot \frac{\mathbf{p}_{i,xyz}}{\|\mathbf{p}_{i,xyz}\|} \\ \bar{I} &= \frac{1}{T} \sum_{j=1}^N I_j, \end{aligned} \tag{4.8}$$

where T is the total number of points in the point cloud. The resulting point cloud is given by:

$$\mathcal{C}' = \{\mathbf{p}'_i | \forall i \in \mathcal{P}\} \cup \{\mathbf{p}_j | \forall j \notin \mathcal{P}\}. \tag{4.9}$$

This concludes the initialisation step of our procedure. An example of an initialised point cloud is given in Figure 4.2c. Its qualitative similarity with the simulation from [Hahner et al., 2021] is perceptible. However, we have produced it without the need for computing $P_{R,\text{fog}}^{\text{soft}}$, $P_{R,\text{fog}}^{\text{hard}}$, or R' .

Algorithm 4: Initialisation and Adversarial Training Procedure

Input: Network N , dataset $\mathcal{D} = \{(\mathbf{p}^{(i)}, \mathbf{y}^{(i)})\}$,
number of training steps S , attack budget ϵ , learning rate α , number of
epochs E

Output: Trained network parameters $\mathbf{W}^{(N)}$

- 1 Initialise network parameters $\mathbf{W}^{(0)}$ (e.g., randomly or using heuristics);
- 2 **for** $t \leftarrow 1$ **to** E **do**
- 3 **for** each mini-batch $\mathcal{B} \subset \mathcal{D}$ **do**
- 4 **for** each input $\mathbf{p}^{(i)} \in \mathcal{B}$ **do**
- 5 Initialise input $\mathbf{p}'^{(i)}$ according to Eq. (4.9);
- 6 Compute adversarial input;;
- 7
$$\mathbf{p}^{*,i} = \arg \max_{\mathbf{p}'^{(i)}} \mathcal{L}(\mathcal{N}(\mathbf{p}'^{(i)}), \mathbf{y}^{(i)}) \quad \text{s.t.} \quad \|\mathbf{p}'^{(i)} - \mathbf{p}^{(i)}\| \leq \epsilon$$
- 8 **end**
- 9 Update network parameters;;
- 10
$$\mathbf{W}^{(s+1)} \leftarrow \mathbf{W}^{(s)} - \alpha \nabla_{\mathbf{W}} \mathcal{L}(\mathcal{N}(\mathbf{p}^*), \mathbf{y})$$
- 11 **end**
- 12 **end**
- 13 **return** $\mathbf{W}^{(S)}$

4.3.2 Adversarial Training on Point Clouds

Once we have initialised a training sample, we update it using an adversarial attack, as described in Section 2.3. However, unlike in a 2D setting, we have four degrees of freedom for each input point: one intensity value, and

three position values. To perform an adversarial attack, we modify the notation in Section 2.3 to a budget *vector* $\epsilon = [\epsilon_I, \epsilon_p]$, which includes attack budgets for the intensity channel and position channels respectively. Moreover, we apply the attack budgets *independently* to each point $\mathbf{p}_i$, rather than *jointly* across all the model inputs as is standard in the noise-based attack literature. We note, however, that the L_∞ -norm frequently applied jointly across inputs as a budget in noise-based adversarial attacks is equivalent to an element-wise L_1 -norm. This is how we approach bounding the intensity channel of each $\mathbf{p}_i$. Concretely, $|I_i - I_i^*| \leq \epsilon_I, \forall i \in |\mathcal{C}'|$, where I_i^* is the intensity channel of the i -th point, $\mathbf{p}_i^*$, in an adversarial attack, $\mathcal{C}^* = \{\mathbf{p}_i^* | \forall i \in |\mathcal{C}'|\}$. For the budget on point position, ϵ_p , we apply the L_∞ norm element-wise, such that the adjusted points are locally bounded to their initial position. This is essential for preserving geometric locality across the point cloud. Concretely, $|\mathbf{p}_{i,xyz} - \mathbf{p}_{i,xyz}^*|_\infty \leq \epsilon_p |\mathbf{p}_{i,xyz}|, \forall i \in |\mathcal{C}'|$, where $|\mathbf{p}_{i,xyz}|$ is a vector of the absolute coordinates x , y , and z for point i . We provide an outline of our training procedure in Algorithm 4.

4.4 Evaluation

For the experiments, we use the lidar portion of the Seeing Through Fog (STF) dataset [Bijelic et al., 2019] which consists of 3,469 clear-weather lidar scans for training as well as 1,847 for testing. Importantly, it also provides 1,732 scans collected in foggy conditions. We use PointRCNN [Shi et al., 2019] as the baseline model. All experiments were run on Ubuntu 22.04 running kernel 6.8.0-49-generic, using an AMD Ryzen Threadripper 3960X with 24 cores, an RTX 3080 Ti, and 256GB of RAM.

Table 4.1: Numerical results showing R40 mAP scores across each class and difficulty. *Clean* indicates scores obtained on clear-weather data only. *Fog* indicates scores obtained on the foggy test set from STF [Shi et al., 2019]. The intersection over union (IoU) thresholds are 0.5, 0.25, and 0.25 for the car, cyclist, and pedestrian classes respectively.

Training Method	Car						Cyclist						Pedestrian					
	Easy		Medium		Hard		Easy		Medium		Hard		Easy		Medium		Hard	
	Clean	Fog	Clean	Fog	Clean	Fog	Clean	Fog	Clean	Fog	Clean	Fog	Clean	Fog	Clean	Fog	Clean	Fog
Standard Training	68.9	39.5	69.4	40.0	65.0	38.3	44.2	21.0	43.7	21.0	41.9	22.7	35.4	19.5	32.8	17.5	31.0	16.0
Simulated [Hahner et al., 2021]	29.5	32.3	30.8	31.9	27.1	29.6	16.6	10.1	16.6	10.1	14.9	10.1	5.58	10.6	3.87	9.96	3.96	8.42
Adversarial Training (Ours)	66.0	35.7	68.1	36.3	62.8	35.0	43.8	24.8	42.7	24.8	40.8	25.2	33.3	27.5	32.0	26.0	28.9	22.5
Clean + Simulated [Hahner et al., 2021]	71.9	43.1	69.7	43.3	63.1	41.5	35.1	18.9	35.7	18.9	33.7	19.5	25.8	26.2	23.8	24.2	21.7	22.6
Clean + AT (Ours)	72.2	45.0	72.8	45.1	67.4	43.3	39.6	19.9	39.5	19.9	37.7	20.5	33.3	32.4	31.7	30.4	28.1	28.1

4.4.1 Model Training

The PointRCNN [Shi et al., 2019] model is trained to predict a bounding box in 3D space around each in-class object in the point cloud, and a classification for the bounding box. The STF dataset [Bijelic et al., 2019] contains three classes: car, cyclist, and pedestrian. PointRCNN consists of two stages: a ground-up bounding box generation stage, and a bounding box refinement stage. The bounding box generation stage takes a raw point cloud as an input and segments the points into ‘foreground’ points, which border an in-class object, and ‘background’ points, which do not. It then proposes a bounding box location for *each* foreground point; this results in multiple bounding box proposals for an object (one for each foreground point lining the object). The bounding box refinement stage takes the bounding box proposals and the foreground mask, along with the original point cloud, and refines the bounding box predictions so that there is one bounding box prediction per in-class object. It also classifies the box into one of the target classes. For a detailed breakdown of the architecture, see [Shi et al., 2019].

The attacks in our adversarial training procedure focus only on the first component of the model: the foreground-background segmentation and bounding box proposal generation. We make this choice as these procedures have differentiable losses. Specifically, PointRCNN uses focal loss when training the point segmentation block, and cross-entropy loss when training the bounding box proposals. Our attacks take place in the raw point cloud input space, as described in Section 4.3.

Each model was trained for 50 epochs using either adversarial training, simulated data (collectively called *synthetic* data), clean data, or, where indicated, an augmentation of the above.

4.4.2 Adversarial Training for Adverse Weather

In Table 4.1 we outline the numerical results of our comparison. The results for our method presented in Table 4.1 all use a position budget $\epsilon_p = 0.001$ and an intensity budget $\epsilon_I = 0$. After experimenting with attacks on both position and intensity, we find that fixing the intensity budget to 0 yields the best performance. We propose this is due to the model struggling to converge when the attack space is too large and that for larger numbers of training epochs a combination of position and intensity attacks may further improve performance. Furthermore, we posit that the relative importance of position over intensity could be due to the model focusing more on point position than intensity when predicting bounding boxes. This makes intuitive sense as different surfaces will have different reflectivities, and the same surfaces may return different received powers when at different angles to the sensor. These effects may lead to poor performance when intensity is over-relied upon by the model.

We observe that, when using the synthetic data only, our approach achieves over double the clean accuracy of the approach in [Hahner et al., 2021], as well as higher scores on the fog test set. Interestingly, standard training beats both approaches in each setting when the synthetic data is not augmented with clean data during training. A noteworthy feature of the results is that the models trained on simulated data perform better in the fog test set than the clear-weather test set for the car and pedestrian classes. In contrast, when trained using adversarial training, the models achieve comparable clean accuracy to the standard training. We believe the adaptive nature of adversarial training combined with the fact that we attack across *all* points in the point cloud prevents our approach from over-fitting to the foggy data in the same way the simulated data does. Moreover, it is well-known that adversarial training comes at a cost to clean performance; however, by using very limited attack budgets

this effect is minimised, as shown by the comparable results between AT and Standard Training.

When we train on a combination of clean and synthetic data, we see performance improvements for both clear-weather and the fog test sets. In some cases, with Clean + AT achieving the highest clean and fog accuracies simultaneously. A notable exception is for the cyclist class, where clean accuracy decreases for AT when adding clean data. This is a counter-intuitive result. A likely explanation lies in the competing loss objectives of AT and standard training. Moreover, while experimenting with adversarial pre-training followed by standard training, we observe a sudden drop in clean and fog performance when the model is first trained with the clean data. This drop gradually recovers to yield the results shown in Table 4.1. This behaviour alludes to a well-known challenge in adversarial training: that performance is highly sensitive to training schedules. We leave an in-depth study on the impact of different training schedules to future work.

The results for the clean + simulated-trained models are significantly improved over the models trained with simulated data alone. The improvement in clean accuracy is particularly prominent. Interestingly, the fog accuracy has improved across the board, suggesting that training on the simulated data alone is insufficient to learn good representations of the in-class objects.

Table 4.2: Numeric results showing the relative performance of adversarial training (AT), simulated data [Hahner et al., 2021], and the random initialisation scheme outlined in Section 4.3. Columns report mAP40 average results for the same thresholds presented in Table 4.1.

Training Method	Car						Cyclist						Pedestrian					
	Easy		Medium		Hard		Easy		Medium		Hard		Easy		Medium		Hard	
	Clean	Fog	Clean	Fog	Clean	Fog	Clean	Fog	Clean	Fog	Clean	Fog	Clean	Fog	Clean	Fog	Clean	Fog
Standard Training	68.9	39.5	69.4	40.0	65.0	38.3	44.2	21.0	43.7	21.0	41.9	22.7	35.4	19.5	32.8	17.5	31.0	16.0
AT on Simulated Data [Hahner et al., 2021]	17.9	19.9	22.2	19.7	21.8	18.6	11.8	7.96	11.8	8.0	11.8	8.0	4.16	10.0	4.31	8.62	4.28	8.79
Random Initialisation Only	69.7	35.7	70.4	36.6	65.1	35.2	37.2	16.6	37.0	16.6	36.6	17.2	27.2	18.2	26.2	17.4	25.2	15.4
AT Only	66.9	33.4	67.0	34.3	61.7	32.8	40.7	15.0	40.7	15.0	40.5	15.0	38.8	17.6	32.9	26.4	32.4	25.0

4.4.3 Ablations

Attack Budgets

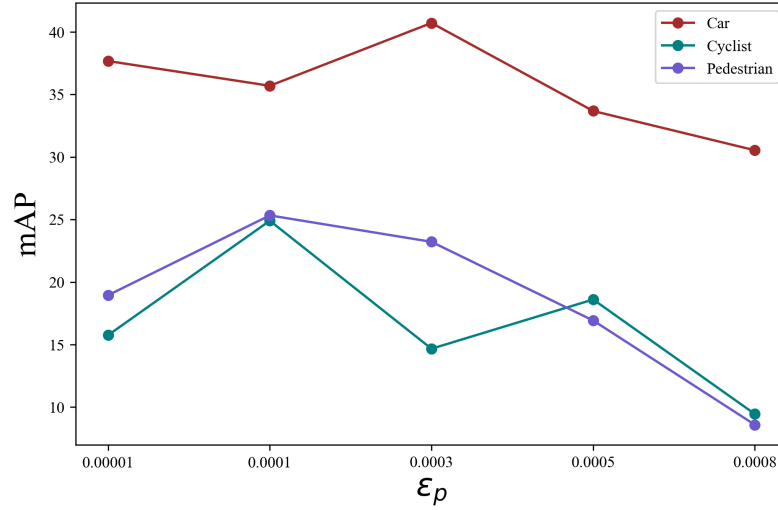


Figure 4.3: Mean Average Precision (mAP) computed using R40 scores for each class plotted against position budget ϵ_p . Results are for 50 epochs of adversarial training.

In Figure 4.3 we show an ablation of the mAP per class against the position budget ϵ_p used in training. We observe two main features: firstly, that the cyclist and pedestrian classes appear more sensitive to the budget scale than the car class, and secondly, that the peak mAP for the pedestrian and cyclist classes appears at a lower attack budget than for the car class. It is important to recall that when performing adversarial training we do not limit the attacks to only points moved under the initialisation scheme. We suggest that the relative sensitivities of classes to attack budgets arise from the relative sizes of the objects in the point clouds; namely, a car is bigger than a cyclist or a pedestrian. Moreover, the point clouds are generated using a car-mounted lidar

scanner; as a result, other cars are more likely to be close to the scanner than pedestrians, further enhancing the effect. We suggest this is responsible for both of the observations mentioned above.

Decomposing training components

In this ablation, we examine the independent contributions from the different training paradigms studied. Specifically, we seek to identify the unique importance of the adversarial training, random initialisation, and simulated data [Hahner et al., 2021].

In Table 4.2 we report numerical results on a combination of the aforementioned training schemes. The numerical results reported here are on the same metrics reported in Section 4.4.2 and we again include the results for standard training to aid comparison. We begin by examining the Random Initialisation Only row, which reports results obtained on models trained using only the initialisation scheme presented in Section 4.3, without any adversarial training. Interestingly, these results are relatively strong, outperforming both the AT on Simulated data and the Pure AT rows. In some cases, they outperform the results for the simulated-only data from Table 4.1. This indicates that our initialisation scheme is effective out of the box - unsurprising given that it is based on the mathematical formulation for the simulation in [Hahner et al., 2021].

When we look at AT Only, we see the performance closely mirrors that for Random Initialisation Only, with the main divergence coming for the smaller object classes, namely Pedestrian. For this class, we see much greater performance from AT Only than Random Initialisation Only. This indicates that these two schemes are each contributing in different ways to the overall performance we see for the combined training scheme in Table 4.1. This highlights the importance of using both schemes jointly to achieve the best results.

An interesting and conspicuous result is for the Simulated Data row, in which

we see a complete breakdown in model learning, achieving the worst results across all benchmarks. We attribute this to the challenge of the learning task being too great for the model to learn in the number of epochs used. Specifically, the simulated data is produced by considering the effect of fog on every point in the cloud. Meanwhile, our initialisation scheme only modifies a small fraction of points, and does not attenuate the intensity of all points, as the simulation does. When we combine the starkly different data produced by the simulation with the dynamic learning objective of adversarial training, the model appears to fail to capture the data distribution, in particularly failing to identify objects in clear weather, as it has not been exposed to these examples in training.

4.5 Conclusions

In this chapter, we have introduced a novel initialisation for adversarial training on lidar-produced point cloud data. Furthermore, we propose a method based on our initialisation and adversarial training to train point cloud-based perception models with the goal of improving performance on OOD data, here represented by adverse weather data. We have shown that the proposed method improves OOD performance and have benchmarked it against a state-of-the-art simulation-based approach, observing an accuracy improvement of up to 6% on the simulation-based benchmark. Though we have demonstrated the potential of adversarial training for use in OOD generalisation for point cloud-based perception models, we would like to further explore the effects of training schedules - such as pre-training, integrated training, and fine-tuning - on the results we observe; we leave this to future work. Moreover, we leave a more detailed study on the effects of attack budgets and initialisation parameters on empirical performance to future work.

Chapter 5

Probabilistic Verification of Bayesian Neural Networks

5.1 Introduction

In the previous chapters, we have considered robustness and verification in a deterministic setting. That is, we have considered models where the parameters are point estimates, fixed after training is complete. In Section 2.2, we introduced Bayesian neural networks and outlined their differences from standard, deterministic neural networks, in addition to highlighting the advantages of this class of models. In Section 2.6 we developed the notion of probabilistic certification on BNNs and covered the relevant notation and previous works. As discussed in Section 2.6, verifying probabilistic networks presents a particular challenge due to the complexity of considering variable parameters with potentially infinite support. Methods currently use loose over-approximations to obtain verification bounds on network outputs. As we have discussed in the deterministic case, loose bounds inhibit our ability to express safety guarantees

on deep models. In this chapter, we expand on a state-of-the-art method introduced in [Wicker et al., 2020] for lower-bounding the probabilistic safety of BNNs. Our goal is to provide probabilistic safety bounds that are provably and experimentally tighter than the previous work.

The rest of the chapter is organised as follows. In Section 5.2 we discuss the details of our approach and introduce the PIE and GIE algorithms for exploring the set of safe weights and computing lower bounds for the probabilistic robustness of BNNs. We also formally show that our approach outperforms the state-of-the-art sampling-based method in Section 5.2.3 by producing provably tighter bounds. We evaluate the performance of our approach against a state-of-the-art approach on the MNIST and CIFAR10 datasets in Section 5.3.

5.2 Computing the Probabilistic Robustness for BNNs

In Section 2.6 we introduced the previous work towards verifying non-deterministic models, along with some important definitions for verification of BNNs. As a reminder, in Definition 7 we define the probabilistic safety criterion P_{safe} , and in Definition 8 we define the safe set of weights $\mathcal{J}$, the region of parameter space for which the model is verifiably robust. Additionally, we outline why directly computing P_{safe} is infeasible.

Given the constraints in directly computing P_{safe} , we instead aim to soundly under-approximate it. Doing so involves finding a valid parameter space for which the model is verifiably robust; an under approximation of $\mathcal{J}$. We denote this under-approximate safe set as $\hat{\mathcal{J}} \subseteq \mathcal{J}$. Given $\hat{\mathcal{J}}$, we can define an under approximation for P_{safe} following the logic from Definition 7,

$$p_{\text{safe}} = \int_{\hat{\mathcal{J}}} q(\mathbf{w}) d\mathbf{w} \leq P_{\text{safe}}(\mathcal{X}, \mathcal{S}). \quad (5.1)$$

A key step in computing p_{safe} is to find the largest possible $\hat{\mathcal{J}}$. In this section we introduce two approaches to this end, building on an orthotope-based approach introduced in Wicker et al. [2020] in which the authors sample fixed-size orthotopes in the parameter space and independently verify the resulting deterministic model (in which parameters are allowed to assume any values contained by the orthotope). Once the algorithm concludes, p_{safe} can be computed by integrating over the union of the safe orthotopes, as in Equation (5.1). A weakness of this approach is that it employs orthotopes of uniform volume regardless of the input specification. For a given input vector to a BNN, it may be possible to construct large orthotopes around the weights’ mean and successfully verify safety using LBP. However, for other inputs to the same network, such large orthotopes may not preserve verification guarantees, necessitating the use of smaller orthotopes. In high-dimensional spaces, these reduced orthotopes yield extremely small volumes, resulting in negligible probabilistic robustness lower bounds—a phenomenon demonstrated in Example 5.

To mitigate this problem, we introduce two approaches that allow us to dynamically adjust the size of orthotopes $\hat{\mathcal{J}}_1, \dots, \hat{\mathcal{J}}_s$. The *Pure Iterative Expansion (PIE)* approach, which we introduce in Subsection 5.2.1, allows us to dynamically *scale* the size of an orthotope $\hat{\mathcal{J}}_i$, and therefore, cover a larger volume in $\mathcal{J}$.

The more sophisticated *Gradient-guided Iterative Expansion (GIE)* approach, which we introduce in Subsection 5.2.2, not only allows the use of dynamic scaling, but also expands the orthotopes further in the dimensions that allow $\hat{\mathcal{J}}_i$ ’s to remain in $\mathcal{J}$, according to the posterior gradient, $\nabla_{\mathbf{w}} B_{\mathbf{w}}$.

Both PIE and GIE can be accompanied by LBP or other approaches. Since

our iterative expansion approach depends on iteratively expanding $\hat{\mathcal{J}}_i$'s and verifying their safety, we use LBP, as described and implemented in [Wicker et al., 2020], because of its significant computational efficiency.

5.2.1 Pure Dynamic Scaling

Given a BNN $B_{\mathbf{w}}$ and safety specification $\varphi = \langle \mathcal{X}, \mathcal{S} \rangle$, recalling that we denote the set of safe weights of $B_{\mathbf{w}}$ with respect to φ as $\mathcal{J}$, the pure sampling and LBP approach [Wicker et al., 2020] starts with the under-approximation $\hat{\mathcal{J}} = \emptyset$ of $\mathcal{J}$ and samples $\mathbf{w}_0 \sim q(\mathbf{w})$. It then defines $\hat{\mathcal{J}}_0 = [\mathbf{w}_0 \pm \lambda \boldsymbol{\sigma}]$, where $\lambda > 0$ is the scaling factor and $\boldsymbol{\sigma}$ is the parameters' standard deviation vector. Then, using IBP or LBP, it checks whether $\hat{\mathcal{J}}_0 \subseteq \mathcal{J}$. If this is the case, $\hat{\mathcal{J}} = \hat{\mathcal{J}} \cup \hat{\mathcal{J}}_0$; otherwise, $\hat{\mathcal{J}}_0$ is discarded. This process is repeated until a given number of weights, s , have been tried.

As we briefly discussed in the beginning of Section 5.2 and demonstrated in Section 5.3, the lower bounds computed using a pure sampling approach are not tight, because of the rigidity of the sampled orthotopes.

The first and most straightforward solution to this problem that we propose is to instead use dynamically-scaled orthotopes, as outlined in Algorithm 5. After sampling an orthotope $\hat{\mathcal{J}}_i = [\mathbf{w}_i \pm \lambda \boldsymbol{\sigma}] \subseteq \mathcal{J}$ and verifying it (line 8 in Algorithm 5), we expand the orthotope to $\hat{\mathcal{J}}_i = [\mathbf{w}_i \pm 2\lambda \boldsymbol{\sigma}]$ rather than immediately moving on to the next sampled parameter vector, as in the pure sampling approach. After the initial expansion, we again check whether $\hat{\mathcal{J}}_i \subseteq \mathcal{J}$. If this is the case, we can repeat the same process for $\hat{\mathcal{J}}_i = [\mathbf{w}_i \pm 3\lambda \boldsymbol{\sigma}]$. We repeat this until $\hat{\mathcal{J}}_i \not\subseteq \mathcal{J}$.

Algorithm 5: Certifying BNNs using PIE**Input:** BNN $B_{\mathbf{w}}$, distribution $\mathbf{w} \sim \mathcal{P}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$, safety specification $\varphi = \langle \mathcal{X}, \mathcal{S} \rangle$, number of samples $s \in \mathbb{N}$, and scaling factor $\lambda > 0$.**Output:** $\hat{\mathcal{J}}$ and p_{safe} .

```

1  $\hat{\mathcal{J}} \leftarrow \emptyset;$ 
2 for  $i \leftarrow 1$  to  $s$  do
3    $\mathbf{w} \sim \mathcal{P}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2);$ 
4    $j \leftarrow 1;$ 
5    $\bar{\mathcal{J}} \leftarrow \emptyset;$ 
6   while  $LBP([\mathbf{w} \pm j\lambda\boldsymbol{\sigma}], \mathcal{X}) \subseteq \mathcal{S}$  do
7      $\bar{\mathcal{J}} \leftarrow [\mathbf{w} \pm j\lambda\boldsymbol{\sigma}];$ 
8      $j \leftarrow j + 1;$ 
9   end
10   $\hat{\mathcal{J}} \leftarrow \hat{\mathcal{J}} \cup \bar{\mathcal{J}};$ 
11 end
12 return  $\hat{\mathcal{J}}, \int_{\hat{\mathcal{J}}} q(\mathbf{w}) d\mathbf{w};$ 

```

Example 4 demonstrates the advantage of the PIE approach over the pure sampling-based approach.

Example 4. Consider a BNN with two Bayesian layers, $B_{\mathbf{w}}(x) = \text{ReLU}(w_2 \cdot \text{ReLU}(w_1 \cdot x))$, where $w, w_2 \sim \mathcal{N}(0, 1)$ and no bias terms. For the input constraint set $\mathcal{X} = \{1\}$ and the output constraint set $\mathcal{S} = \{y \in \mathbb{R} : y \leq 1\}$, the set of safe weights is given by

$$\mathcal{J} = \left\{ \begin{pmatrix} w_1 & w_2 \end{pmatrix}^T \in \mathbb{R}^2 : \text{ReLU}(w_2 \cdot \text{ReLU}(w_1)) \leq 1 \right\}. \quad (5.2)$$

The set $\mathcal{J}$ is shown in blue in Figures 5.1a to 5.1c for the same sampled weights $\mathbf{w}_1, \dots, \mathbf{w}_4 \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$ and scaling factor $\lambda = 1$; the areas covered by the pure sampling approach, PIE for 2 iterations, and GIE (cf. Section 5.2.2) for 2 iter-

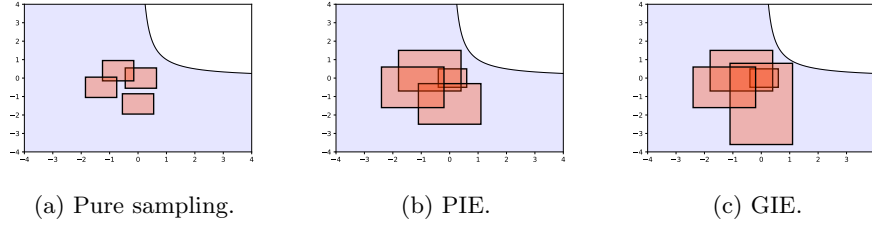


Figure 5.1: A comparison between the area of the set of safe weights $\mathcal{J}$ in Example 4 that is covered by the pure sampling (5.1a), PIE after two iterations (5.1b), and GIE after two iterations (5.1c) for the same initial weights $\mathbf{w}_1, \dots, \mathbf{w}_4 \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$. See Section 5.3 for detailed comparisons, including comparison with equal compute.

ations are shown in Figures 5.1a, 5.1b, and 5.1c, respectively. We observe that the GIE approach (Figure 5.1c) has the greatest coverage of $\mathcal{J}$, followed by the PIE approach (Figure 5.1b), which has greater coverage than the pure sampling approach (Figure 5.1a).

5.2.2 Gradient-guided Dynamic Scaling

The goal of this subsection is to improve Algorithm 5 further by using the gradients of the network $\nabla_{\mathbf{w}} B_{\mathbf{w}}$, to maximise coverage of the safe set of weights $\mathcal{J}$, as summarised in Algorithm 6. Recall that for a given point in the parameter space, $\mathbf{w} \in \mathcal{J}$, our goal is to explore $\mathcal{J}$ starting from $\mathbf{w}$. Instead of expanding $\hat{\mathcal{J}}_i$ uniformly in all dimensions, we can use the parameter-wise BNN gradient, $\nabla_{\mathbf{w}} B_{\mathbf{w}}$, to expand $\hat{\mathcal{J}}_i$ further in dimensions that do not violate $\hat{\mathcal{J}}_i \subseteq \mathcal{J}$.

From Equation (2.42), we have that

$$\mathcal{J} = \{\mathbf{w} \in \mathbb{R}^{n_{\mathbf{w}}} : \mathbf{a}^\top \cdot B_{\mathbf{w}}(B_\epsilon(\mathbf{x})) \geq b\},$$

for some $\mathbf{a} \in \mathbb{R}^n$, $\mathbf{x} \in \mathbb{R}^m$, $b \in \mathbb{R}$, and $\epsilon > 0$. To maximise the volume of a candidate orthotope, $\hat{\mathcal{J}}_i$, while staying within the bounds of $\mathcal{J}$, we propose

expanding further in directions with *zero* or *positive gradients* in $\nabla_{\mathbf{w}} \mathbf{a}^\top \cdot B_{\mathbf{w}}(\mathbf{x})$.

Concretely, let

$$\mathcal{E}^- = \{\mathbf{e}_i \in \mathbb{R}^{n_w} : \nabla_{\mathbf{w}}(\mathbf{a}^\top \cdot B_{\mathbf{w}})(\mathbf{x}) \cdot \mathbf{e}_i < 0\},$$

$$\mathcal{E}^+ = \{\mathbf{e}_i \in \mathbb{R}^{n_w} : \nabla_{\mathbf{w}}(\mathbf{a}^\top \cdot B_{\mathbf{w}})(\mathbf{x}) \cdot \mathbf{e}_i > 0\},$$

$$\mathcal{E}^0 = \{\mathbf{e}_i \in \mathbb{R}^{n_w} : \nabla_{\mathbf{w}}(\mathbf{a}^\top \cdot B_{\mathbf{w}})(\mathbf{x}) \cdot \mathbf{e}_i = 0\},$$

where $\{\mathbf{e}_1, \dots, \mathbf{e}_{n_w}\}$ is the standard basis for $\mathbb{R}^{n_w}$.

Intuitively, we can cover more of $\mathcal{J}$ by moving further in directions that take us away from the boundary at $\mathbf{a}^\top \cdot B_{\mathbf{w}}(\mathbf{x}) = b$. When the activations are ReLU functions, we have $\nabla_{\mathbf{w}}(\mathbf{a}^\top \cdot B_{\mathbf{w}_i})(\mathbf{x}) \cdot \mathbf{e}_i = 0$ for all i corresponding to weights feeding directly to inactive nodes. Since the output value of an inactive node is locally constant, we can move in directions corresponding to these weights without changing the network's output, thereby remaining inside $\mathcal{J}$. This assumption holds only locally - larger deviations in the parameter space will alter the network's activation pattern. Nevertheless, assuming local linearity yields a substantial increase in the volume of verifiable $\hat{\mathcal{J}}_i$'s.

With this knowledge, let us define

$$\mathbf{e}^+ = \sum_{\mathbf{e}_i \in \mathcal{E}^+ \cup \mathcal{E}^0} \mathbf{e}_i, \quad \text{and} \quad \mathbf{e}^- = \sum_{\mathbf{e}_i \in \mathcal{E}^- \cup \mathcal{E}^0} \mathbf{e}_i.$$

Algorithm 6: Certifying BNNs using GIE**Input:** BNN $B_{\mathbf{w}}$, distribution $\mathbf{w} \sim \mathcal{P}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$, safety specification $\varphi = \langle \mathcal{X}, \mathcal{S} \rangle$, number of samples $s \in \mathbb{N}$, scaling factor $\lambda > 0$, and
gradient-based scaling factor $\rho > 0$.**Output:** $\hat{\mathcal{J}}$ and p_{safe} .

```

1   $\hat{\mathcal{J}} \leftarrow \emptyset$ ;
2  for  $i \leftarrow 1$  to  $s$  do
3       $\mathbf{w} \sim \mathcal{P}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$ ;
4       $\mathcal{E}^- \leftarrow \{\mathbf{e}_i \in \mathbb{R}^{n_{\mathbf{w}}} : \nabla_{\mathbf{w}}(\mathbf{a}^\top \cdot B_{\mathbf{w}})(\mathbf{x}) \cdot \mathbf{e}_i < 0\}$ ;
5       $\mathcal{E}^+ \leftarrow \{\mathbf{e}_i \in \mathbb{R}^{n_{\mathbf{w}}} : \nabla_{\mathbf{w}}(\mathbf{a}^\top \cdot B_{\mathbf{w}})(\mathbf{x}) \cdot \mathbf{e}_i > 0\}$ ;
6       $\mathcal{E}^0 \leftarrow \{\mathbf{e}_i \in \mathbb{R}^{n_{\mathbf{w}}} : \nabla_{\mathbf{w}}(\mathbf{a}^\top \cdot B_{\mathbf{w}})(\mathbf{x}) \cdot \mathbf{e}_i = 0\}$ ;
7       $\mathbf{v}^+ \leftarrow \lambda \boldsymbol{\sigma}(\mathbf{1} + \rho \sum_{\mathbf{e}_i \in \mathcal{E}^+ \cup \mathcal{E}^0} \mathbf{e}_i)$ ;
8       $\mathbf{v}^- \leftarrow \lambda \boldsymbol{\sigma}(\mathbf{1} + \rho \sum_{\mathbf{e}_i \in \mathcal{E}^- \cup \mathcal{E}^0} \mathbf{e}_i)$ ;
9       $j \leftarrow 1$ ;
10      $\bar{\mathcal{J}} \leftarrow \emptyset$ ;
11     while  $LBP([\mathbf{w}_i - j\mathbf{v}^-, \mathbf{w}_i + j\mathbf{v}^+], \mathcal{X}) \subseteq \mathcal{S}$  do
12          $\bar{\mathcal{J}} \leftarrow [\mathbf{w}_i - j\mathbf{v}^-, \mathbf{w}_i + j\mathbf{v}^+]$ ;
13          $j \leftarrow j + 1$ ;
14     end
15      $\hat{\mathcal{J}} \leftarrow \hat{\mathcal{J}} \cup \bar{\mathcal{J}}$ ;
16 end
17 return  $\hat{\mathcal{J}}, \int_{\hat{\mathcal{J}}} q(\mathbf{w}) d\mathbf{w}$ ;

```

Now, instead of starting from the orthotope $\hat{\mathcal{J}}_i = [\mathbf{w}_i \pm \lambda \boldsymbol{\sigma}]$ as in the PIE approach, we start from $\hat{\mathcal{J}}_i = [\mathbf{w}_i - \lambda \boldsymbol{\sigma}(\mathbf{1} + \rho \mathbf{e}^-), \mathbf{w}_i + \lambda \boldsymbol{\sigma}(\mathbf{1} + \rho \mathbf{e}^+)]$, where $\rho > 0$ is the *gradient-based scaling factor*. Then, we check whether $\hat{\mathcal{J}}_i \subseteq \mathcal{J}$ (line 11 in Algorithm 6) and iteratively expand $\hat{\mathcal{J}}_i$, similarly to the PIE approach, until $\hat{\mathcal{J}}_i \not\subseteq \mathcal{J}$.

5.2.3 Theoretical Comparison

Here we compare Algorithms 5 and 6 against a state-of-the-art sampling-based approach [Wicker et al., 2020] and show that both algorithms introduced produce provably tighter lower bounds for the probabilistic robustness of BNNs. We start this by comparing Algorithms 5 and 6 against the static orthotopes approach of Wicker et al. [2020]. We show that Algorithms 5 and 6 always provide tighter lower bounds than the static orthotopes approach. We provide an intuition for the three approaches in Figure 5.2.

Proposition 5. *For every BNN, $B_{\mathbf{w}}$, and specification, $\phi = \langle \mathcal{X}, \mathcal{S} \rangle$, let p be the probabilistic robustness of $B_{\mathbf{w}}$ with respect to ϕ , and p_s, p_p , and p_g be the probabilistic robustness lower bounds computed with the same sampled weights $\mathbf{w}_1, \dots, \mathbf{w}_s$ using the pure sampling, PIE, and GIE approaches, respectively. Then we have that $p_s \leq p_p, p_g \leq p$.*

Proof. It is trivial that $p_s, p_p, p_g \leq p$. We show that $p_s \leq p_p$. Let $\hat{\mathcal{J}}_s$ and $\hat{\mathcal{J}}_v$ be the robustness certificates computed using $\mathbf{w}_1, \dots, \mathbf{w}_s$ by the pure sampling approach [Wicker et al., 2020] and the PIE approach (Algorithm 5), respectively. For each $\mathbf{w}_i$, the pure sampling approach checks whether $LBP([\mathbf{w}_i \pm \lambda\sigma], \mathcal{X}) \subseteq \mathcal{S}$. This is also checked in line 6 of Algorithm 5. Therefore, if $[\mathbf{w}_i \pm \lambda\sigma] \subseteq \hat{\mathcal{J}}_s$, then $[\mathbf{w}_i \pm \lambda\sigma] \subseteq \hat{\mathcal{J}}_v$. Thus, $\hat{\mathcal{J}}_s \subseteq \hat{\mathcal{J}}_v$ and $p_s = \int_{\hat{\mathcal{J}}_s} B_{\mathbf{w}}(\mathbf{x}) d\mathbf{w} \leq \int_{\hat{\mathcal{J}}_v} B_{\mathbf{w}}(\mathbf{x}) d\mathbf{w} = p_p$. The proof for $p_s \leq p_g$ is similar. $\square$

Intuitively, one may expect that $p_p \leq p_g$; however, this does not necessarily hold. For instance, if the gradient-based scaling factor ρ is too large, then $\mathbf{w} + j\mathbf{v}^+$ may well lie within the set of unsafe weights.

As presented in Section 5.3, both PIE and GIE allow us to compute bounds for the probabilistic robustness of BNNs that are significantly tighter than the state of the art. Before presenting their performance, we would like to emphasise

the theoretical significance of the iterative expansion, used in both approaches, in Example 5.

Example 5. Consider a BNN with k nodes and posterior $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$; the tightest lower bound that can be computed using a single orthotope results from an orthotope centred at $\boldsymbol{\mu}$. In the pure sampling approach, we have orthotopes of the form $\mathcal{O} = [\boldsymbol{\mu} \pm \lambda \boldsymbol{\sigma}]$ (cf. Algorithm 5). For a fixed $\lambda > 0$, the tightest lower bound that can be obtained for probabilistic robustness is p_λ^k , where $p_\lambda = \text{Prob}_{x \sim \mathcal{N}(0,1)}(|x| \leq \lambda)$. This means that the probabilistic robustness approximation computed using only pure sampling is exponentially vanishing as the parameter count of the network increases. This cannot be addressed by increasing the number of sampled orthotopes, as exponentially many orthotopes would be needed.

To better understand this effect, we observe that for a BNN with only 10^3 parameters and $\lambda = 3$, this gives us a lower bound of $p_3^{1,000} \approx 0.9974^{1,000} < 0.08$.

5.3 Evaluation

We evaluate the performance of our approach against the previously discussed state-of-the-art approach for computing lower bounds for probabilistic robustness [Wicker et al., 2020]. As in previous studies, the approach is evaluated on the MNIST dataset [LeCun et al., 2010] and additionally the CIFAR10 dataset [Krizhevsky, 2009]. All experiments were run on an AMD Ryzen Threadripper 3970X 32-core with 256GB RAM.

The evaluation is divided into three parts: 1) benchmarking against the state-of-the-art pure sampling approach in [Wicker et al., 2020], 2) an ablation study on the use of gradient-based dynamic scaling factors, and 3) an ablation study on the number of expansion iterations.

For the MNIST baselines, we use 6 fully-connected BNNs from [Wicker et al.,

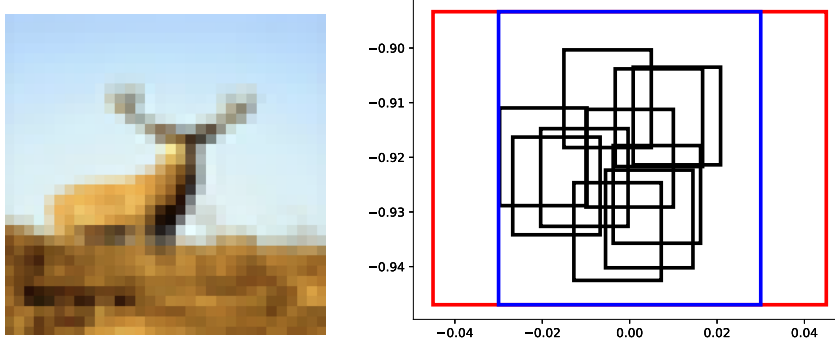


Figure 5.2: A 2-dimensional cross-section of the safe region $\mathcal{C}$ covered by pure sampling (black), PIE (blue), and GIE (Red) on the same image in CIFAR dataset.

2020]. However, we note that these networks exhibit narrow posterior distributions (standard deviation $\approx 10^{-5}$). Therefore, we also use two networks trained for broader posterior distributions (standard deviation $\approx 10^{-3}$) that achieve comparable clean accuracy and show greater natural resilience to adversarial attack, a key strength of BNNs [Bortolussi et al., 2022]; we discuss training details for these models in Section 5.5. MNIST networks are identified using an $\ell \times n$ nomenclature with ℓ , the number of hidden layers, and n , the number of nodes per hidden layer.

We also include a Convolutional Neural Network (CNN) trained on the CIFAR10 dataset. The CNN consists of 5 deterministic convolutional layers followed by 3 fully-connected Bayesian layers with 256, 64, and 10 nodes, respectively. As in [Wicker et al., 2020], all parameter posteriors are Gaussian.

5.3.1 Comparison of Pure Sampling, PIE, and GIE Approaches

The performances of sampling-based methods are biased by the compute available and the maximum number of iterations. To remove this bias, we enforce an equal limit on the total number of LBP calls per image. The results are presented in Table 5.1. We note that the GIE approach must also compute network gradients; a step that neither PIE nor the pure sampling approach requires. Nevertheless, in practice, we find that the computation of network gradients is much faster than the LBP used in all approaches. We quantify and discuss the impact on runtime in Section 5.4.

The results show a significant improvement over the pure sampling method of [Wicker et al., 2020] in all cases with gradient-based scaling providing the tightest guarantees. Moreover, we remark that in obtaining these results, the method from [Wicker et al., 2020] depends on careful selection of hyperparameters, namely the orthotope side length, where the use of suboptimal values yields trivially small guarantees - for the results in Table 5.1, we used grid-search to tune the hyperparameters in the pure sampling approach. In contrast, by using a small value for the initial orthotope's side length, our approach is able to adapt to each image and each network with little consideration for tuning hyperparameters.

In summary, the approach presented here considerably outperforms the present state of the art in estimating probabilistic robustness, leading to bounds that often more than double the verification bounds in [Wicker et al., 2020].

Dataset	Network	Accuracy (%)	Adversarial Accuracy (%)	Input Perturbation, ϵ	Probabilistic Certification (%)		
					Sampling (Wicker et al.)	Ours (PIE)	Ours (GIE)
MNIST	1x128	96	40	0.025	26.0	56.9	58.4
	1x256	94	42		22.0	50.0	51.2
	1x512	94	36		10.0	35.0	37.3
	2x256	92	16	0.001	72.0	74.0	74.0
	2x512	82	2		22.0	22.0	26.1
	2x1024	68	0		0.0	0.0	0.0
	2x50	96	70	0.01	58.6	67.3	71.8
	2x150	92	54	0.001	49.8	54.9	56.4
CIFAR10	CNN	72	2	0	22.0	27.1	35.6

Table 5.1: Probabilistic robustness results for the pure sampling method from [Wicker et al., 2020] and ours. Adversarial accuracies were obtained using an $\epsilon = 0.05$ bounded PGD attack for all networks. Probabilistic certification denotes the mean p_{safe} by each approach. Also, see Section 5.6 for a comparison of the probabilistic certification results obtained by each approach with the empirical probabilistic robustness values for each network.

5.3.2 Ablation Study on the Use of Gradient-Based Dynamic Scaling Factors

Here we explore the impact of using gradient-based dynamic scaling factors. In Figure 5.3, we plot the *probabilistic certification*, i.e., the mean p_{safe} , attained for a range of gradient-based scaling factors and networks. Consistent with [Wicker et al., 2020], each result is averaged over 50 images from the MNIST test set. In each case, we observe that the optimal scaling factor is larger than 0 - which represents no gradient-guided scaling and is marked by the red dashed line. The optimal gradient-based scaling factor varies between networks, as does its effect on the overall p_{safe} . Examining the results for individual images with a gradient-based scaling factor of 0 goes towards explaining this behaviour.

For the 1x256 network, we find that 89.1% of the correctly classified images have p_{safe} values greater than 99.9% or less than 0.1%. For these networks, most of the images are either trivially verifiable or prohibitively challenging, with only 10.9% in between. In such instances, the use of gradient-based scaling factors has little effect, as p_{safe} is either very close to 1, or adversarial examples are

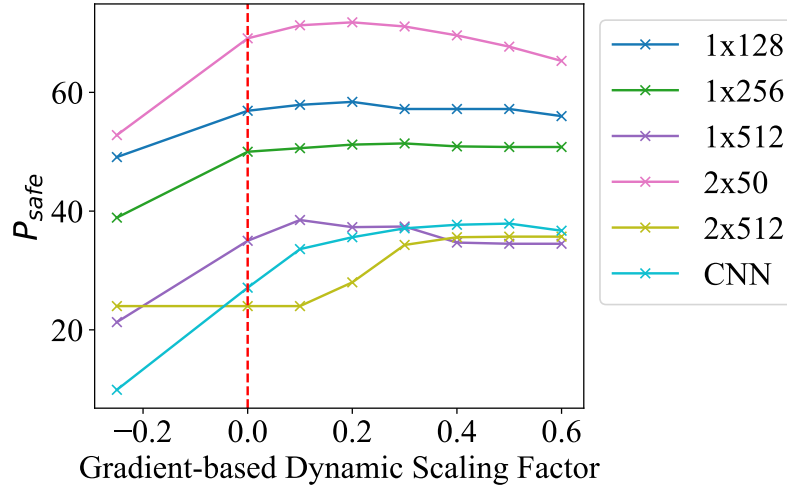


Figure 5.3: Probabilistic certification as a function of ρ , the gradient-based scaling factor in Algorithm 6.

found before p_{safe} can reach substantial values.

To further illustrate this point, we examined the CNN network, which was most sensitive to the gradient-based scaling factor, in the same way. We found that only 68.6% of the correctly classified images lay outside the 0.1% - 99.9% boundary. To remove this dilution effect, we examined how gradient-based dynamic scaling impacts a single image; for which we use the same example given in Figure 5.2. Using no gradient-based dynamic scaling on this image, we achieve $p_{\text{safe}} = 36.5\%$ while using a gradient-based scaling factor of 0.2 we reach $p_{\text{safe}} = 97.8\%$, highlighting the importance of gradient-based dynamic scaling on certain inputs.

5.3.3 Ablation Study on the Number of Iterations

To better understand the inner workings of the iterative expansion approaches presented in this chapter, we calculated the number of iterations it took Algo-

Algorithm 5 to reach the maximum probabilistic certification presented in Table 5.1. The results are available in Figure 5.4. We observe that Algorithm 5 takes 8 iterations to start computing a meaningful lower bound and at most 13 iterations to reach maximum probabilistic certification for the MNIST networks. The minimum number of iterations to obtain a meaningful lower bound can be reduced, e.g., to a single iteration, by increasing the value of λ in line 8 of Algorithm 5; however, this is disadvantageous for two reasons: (1) one has to manually search for an optimal value for λ in a manner similar to the pure sampling approach, and (2) this can reduce the certified robustness obtained in later iterations because the expansion rate of PIE increases with λ .

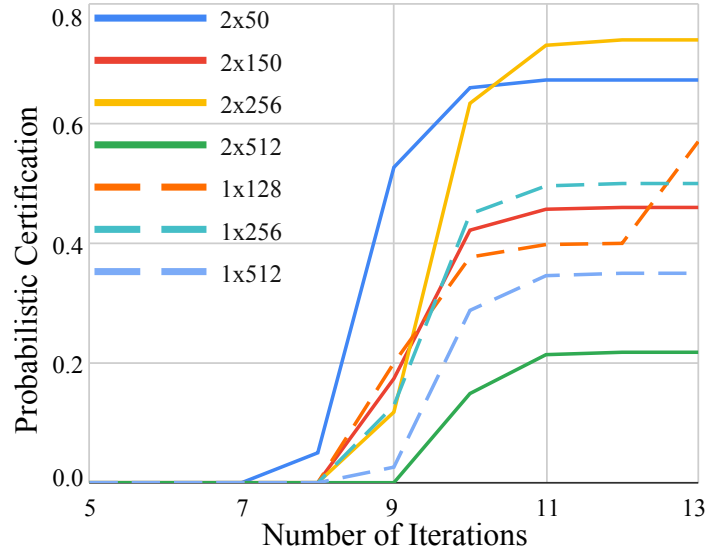


Figure 5.4: The number of expansion iterations before PIE reaches max probabilistic certifications in Table 5.1.

5.4 Evaluation of Timing for PIE Algorithm

In Table 5.2 we examine the impact on runtime of generating the gradients required for the PIE algorithm. We observe that in each case there is an increase

Network	Relative Runtime ($\pm\%$)
1x128	+3.3
1x256	+2.5
1x512	+1.4
2x256	+0.8
2x512	+0.6
2x1024	+0.2
2x50	+2.3
2x150	+0.4
CNN	+0.5

Table 5.2: Impact of computing gradients for PIE algorithm on runtime. The data are presented as a percentage increase or decrease in runtime as compared to the [Wicker et al., 2020] approach on the same networks.

in runtime, peaking at 3.3% for the 1x128 network. The relative cost decreases as network size increases as a result of the gradient computation scaling better than the LBP computation with network size.

5.5 Training Details

In addition to evaluating our approaches on the state-of-the-art benchmarks, we evaluated our approach on two fully connected BNNs and a convolutional BNN (only the fully connected layers are Bayesian). Similarly to the state-of-the-art approach, we used normal distributions for the prior (with mean 0 and standard deviation 1) and the posterior of the Bayesian layers. Using a wider standard deviation than the state-of-the-art, as expected, has resulted in networks that are more adversarially robust (cf. Table 5.1). We trained the networks using variational inference [Blundell et al., 2015] using a learning rate of 0.001 with Adam optimiser [Kingma and Ba, 2015]. All the hidden layers in the networks use ReLU activations.

The fully connected networks are trained on the MNIST dataset and each

have 2 hidden layers with 50 and 150 nodes, respectively. The convolutional network is trained on the CIFAR10 dataset and has 5 convolutional layers with filter size 3 and stride 2, followed by 3 fully connected Bayesian hidden layers of sizes 256 and 64.

5.6 Comparison Against Empirical Probabilistic Robustness

In Figure 5.5, we compare the pure sampling approach of [Wicker et al., 2020], as well as the PIE and GIE approaches, presented in this chapter, against the empirical values we obtained for the probabilistic robustness of the networks in Table 5.1. We calculated the values for the empirical probabilistic robustness by sampling from the BNN posterior and using IBP to check whether the samples were classified correctly. Each value for the empirical accuracy shows the percentage of samples that resulted in correct classification.

We observe that PIE and GIE always perform favourably compared to pure sampling, and in some instances, such as for 1-layer networks, PIE and GIE significantly outperform pure sampling. Nevertheless, for most networks, there is still a gap between the certified robustness obtained by each of the approaches and the empirical probabilistic robustness.

5.7 Conclusions

As discussed in Section 5.1, BNNs have been proposed as an inherently robust architecture. This is particularly important in safety-critical applications, where robust solutions are essential, and models need to be shown to be highly robust before deployment.

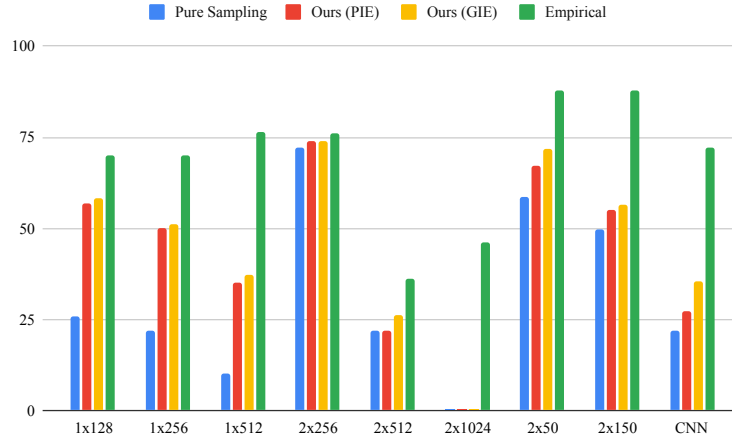


Figure 5.5: Comparison of the pure sampling, PIE, and GIE approaches against the empirical probabilistic robustness, obtained by sampling parameters from the BNN posterior and using IBP to verify whether they result in correct classification. 50 samples were used for calculating each empirical accuracy.

Given this, it is essential to ascertain the robustness of BNNs in production. Unlike deterministic neural networks, where robustness verification can be recast as a deterministic optimisation problem, establishing the robustness of BNNs is a probabilistic problem. Finding an exact solution to this problem is infeasible even for small models. As reported, proposals have been put forward to approximate the probabilistic safety of BNNs by sampling regions in the weight space; however, such approximations are loose and underestimate the actual robustness of the model, thereby providing relatively little information. This limits the possibility of using BNNs in tasks where robustness needs to be verified before deployment.

We have presented an alternative approach to search the weight space and, therefore, approximate the lower bounds of probabilistic safety in BNNs. The algorithms proposed provably produce tighter bounds than the state-of-the-art sampling-based approaches. This result was validated in all experiments conducted. The approaches were evaluated on all the previously proposed bench-

marks, as well as new benchmarks introduced in this chapter, reporting a marked improvement in the bounds for all benchmarks. In some cases, we obtain bounds that are 50% higher than the previously known bound of just over 20%, opening the way to better evaluate the robustness of BNNs before deployment.

Methods for verifying the probabilistic robustness of BNNs are extremely sensitive to parameter count - this remains true for both algorithms proposed here. Therefore, scaling to larger networks is a challenge due to two primary effects: firstly, as network size increases, so does the parameter count and the respective dimensionality of the probability space. The higher this dimensionality, the more important sampling *large* orthotopes becomes, and the less effective sampling *multiple* orthotopes becomes. This reliance on being able to verify single large orthotopes makes these algorithms vulnerable to loose verification techniques. Secondly, as network size increases, it is well known that the tightness of bound propagation techniques suffers, thus impacting the ability of our algorithms to verify any orthotopes. Both of these limitations can be alleviated by using tighter verification procedures in place of bound propagation.

Chapter 6

Probabilistic Counterfactual Explanations on Bayesian Neural Networks

6.1 Introduction

In Section 2.7 we introduced counterfactual explanations (CFX) as a method for improving AI safety through interpretability. We also highlighted some key desiderata for CFXs, and outlined where current approaches fall short. Expanding on our investigation of AI safety methods in probabilistic models from Chapter 5, here we introduce the notion of counterfactual explanations for BNNs. In particular, we posit that the limitations of CFXs stem from the deterministic nature of traditional neural networks, which fail to capture inherent uncertainty in the data. To address this fundamental issue, we propose a novel framework for generating counterfactual explanations using BNNs. The approach leverages

the uncertainty quantification capabilities of BNNs to produce CFXs that are more plausible and robust than those generated by deterministic models.

To validate the approach, we conduct a comprehensive empirical evaluation across multiple datasets, including *MNIST* [LeCun et al., 2010] as well as several tabular datasets, including *German Credit Risk* [Dua and Graff, 2017], *Diabetes* [Smith et al., 1988], *News Categorisation* [Fernandes et al., 2015], and *Spam Base* [Hopkins et al., 1999], covering various domains such as finance and healthcare. Our results consistently show that BNN-based CFXs outperform deterministic baselines across various metrics, including plausibility and robustness. Notably, we find that BNN-based CFXs also tend to outperform CFXs built using ensembling techniques.

The remainder of this chapter is organised as follows. We present the key contribution in Section 6.2, where we formally define counterfactual explanations for BNNs and show how they can be computed. We validate the approach in Section 6.3 and present an extensive experimental evaluation using standard datasets from the CFX literature. Finally, we discuss the broader implications of the work for the field of explainable AI and the practical deployment of machine learning models in Chapter 7.

6.2 Counterfactual Explanations for BNNs

Counterfactual explanations do not have a commonly accepted definition in probabilistic (non-causal) models and have never been formally defined for BNNs. Here, we propose a framework for defining and computing counterfactuals specific to BNNs. In contrast to deterministic networks, the parameters of a BNN are modelled as distributions rather than fixed values, complicating the definition of counterfactuals. Specifically, while counterfactuals for deterministic networks usually require the computation of model gradients, the gradient

of a BNN's output with respect to its input is distributional, and its expected value is difficult to compute exactly. Moreover, to compute the true gradient of a BNN with respect to its input, we differentiate Equation (2.17) with respect to $\mathbf{x}$:

$$\begin{aligned}\partial_{\mathbf{x}}B(\mathbf{x}) &= \partial_{\mathbf{x}} \left(\int_W B(\mathbf{x}|\mathbf{w})P(\mathbf{w}) d\mathbf{w} \right) \\ &= \int_W \partial_{\mathbf{x}}B(\mathbf{x}|\mathbf{w})P(\mathbf{w}) d\mathbf{w}.\end{aligned}\tag{6.1}$$

However, both Equation (2.17) and its gradient in Equation (6.1) are intractable to compute directly. Similar to Equation (2.18), we can approximate the expected gradient of a BNN through Monte Carlo sampling,

$$\partial_{\mathbf{x}}\tilde{B}(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^N \partial_{\mathbf{x}}B(\mathbf{x}|\mathbf{w}_i),\tag{6.2}$$

where $\mathbf{w}_i$ are samples from the parameter posterior. In this setting, the gradient $\partial_{\mathbf{x}}B(\mathbf{x}|\mathbf{w}_i)$ can be computed in the same way as a standard, deterministic MLP.

Proof of Monte Carlo Approximation. Let $\mathbf{w}_1, \dots, \mathbf{w}_N$ be i.i.d. samples from $P(\mathbf{w})$. Define

$$X_i \triangleq \partial_{\mathbf{x}}B(\mathbf{x}|\mathbf{w}_i), \quad i = 1, \dots, N.$$

Since

$$\mathbb{E}[X_i] = \int_W \partial_{\mathbf{x}}B(\mathbf{x}|\mathbf{w})P(\mathbf{w}) d\mathbf{w} = \partial_{\mathbf{x}}B(\mathbf{x}),$$

the estimator in Equation (6.2) is the sample mean of $\{X_i\}_{i=1}^N$. By the *Strong Law of Large Numbers*, we have

$$\frac{1}{N} \sum_{i=1}^N X_i \xrightarrow{a.s.} \mathbb{E}[X_i] = \partial_{\mathbf{x}}B(\mathbf{x}),$$

as $N \rightarrow \infty$. Thus, the Monte Carlo estimator converges almost surely to the

true gradient in Equation (6.1).

Furthermore, by the *Central Limit Theorem*,

$$\sqrt{N} \left(\frac{1}{N} \sum_{i=1}^N X_i - \partial_{\mathbf{x}} B(\mathbf{x}) \right) \xrightarrow{d} \mathcal{N}(0, \sigma^2),$$

where $\sigma^2 = \text{Var}(\partial_{\mathbf{x}} B(\mathbf{x}|\mathbf{w}))$. This shows that the error decays as $O(1/\sqrt{N})$. If $\partial_{\mathbf{x}} B(\mathbf{x}|\mathbf{w})$ is bounded, concentration inequalities (e.g., Hoeffding's inequality) can also provide error bounds. $\square$

Having established the mathematical framework, we can now formally define probabilistic counterfactual explanations for BNNs. This definition not only computes counterfactual explanations with minimal distance from the original input but also incorporates the model's inherent uncertainty.

Definition 9 (Probabilistic Counterfactual Explanation). Given a BNN B , an input $\mathbf{x}$, with observed negative outcome $B(\mathbf{x})_l = 0$, a *probabilistic counterfactual explanation* is an input $\mathbf{x}_c$ such that $B(\mathbf{x}_c)_l = \mathbb{E}_{\mathbf{w} \sim P(\mathbf{w})}(B(\mathbf{x})_l) = 1$. Formally,

$$\mathbf{x}_c = \arg \min_{\mathbf{x}'} d(\mathbf{x}, \mathbf{x}') \text{ s.t. } \arg \max_l B(\mathbf{x}')_l = 1. \quad (6.3)$$

Equation (6.3) describes the output constraint for the classification setting on which we focus. For regression tasks, we can replace the constraint with a bound on the output of the network, $l_i \leq B(\mathbf{x}_c)_i \leq u_i, i = 1, \dots, n$ where n is B 's output dimension. Moreover, whilst we have focused on the binary classification setting, this definition can be extended to the multi-class case by replacing the negative outcome with the original class y , and the positive outcome with some target class, t . We continue with this more generalised notation for the classification setting.

To compute CFXs, we parallel the optimisation formulation given in Equation (2.44). Concretely, and focusing on the classification case, we use a linear

loss, $\mathcal{L}_{\text{lin}}$, for a specified target class, t , and write our objective function as

$$\arg \min_{\mathbf{x}_c} \mathcal{L}_{\text{lin}}(\tilde{B}(\mathbf{x}_c), t) + \lambda \cdot d(\mathbf{x}, \mathbf{x}_c), \quad (6.4)$$

where $\mathcal{L}_{\text{lin}}(\tilde{B}(\mathbf{x}_c), t) = \tilde{B}(\mathbf{x}_c)_y - \tilde{B}(\mathbf{x}_c)_t$ with $\tilde{B}(\mathbf{x}')_y$ being the average value of the output marginal predictive probability corresponding to the observed class y and $\tilde{B}(\mathbf{x}_c)_t$ that for the target class. We have selected linear loss due to its computational efficiency and its frequent application in the robustness literature, where it is known for prompting rapid changes in model outputs [Carlini and Wagner, 2017b]. However, alternative loss functions, such as cross-entropy, may also be employed, as discussed in Section 6.3. Echoing Equation (2.44), the first term in the objective of Equation (6.4) accounts for the validity of candidate CFXs, while the second term in Equation (6.4) promotes CFXs that are closer to the original input $\mathbf{x}$.

Based on this objective function, we outline our algorithm for computing probabilistic counterfactuals in Algorithm 7. The procedure begins by initialising the counterfactual with the original input vector and proceeding to the main loop. Within the main loop, we alternate between computing approximate gradients using Equation (6.2), and stepping the counterfactual according to the gradient. In our experiments, we set L and U as the upper and lower bounds on the input, though it is possible to limit this to an l_p ball if there is a pre-defined budget for the counterfactuals. We note that, as for the deterministic setting, this algorithm does not guarantee a valid counterfactual and that the choice of λ , ϵ , and N will dictate this as tunable parameters.

At this point, it is worth stepping back to highlight the similarities between CFX and the adversarial attacks we have previously discussed and which are defined in Definition 3. Mathematically, these two inputs are computed similarly via input sub-gradients, with an objective function which involves changing the

output class or value sufficiently. However, the semantic objectives are quite different: adversarial attacks generally prioritise cost and detectability, while counterfactuals, as we explore in this chapter, have a more varied list of desirable characteristics. For example, a CFX might prioritise the plausibility of an input above the cost alone, due to the necessity for CFX to be actionable. The different high-level goals between CFX and adversarial attacks result in different algorithmic constraints and evaluation criteria which must be considered in the design of their respective algorithms.

Algorithm 7: Generating Counterfactual Explanations (CFX) for BNNs

Input: BNN B , input sample $\mathbf{x}$, target class y , step size ϵ ,

distance weight λ , number of iterations N , input bounds L and U

Output: Counterfactual explanation $\mathbf{x}_c$

```

1 Initialise counterfactual:  $\mathbf{x}_c \leftarrow \mathbf{x}$ ;
2 for  $n \leftarrow 1$  to  $N$  do
3   Compute gradient of loss w.r.t. input;;
4   
$$\delta \leftarrow \partial_{\mathbf{x}_c} \left[ \mathcal{L}_{\text{lin},t}(\tilde{B}(\mathbf{x}_c)) + \lambda \|\mathbf{x} - \mathbf{x}_c\|_p \right]$$

5   Update counterfactual:  $\mathbf{x}_c \leftarrow \mathbf{x}_c + \epsilon \cdot \delta$ ;
6   Clip counterfactual to input bounds:  $\mathbf{x}_c \leftarrow \text{clip}(\mathbf{x}_c, L, U)$ ;
7 end
8 return  $\mathbf{x}_c$ 
```

6.3 Evaluation

In this section, we evaluate counterfactual explanations produced for BNNs. We focus on three main properties: validity, robustness, and plausibility, and show that CFXs obtained for BNNs outperform those produced for traditional

Multi-Layer Perceptrons (MLPs). We also test other methods for uncertainty quantification, namely ensemble methods, and show that BNNs produce better CFXs in most instances. We conducted experiments on various popular datasets to cover different data types and classification tasks. They include one vision dataset, MNIST [LeCun et al., 2010], and four tabular datasets: *credit* [Dua and Graff, 2017], *diabetes* [Smith et al., 1988], *news* [Fernandes et al., 2015], and *spambase* [Hopkins et al., 1999]. For each dataset, we trained the following models: a single standard, deterministic multi-layer perceptron (*MLP*), an ensemble of 50 randomly initialised MLPs (*Ensemble*), and a Bayesian Neural Network (*BNN*). We keep the architectures of the three models consistent with 2 hidden layers, 150 nodes each, to facilitate comparison. In training the BNNs, we use an adaptive variant of the HMC algorithm called NUTS provided as part of the `numpyro` package.

We have chosen these benchmarks as they represent BNNs’ closest deterministic and uncertainty-aware counterparts. An MLP is the least complex form of deep neural network and is also used exhaustively in the CFX literature as a case study, qualifying it as a key benchmark. We also compare with ensembles of MLPs as these have previously been studied in the context of uncertainty-aware CFX by [Schut et al., 2021]. Moreover, MLP ensembles have functional similarities to BNNs - as we use a sampling-based Bayesian inference algorithm, our BNNs can be considered as a finite ensemble of samples in the same way as an ensemble of MLPs. In these comparisons, we hypothesise that the BNN will be better able to capture the underlying data manifold, leading to more robust and plausible counterfactuals in practice.

Counterfactual explanations are generated using gradient-based optimisation methods tailored to each model type. For MLPs and ensembles, we used standard projected gradient descent to find minimal input perturbations that change

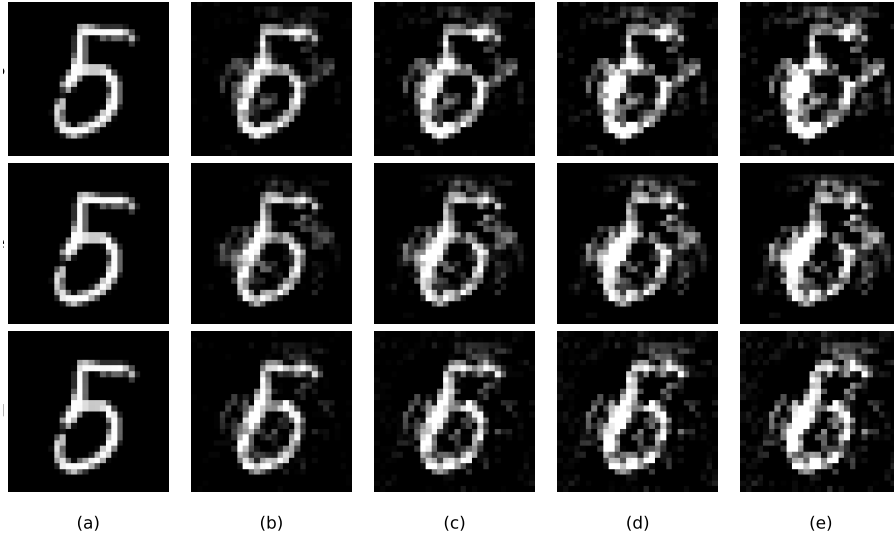


Figure 6.1: Generation of a counterfactual for an MLP, an Ensemble, and a BNN (top to bottom, respectively). a) shows the original image after which a counterfactual with target ‘6’ is progressively generated. b)-d) show snapshots of the image as number of iterations increases.

the model’s prediction according to Equation (2.44). For BNNs, we utilise the probabilistic counterfactual framework as defined in Section 6.2, leveraging Monte Carlo sampling to approximate gradients and defined in Equation (6.2). For every dataset, we compute counterfactuals for 50 random samples from the test set. All experiments are performed on an RTX 3080 GPU and AMD Ryzen Threadripper 3960x 24-core CPU with 256GB of RAM running Ubuntu 22.04.

In the rest of this section, we first look at a visual example from the MNIST dataset in Section 6.3.1, before defining our metrics and discussing numerical results in Sections 6.3.2 and 6.3.3. Finally, we compare against previous work on uncertainty-aware CFX in Section 6.3.4.

6.3.1 Visual Interpretation

We begin with an example from the MNIST image classification dataset, where we randomly select a target class for each image. Figure 6.1 shows snapshots of computing a counterfactual explanation on the three model types (MLP, ensemble, and BNN from top to bottom). Each row is run with the same hyperparameters, original image, and target class.

We observe that as the number of iterations increases from left to right, the images increasingly resemble the number six. However, the MLP suffers some erroneous fragments on either side of the number in b) and c). By e) the MLP’s CFX has begun to degrade, particularly on the right-hand side, to a point where it is nearly unrecognisable as a 6 to the human eye. The ensemble row also shows some noise around the number, but the degradation of ‘key’ pixels is less prominent and the final image is more recognisable as a six. The BNN also suffers from noise above and to the sides of the number at the b) and c) stages; however, the noise is less pronounced in these stages than we observe for the Ensemble and MLP. At stage e) the BNN’s CFX is considerably more complete than for the Ensemble or MLP; despite the pronounced noise, the key pixels are largely preserved and the resulting number six is more prominent.

We attribute the improved preservation of key pixels in the BNN to the better representation of the data distribution captured by this model. Similarly, we propose that the model averaging in the Ensemble prevents the major deterioration of the key pixels that we observe for the MLP. Specifically, for pixels to significantly change in the Ensemble model, they must have a significant impact on the output across all models of the ensemble. This helps to mitigate any local minima we might observe in any single MLP. We emphasise that these counterfactuals are produced with no CFX-specific regularisation scheme in either training or the CFX algorithm, with the intention of examining the

explanations produced by these models in an unmodified state.

6.3.2 Metrics

We use three metrics for evaluating the counterfactuals produced for each model: *Local Outlier Factor* [Breunig et al., 2000] (LOF) provides a measure of how closely a counterfactual lies to the manifold of training data. It is frequently used as a measure of *plausibility* in the CFX literature. We also use the *Implausibility* measure from [Altmeyer et al., 2024] as a secondary measure of plausibility. As outlined in [Altmeyer et al., 2024], this metric considers the sample-averaged Euclidean distance between a counterfactual and any in-class sample from the training set. Finally, we define a novel metric, the *CFX Sensitivity*, to measure the sensitivity of explanations to initial points. For a CFX algorithm to be effective, it is essential to generate counterfactuals that are locally consistent (low sensitivity) to ensure fairness of the produced explanations; this concept is explored by [Slack et al., 2021]. In the broader CFX literature, this notion is often referred to as the *robustness* of counterfactuals [Artelt et al., 2021, Leofante and Potyka, 2024].

Definition 10 (CFX Sensitivity). Given an original input, $\mathbf{x}$, and a counterfactual explanation, $\mathbf{x}_c$, based on $\mathbf{x}$. We compute a second counterfactual, $\mathbf{x}'_c$, on a point $\mathbf{x}'$ where $\mathbf{x}'$ is sampled uniformly from the neighbourhood of $\mathbf{x}$: $\|\mathbf{x} - \mathbf{x}'\|_p$ with some budget b . We term the distribution governing these samples $U_b(\mathbf{x})$. The CFX Sensitivity is then defined as the ratio of the distance between the two counterfactuals, $\mathbf{x}_c$ and $\mathbf{x}'_c$, and the cost of the initial counterfactual (see Figure 6.2). Formally,

$$\text{CFX Sensitivity} = \mathbb{E}_{\mathbf{x}' \sim U_b(\mathbf{x})} \left[\frac{\|\mathbf{x}'_c - \mathbf{x}_c\|_p}{\|\mathbf{x}_c - \mathbf{x}\|_p} \right]. \quad (6.5)$$

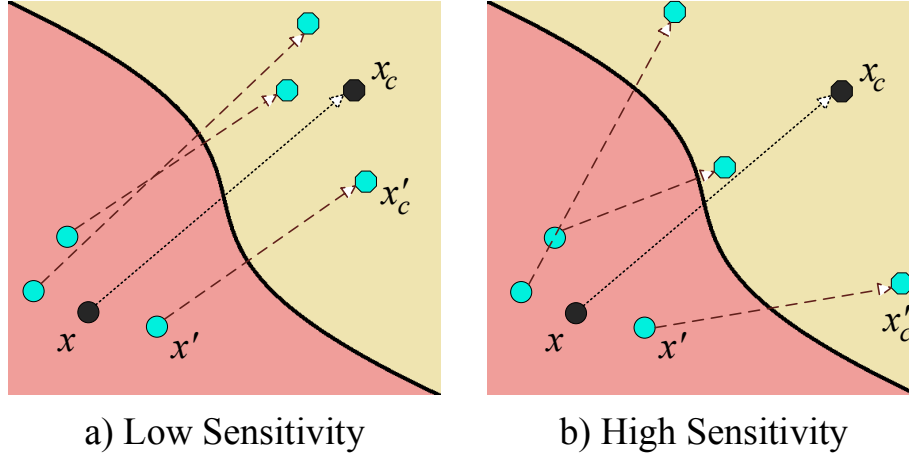


Figure 6.2: Figure 6.2a shows low sensitivity (high robustness), where the distance between x_c and x'_c is smaller compared to the high sensitivity (low robustness) scenario in Figure 6.2b.

We compute the expected value in Equation (6.5) using Monte Carlo sample averaging of $U_b(\mathbf{x})$. For all our experiments, we instantiate Definition 10 using l_2 norms, although this metric can be applied with any l_p norm in general. For the budget vector, we use 5% of the element-wise input domain. Where a feature has no obvious predefined input domain, we use the range of that feature in the training set. Any instance where we are unable to find a valid counterfactual is discarded when computing the CFX Sensitivity.

6.3.3 Numeric Results

Table 6.1: Numeric results for counterfactuals produced on the MNIST, credit, diabetes, news, and spambase datasets, and the MLP, Ensemble, and BNN model types. For each dataset/model pair we report three metrics covering the plausibility and robustness of the counterfactuals. We also report the clean accuracy, percentage of *valid* counterfactuals found for each pair, and the mean l_2 cost. The arrows indicate for each metric whether high is better ($\uparrow$) or lower is better ($\downarrow$).

Dataset	Model	Clean Accuracy (%)	Valid CFX (%)	Metric					l_2 Cost $\downarrow$
				LOF $\uparrow$	Implausibility $\downarrow$		CFX Sensitivity (10^{-3}) $\downarrow$		
					Mean	Std. Dev	Mean	Std. Dev	
MNIST	MLP	95.6	88.0	0.721	87.4	0.196	35.8	9.52	26.8
	Ensemble	97.3	82.0	0.463	91.0	0.040	3.94	0.113	83.4
	BNN	98.2	74.0	0.838	87.2	0.061	40.8	3.96	12.0
credit	MLP	75.5	100.0	1.0	3.45	0.0084	36.9	1.91	0.394
	Ensemble	76.5	96.0	1.0	3.36	0.0131	36.3	4.82	0.274
	BNN	71.0	88.0	1.0	3.31	0.0194	26.5	8.55	0.904
diabetes	MLP	77.9	100.0	0.949	0.427	0.0019	14.9	2.69	0.246
	Ensemble	76.0	100.0	0.897	0.425	0.0024	14.2	2.56	0.242
	BNN	72.7	100.0	1.0	0.435	0.0012	21.4	5.18	0.195
news	MLP	65.0	92.0	1.0	73.0	16.8	607	48.8	12.6
	Ensemble	65.9	100.0	1.0	66.5	0.57	381	9.23	17.1
	BNN	65.9	80.0	1.0	66.3	2.86	454	11.1	12.3
spambase	MLP	92.9	80.0	0.886	56.0	2.84	475	35.3	51.6
	Ensemble	92.9	92.0	0.902	59.2	3.00	377	22.2	52.1
	BNN	92.9	78.0	0.882	59.2	1.41	334	86.5	51.8

In Table 6.1 we report numerical results across three model types and five datasets. In addition to the metrics, we report the clean accuracy of the models and the percentage of valid counterfactuals found. *Valid Counterfactuals* is defined as the percentage of counterfactuals that our algorithm obtained that successfully change the model output to the intended target class. A validity of 100% implies that we found a valid counterfactual for all 50 sampled test inputs. For these results, we performed hyperparameter tuning of ϵ , λ , and N in Algorithm 7 independently for each dataset/model run, and excluded any run which obtained a *Valid Counterfactuals* score of less than 70%.

Focusing initially on LOF, the results show that the BNN achieved the largest or equivalent LOF score across all datasets except spambase. This indicates that for the BNN model, the counterfactuals we find better represent the training data distribution. We observe similarly that the Implausibility score was better or the same across all datasets except for spambase. We note in the experiments that there appears to be a trade-off between LOF and CFX Sensitivity. In Table 6.1, we prioritise LOF when selecting the best hyperparameter sets as this is a highly recognised metric in the CFX literature. For each dataset/model pair, we also report the mean l_2 cost of counterfactuals found. Although this is not a metric pertaining to plausibility or robustness, the literature on counterfactuals generally considers cheap counterfactuals to be desirable. The results show that in three of the five datasets, the counterfactuals produced on the BNN were the cheapest, as well as maintaining high scores over the full metric suite. In two cases, they provide both the cheapest counterfactuals and the highest LOF score; these results support our hypothesis. It is important to note that this is achieved with no hyperparameter tuning for low-cost CFX.

In Table 6.2, we provide the best scores for each metric across all runs. Here

Table 6.2: Best performance results by metric across all hyperparameter tuning runs. Arrows indicate for each metric whether high is better ($\uparrow$) or lower is better ($\downarrow$).

Metrics	Model	Datasets				
		MNIST	credit	diabetes	news	spambase
LOF $\uparrow$	MLP	0.721	1.0	0.949	1.0	0.886
	Ensemble	0.463	1.0	0.897	1.0	0.902
	BNN	1.0	1.0	1.0	1.0	0.882
Implausibility $\downarrow$	MLP	87.2	3.44	0.424	61.7	51.0
	Ensemble	90.9	3.35	0.422	65.5	55.8
	BNN	87.1	3.28	0.390	61.6	54.2
CFX Sensitivity (10^{-3}) $\downarrow$	MLP	16.9	34.3	11.1	513	409
	Ensemble	3.72	27.6	9.46	405	351
	BNN	33.4	9.79	10.5	289	162

we see a clear divide between the models with some component of averaging (Ensemble and BNN) and the MLP, which only achieves a best result in *Implausibility* on *spambase*. When comparing the Ensemble and BNN models, the results are similar, with the BNN yielding improved scores in nine of the twelve head-to-heads with the Ensemble model. We note that these scores are to give a full picture only and that prioritising a single metric is usually at heavy detriment to other metrics or the l_2 cost of the counterfactual.

6.3.4 Comparison with Ensembles

In [Schut et al., 2021], the authors propose producing counterfactuals that consider aleatoric and epistemic uncertainty. They use softmax output and an ensemble of MLPs to capture the aleatoric and epistemic uncertainties respectively. Unlike here, their counterfactuals are generated using cross-entropy loss rather than our linear loss function $\mathcal{L}_{lin}$. Moreover, in [Schut et al., 2021] the authors apply a variation on the Jacobian-based saliency map (JSMA) originally applied in adversarial attacks [Papernot et al., 2016]. Specifically, the JSMA limits updates to the input to only consider the input dimension with the largest partial gradients.

Here we apply these modifications to align our CFX algorithm with that from [Schut et al., 2021] and compare the CFXs produced by the Ensemble models and our BNNs. As in Section 6.3.3 we compare each model/dataset pair using three established metrics and show the results in Table 6.3.

The results in Table 6.3 show that the BNN outperforms or ties the Ensemble in LOF for every benchmark, and where there is a tie, the BNN maintains a lower CFX Sensitivity. The performance is more consistently in favour of the BNN than we see in Table 6.1. However, we note that the costs of the BNN’s counterfactuals are often higher than the Ensemble’s, in contrast to what we observe in Table 6.1.

Table 6.3: Numeric results for MNIST, credit, diabetes, news, and spambase datasets on the [Schut et al., 2021] MLP Ensembles and our BNNs. For each metric arrows indicate whether higher is better($\uparrow$) or lower is better ($\downarrow$).

Dataset	Model	Clean Accuracy (%)	Valid CFX (%)	Metric			l_2 Cost $\downarrow$
				LOF $\uparrow$	Implausibility $\downarrow$	CFX Sensitivity (10^{-3}) $\downarrow$	
MNIST	Ensemble	97.3	100.0	0.560	98.3	0.0172	14.0
	BNN	98.2	74.0	0.946	94.5	0.0081	3.40
credit	Ensemble	76.5	100.0	1.0	3.70	0.0508	1.58
	BNN	71.0	100.0	1.0	4.03	0.2825	2.38
diabetes	Ensemble	100.0	100.0	0.949	0.417	0.0448	0.202
	BNN	72.7	100.0	1.0	0.435	0.0422	0.277
news	Ensemble	65.9	74.0	1.0	66.8	0.0018	19.0
	BNN	65.9	80.0	1.0	64.1	0.0019	22.5
spambase	Ensemble	92.9	76.0	0.879	52.7	0.0022	47.8
	BNN	92.9	78.0	0.882	54.3	0.0019	47.0

In Figure 6.3 we compare counterfactuals produced by the Ensemble and BNN models. Here we observe a much clearer counterfactual for both models than in Figure 6.1; this is due to the JSMA filtering applied. However, we note that the Ensemble model continues to produce more prominent erroneous artefacts than the BNN.

6.4 Conclusions

In this chapter, we have presented the first formal study of counterfactual explanations for Bayesian Neural Networks. We proposed a definition for CFX within

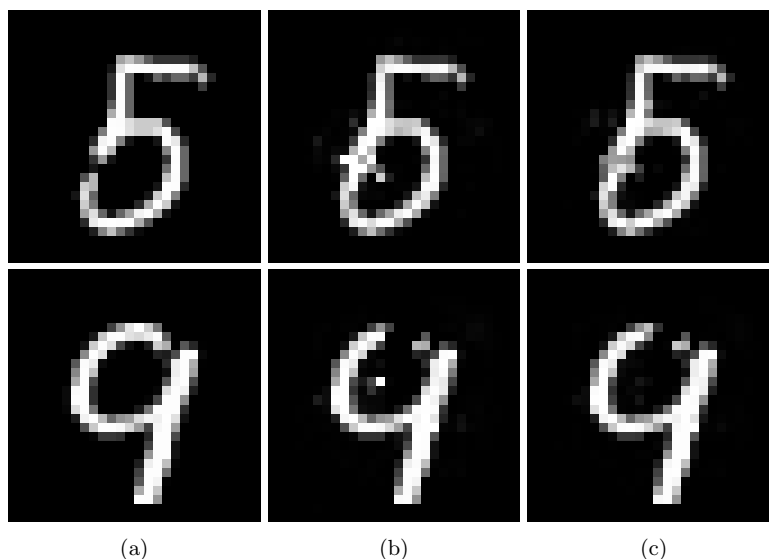


Figure 6.3: Visual comparison of two counterfactuals produced under the same setting as in [Schut et al., 2021] for (b) an Ensemble model, and (c) a BNN. Original inputs are shown in (a). The top counterfactual is for an original input of 5, with a target of 6 and the lower counterfactual is for an original input of 9 with target class 4.

the context of BNNs as well as a framework for computing them in practice. We reported results on five commonly used datasets from the CFX literature and compared the performance of our method against two baselines: MLPs and MLP ensembles. We have shown that BNNs often produce cheaper, more robust, and more plausible explanations. We observe that some state-of-the-art metrics appear to exhibit trade-off behaviour in all models. Notably, obtaining highly robust explanations is observed to be more costly, confirming previous observations made in the context of deterministic models [Jiang et al., 2024].

This work opens up several interesting avenues for future work. First, in this chapter, we have explored CFXs produced for BNNs by solving Equation (6.4) without applying additional regularisations to promote robustness or plausibility as sometimes used in the literature [Karimi et al., 2023, Jiang et al., 2024]. It would be interesting to see how our results hold up in such a setting and we

leave this investigation for future work. Furthermore, we consider only HMC-based Bayesian inference algorithms and a single network architecture for ease of comparison. We leave exploring CFX for BNNs produced by less precise inference algorithms and varying architectures as points for future work. Finally, we believe it would be interesting to conduct user studies to evaluate whether CFXs generated for BNNs are better suited for communicating uncertainty to lay users when compared to explanations generated for non-probabilistic models.

Chapter 7

Conclusions

This thesis has presented several contributions towards addressing the critical challenges of safety, robustness, and interpretability in deep neural networks. As AI systems are increasingly integrated into safety-critical applications, from autonomous vehicles to medical diagnostics, ensuring their reliability and trustworthiness has never been more important. Through this work, we have advanced the state-of-the-art in neural network verification, robustness training, and explainability across both deterministic and probabilistic settings.

7.1 Summary of Contributions

Throughout this thesis, we have made four key contributions that advance the field of safe and trustworthy AI:

In Chapter 3, we introduced a novel piecewise-linear approximation technique for verifying neural networks against geometric transformations. Unlike previous approaches that relied on loose linear or interval bounds, our method captures the non-convex nature of geometric perturbations more precisely. The experimental results demonstrated that our approach produces bounds up to

30% tighter than the previous state-of-the-art, enabling verification of up to 32% more cases in our benchmarks. This work is particularly significant for computer vision applications where geometric robustness against rotations, translations, and other spatial transformations represents more realistic threat models than traditional ℓ_p -norm bounded noise.

Chapter 4 presented a novel application of adversarial training to improve out-of-distribution performance in LiDAR-based 3D object detection systems. By developing a computationally efficient initialisation scheme inspired by fog physics and combining it with targeted adversarial training, we achieved up to 6% improvement in bounding box precision under adverse weather conditions compared to expensive simulation-based approaches. This contribution demonstrates that adversarial training techniques can be effectively adapted beyond their traditional security applications to address real-world distribution shifts in autonomous driving scenarios.

In Chapter 5, we developed two algorithms, Pure Iterative Expansion (PIE) and Gradient-guided Iterative Expansion (GIE), that significantly advance the state-of-the-art in probabilistic verification. Our approach dynamically adjusts the size and shape of verification regions based on the local properties of the parameter posterior, leading to probabilistic robustness certificates that are up to 40% tighter than previous methods. Importantly, our algorithms automatically adapt to different network architectures without requiring extensive hyperparameter tuning, making them more practical for real-world deployment.

Chapter 6 introduced the first formal framework for generating counterfactual explanations for Bayesian Neural Networks. By leveraging the uncertainty quantification inherent in BNNs, we demonstrated that probabilistic counterfactuals exhibit superior plausibility and robustness compared to their deterministic counterparts. Our extensive evaluation across multiple datasets showed that

BNN-based counterfactuals outperformed both standard MLPs and ensemble methods in 12 out of 15 head-to-head comparisons, suggesting that probabilistic models naturally encode properties beneficial for generating meaningful explanations.

7.2 Broader Impact and Limitations

The research presented in this thesis is founded on the premise that AI safety must evolve by adapting existing methodologies to support novel model architectures and realistic safety specifications. Specifically, this work extends safety methods to BNNs - an architecture that has remained relatively under-explored in the verification community - while also contributing to improvements in the semantic relevance of verification specifications. As the rate of AI development continues its exponential trajectory, moving from specialised tasks to a more general functionality, it is critical that safety research produces methods that are both architecture-agnostic and directly transferable to real-world environmental conditions. It is the author's hope that these contributions serve as a foundational step toward building truly dependable AI systems.

However, despite these advancements, this work represents a stepping stone rather than a complete solution. The field continues to be constrained by scalability challenges, particularly within formal verification. While this thesis contributes toward more efficient verification, it does not integrate certified training, a pervasive strategy for improving model-verifier compatibility, and as a result, empirical evaluations remain focused on small-to-medium-scale benchmarks. Furthermore, while this work independently explores explainability, verification, and robustness, it does not explore fusion safety criteria to unify these approaches. There is a clear logical interdependence between these domains. For example, a system supported by counterfactual explanations may permit

more flexible OOD guarantees, as a human practitioner can better adjudicate edge-case failures. The exploration of such multi-modal safety certificates remains an avenue for future research.

7.3 Future Work

While this thesis has made significant strides in neural network safety and interpretability, several promising avenues for future research emerge from our work:

7.3.1 Scaling Verification to Modern Architectures

Although our geometric verification approach significantly improves upon previous methods, current verification techniques still struggle with large-scale models such as transformers and modern convolutional architectures. Future work should explore hierarchical verification approaches that can leverage model structure, as well as investigating how our piecewise-linear approximation techniques might be extended to other non-convex input specifications beyond geometric transformations. Moreover, we explored constraints limited to two piecewise segments. Future work could explore additional piecewise segments and explore the combination of such constraints with multiple verifier calls for handling highly non-linear queries.

7.3.2 Unified Framework for Natural Perturbations

Our work on weather-based OOD generalisation via adversarial training opens the door to a more general framework for handling natural distribution shifts. Future research could explore how to automatically identify and generate training perturbations that correspond to real-world variations, potentially combin-

ing physics-based simulations with learned adversarial patterns. Additionally, investigating how these techniques transfer across different sensor modalities, e.g., from LiDAR to camera-based and multimodal systems, would be valuable.

7.3.3 Verification-Aware Training

An exciting direction is the integration of verification objectives directly into the training process, often called certified training, for both deterministic and Bayesian neural networks. By encouraging networks to be more easily verifiable during training, we can achieve models that are both highly accurate and amenable to formal analysis. Although there is prominent work in this direction for deterministic settings, it remains unclear how such a training scheme could promote probabilistically certifiable models. Furthermore, the majority of work in this direction focuses on simple threat models such as norm-based and kernel-based input sets. Extending these training regimes to include more complex threat models such as those discussed in Chapter 3 is left to future work.

7.3.4 Richer Explanation Frameworks

While our work on probabilistic counterfactuals demonstrates clear advantages, there remains significant room for developing more sophisticated explanation methods that leverage uncertainty. Future work could explore hierarchical explanations that provide different levels of detail based on the model's confidence, or investigate how counterfactual explanations can be combined with other interpretability techniques to provide more comprehensive model understanding.

7.3.5 Cross-Domain Safety Metrics

As AI systems are deployed across increasingly diverse domains, there is a need for safety metrics that can meaningfully compare and aggregate different types of guarantees. Future research could work towards unified frameworks that combine verification bounds, robustness certificates, and explainability metrics into holistic safety assessments.

7.4 Closing Remarks

The journey towards safe and trustworthy AI is far from complete, but this thesis has contributed several important stepping stones. By developing tighter verification bounds, more effective robustness training methods, and principled approaches to probabilistic safety, we have expanded the toolkit available to researchers and practitioners working to deploy AI systems responsibly. Most importantly, this work highlights the value of considering safety from multiple perspectives. Formal verification provides mathematical guarantees but can be conservative; adversarial training improves empirical robustness but lacks formal assurances; explainability techniques offer human-understandable insights but may not capture all failure modes. It is through the combination and continued advancement of these complementary approaches that we will ultimately achieve AI systems worthy of the trust we place in them. As neural networks continue to grow in capability and application scope, the importance of the safety properties studied in this thesis will only increase. We hope that the methods and insights presented here will contribute to a future where AI systems are not only powerful and accurate, but also verifiably safe, robust to real-world conditions, and transparent in their decision-making processes.

Bibliography

Steven Adams, Andrea Patane, Morteza Lahijanian, and Luca Laurenti. BNN-DP: robustness certification of bayesian neural networks via dynamic programming. In *International conference on machine learning, ICML*, pages 1613–1622. PMLR, 2023. [66](#)

Patrick Altmeyer, Mojtaba Farmanbar, Arie van Deursen, and Cynthia C. S. Liem. Faithful model explanations through energy-constrained conformal counterfactuals. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan, editors, *AAAI Conference on Artificial Intelligence*, pages 10829–10837. AAAI Press, 2024. [70](#), [143](#)

André Artelt, Valerie Vaquet, Riza Velioğlu, Fabian Hinder, Johannes Brinkrolf, Malte Schilling, and Barbara Hammer. Evaluating robustness of counterfactual explanations. In *IEEE Symposium Series on Computational Intelligence, SSCI*, pages 1–9. IEEE, 2021. [69](#), [70](#), [143](#)

M. Balunović, M. Baader, G. Singh, T. Gehr, and M. Vechev. Certifying geometric robustness of neural networks. In *Advances in Neural Information Processing Systems, NeurIPS*. Curran, 2019. [14](#), [63](#), [64](#), [73](#), [75](#), [76](#), [84](#), [87](#), [88](#), [89](#), [90](#), [91](#), [93](#), [95](#)

Solon Barocas, Andrew D. Selbst, and Manish Raghavan. The hidden assump-

- tions behind counterfactual explanations and principal reasons. In *Conference on Fairness, Accountability, and Transparency, FAT**, pages 80–89, 2020. 68
- Ben Batten and Alessio Lomuscio. Improving weather-based ood generalisation in lidar-based object detection models via adversarial training. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR) Workshops*, pages 4321–4329, June 2025. 98
- Ben Batten, Yang Zheng, Alessandro De Palma, Panagiotis Kouvaros, and Alessio Lomuscio. Verification of geometric robustness of neural networks via piecewise linear approximation and lipschitz optimisation. In *European Conference on Artificial Intelligence*, 2024. URL <https://api.semanticscholar.org/CorpusID:271947205>. 72
- Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade: Second edition*, pages 437–478. Springer, 2012. 37
- James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012. 37
- Leonard Berrada, Sumanth Dathathri, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Jonathan Uesato, Sven Gowal, and M. Pawan Kumar. Make sure you’re unsure: A framework for verifying probabilistic specifications. In *Advances in Neural Information Processing Systems, NeurIPS*, pages 11136–11147, 2021. 65, 66, 67
- Mario Bijelic, Tobias Gruber, Fahim Mannan, Florian Kraus, Werner Ritter, Klaus C. J. Dietmayer, and Felix Heide. Seeing through fog without seeing fog: Deep multimodal sensor fusion in unseen adverse weather. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages

- 11679–11689, 2019. URL <https://api.semanticscholar.org/CorpusID:211069740>. 97, 106, 108
- Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006. 34, 39, 40, 42
- David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017. 42
- Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *International conference on machine learning, ICML, ICML’15*, pages 1613–1622. JMLR.org, 2015. 42, 130
- Luca Bortolussi, Ginevra Carbone, Luca Laurenti, Andrea Patane, Guido Sanguinetti, and Matthew Wicker. On the robustness of bayesian neural networks to adversarial attacks, 2022. 125
- E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, and R. Misener. Efficient verification of relu-based neural networks via dependency analysis. In *AAAI Conference on Artificial Intelligence*, volume 34, pages 3291–3299, 2020a. 62, 88, 89
- Elena Botoeva, Panagiotis Kouvaros, Jan Kronqvist, Alessio Lomuscio, and Ruth Misener. Efficient verification of relu-based neural networks via dependency analysis. In *AAAI Conference on Artificial Intelligence*, 2020b. 19
- Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 93–104. Association for Computing Machinery, 2000. 143

- Christopher Brix, Mark Niklas Müller, Stanley Bak, Taylor T. Johnson, and Changliu Liu. First three years of the international verification of neural networks competition (vnn-comp), 2023. URL <https://arxiv.org/abs/2301.05815>. 19
- Christopher Brix, Stanley Bak, Taylor T. Johnson, and Haoze Wu. The fifth international verification of neural networks competition (vnn-comp 2024): Summary and results, 2024. URL <https://arxiv.org/abs/2412.19985>. 19
- Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of Markov Chain Monte Carlo*. CRC press, 2011. 43
- Robin A Brown, Edward Schmerling, Navid Azizan, and Marco Pavone. A unified view of sdp-based neural network verification through completely positive programming. In *International conference on artificial intelligence and statistics*, pages 9334–9355. PMLR, 2022. 59
- R. Bunel, I. Turkaslan, P. Torr, M P. Kumar, J. Lu, and P. Kohli. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21, 2020a. 60, 63
- Rudy Bunel, Alessandro De Palma, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip HS Torr, and M Pawan Kumar. Lagrangian decomposition for neural network verification. In *Conference on Uncertainty in Artificial Intelligence*, 2020b. 63
- Ruth M. J. Byrne. Counterfactuals in explainable artificial intelligence (XAI): evidence from human reasoning. In *International Joint Conference on Artificial Intelligence, IJCAI*, pages 6276–6282, 2019. 68
- Ginevra Carbone, Matthew Wicker, Luca Laurenti, Andrea Patané, Luca Bertolussi, and Guido Sanguinetti. Robustness of bayesian neural networks to

- gradient-based attacks. In *Advances in Neural Information Processing Systems, NeurIPS*, 2020. 65
- 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*, 2019a. 65
- Luca Cardelli, Marta Kwiatkowska, Luca Laurenti, and Andrea Patane. Robustness guarantees for bayesian inference with gaussian processes. In *AAAI Conference on Artificial Intelligence*, 2019b. 65, 66
- Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. Ieee, 2017a. 45
- Nicholas Carlini and David A. Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, 2017b. 138
- Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999. 33
- Alessandro De Palma, Harkirat Singh Behl, Rudy Bunel, Philip H. S. Torr, and M. Pawan Kumar. Scaling the convex barrier with active sets. In *International Conference on Learning Representations*, 2021. 63
- Terrance Devries and Graham W. Taylor. Learning confidence for out-of-distribution detection in neural networks. *ArXiv*, abs/1802.04865, 2018. URL <https://api.semanticscholar.org/CorpusID:3271220>. 48
- Amit Dhurandhar, Pin-Yu Chen, Ronny Luss, Chun-Chen Tu, Pai-Shun Ting, Karthikeyan Shanmugam, and Payel Das. Explanations based on the missing:

- Towards contrastive explanations with pertinent negatives. In *Advances in Neural Information Processing Systems, NeurIPS*, pages 590–601, 2018. 70
- Choubo Ding and Guansong Pang. Improving out-of-distribution detection with disentangled foreground and background features, 2024. URL <https://arxiv.org/abs/2303.08727>. 48
- Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016. 40
- Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. <http://archive.ics.uci.edu/ml>. 135, 140
- Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 503:92–108, 2022. ISSN 0925-2312. doi: <https://doi.org/10.1016/j.neucom.2022.06.111>. URL <https://www.sciencedirect.com/science/article/pii/S0925231222008426>. 26
- R. Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *ATVA17*, volume 10482, pages 269–286. Springer, 2017. 18, 19, 49
- Lawrence Evans and Ronald Gariepy. *Measure Theory and Fine Properties of Functions, Revised Edition*. Chapman and Hall/CRC., 04 2015. ISBN 9780429161483. doi: 10.1201/b18333. 82
- Kelwin Fernandes, Pedro Vinagre, Paulo Cortez, and Pedro Sernadela. Online News Popularity. UCI Machine Learning Repository, 2015. <https://doi.org/10.24432/C5NS3V>. 135, 140
- Claudio Ferrari, Mark Niklas Mueller, Nikola Jovanović, and Martin Vechev. Complete verification via multi-neuron relaxation guided branch-and-bound. In *International Conference on Learning Representations*, 2022. 63

- Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International conference on machine learning, ICML*, pages 1050–1059. PMLR, 2016. 39
- Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, 2018. 63
- Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE, 2012. 17
- I J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014. 17, 44, 45, 46
- Ian Goodfellow. *Deep learning*, volume 196. MIT press, 2016. 34, 36
- Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008. 35
- Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of field robotics*, 37(3):362–386, 2020. 17
- Riccardo Guidotti. Counterfactual explanations and how to find them: literature review and benchmarking. *Data Min. Knowl. Discov.*, 38(5):2770–2824, 2024. 68
- Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning, ICML*, pages 1321–1330. PMLR, 2017. 39

- Martin Hahner, Christos Sakaridis, Dengxin Dai, and Luc Van Gool. Fog simulation on real lidar point clouds for 3d object detection in adverse weather. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 15263–15272, 2021. URL <https://api.semanticscholar.org/CorpusID:236976003>. 11, 14, 97, 98, 99, 100, 102, 103, 104, 107, 109, 111, 113
- Martin Hahner, Christos Sakaridis, Mario Bijelic, Felix Heide, Fisher Yu, Dengxin Dai, and Luc Van Gool. Lidar snowfall simulation for robust 3d object detection. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16343–16353, 2022. URL <https://api.semanticscholar.org/CorpusID:247778401>. 97
- Trevor Hastie. The elements of statistical learning: data mining, inference, and prediction, 2009. 31, 40
- K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR16*, pages 770–778, 2016. 29
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 38
- Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *ArXiv*, abs/1610.02136, 2016. URL <https://api.semanticscholar.org/CorpusID:13046179>. 48
- Dan Hendrycks, Mantas Mazeika, and Thomas Dietterich. Deep anomaly detection with outlier exposure. *Proceedings of the International Conference on Learning Representations*, 2019. 48
- P. Henriksen and A. Lomuscio. Deepsplit: An efficient splitting method for

- neural network verification via indirect effect analysis. In *International Joint Conference on Artificial Intelligence, IJCAI*, 2021. 56, 63
- Patrick Henriksen and Alessio R. Lomuscio. Efficient neural network verification via adaptive refinement and adversarial search. In *24th European Conference on Artificial Intelligence, ECAI*, 2020. 56
- Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006. 30
- Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970. 34
- Mark Hopkins, Erik Reeber, George Forman, and Jaap Suermondt. Spambase. UCI Machine Learning Repository, 1999. <https://doi.org/10.24432/C53G6X>. 135, 140
- Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. ISSN 0893-6080. doi: [https://doi.org/10.1016/0893-6080\(89\)90020-8](https://doi.org/10.1016/0893-6080(89)90020-8). URL <https://www.sciencedirect.com/science/article/pii/0893608089900208>. 26
- Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine learning*, 110(3):457–506, 2021. 38
- Junqi Jiang, Francesco Leofante, Antonio Rago, and Francesca Toni. Robust counterfactual explanations in machine learning: A survey. In *International Joint Conference on Artificial Intelligence, IJCAI*, pages 8086–8094. International Joint Conferences on Artificial Intelligence Organization, IJCAI, 2024. 69, 149

- M. Jordan and A. G. Dimakis. Exactly computing the local lipschitz constant of relu networks. *arXiv preprint arXiv:2003.01219*, 2020. 82
- Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017. 26
- Amir-Hossein Karimi, Gilles Barthe, Bernhard Schölkopf, and Isabel Valera. A survey of algorithmic recourse: Contrastive explanations and consequential recommendations. *ACM Comput. Surv.*, 55(5):95:1–95:29, 2023. 68, 70, 149
- G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification, CAV*, pages 97–117, 2017. 19, 49, 67
- Eoin M. Kenny and Mark T. Keane. On generating plausible counterfactual and semi-factual explanations for deep learning. In *AAAI Conference on Artificial Intelligence*, pages 11575–11585. AAAI Press, 2021. 68
- Velat Kilic, Deepti Hegde, Vishwanath A. Sindagi, A. Brinton Cooper, Mark A. Foster, and Vishal M. Patel. Lidar light scattering augmentation (lisa): Physics-based simulation of adverse weather conditions for 3d object detection. *ArXiv*, abs/2107.07004, 2021. URL <https://api.semanticscholar.org/CorpusID:235899299>. 97
- Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations, ICLR*, 2015. 130

- P. Kouvaros and A. Lomuscio. Formal verification of cnn-based perception systems. *arXiv preprint arXiv:1811.11373*, 2018. 63, 75
- Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. 124
- Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012. 89
- Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951. 42
- Thibault Laugel, Marie-Jeanne Lesot, Christophe Marsala, Xavier Renard, and Marcin Detyniecki. The dangers of post-hoc interpretability: Unjustified counterfactual explanations. In Sarit Kraus, editor, *International Joint Conference on Artificial Intelligence, IJCAI*, pages 2801–2807. ijcai.org, 2019. 69
- Mathias Lechner, Đorđe Žikelić, Krishnendu Chatterjee, and Thomas Henzinger. Infinite time horizon safety of bayesian neural networks. In *Advances in Neural Information Processing Systems, NeurIPS*, volume 34, pages 10171–10185. Curran Associates, Inc., 2021. 66
- Y. LeCun, C. Cortes, and C. J. Burges. MNIST handwritten digit database, 2010. 89, 124, 135, 140
- Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, 1998. 28
- Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521 (7553):436–444, 2015. 16, 30

- Francesco Leofante and Nico Potyka. Promoting counterfactual robustness through diversity. In *AAAI Conference on Artificial Intelligence*, pages 21322–21330. AAAI Press, 2024. 70, 143
- Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017. 17
- Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher Strong, Clark Barrett, Mykel J Kochenderfer, et al. Algorithms for verifying deep neural networks. *Foundations and Trends® in Optimization*, 4(3-4):244–404, 2021. 51
- Weitang Liu, Xiaoyun Wang, John Douglas Owens, and Yixuan Li. Energy-based out-of-distribution detection. *ArXiv*, abs/2010.03759, 2020. URL <https://api.semanticscholar.org/CorpusID:222208700>. 48
- Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward relu neural networks, 2017. 49
- 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*. OpenReview.net, 2018. 45, 46
- Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943. 24
- Gustav Mie. Beiträge zur optik trüber medien, speziell kolloidaler met-

- allösungen. *Annalen der Physik*, 330(3):377–445, 1908. doi: 10.1002/andp.19083300302. 97
- Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artif. Intell.*, 267:1–38, 2019. 68
- Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T Dudley. Deep learning for healthcare: review, opportunities and challenges. *Briefings in bioinformatics*, 19(6):1236–1246, 2018. 17
- Kiarash Mohammadi, Amir-Hossein Karimi, Gilles Barthe, and Isabel Valera. Scaling guarantees for nearest counterfactual explanations. In *AAAI/ACM Conference on AI, Ethics, and Society*, pages 177–187. ACM, 2021. 69
- Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012a. 39, 41
- Radford M Neal. Mcmc using hamiltonian dynamics. *arXiv preprint arXiv:1206.1901*, 2012b. 43
- Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 372–387, 2016. 44, 147
- K. Pei, Y. Cao, S. Yang, and S. Jana. Towards practical verification of machine learning: The case of computer vision systems. *CoRR*, abs/1712.01785, 2017. 63
- Luca Pulina and Armando Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *Computer Aided Verification, CAV*, volume 6174 of *Lecture Notes in Computer Science*, pages 243–257. Springer, 2010. 49

- A. Raghunathan, J. Steinhardt, and P. Liang. Semidefinite relaxations for certifying robustness to adversarial examples. *arXiv preprint arXiv:1811.01057*, 2018. [59](#)
- R. Rasshofer, M. Spies, and H. Spies. Influences of weather phenomena on automotive laser radar systems. *Advances in Radio Science*, 9, 07 2011. doi: 10.5194/ars-9-49-2011. [99](#), [103](#), [104](#)
- David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986. [24](#), [35](#)
- Lisa Schut, Oscar Key, Rory McGrath, Luca Costabello, Bogdan Sacaleanu, Medb Corcoran, and Yarin Gal. Generating interpretable counterfactual explanations by implicit minimisation of epistemic and aleatoric uncertainties. In Arindam Banerjee and Kenji Fukumizu, editors, *International Conference on Artificial Intelligence and Statistics, AISTATS*, volume 130 of *Proceedings of Machine Learning Research*, pages 1756–1764. PMLR, 2021. [13](#), [15](#), [140](#), [147](#), [148](#), [149](#)
- Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John P. Dickerson, Christoph Studer, Larry S. Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! In *Advances in Neural Information Processing Systems, NeurIPS*, 2019. [18](#), [47](#)
- Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointcnn: 3d object proposal generation and detection from point cloud. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–779, 2019. doi: 10.1109/CVPR.2019.00086. [14](#), [106](#), [107](#), [108](#)
- G. Singh, T. Gehr, M. Püschel, and M. Vechev. An abstract domain for certi-

- fying neural networks. In *PACMPL19*, volume 3, pages 1–30, 2019a. [11](#), [63](#), [73](#), [74](#), [75](#), [93](#), [95](#)
- Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems, NeurIPS*, 2018. [54](#), [56](#), [63](#)
- Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 2019b. [56](#), [59](#), [63](#)
- Aman Sinha, Hongseok Namkoong, and John C. Duchi. Certifiable distributional robustness with principled adversarial training. In *International Conference on Learning Representations, ICLR*, 2017. [67](#)
- Dylan Slack, Anna Hilgard, Himabindu Lakkaraju, and Sameer Singh. Counterfactual explanations can be manipulated. In *Advances in Neural Information Processing Systems, NeurIPS*, pages 62–75, 2021. [69](#), [70](#), [143](#)
- Jack Smith, J. Everhart, W. Dickson, W. Knowler, and Richard Johannes. Using the adap learning algorithm to forecast the onset of diabetes mellitus. *Proceedings - Annual Symposium on Computer Applications in Medical Care*, 10, 11 1988. [135](#), [140](#)
- Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014. [35](#)
- Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58(1):267–288, 1996. [34](#)

- V. Tjeng, K. Xiao, and R. Tedrake. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017. [63](#)
- Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. [17](#)
- Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian J. Goodfellow, Dan Boneh, and Patrick D. McDaniel. Ensemble adversarial training: Attacks and defenses. In *International Conference on Learning Representations, ICLR*. OpenReview.net, 2018. [18](#)
- H. Tran, S. Bak, W. Xiang, and T. Johnson. Verification of deep convolutional neural networks using imagestars. In *International Conference on Computer Aided Verification*, pages 18–42. Springer, 2020. [18](#)
- Berk Ustun, Alexander Spangher, and Yang Liu. Actionable recourse in linear classification. In *Conference on Fairness, Accountability, and Transparency, FAT**, pages 10–19. ACM, 2019. [70](#)
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems, NeurIPS*, volume 30, 2017. [17](#)
- VNN-COMP. Verification of neural networks competition, 2020. [89](#)
- Sandra Wachter, Brent D. Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harv. JL & Tech.*, 31:841, 2017. [69](#), [70](#)

- S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems, NeurIPS*, pages 6369–6379, 2018a. [52](#), [56](#)
- Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symposium, USENIX*, 2018b. [58](#)
- 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 complete and incomplete neural network verification. In *Advances in Neural Information Processing Systems, NeurIPS*, 2021. [19](#), [63](#)
- Matthew Wicker, Luca Laurenti, Andrea Patane, and Marta Kwiatkowska. Probabilistic safety for bayesian neural networks. In *Conference on Uncertainty in Artificial Intelligence*, 2020. [15](#), [65](#), [66](#), [116](#), [117](#), [118](#), [123](#), [124](#), [125](#), [126](#), [127](#), [130](#), [131](#)
- Matthew Wicker, Andrea Patane, Luca Laurenti, and Marta Kwiatkowska. Adversarial robustness certification for bayesian neural networks, 2023. [66](#)
- Hjalmar Wijk, Benjie Wang, and Marta Kwiatkowska. Robustness guarantees for credal bayesian networks via constraint relaxation over probabilistic circuits. In *International Joint Conference on Artificial Intelligence, IJCAI*, pages 4885–4892, 2022. [66](#)
- Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In *International Conference on Learning Representations*, 2021. [63](#)
- Mingyang Yi, Lu Hou, Jiacheng Sun, Lifeng Shang, Xin Jiang, Qun Liu,

- and Zhiming Ma. Improved ood generalization via adversarial training and pretraining. In Marina Meila and Tong Zhang, editors, *International conference on machine learning, ICML*, volume 139 of *Proceedings of Machine Learning Research*, pages 11987–11997. PMLR, 18–24 Jul 2021. URL <https://proceedings.mlr.press/v139/yi21a.html>. 48, 97
- Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *ieee Computational intelligence magazine*, 13(3):55–75, 2018. 17
- Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems, NeurIPS*, 2018. 52, 56
- Huan Zhang, Shiqi Wang, Kaidi Xu, Linyi Li, Bo Li, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. General cutting planes for bound-propagation-based neural network verification. In *Advances in Neural Information Processing Systems, NeurIPS*, 2022. 63
- Xin Zou and Weiwei Liu. On the adversarial robustness of out-of-distribution generalization models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 68908–68938. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/d9888cc7baa04c2e44e8115588133515-Paper-Conference.pdf. 48