Adversarial Robustness of Bayesian Neural Networks

Matthew Wicker

Wolfson College

University of Oxford

A thesis submitted for the Degree of Doctor of Philosophy

Michaelmas 2021

Abstract

This thesis puts forward methods for computing local robustness of probabilistic neural networks, specifically those resulting from Bayesian inference. In theory, applying Bayesian inference to the learning of neural network parameters carries the promise of solving many practically vexing problems that arise under the frequentist learning paradigm. In particular, Bayesian learning allows for principled architecture comparison and selection, the encoding of prior knowledge, and calibration of predictive uncertainties. Recent studies have shown that Bayesian learning can lead to more adversarially robust predictors. Though theoretically this is the case, and empirically has been shown in particular instances, anecdotal evidence of heightened robustness does not provide sufficient assurances for those who wish to deploy Bayesian deep learning in a safety-critical context. While methods exist for arriving at guarantees of robustness for deterministic neural networks, the probabilistic nature of Bayesian neural network weights renders these methods inoperable.

In this thesis, we investigate concepts of robustness for Bayesian neural networks, which allow for robustness guarantees which consider both the stochasticity of the model as well as the model's decision. We provide methodologies which compute these quantities for a given Bayesian neural network with either *a priori* statistical guarantees on the precision of our estimates, or probabilistic upper and lower bounds which are provably sound. Finally, we consider robustness as a primary desideratum in the Bayesian inference of neural network parameters and demonstrate how to modify the likelihood in order to infer a posterior distribution with favorable robustness properties. The modification of the likelihood make our method transparent to the approximate inference technique for Bayesian neural networks.

We assess the practical applicability of our proposed methodology using Bayesian neural networks trained on several real-world datasets includ-

ing airborne collision avoidance and traffic sign recognition. Additionally, we assess the robustness of Bayesian posterior distributions approximately inferred using five different approximate inference methods. We find that our methodology provides the first provable robustness guarantees for Bayesian neural networks, thus enabling their deployment in safety-critical scenarios. Further, our proposed methodology for robust Bayesian inference of neural network parameters enables us infer posterior distributions which have greatly heightened provable robustness even on full-color images.

Acknowledgements

There are many people without whom this thesis would not exist. First and foremost, I express my deepest gratitude to my supervisor and mentor Marta Kwiatkowska. From providing my first exposure to research as an undergraduate, to the past three years of guidance, support, and utmost patience, not only would this thesis not exist without her support but my passion for research would not be what it is today.

The work in this thesis is also particularly indebted to my closest collaborators and good friends Luca Laurenti and Andrea Patanè. Their kindness, friendship, and insightful discussions made my DPhil process a thoroughly enjoyable one, and their constant curiosity throughout this quest pushed me to understand and explore research in new ways.

I must also thank the many collaborators and teachers who greatly shaped my academic career during my time at the University of Oxford and prior. To my collaborators Alessandro Abate, Sandy Anderson, Luca Bortolussi, Liming Cai, Ginevra Carbone, Luca Cardelli, Yarin Gal, Xiaowei Huang, Artem Kaznatcheev, Rhiannon Michelemore, Nicola Paoletti, Wenjie Ruan, Guido Sanguinetti, and Min Wu, you were all truly a pleasure to work with. Many thanks to the teachers and administrators for their valuable guidance: Lee Taylor, Jessica Hunt, Avril Lethbridge, and Anita Hancox.

Thanks is also due to my Mom and Dad for their unrelenting love and support in every way imaginable. To my brothers Daniel and Jacob for always being there. Thanks to Aline Wicker and Gloria Perlstein. And to my close friends Julia Weil, Tom Maltas, Catherine Higgins, Katie Henley, Ashevik Vishwanathen, and Yaz Açıkgöz.

Some of the outcomes reported in this thesis are aligned with the FUN2MODEL (grant agreement No. 834115) project led by Marta Kwiatkowska, funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Classical computer science concerns itself with the creation of programs that solve a given problem. Correspondingly, classical program verification is the task of ensuring (typically via formal proof) that a given program correctly solves a given problem in every instance [6]. In recent years, computer scientists have broadened the class of problems that they would like to solve to tasks which are either too complex or poorly defined to be handled by the classical programming paradigm. Where programs can no longer be devised by humans, they can be learned by example [57]. As learned solutions become much better than their hand-coded counterparts, the domains in which they are employed become more complex. It is no surprise that the domains in which learning has the greatest potential impact also carry the greatest risk of harm [1, 10]. Learned solutions aimed at such tasks, including medical diagnosis and self-driving cars, must be guaranteed to be safe before they are deployed and trusted by the general public. Unfortunately, the same hindrances to writing classical programs for these tasks hinders their formal verification [79]. Further, initial attempts to check basic stability of learned solutions revealed their remarkable fragility [136]. This fragility manifests itself in the form of over-confident, incorrect predictions, which can be produced for nearly every input given to learning algorithm. Thus, if we would like to capitalize on the bright-future of machine learning algorithms, we must ensure they are *safe* before deployment.

In this thesis, we will focus on the most popular and powerful learning algorithm to date: deep neural networks.[1] Neural networks are powerful function approximators, which hold the promise of continued and significant contribution to advancement of state-of-the-art performance in a wide range of tasks. Neural networks have already achieved remarkably strong performance in safety-critical domains such as medical

[1]The definitions and philosophy behind our primary definitions will hold for any probabilistic classifier; however, methodology herein will focus on neural networks.

diagnosis and pathology as well as control and planning. Yet, a primary roadblock to the adoption of neural networks in such domains is the lack of interpretability and reliability of their predictions [1].

We will use two primary vulnerabilities to motivate the study of robustness of Bayesian neural networks (BNNs), which are neural networks with a distribution over their parameters inferred by Bayes' rule. The first potential vulnerability is deterministic neural networks' (DNNs') lack of calibrated uncertainty, that is, *knowing what they don't know* [81]. This is a particular challenge when deterministic neural networks are used for inference on data points that statistically deviate from the training data. In this case, the DNN can often make highly confident, incorrect predictions, which can lead to bad actions if relied upon [104]. The second vulnerability is that of adversarial examples [136]. An adversarial example is an input which is crafted to be indistinguishable from a naturally occurring input, but which causes the neural network to make a wrong classification or an unsafe change in output. In the case of medical diagnosis, this could be predicting that a patient has cancer due to slight changes in the hue of a pathology slide, or in autonomous navigation could represent a large change in the predicted steering angle based on slight changes to lighting conditions [105]. Adversarial attacks have been proven to be a safety concern not just in image classification [58], but also in audio recognition [163], malware identification [126], and natural language processing [41]. These pose a great security risk to safety and security-critical applications. Naturally, proving safety to adversarial examples is a prerequisite to deployment of any neural network in a safety-critical setting.

Proving safety of neural network predictions has been an important and active area of research over the past few years and has made great strides in being able to efficiently prove the non-existence of adversarial examples [79, 22, 152]. While this satisfies one of our desires (lack of adversarial examples), deterministic neural networks still provide very little in the way of calibrated uncertainty. In particular, given a deterministic neural network and an input we would like to classify, it is typically the case that, if an adversarial example exists, then it is incorrectly classified with very high confidence [58]. This means that, based on the output, there is no way to reason about whether an input is potentially incorrect or corrupted. Moreover, there are works for deterministic neural networks which show that for many tasks the existence of adversarial examples is inevitable [47, 46] and further that robust deterministic learning is impossible [59]. While sound local verification (proving the non-existence of adversarial examples) is necessary for providing users assurances of

correct performance in particular instances, the Bayesian learning paradigm offers a systematic method for alleviating the concerns of these impossibility results at a more general level. By introducing calibrated uncertainty, Bayesian neural networks have been shown to be more robust to adversarial examples both theoretically and empirically, and can potentially weaken or defeat impossibility results for deterministic networks [53, 23, 7] Therefore, they seem like a natural and viable candidate for deployment in safety-critical scenarios in which proofs of safety and robustness are necessary.

Despite their many attractive qualities, Bayesian neural networks cannot be analysed straightforwardly with the techniques developed for deterministic neural networks [168]. The primary difference between Bayesian and deterministic networks is that the former has a posterior distribution over its parameter values. In order to verify robustness of such a model, one must find a way to perform the correctness analysis that is available for deterministic neural networks while taking into consideration the range or probable parameter values in a sound way. Doing so is a necessary prerequisite for the safe deployment of Bayesian neural networks in safety-critical scenarios.

In this thesis, we develop tools that allow us to leverage the advances in robustness quantification of deterministic neural networks in a Bayesian setting. In particular, we investigate two notions of robustness for Bayesian neural networks which allow practitioners to quantify the worst-case behavior of a given Bayesian neural network prior to its deployment. The first notion of robustness for Bayesian neural networks is the probabilistic robustness (defined in Chapter 4). This allows practitioners to understand the interaction between the model's inherent stochasticity and its adversarial robustness and can also be seen as a worst-case measure of uncertainty. The second notion of robustness is the Bayesian decision robustness. Apart from simply having a distribution over their weights, Bayesian neural networks are distinct from their deterministic counterparts in that we must reason about their predictive distributions and the risk or loss of an incorrect decision in order to make prediction. The decision robustness takes into account the decision-making process of the Bayesian model under consideration and allows us to certify that the correct decision is being issued even in the presence of an adversary. These definitions allow us to quantify the correctness of a Bayesian neural network probabilistically.

In addition to our study of novel notions of robustness for BNNs, we also establish practical procedures for computation of these robustness quantities such that practitioners have guarantees of worst-case estimates. While worst-case estimates on their

own can give us confidence in the correctness of our model, having guarantees on these estimates can not only provide further reassurance at deployment time, but can also be critical for regulators who may require concrete proof of a model's correctness. Here, we provide two forms of guarantees for our robustness quantities. The first is a statistical guarantee. This guarantee gives the practitioner or regulator *a priori* control of the error and confidence of our robustness estimates. With these guarentees, we can make statements of the form: we are 99% confident that the given BNN is robust w.r.t. provided specification with error at most 1%. The other form of guarantees we provide are probabilistic. These eliminate the confidence term from the statistical guarantees and allow us to make statements such as: the provided BNN's decision is robust to the given specification. While this is a much stronger statement than the statistical one, it is also much more difficult to compute and relies on more sophisticated techniques such as convex relaxation.

The final methodological contribution of this thesis considers not just quantifying the robustness of a given Bayesian neural network, but inferring a Bayesian posterior that takes adversarial inputs into consideration at training time. We develop a novel likelihood approach to adversarial training, which is distinct from the bootstrapping approaches that are classically taken. This allows us to treat adversarial robustness probabilistically during inference. Ultimately, our training approach is applicable to any approximate Bayesian inference method and allows for flexible definitions of robustness to be considered during inference.

The ability to quantify the level of local robustness of a Bayesian neural network with guarantees is not just of inherent interest to those who wish to deploy Bayesian neural networks in safety critical scenarios, but also should be of interest to practitioners who seek to develop better, more reliable neural networks. For example, the methods developed in this thesis can be directly leveraged to compare two posteriors (or indeed priors) and to reason about which ought to be selected where robust performance is a concern.

Further contributions of this thesis are empirical. In order to evaluate each of the methodologies outlined above, we have implemented a tool that performs approximate Bayesian inference for neural networks expressed in Tensorflow 2.0 or Keras and implements all of the robustness quantification methodology described in this thesis. Using this tool, we are able to infer approximate Bayesian posteriors on real-world datasets such as the vertical airborne collision avoidance (VCAS) dataset [77], a variety of regression benchmarks from the University of California Irvine machine learning repository (UCI datasets) [38], the MNIST handwritten digit-recognition

dataset (MNIST) [93], the CIFAR-10 image recognition dataset [86], and the German Traffic Sign Recognition Benchmark (GTSRB) [70]. The smallest of these containing only a couple dozen input features and the largest containing a few thousand. In Chapter 5, we show how our proposed methodology can evaluate the robustness of Bayesian neural networks deployed for image recognition in self-driving cars. In Chapter 6, we perform sound verification of a Bayesian neural network trained on an airborne collision avoidance system. Finally, in Chapter 7, we show that our methodology can infer certifiably robust approximate Bayesian posteriors for convolutional neural networks trained on the CIFAR-10 image dataset.[2]

In conjunction, these contributions allow those who wish to deploy Bayesian neural networks in safety-critical scenarios access to different kinds of worst-case robustness information about their network and gives them control over the level of guarantee that their application calls for. Further, our methodology not only quantifies the robustness of Bayesian neural networks, but also allows for the incorporation of robustness specifications into the inference procedure. Having access to these quantities allows for Bayesian neural networks to be deployed in safety-critical scenarios with assurances of correct performance.

1.1 Contributions

Below we detail an itemized list of the technical contributions of this thesis. In particular, we highlight novel methodological contributions which are presented formally in this thesis.

- We investigate two notions of adversarial robustness for Bayesian neural networks. First, we develop the notion of probabilistic model robustness which can intuitively be interpreted as the probability that a model drawn from a given (potentially approximate) Bayesian posterior is statistically robust to a provided specification. Further, we discuss the theoretical view of this quantity as a probabilistic measure of the predictive stability of the models included in our Bayesian posterior and the interpretation of this measure as a worst-case uncertainty.

- We define decision robustness for Bayesian neural networks as a direct analogue of the notions of robustness that are commonly used to provide assurances of

[2]Code for reproducing the results in this thesis can be found at: https://github.com/matthewwicker/AdversarialRobustnessOfBNNs

correctness for deterministic neural networks. Starting from the distinction between the role of a likelihood and loss function in the classical Bayesian setting, we establish that, if the posterior predictive quantity which minimizes the loss function for a given problem is the same (or sufficiently similar) for all inputs in a given input set, then the decision of the Bayesian neural network is robust. Proving the robustness of the Bayesian decision-theoretic quantity directly translates to proving the absence of adversarial examples for the provided Bayesian posterior. This is, to the best of our knowledge, the first time that anyone has verified the correctness of a Bayesian neural network decision in an adversarial setting, a necessary step in the deployment of Bayesian neural networks in safety-critical domains.

- We provide theoretical and practical perspectives on estimating both the probabilistic model robustness and the Bayesian decision robustness with *a priori* statistical guarantees on the error and confidence. Through the use of concentration inequalities and by leveraging advancements in statistical model checking, we are able to efficiently guarantee that the computed values corresponding to both definitions are statistically tight for a given Bayesian neural network and robustness specification. To this end, we show how this methodology can be used to evaluate Bayesian neural network robustness in regression and classification settings and for a variety of posteriors arising from different approximate inference techniques.

- We establish algorithms which provide sound bounds on both probabilistic model robustness and decision robustness of Bayesian neural networks. In particular, we leverage advances in convex relaxation of neural network computations in order to arrive at sound lower and upper bounds on the result of marginalization over the posterior predictive distribution. This in turn allows us to provide concrete proofs of correctness (e.g., the absence of adversarial examples) for Bayesian neural networks. We show how doing so can allow us to prove sufficient safety of Bayesian neural networks for deployment in airborne collision avoidance tasks and we further investigate the scalability of this algorithm to larger input domains and network architectures.

- We provide a principled probabilistic perspective on the incorporation of adversarial examples into the inference or approximate inference procedure of Bayesian neural networks. By developing a novel likelihood function which

takes into account a worst-case adversary, we show how to infer Bayesian posterior distributions with favorable adversarial robustness properties. Given that this methodology involves the development of a novel adversarial likelihood (and corresponding error model), our approach is compatible with all forms of approximate Bayesian inference for neural networks. Further, we show how different approximate inference methods perform when used in conjunction with our proposed likelihood. Ultimately, we find that our method leads to substantial gains in robust performance of the resulting posteriors.

- Finally, we discuss and highlight future works of interest for the sound quantification of robustness for Bayesian neural networks. We highlight the limitations of the methodology presented in this thesis as well as the remaining necessary requirements for developing a comprehensive understanding of Bayesian deep learning in an adversarial setting.

1.2 Thesis Organization

The thesis begins with a contextualization of our contribution with the relevant literature. We make an effort to survey some of the most important developments in the study of adversarial robustness and approximate Bayesian inference for neural network parameters as well as a brief summary of their overlap. Following this, we dive deeper into several select works and provide formal mathematical definitions that will lay the foundations for our contributions, which are made in Chapters 4, 5, 6, and 7. In Chapter 4 we introduce the shared problem formulation for the definitions of robustness that are computed with guarantees in subsequent chapters. In Chapter 5 we present the notion of statistical guarantees and then show how one can compute them for Bayesian neural networks. In Chapter 6, we provide a methodology to compute sound probabilistic upper and lower bounds on the definitions of interest. We follow this with Chapter 7, which presents a probabilistically principled incorporation of adversarial robustness into Bayesian learning for neural networks. Finally, in Chapter 8 we summarize the strengths and weaknesses of our proposed methodology, and summarise future work in robustness quantification for Bayesian neural networks.

1.3 Publications

This thesis is based on several works which were published throughout the thesis. In this section I detail my contribution to each of the works [24, 104, 150, 151, 158, 23].

Chapter 5 is largely based on the statistical guarantees work presented in [24] (which appeared in IJCAI 2019). I designed and implemented all of the experiments, as well performed the visualizations and analysis of the experiments. This involved implementing several Bayesian inference methods and adversarial robustness quantification methods in order to form systematic comparisons between approximate inference methods. The extension of this work to decision robustness that can be found in Chapter 5 was done independently and has not appeared in any published materials yet. The work presented in [24] was also extended in [104] (which appeared in ICRA 2020) to incorporate time-based loops, particularly in the setting of autonomous navigation. My contributions to this work were more on the theoretical side than that of [24], though I still designed all of the experiments and performed the visualizations. The driving idea for the work in Chapter 6 (originally published in [150], and appeared in UAI 2020) was my own. Through collaboration with others, we finalized the details involving interval and linear bound propagation. The driving idea, implementation and write up of [150] was my contribution; however, the proof of the theorems stated in the paper (and included in this thesis for completeness with appropriate credit given) are due to other authors. Following a similar pattern to the work of Chapter 5, [150] was extended to time-based control loops in [151] (which appeared in UAI 2021). The motivation, methodology, implementation, and initial proof sketches in this work was my own. The presentation of the algorithm and proof details in [151] are due to my co-authors. Finally, works whose methodology are not presented in this thesis but which contribute to the intuitions presented and the experimental evaluations are [158] (which appeared in AABI 2021) and [23] (which appeared in NeurIPS 2020). For both of these works, I developed my own approximate inference framework for learning neural networks parameters, which is based on Tensorflow 2.0. This framework is used throughout this thesis.

Chapter 2

Related Works

Contents

In this chapter we discuss works that help contextualize the contribution of this thesis. We provide interested readers with a summary of seminal works on top of which this thesis builds, concurrent works with the contributions of this thesis, and related works that are of interest to those who study local/adversarial robustness of Bayesian methods. Here, we describe the contributions of each work and their relation to this thesis. In the following chapter, we will introduce preliminaries and dive deeper into selected works on which the methodology of this thesis relies.

We begin by covering works in robustness, specifically adversarial attacks, defences, and verification. Then, we will cover concurrent and prior works which study these phenomena in the Bayesian setting.

2.1 Adversarial Robustness

The primary focus of this thesis lies at the intersection of adversarial robustness and Bayesian deep learning. In our first subsection we cover the seminal developments in adversarial robustness. In particular, we are interested in *local* adversarial robustness. That is, we are interested in the sensitivity of a neural networks predictions to small manipulations of a particular, given input. We end this section with a comparative discussion of local and adversarial robustness with other notions of robustness considered in machine learning, optimization, and statistics.

2.1.1 Adversarial Examples

Adversarial testing predates machine learning and probably computing systems in general. The central idea is to make the assumption that an attacker would like to realise a *worst-case* behavior of a deployed system. In machine learning this *worst-case* behavior is modeled as receiving some undesirable output (i.e., incorrect prediction). Adversarial testing of machine learning models has been conducted for at least the past decade [10]. Despite this, we are primarily interested in adversarial testing of neural networks, which was popularized in roughly 2013 with the publication of [136]. This work used a L-BFGS optimization method over the input space to find small *imperceptible* modifications to inputs which cause misclassifications. They find that they could achieve misclassification for nearly every input given to a state-of-the-art classifier. Later, [58] showed that a sophisticated second-order method was not necessary to find adversarial examples and that a single first-order gradient step would suffice. The attack in [58] was called the fast gradient sign method (FGSM). A series of works followed this which, refined the first-order method and setting of attacking neural networks, notably: [108, 89, 26, 101]. In [108, 89] the authors take what can be seen as an iterative version of the approach of [58] and notice heightened success in forcing misclassifications. In [26], the authors augment the first-order methods, typically used with momentum and explore different differential loss functions for the optimization. The systematic treatment of the optimization procedure for finding adversarial examples presented in [26], named Carlini and Wagner attacks (CW attacks), allowed this method to remain one of the most powerful ways to find misclassifications. In [101], the authors utilize a projected gradient descent methodology to attack inputs, which sees strong performance in the limit of the number of iterations. The aforementioned imperceptible changes were in the order of 1/255 (where pixel

values are scaled between 0 and 1). These minute changes were able to reduce state-of-the-art image classifiers to random guessing. Such a strong and counter-intuitive result brings with it questions of both practical and theoretical interest. Below we detail some of the important developments in both directions which help motivate the methods presented in this thesis.

Practical Perspectives on Adversarial Attacks In [118], the authors take a cyber-security flavored approach to attacks and characterize the attack setting also known as the *threat model*. An adversarial threat model refers to the act of modeling what information the attacker may have access to, as well as the medium of the attack. Defining a taxonomy for these concerns made the study of *practical* threats feasible.

In [118], the authors point out that a realistic attacker will likely not have access to the weights of a model that is deployed. If one has access to the weights and architecture, then the threat model is deemed a *white-box* threat model. In the case that one does not have access to the weights or architecture, the attack is considered *black-box*. The attacks that we have seen until now [136, 58, 108, 101, 26] have been white-box attacks. In [118], the authors present one of the first black-box attack methods, named the Jacobian saliency matrix attack (JSMA). The idea behind this attack was to build a finite-difference estimation of the Jacobian gradient (i.e., the first-order partial derivative wrt the input) and use that to attack the network with a similar procedure to that defined in [58]. Other such black-box attacks include zeroth order optimization [31], optimization with surrogate gradients [4], game-based approaches [148, 153] and genetic algorithms [2]. These kinds of attacks showed that networks that were deployed online (where users were allowed query access) could be attacked and very successfully fooled in practice [69].

A similar concern was the medium of attack. Machine learning practitioners were quick to respond to adversarial examples by pointing out that the 1/255 change to an image's pixel values would likely be washed out by particular lighting conditions. By changing the attack medium from a digital injection to a physical attack, several papers were able to show that attacks do manifest in the physical world in a robust way [149, 5, 89]. These works showed that traffic signs, traffic lights and a host of other objects were still misclassified even after printed onto physical media. In a similar vein, [20] use the methodology of [107] in conjunction with constraints to realize a small circular patch which can be placed onto *any* image which causes it to be misclassified.

Finally, a major concern was that, while attacks do present an inherent flaw in deep learning, simply being robust to adversarial attacks is not enough to declare a network *safe* for deployment [19], [66], [55], [144].

The existence of realistic attack algorithms and practical threat vectors underscores the necessity of developing robust machine learning along with the establishment of practical guarantees prior to the deployment of algorithms in the real world.

Theoretical Perspectives from Adversarial Attacks Adversarial examples are not only a realistic concern for machine learning practitioners, but also represent a fundamental challenge for machine learning theorists. In particular, the phenomenon of adversarial examples was not just shown to affect every tested image classification network in [58, 101, 26, 108], but also affected state-of-the-art networks in natural language processing [41], audio classification [163], 3D deep learning [149], malware classification [62], and others [1, 30]. The fact that these models were all able to capture super-human performance on natural data, but unable to withstand minor adversarial noise, prompted the question of whether *any* classifier could be both robust and accurate.

2.1.2 Impossibility Results for Adversarial Robustness

The entire following subsection is considered a correction as it is an expansion of the previous text that existed in the related works. The empirical fact that, for undefended neural networks, adversarial examples could fool *every* prediction made by a classifier led to a stream of research which studied if, from a theoretical perspective, adversarial examples were inevitable. Such results are known as *impossibility* results for adversarial robust classification. One of the first theoretical works in this direction was reported in [48, 47]. Under the assumption that the residuals of the model prediction have a known relationship to the distance of an input to the decision boundary, the authors provide an upper bound on the adversarial performance of linear and quadratic models. Other works have reasoned about the inevitability of adversarial examples not by making assumptions about the model/classifier, but instead making assumptions about the data that we wish to classify. In [46] the authors assume the data comes from a smooth generative model and derive upper bounds on the adversarial performance of any model in this setting. In [138] the authors generate a binary classification task (i.e., learning problem) in which there is a provable trade-off between accuracy and robustness: models which are more accurate are inherently less robust. The works of [46] and [138] were unified in a

more general setting in [36], which makes assumptions about the smoothness of the data-generating distribution.

Further learning problems (similar in nature to that given in [138]) for which we can prove the inevitability of adversarial examples were formulated in [129], where the authors prove results for data distributed on the unit cube and unit sphere. The work of [56] expands research on the unit sphere and studies high-dimensional classification of concentric spheres. Similarly to previous settings, [56] provide a proof that adversarial examples must exist within a certain magnitude for *any* classifier and show empirically that this theorem holds for neural networks in their setting. They also show that in their setting adversarial examples necessarily exist with distance $\mathcal{O}(1/\sqrt{n})$ of all input points. In [59], the authors focus on the Boolean hypercube setting and show, using the PAC-learning framework, that no non-trivial concept class can be robustly learned in the distribution-free setting against an adversary who can perturb just a single input bit. We will return to this discussion of the inevitability of adversarial examples from the Bayesian perspective in Section 2.2.

2.1.3 Verification and Certification

The widespread existence of adversarial examples is cause for concern when deploying neural networks in safety-critical scenarios. This concern can be quelled if one is able to *prove* that no adversarial examples exist for a given network and input-output specification. In the next section, we will formalize this notion, but here we will discuss different methodologies which have been put forth to prove that no adversarial examples exist within a given input set. While many of the methodologies proposed below are computationally expensive, we highlight that, without a formal proof or guarantee of correctness for these machine learning systems, their deployment would be either infeasible or imprudent. Verification is a natural pre-requisite for deployment of deep learning systems in safety-critical scenarios.

One of the first works to consider the verification problem for neural networks was presented in [160], where the authors consider verifying neural networks aimed at replacing large look up tables that would normally be used in aviation applications by a brute force approach: sampling the input space in a fine grid in order to arrive at guarantees. While this does in fact provide a form of guarantees that no adversarial examples exist, it becomes computationally intractable for problems of even a modest dimensionality. Below, we taxonomize the different popular methods for showing no adversarial example exists based on their underlying algorithmic approach.

SAT & SMT-Based Verification Satisfiability (SAT) testing and satisfiability modulo theories (SMT) are two well studied decision problems in computer science [6]. Algorithms for solving SAT and SMT problems have been studied over the past decades and powerful heuristics for solving these NP-complete problems have made solving larger scale SAT and SMT instances more feasible.

When applied to neural networks, SAT solving requires the neural networks weights and activation functions to be expressed as Boolean variables and functions (conjunctions and disjunctions). While most neural networks are not binary, the algorithms in [111, 112] show how, given a binary neural network, one can transform the proof that no adversarial example exists into a satisifiability (SAT) problem which can be solved with any SAT solver.

More powerful approaches extend the Boolean logic formulas to satisfiability modulo theories. In particular, for deterministic neural network (DNN) verification, by including a theory over real arithmetic (addition and multiplication) the methods of [79, 80, 42, 43, 77] are able to verify not just Boolean networks, but also networks which employ piece-wise linear activation functions such as the common rectified linear units (ReLU). Further contributions from [79] include the proof that the general problem of finding (optimal) adversarial examples is NP-hard. In a similar vein to these SMT approaches [71] uses SMT to propagate adversarial properties layer-by-layer until the output, at which point one can check if a safety criterion is met.

Abstract Interpretation Abstract interpretation is a classical form of program analysis. Rather than building logical formulas, abstract interpretation uses sound over-approximation of different operations in order to test if a property holds. This was first investigated for neural networks in [54]. When applied to deep neural networks, the central idea of abstract interpretation is to build an over-approximation (i.e., a superset) of the inputs that one would like to verify does not contain an adversarial example. Throughout the thesis we will will refer to over-approximations generally as analytically-convenient (i.e., easy to compute or reason about) supersets of a set of interest. For instance, abstract interpretation can handle arbitrarily complex perturbation sets by creating a set T' such that $T \subset T'$. These supersets can take the form of intervals, zonotopes, or polyhedra. By then defining semantics which propagate these shapes through the network (known as abstract transformations), one arrives at an over-approximation of the reachable outputs from these different input sets. Specifically, one arrives at a set S' such that we know no point of T' is mapped outside of S'. We can then check if the reachable outputs set, S', contains an unsafe

output. If it does not, then we are able to say soundly that the network under consideration is safe, i.e., $S' \subset S \implies \forall x \in T,\ f^{\theta}(x) \in S$. This approach has been further investigated in [95, 131, 106, 132]. The general consensus is that while abstract interpretation is computationally faster than the methods provided by SAT and SMT, they are not always *exact* and thus lose the property of completeness (more on this in Chapter 3).

One of the benefits of abstract interpretation is that it can be very computationally efficient if the abstract domain is chosen to be sufficiently simple [165, 145]. The box or interval abstract domain simply over-estimates the input set with an upper and lower bound for each feature dimension. Interval arithmetic can then be used to propagate these upper and lower bounds through the neural network. Arriving at the final output set in this manner is only twice as expensive as a standard forward pass. Of course, we know from [79] that solving the robustness problem exactly is NP-hard, so we can intuit that the interval domain makes pretty harsh approximations of the output set. While this is true, some works have found that the box/interval domain is empirically rather tight [60].

Linear Programming The final method we provide exposition on is the family of linear programming methods. Unlike the previous two methods discussed, linear programming does not have its roots in classical computer science, but rather in optimization.

The central idea behind the linear programming approach (and similarly mixed integer linear programming, MILP) is that we encode both our input properties and neural network computations as a set of linear constraints on an optimization problem. Solving the corresponding optimization problem to these constraints results in the ability to declare a network robust to adversarial examples. One of the first attempts at doing so leveraged mixed integer linear programming [137] and formulated rules for constraint building. Other methods include semidefinite programming, which is considered in [121]. The work of [152] and [39] use a dual formulation and derive bounds on the error and exactness of solving for the worst-case adversary with linear programming. Moreover, the mixed integer linear programming approach can be combined with branch-and-bound in order to obtain exact verification without the use of the dual used in [39] this is explored in [22, 21].

2.1.4 Adversarial Defenses

Below, we break down adversarial defenses into two broad categories. The first is an engineering-based defense category, which captures modifications to the architecture or some pre-processing of inputs in the hope of becoming more adversarially robust. The second category is adversarial training, which sees the introduction of adversarial signal at training time.

Engineering-based Defense After the popularization of adversarial examples in [136, 58], there was a rush of research attempting to *solve* the problem of adversarial examples. Here we detail a few defenses that do not directly see the injection of adversarial attacks into the training dataset (which is discussed below). In [117] the authors propose adding a temperature value into the logits of a neural network in order to prevent the softmax outputs from saturating. They then retrain a smaller network not using the true one-hot labels but using the labels from the tempered network. This is known as defensive distillation and achieved strong adversarial robustness results until it was thwarted in [29]. Similarly, feature squeezing was a method which simply rounds pixel values in an attempt to smooth out injected adversarial attacks [155, 28], but these attacks can be defeated either by expectation over transformations [5] or by having an attack with a smartly chosen attack radius [130]. Other methods seek to make it difficult to compute the gradient of a network, thus making it difficult to attack with the white-box methods we have seen above. This was a very popular defense method which was shown to be universally unsound in [4]. The key take-away point from each of these case studies is that, in general, engineering-based defenses can be readily bypassed with a modification to the adversary under consideration [4, 5]. The only method that was endorsed as an appropriate defense by the authors of [4] was the adversarial training method of [101].

Adversarial Training Adversarial training generally describes the idea of modifying the data seen at training time to include perturbed data. This modification was first proposed alongside the fast gradient sign method (FGSM) in [58]. Here, the authors propose a simple approach, where the likelihood is taken to be the sum of the likelihood between normal and adversarial data. They find that indeed adding adversarial noise to the training data caused the attack to be less successful during testing time [58]. Further approaches, such as [89, 78], propose modifications to this approach such as pairing the logits of natural and adversarial attacks and minimizing

their discrepancy. A similar approach was to bootstrap a likelihood but based on randomly sampled noise [167], and to attempt to regularize the input gradient similarly to what was proposed in [37], far before adversarial examples were ever discovered. While this model did have some improved robustness, it was found that increasing the strength of the attack caused the performance to degrade rapidly.

It was widely observed (and notably observed in [101]) that the adversarial training methods proposed in [58, 167] are only robust to the kinds of attacks they have seen at training time. That is, when a network trained with FGSM adversarial examples is attacked with PGD, it is found to be only slightly more robust than an undefended model [101]. Similarly, more heuristic approaches such as adversarial logit pairing [78] were also readily defeated by attacks different from what was seen at training time [45]. The authors of [101] attempt to remedy this in two ways. The first attempt to remedy this is called *robust optimization* and consists of training exclusively on adversarial examples. Concretely, robust optimization works by approximately computing an adversarial example for each point that we train on prior to passing it through the network and we evaluate the model likelihood on this adversarial data rather than the clean data. The second way they reason about improving robustness is by making the argument that, in the limit of the number of iterations, their PGD attack is a universal attack that will find a worst-case adversarial example. In conjunction, the authors reason that training on the worst-case adversarial example should lead to provably robust models. The latter point, finding the optimal adversarial example, is not realisable in practice which hinders the authors argument with regards to provable robustness. However, the robust optimization approach of [101] achieved state-of-the-art adversarial robustness and makes compelling theoretical arguments about how adversarial training should be done. In [164] the authors combine the robust optimization approach [101] with the adversarial regularization approach similar to [37, 167] and find that this is even better than just using PGD and robust optimization.

While [101] are unable to provide worst-case adversarial attacks, methods from convex relaxation such as those introduced in [152, 39] are able to provide an over-approximate worst-case loss for a neural network and given input. Abstract interpretation was also investigated to the same effect [106]. In [152, 120, 60], the authors push the state-of-the-art robustness even further by training with an over-approximate worst-case likelihood. The primary benefit of this approach is that it trains networks which are provably robust w.r.t. *any* attack.

2.2 Bayesian Robustness

In this section, we begin by considering the impossibility of robust performance from the perspective of Bayesian models rather than deterministic models as above. After establishing that Bayesian models may be inherently more adversarially robust, we cover previous and concurrent works that study Bayesian models in an adversarial setting.

2.2.1 Impossibility Results through a Bayesian Lens

In Section 2.2.1 we briefly cover impossibility results for robust learning. These results usually leverage assumptions on the form of the model or on the distribution of the data. Each work, however, discusses impossibility results with respect to a deterministic model. In [129], the authors provide a lengthy discussion of how one might avoid such impossibility results. They provide three primary perspectives: 1: alter the data representation to avoid the assumptions made, 2: add a "do not know" class allowing classifiers to abstain from prediction, 3: make it computationally infeasible to compute adversarial examples. In the literature it has been found that each of these components can be theoretically realised in the Bayesian setting, and that Bayesian learning can produce neural networks which are empirically much more robust.

In [7] the authors study the same concentric sphere classification problem as [56] but using Bayesian neural networks. They first establish that even linear models with careful regularization have adversarial examples in this case, thus ruling out robustness for MAP estimates. They then show that bootstrapping (via an ensemble of networks) is insufficient in this case. Finally, they show empirically how on adversarial examples the network is much more uncertain and reason that having high predictive uncertainty (from MCMC) allows them to learn a classifier which can detect adversarial examples.

In [53] the authors take the uncertainty argument one step further than the empirical study done in [7]. The authors provide a theoretical line or reasoning which shows that Bayesian neural networks should not have adversarial examples (defined as highly confident incorrect predictions). Their argument hinges on unambiguous labels for your data and a condition on the quality of the posterior inference. They also provide a range of experimental results backing this claim.

Finally, in [23] the authors[1] provide a proof and strong empirical evidence that it is infeasible to compute strong adversarial examples using first-order methods for Bayesian neural networks. Specifically, this work shows that under the assumptions of infinite data and exact inference, the input gradient used to compute adversarial examples should be 0, thus making first-order optimization intractable. Further investigation in [158] demonstrates that gradient-free optimization also struggles.[2]

While this is still very much an open area of research, each of these works agrees about the following facets: Bayesian neural networks ought to be more robust. As hypothesized in [129], avoiding the impossibility theorems for adversarial robustness relies on both abstention from a decision (in the Bayesian case based on predictive uncertainty) and on the intractability of computing adversarial examples. Empirically, there is also a lot of agreement and interesting lines of study posed by these works. Firstly, each work agrees that the *quality* of the posterior inference plays a vital role in robustness to adversarial examples. Specifically, [7, 23] both highlight situations in which bootstrapping with ensembles is not sufficient for adversarial robustness. Works offer varying observations on how well approximate inference performs. In [7], the authors find that Laplace method provides strong uncertainties which leads to robustness. In [23] the authors find this to only be the case on simple datasets such as MNIST and observe that the local robustness performance of approximate inference methods degrades on more complicated tasks such as FashionMNIST.

2.2.2 Robustness Verification for Bayesian Models

A concurrent work with this thesis is [9], which considers both Bayesian neural networks and probabilistic input specifications. The authors consider verifying the expectation of the model w.r.t. the input specification and posterior distribution over the weights. This is similar, but not exactly the same as the notion of decision robustness highlighted in Chapter 4, as decision robustness does not necessitate the consideration of the mean. Further, in [9], the authors provide verification that out-of-distribution samples are not classified with high confidence. This is a specification that is not empirically studied in this thesis but is highly interesting.

Concurrently with this thesis, however, many works have studied guarantees on the robustness of Gaussian processes in adversarial settings [25, 14, 119]. While these have principled connections to Bayesian neural networks in the limit of the width of

[1]This is a published work that I am first author on and was completed concurrently with the primary works of this thesis.

[2]This is also a publication for which I am an author.

a Bayesian neural network [113, 44], gaining guarantees on a GP which approximates a BNN is likely not to be rigorous enough as a proof of safety to warrant deployment of the BNN, as GPs are known to be more robust than BNNs and so approximation with a GP may hide considerable faults in the Bayesian neural network.

2.2.3 Detecting Adversarial Examples with Uncertainty

In [81] the authors discuss the difference between epistemic uncertainty (arising from our uncertainty about the model parameters) and aleatoric uncertainty (arising from our uncertainty about the data) and discuss exactly how uncertainty can be used in safety-critical scenarios using segmentation and autonomous driving as an example. The authors primarily suggest using uncertainty to detect out-of-distribution samples such as in [124, 125]. Several works have applied the a similar treatment to adversarial examples, seeking to use the predictive uncertainties in order to flag when an input may have been provided by an adversary [134]. The authors of [134] study out-of-distribution uncertainty of Bayesian Neural Networks by studying how the predictive distribution changes as we interpolate between two test inputs of differing classes. Further, in [134], the authors study the predictive uncertainty on adversarial inputs and empirically find that uncertainty estimate tends to increase for adversarial points. Further empirical evidence was found in favor of using uncertainty to detect adversarial examples in [123, 92, 49]. In [123] the authors study adversarial attacks and the addition of random noise to test inputs for a variety of approximate inference methods for Bayesian Neural Networks. They find that for both adversarial attacks and test inputs corrupted with noise, one can find an increase in predictive uncertainty from the Bayesian posterior. In addition, they find that the correlation between increased uncertainty and adversarial attacks is higher for less-approximate inference methods. Namely, they find that MC-Dropout did not demonstrate strong uncertainty properties in the face of adversarial attacks. In [96], the authors train MC-Dropout approximate posteriors using an α-divergence objective and show that this can strengthen the performance of these networks on both out-of-distribution samples as well as on adversarial attacks. In [49] the authors rely on MC-Dropout posteriors, but perform density estimation in the feature space of the MC-Dropout posterior in order to develop a more reliable method for uncertainty quantification. They find that the combination of these two methods allows uncertainty to flag adversarial examples with some success. In [104] the authors propose the use of uncertainty to flag out-of-distribution samples for autonomous driving and find, in a software simulator, that they are able to stop a large proportion of collisions that would not have

been avoided without uncertainty. Moreover, their study corroborates the findings of [123] that more sophisticated approximate inference techniques lead to better uncertainty. This correlation was observed theoretically in [23] where the authors provide a proof that under the conditions of exact inference, Bayesian Neural Networks should not be susceptible to gradient-based adversarial attacks.

2.2.4 Adversarial Attacks on Bayesian Neural Networks

Though some works have shown how uncertainty can be practically useful in flagging adversarial examples, there has also been some development of attacks specifically for Bayesian Neural Networks and other Bayesian methods such as Gaussian Processes. In [63], the authors study adversarial examples in a Bayesian setting and find that, despite their ability to capture uncertainty, Bayesian neural networks whose posterior distributions are only approximately inferred can be fooled into producing incorrect predictions with high confidence and low uncertainty. In [168], the author makes the observation that adversarial attack algorithms for deterministic neural networks cannot be directly translated to the case of Bayesian Neural Networks. Specifically, the observation made was that the attacks must be optimized to attack the full posterior distribution, rather than just a sample from the posterior. In [4], the authors formulate the expectation over transformation (EOT) attack which generalizes the observation of [168] to any randomization of a neural networks predictions (e.g., smoothing). The authors of [4] empirically show that Bernoulli approximate posteriors such as those resulting from MC-Dropout approximate inference [51] are not more robust to adversarial examples than deterministic neural networks. In [158], the authors consider non-gradient based attacks for Bayesian Neural Networks and find empirically that genetic algorithms can explore the space of adversarial attacks better than gradient-based optimization in the case of Bayesian neural networks.

2.2.5 Adversarial Defenses for Bayesian Neural Networks

Another line of work is to consider the intersection of Bayesian inference and adversarial training (which is done in detail in Chapter 7 of this thesis). In [146], the authors consider tuning the dropout rate of neural network parameters according to what gives the best adversarial robustness performance. By learning the optimal dropout rate, they find that they can increase the adversarial robustness of MC-Dropout networks. This is similar in spirit to the works on improving the performance of MC-Dropout's uncertainty properties against adversarial examples in [96, 49]. In [98], the authors

formulate the update similar to that of [16], but specifically for the bootstrapped adversarial likelihood provided in [58]. It was seen that the adversarial training procedure proposed in [98] was not evaluated on attacks properly crafted for Bayesian Neural Networks and that following the appropriate attack schemes that their method did not significantly increase the robustness of the Bayesian posterior [168]. In [157], the authors do not consider a worst-case adversarial example, but instead consider Bayesian inference against a distributional adversary. That is, in [157], the authors consider performing inference which takes into account a local distribution over each input point. Further, the work of [156] can be seen as an adversarial training method which operates by selecting a prior based on observing the function space properties (e.g., adversarial robustness) of a particular weight setting and then placing higher probability on weights with favorable properties.

2.3 Non-Local Notions of Robustness

The above methods for evaluation and training, as well as the methods developed in this thesis we will focus on local notions of robustness (i.e. robustness of the predictions of a network with respect to a single input). Before discussing these in further detail it is critical to distinguish the notion of robustness we will use from the many important but distinct notions which have been developed in the literature. In this section we will first cover specifications that reason about more than just a single input (and are thus deemed *global*), next we will discuss the relationship between the definitions in this thesis and those that are developed in the field of robust statistics, finally, we will discuss some works in robustness to misspecification in Bayesian models.

Global Robustness for Neural Networks Local robustness property involves a given x^* and involves finding another input x' in an allowed perturbation set such that the predictions differ in an undesirable way. *Global* properties of neural networks are instead require that for every input in a region of input space $Q \subseteq \mathbb{R}^{in}$ that a local property holds. Such properties clearly subsume local robustness properties. Global properties are also strictly more difficult to compute than their local counterparts. In [127] the authors provide global guarantees on the ℓ_0 magnitude of adversarial manipulations. Despite this difficulty, there are still promising works in this direction and applications for which global guarantees are necessary. Many works on global robustness focus on the global lipschitz constant of the function under consideration

which can be vacuous for unregularized networks. And methods which do seek to regularize global lipschitz constants must be careful not to fall into the pitfalls discussed in [74]. In [139] the authors propose to add reduction of the global lipschitz constant to the learning objective of neural networks. In [94] the authors propose yet another method for regularizing global lipschitz constants in addition to allowing the model to abstain from prediction and find more scalable results than previous methods.

Robust Statistics The field of robust statistics was largely popularized in some ground breaking work by Huber as early as 1960 [140, 18, 72]. In general the field of robust statistics studies the performance of statistical models when the observed data vary only slightly from the modellers assumptions either by not following an assumed parametric distribution or due to the existence of outliers. It was found that only minor modifications to the underlying assumptions of a model (i.e., assumed normality) can have catastrophic affects on the quality of inference [73]. In particular, in [140] the author shows that contamination of only two points from 1000 sampled from a Gaussian distribution can lead to poor performance of classical statistical procedures (relative to a robust procedures). The collection of methods united under the name robust statistics (see [73] for a thorough treatment), seeks largely to (1) develop tools to understand and compare how poorly classical statistical models perform when the data deviates slightly from the modellers assumptions and (2) to develop statistical models which are robust to such changes in assumptions. For further distinction on these two see [18] where the authors distinguish these two as criterion robustness and inference robustness. An early and classical model of such assumption-breaking data distribution comes in the form of Huber's contamination model [73] in which the data is distributed according to the following mixture: $x \sim (1-\gamma)p(x) + \gamma q(x)$ where $p(x)$ is taken to be the clean data distribution and $q(x)$ is taken to be the contamination distribution. The goal of a method in robust statistics is to devise a scheme for accurately estimating parameters/properties of $p(x)$ despite the contamination arising from $q(x)$. Many works in robust statistics seek to quantify the effects of such contamination (or similar settings) in order to produce estimators which can minimize the effect of the contamination on the resulting estimate. Early discussion of practical methods for dealing with and adapting estimators can be found in [87] with more detailed discussion of approaches in [3, 73].

There are several distinctions between this line of work and the local adversarial robustness considered throughout this Thesis. A key assumption of Huber's contamination model is that the data on which we perform inference with is contaminated.

This is not the case for the robust training method discussed in Chapter 7. Rather, our formulation must assume that all of the data has the correct/true label (subject to noise modelled accurately in the likelihood) in order to properly model an adversary. Though the model that we study throughout this thesis is relatively orthogonal to that of robust statistics, there is an interesting union between robust statistics and local adversarial robustness which is called *poisoning attacks* which is discussed in [116, 128].

Robustness to Model Misspecification Another important form of robustness for Bayesian models to have is misspecification robustness. A Bayesian model is usually said to be misspecified if a 'true' or data generating model has no support in the prior distribution either due to the form of the data (e.g., non-linear data, linear model) or simply a poorly chosen prior (e.g., a Dirac delta prior mass on an incorrect model). In such cases, many of the Bayesian arguments regarding convergence properties (e.g., Von Mises theorems) no longer hold. In learning settings related to what is covered in this thesis [65] show that misspecified Bayesian linear regression has posterior mass which diverges away from the error-minimizing and KL minimizing solution included in the prior. Moreover, in [166] they study the same settings and find that BNNs can be similarly inconsistent. In general, the "solution" proposed by both works is a modification to the objective. In [65] they find that in misspecified settings adding a weighting factor η to the likelihood where $\eta = 0$ means we do not update the prior distribution. In [64] the authors propose *SafeBayes* to choose an optimal η to safe-guard against model misspecification and find that this is able to mitigate against the perils of misspecification. In [166] the authors find that variational methods are affected less by misspecification and hypothesize that the difference in their optimization objective protects from misspecification.

Work on misspecification robustness is orthogonal to the local robustness presented in this thesis. Where misspecification robustness concerns model selection we will assume that the modeler has chosen an appropriate model prior to learning. Moreover, the methods for arriving at robustness guarantees in this Thesis can be done on any posterior regardless of if it inconsistent due to misspecification.

Chapter 3

Background

Contents

In this chapter, we introduce the notation and preliminary background to frame our contribution. In particular, we start by briefly covering the standard frequentist learning paradigm and using this to introduce deep learning. Next, we define the concept of adversarial examples and local robustness. Using the widespread vulnerability of deep neural networks to adversarial examples, we motivate the use of uncertainty in deep learning and thus Bayesian learning of neural network parameters.

Notation A note on the mathematical notation used in this thesis is in order before a discussion of methods and techniques. Real and natural valued variables will be in standard italicized type, e.g., i. Vectors will be denoted by bold, lower-case type, $\mathbf{x}$ with dimensions indexed by subscript, i.e., $\mathbf{x}_i$ refers to the i^{th} entry (equivalently,

feature dimension) of the vector $\mathbf{x}$. Unless otherwise noted, Greek letters (i.e., θ) will represent vectors. Random variables and vectors of random variables will be denoted using bold, italicized script, $\boldsymbol{x}$. Similarly, a matrix or finite collection of matrices will be denoted with an upper-case, bold, $\mathbf{M}$, with italicized counterpart referring to random variables over the matrix or collection of matrices $\boldsymbol{M}$. We will use the notation $[n]$ to indicate the finite set of natural numbers, $\{1, 2, \cdots, n\}$.

3.1 Deep Learning

Below, we cover the supervised learning paradigm in general and then proceed to discuss deep learning as a parameterized class of functions for solving the supervised learning problem. After this, we will discuss the engineering choices that dictate the parameterization and architectural form of a given neural network.

3.1.1 Supervised Learning

One of the primary motivations for developing machine learning systems is to mimic complex, observed phenomena for which simple generating or discriminating processes are difficult to express in closed mathematical form or algorithmically. We can model the process we would like to understand as a function: $g : \mathbb{R}^{n_{\text{in}}} \longrightarrow \mathbb{R}^{n_{\text{out}}}$. Given that the task at hand, understanding the function g under observational noise, does not have an intuitive underlying mathematical structure, we pose learning as trying to approximate g. By way of example, image classification poses the problem of extracting semantic information from high-dimensional matrices. As it is not straightforward to express what dictates *semantic information* in the space of matrices, we attempt to learn to extract this information from a set of examples.

One of the most common settings used to *learn* a function g is the supervised learning setting [11, 57]. In this case, we assume to have a set of observed inputs and outputs from g which comprise our *dataset*. The dataset is defined as a finite collection of input vectors $\mathbf{x}^{(j)}$, and corresponding output vectors, $\mathbf{y}^{(j)}$, according to the true function g (some noise in the labels can be considered as an extension of this setting). A collection of n_{data} of these observations forms our dataset: $\mathcal{D} := \{(\mathbf{x}^{(j)}, \mathbf{y}^{(j)})\}_{j=0}^{n_{\text{data}}}$, Where $\mathbf{y}^{(j)}$ encodes the semantic meaning of $\mathbf{x}^{(j)}$ that we would like to extract. In this thesis, we will primarily consider two specific supervised learning settings. Firstly, we consider the *classification* setting where $\mathbf{y}$ values are a finite collection of elements called *classes*. Secondly, we consider the *regression* setting where $\mathbf{y}$ values are continuous.

The central idea behind supervised learning is that a function that performs *well* at reproducing $\mathbf{y}^{(j)}$ given $\mathbf{x}^{(j)}$ from the dataset will also perform well given some unseen (i.e., not in $\mathcal{D}$) input point $\mathbf{x}^*$ [11, 141]. Performance on unseen data is known as *generalization*. Crafting a model which learns to generalize well is the main challenge in supervised learning. Intuitively, if one performs well even on unseen data, we reason that we have learned to extract meaningful semantic information from the input vectors. There is a large body of work on generalization guarantees, theoretical analysis of generalization, and empirical studies of generalization, all of which are beyond the scope of this thesis [142, 141].

3.1.2 Deep Learning Architectures

Given a particular dataset, a neural network is a carefully chosen function f^θ with parameters $\theta \in \mathbf{\Theta} \subseteq \mathbb{R}^{n_{\text{param}}}$, which one hopes can approximate g, thus solving the supervised learning problem. The dimensionality of a neural network's parameters θ are determined by the *architecture*. Together, the parameters and architecture determine how a neural network maps the input space, $\mathbb{R}^{n_{\text{in}}}$ to the output space $\mathbb{R}^{n_{\text{out}}}$. Recall that we would like for this mapping to produce the output $\mathbf{y}^{(j)}$ for each given $\mathbf{x}^{(j)}$. Practitioners must make several important decisions about the kind of neural network they would like to employ [57]. This series of decisions constitutes architecture selection and plays a vital role in accurately capturing the desired input-output relationship. The primary decisions include: selecting the number of layers, selecting the kind (and order) of layers, selecting each layer's activation function, and finally selecting a suitable likelihood function. Poor selection of any one of these variables can lead to an unstable neural network or one which cannot approximate the function of interest well. In this section, we cover different kinds of layers used in neural networks, then we discuss activation functions, and finally discuss appropriate likelihood functions for different problem settings.

3.1.2.1 Layers

The term *deep* neural networks references the potentially many parameterized transforms (layers) that are applied sequentially in order to map the input to the output space. In this section, we will cover some of the most foundational layers and describe the elementary operations that make up common deep neural networks.

Fully-Connected Networks Historically, the first and still the most important neural network layer is the *feed-forward fully-connected* layer [102, 57]. To build a description of this network, we start with the neuron. A single neuron is composed of two operations: a sum, and a non-linear transformation. A neuron η has a vector of weight parameters $\mathbf{w} \in \mathbb{R}^k$ and a bias parameter $b \in \mathbb{R}$. As a function, the neuron η takes an input $\mathbf{x} \in \mathbb{R}^k$ and performs the following computation: $\sigma(b + \sum_{i=0}^{k} w_i x_i)$, where σ is the non-linear activation function (discussed in Section 3.1.2.2).

Now that we have covered the basic neuron, we can build a definition of a neural *network*. The term network refers to a finite set of inter-connected neurons and their corresponding computations as graphs (with each node being a vertex in the graph and each input being an edge). To demonstrate one of the most fundamental ways to connect a collection of nodes, we describe the fully-connected layer. The computational graph between two adjacent layers of a fully-connected network forms a *complete, bipartite* graph, which means that every neuron in layer l is connected to (i.e., serves as input to) every neuron in layer $l+1$. Thus, the output of the i^{th} neuron on layer $l+1$ can be expressed as: $\eta_i^{1+1} = \sigma(b + \sum_{j=0}^{k} w_j \eta_j^l)$, where k is the number of neurons in the previous layer.

Notationally and conceptually, a fully-connected layer is best thought of as matrix multiplication. This becomes more obvious when we think about the *adjacency* matrix of a fully-connected bipartite graph. Between layers l and $l+1$ the adjacency matrix of the bipartite graph is a matrix $\mathbf{W}^l \in \mathbb{R}^{m \times k}$, where m is the number of neurons in layer l and k is the number of neurons in layer $l+1$. Each row in the matrix (elements of $\mathbb{R}^k$) represents the parameters of a single neuron. Further, each element of the adjacency matrix $\mathbf{W}_{i,j}^l$ represents the weight between the i^{th} node in layer l and the j^{th} node in layer $l+1$. Thus, we will henceforth call this adjacency matrix the *weight* matrix. We can notice the notational convenience of this representation by succinctly providing the element-wise formulation of a fully-connected feed-forward neural network:

$$\mathbf{z}_j^{(0)} = \mathbf{x}_j, \quad \boldsymbol{\phi}_j^{(i)} = \sum_{k=1}^{n_{i-1}} \mathbf{W}_{jk}^{(i)} \mathbf{z}_k^{(i-1)} + \mathbf{b}_j^{(i)}, \quad \mathbf{z}_j^{(i)} = \sigma^{(i)}\left(\phi_j^{(i)}\right), \quad j = 1, \ldots, n_i. \tag{3.1}$$

The sum in the above equation is simply performing an inner product, thus, the output of layer $l+1$ can be written simply as: $\sigma(\mathbf{W}^{l+1}\mathbf{z}^l + b^{l+1})$. Thus, rather than focusing on element-wise notation, the layer-wise formulation of a fully-connected

layer can be simplified to the following forward iterative equations:

$$\mathbf{z}_j^{(0)} = \mathbf{x}^{(j)},$$
$$\boldsymbol{\phi}^{(i)} = \mathbf{W}^{(i)}\mathbf{z}^{(i-1)} + \mathbf{b}^{(i)}, \tag{3.2}$$
$$\mathbf{z}_j^{(i)} = \sigma^{(i)}\left(\boldsymbol{\phi}^{(i)}\right), \tag{3.3}$$

Now that we have introduced the concept and mathematical definition of a fully-connected layer, a word on terminology for different terms in the forward iterative equations. Equation (3.2) is referred to throughout this thesis as the *preactivation* of layer i as it is the value after the affine transformation (in this case an inner product) but before the application of the activation function. The result of Equation (3.3) is termed the *output* of layer i.

In principle, fully-connected networks which take an all-to-all connection approach between layers are desirable as it is well known that with a wide enough fully-connected layer one can approximate any continuous function with arbitrary precision [34]. Moreover, their simple implementation as the application of an element-wise non-linearity on an inner product makes their implementation on modern graphics processing units (GPUs) very efficient.

3.1.2.2 Activation Functions

In previous sections on architectural choices we have alluded to the non-linear activation function σ. In this subsection we define different activation functions. Without an activation function, the neural networks we have discussed so far would simply linear mappings from $\mathbb{R}^{n_{\text{in}}} \longrightarrow \mathbb{R}^{n_{\text{out}}}$. In order to capture more complex, non-linear relationships between the domain and co-domain, one introduces non-linearities at the neuron level, which allows the network to conform to complex non-linear relationships that exist in the data. There are several activation functions that are very commonly used in neural network architectures.

In recent years, the most ubiquitous activation function has been the rectified linear unit (Relu). The Relu activation function, $\sigma(x) = max(0, x)$, is simply a threshold: if the value of the inner product of the neuron is positive, propagate it forward; if it is negative, propagate a zero. The practical benefit of Relu units is that they are fast to compute and differentiate and have shown to empirically possess strong convergence properties even for large and complicated neural network architectures. Moreover, Relu units are piece-wise linear activation functions, which makes networks that exclusively use Relu amenable to verification procedures such

as the simplex method and mixed integer linear programming. Very related to the Relu function is the LeakyRelu, which modifies Relu with the introduction of an α parameter: $\sigma(x) = max(\alpha x, x)$. By selecting $\alpha << 1$ we have that the LeakyRelu performs a similar operation to Relu, but propagates ("leaks") negative values forward through the network, albeit only a fraction of them (dictated by α). The motivation for this modification is that the gradient of negative values in a leaky Relu will have a non-zero derivative, whereas for standard Relu the gradient of the negative values will evaluate to 0. This has shown some practical advantages over Relu, and maintains the piece-wise linear nature of the Relu activation function.

Another common activation function is the hyperbolic tangent function (Tanh). Unlike the varieties of Relu activations, Tanh is not piece-wise linear: $\sigma(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$. This removes the benefit of being amenable to sound and complete verification enjoyed by Relu units. The benefit of this activation is that its co-domain is bounded: $[-1, 1]$. This can make it particularly appealing as the final activation of a regression problem with this range (further discussion below). However, its primary downside is that the gradients saturate to 0 for input values with absolute value greater than 3, which can make gradients vanish in this range.

Final Layer Activations In general, the selection of activation functions for intermediate layers (i.e., non-final layers) is done via trial and error (or a more systematic/principled version of trial and error such as Bayesian optimization). That is to say, there is no principled choice of activation function for a given network and dataset. This is not the case for the final activation function, however, as the co-domain of the final activation ought to match the co-domain of g, the underlying function we are trying to mimic.

In the classification setting, one is interested in predicting the "class" given some unknown feature vector $\mathbf{x}$. The classes are a set of finite, mutually-exclusive groupings of the observed feature vectors. Given that we have a set of $n_{\text{classes}} \in \mathbb{N}$ classes, the typical deep learning framework models these classes via a one-hot encoding, whereby the potential classes are each independently encoded as one of the unit basis vectors of $\mathbb{R}^{n_{\text{class}}}$ which is then taken to be $\mathbb{R}^{n_{\text{out}}}$. Typically, the output prediction of a neural network is then a vector from $\mathbb{R}^{n_{\text{class}}}$, which is normalized to sum to one. This is sometimes interpreted as the parameters of a multinoulli distribution [11]. The normalization of the class confidences is done via the application of the softmax

function:

$$\sigma_c(f, \mathbf{x}) = \frac{exp(f_c^\theta(\mathbf{x}))}{\sum_{l=1}^{n_{\text{class}}} exp(f_l^\theta(\mathbf{x}))} \tag{3.4}$$

The softmax function results in a vector whose sum (over all values of c) is 1, and which can be compared with the true/expected one-hot encoding vector. Throughout the thesis, when in a classification setting, we will omit σ and refer to $f_c^\theta(\mathbf{x})$ as the c^{th}-class softmax probability.

In regression, the choice is slightly less obvious as the co-domain of the function of interest, g, is problem dependent. For example, if one would like to learn the optimal drug dosage based on disease features, it makes sense to scale the output between 0 and 1 (continuous), where 0 represents the prediction that no drug should be administered and 1 represents the prediction that the patient should be given the maximum tolerable dosage. For such problems, the sigmoid activation function is normally used, $\sigma(x) = \frac{1}{1+e^x}$, as it has co-domain [0,1]. For other problems, normalization is not necessary or conducive to a better model. For non-normalized functions it is often that a linear activation, $\sigma(x) = x$, is used as it has unbounded co-domain. The final layer activation functions will also play a key role in determining the *goodness-of-fit* of our particular model.This is measured by the likelihood function which will be discussed further in Section 3.3.2.

In this section, we have described how different layers correspond to different parameterizations and transformations of our features. Without loss of generality, we will assume for the rest of the thesis that, for a given architecture we have a parameter vector $\theta \in \mathbb{R}^{n_{\text{params}}}$, which encodes all of the weights and biases for our neural network. As an example, a neural network with K fully-connected layers has $\theta = \{\mathbf{W}^{(k)}, \mathbf{b}^{(k)}\}_{k=0}^{K}$. We also highlight that for all of the neural networks we have discussed thus far, a particular parameter setting θ exactly determines the input-output mapping of our network. As such, we call these neural networks *deterministic*.

3.2 Adversarial Examples & Local Robustness

In Section 3.1.1, we established that the goal of supervised learning was to accurately mimic the output of a function g. We referenced the notion of *generalization* which is the property that a function which performs well on the training set ideally performs well on the test dataset as well. We also discussed the notion of a likelihood as a way of minimizing error on the test set which ideally leads to good generalization [141]. In

the seminal work of Szegedy et al [136], it was discovered that, despite strong generalization performance, neural networks possess a concerning vulnerability to adversarial examples (for other machine learning models see [10]). An adversarial example is an input to a neural network which, despite small, potentially imperceptible modifications to the input, causes large changes in the neural network's output. It was found that for nearly every input $\mathbf{x}$ to a state-of-the-art classifier an imperceptibly different input $\mathbf{x}'$ could be found such that the network misclassifies the input, despite being correct on the original $\mathbf{x}$. More concretely, it was found that a vector γ could be found such that $f^\theta(\mathbf{x}) \neq f^\theta(\mathbf{x}+\gamma)$ despite the fact that the magnitude of $|\gamma|$ is small (further formalization in the next section, Section 3.2.1).

We will begin by defining *local robustness properties* which, if satisfied, precludes the existence of adversarial examples. Finally, we will discuss definitions pertinent to the falsification and verification of local robustness properties.

3.2.1 Local Robustness Properties

Above, we introduced the concept of adversarial examples: inputs similar to those naturally occurring in the test set that cause a neural network to produce an erroneous or perhaps dangerous output. Here, we formally define adversarial examples as violations of a *local robustness property*. In particular, we say that a neural network f^θ is *robust* with respect to an input $\mathbf{x}$ if there is no perturbation of $\mathbf{x}$ within an allowed input set T which causes the output of f^θ to be incorrect or dangerous according to a safe output set S. The adversarial examples in Szegedy et al [136] were particularly worrisome because they are visually indistinguishable from the unmodified examples. This property can intuitively be made more general by collecting all inputs, $\mathbf{x}'$, which are semantically similar to $\mathbf{x}$, into a set T. In order to fully rule out the possibility of adversarial examples, we are then interested in knowing if any input in T, typically centered at an input $\mathbf{x}$, causes the network to make an incorrect or unsafe decision. Similarly to how we defined T, we collect all of the possible *safe* outputs in an output set S. For classification, the set S may consist of all vectors corresponding to a correct classification (e.g., a green traffic light). For regression, the set S may consist of all output vectors within some tolerable error.

Definition 1 (Local Robustness Property)**.** *Given a neural network f^θ, an input $\mathbf{x}$, an input set $T \subseteq \mathbb{R}^{n_{in}}$ including $\mathbf{x}$, and a set $S \subseteq \mathbb{R}^{n_{out}}$ of safe outputs, we say the network is $\{T, S\}$-locally robust if and only if*

$$f^\theta(\mathbf{x}') \in S \quad \forall \mathbf{x}' \in T \tag{3.5}$$

We can then simply state the formal definition of an adversarial example as a violation of this local robustness property (equivalently, when local robustness is not satisfied):

Definition 2 (Adversarial Example)**.** *Given a neural network f^θ, an input $\mathbf{x}$, an input set $T \subseteq \mathbb{R}^{n_{in}}$ such that $\mathbf{x} \in T$, and a set $S \subseteq \mathbb{R}^{n_{out}}$ of safe outputs, we say that an input $\mathbf{x}'$ is an adversarial example iff:*

$$\mathbf{x}' \in T \quad \wedge \quad f^\theta(\mathbf{x}') \notin S \tag{3.6}$$

Given both of these definitions, we will refer to a *local robustness property* as a two-tuple which throughout the thesis will be denoted $\{T, S\}$. If a network satisfies the conditions of Definition 1 we say it is *robust*. Conversely, if it satisfies Definition 2 then we say it is *vulnerable* or *non-robust*. One may also describe a network which is locally robust at an input to be *adversarially* robust. Throughout this thesis, we will make the distinction that *adversarial robustness* refers to the property of a network being robust when measured on many inputs while *locally robust* concerns only a single input.

On the selection of T Philosophically, one ought to select an input specification, T, that models all inputs which are perceptually similar to the input of interest $\mathbf{x}$ (in Definition 1). As T typically depends on a particular $\mathbf{x}$ we may write $T_\mathbf{x}$ to denote the local input property w.r.t. a given $\mathbf{x}$. Practically, a tidy mathematical expression of such semantic similarity rarely exists and is highly problem dependent. Despite the lack of a one-size-fits-all input specification, it is often the case that an ℓ_p-norm ball is a general and interesting specification for most domains. We recall the definition of an ℓ_p-norm ball centered at $\mathbf{x}$ here:

Definition 3. *Given a vector $\mathbf{x} \in \mathbb{R}^{in}$ the ℓ_p norm ball of radius ϵ around $\mathbf{x}$ is defined as:*

$$\mathcal{B}_{\epsilon,p}(\mathbf{x}) := \{\mathbf{x}' \; s.t. \; ||\mathbf{x}' - \mathbf{x}||_p \leq \epsilon\} \tag{3.7}$$

One can select parameters of the norm ball (ϵ and p) in order to fine-tune the input property to the domain. While this ball does not typically capture *all* of the semantically similar inputs, it can be tuned to capture a subset of them; thus, finding a counter example in this setting is still of great interest. The ℓ_p-norm ball setting is able to express different specifications with different settings of p, namely $p \in \{0, 1, 2, \infty\}$. For $p = 0$, we recover the Hamming distance, which is simply a count

of the number of dimensions which differ between $\mathbf{x}'$ and $\mathbf{x}$. For $p = 1$, we recover the 'Manhattan' distance which is simply the distance one would have to walk between $\mathbf{x}'$ and $\mathbf{x}$ if you were following a strict grid (as in a city blocks, hence Manhattan) in each dimension. More intuitively, from a mathematical point of view, the Manhattan or ℓ_1 distance is just the sum of the absolute differences in each dimension. For $p = 2$ we have the straight-line or Euclidean distance, which is perhaps the most widely used distance metric in applied mathematics. In the study of adversarial robustness, however, $p = \infty$ is the most widely used metric. Also known as the sup norm, by definition, the ℓ_∞ norm measures the maximum difference between any two dimensions of the inputs. In other words, $||\mathbf{x} - \mathbf{x}'||_\infty = max_{i \in [n]} |\mathbf{x}_i - \mathbf{x}'_i|$ given that $\mathbf{x} \in \mathbb{R}^{in}$. We note that the selection of ϵ ought to depend greatly on the selection of p. For instance, if one chooses ℓ_0 then the value of ϵ will be an integer between 1 and n. For ℓ_∞, on the other hand, we would expect it to be in the interval $[0, 1]$ if the dimensions are normalized into that range. Further works, generally in fairness (which has strong links to robustness [110]), investigates "balls" of input points within a certain Mahalanobis distance of one another which allows the modeller to account for intracorrelation between input features [159].

Before moving on to our discussion of measuring sensitivity in the output domain, we discuss some popular domain-specific choices for ϵ and p to establish some more concrete intuition for what is happening here. Given an image $\mathbf{x} \in \mathbb{R}^{W \times H \times C}$ (where W is the width, H the height, and C the number of channels), it is common to select $p = \infty$ and to select ϵ in increments of $1/255$. For $\epsilon = k/255$, we interpret the norm ball to test the property that if we change (at most) every pixel's color intensity by at most $k/255$ then the output should remain safe. For robotics, one may use the ℓ_2 distance with an ϵ which corresponds to the expected positional noise of your sensors. This property would correspond to the intuition that instrumental noise should not greatly affect the prediction of the network, even in the worst case.

On the selection of S It is clear that, if a traffic light is red and is predicted to be green (due perhaps to some noise in $\mathcal{B}_{1/255,\infty}$), we would say that prediction is unsafe. However, in general, it is difficult to determine what kind of change in prediction is *unsafe*. Similarly to the input specifications there are some general measures which are typically of at least some interest to each application, though it is usually true that the more specific a property is to a given application the more interesting it is to practitioners. For classification, generally one checks if there is an $\mathbf{x}' \in T$ which causes the classification to change arbitrarily. This would be modelled as S being

all softmax vectors such that the true class is non-maximal. One could similarly check if there is an $\mathbf{x}' \in T$ which causes the network to output a particular, unsafe class. This would be modelled as S being all of the softmax vectors such that the particular unsafe class is maximal. For regression tasks, there is often some minor tolerated noise in the output and this is typically expected to be bounded w.r.t. the ℓ_2 magnitude.

One typical specification of interest is the Lipschitz property of a neural network. Formally, a network is Lipschitz continuous or k-Lipschitz continuous if it satisfies the following property for all $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}$: $||f^{\theta}(\mathbf{x}^{(1)}) - f^{\theta}(\mathbf{x}^{(2)})||_p < k||\mathbf{x}^{(1)} - \mathbf{x}^{(2)}||_p$. We note that in the case of using $\mathcal{B}_{\epsilon,p}$ as well as an ℓ_q norm in the output space, one is effectively providing a Lipschitz constraint on the function learned by the neural network.

3.2.2 Computing Robustness Properties

While the task of defining local testing criteria for learning algorithms is challenging, computing the satisfaction of said robustness properties can be more challenging. Over the past few years, techniques from optimization, convex relaxation, satisfiability testing, and statistics have been used to great effect in computing local robustness properties. Given the safety-critical nature of understanding the worst-case performance of learning algorithms, it is imperative that these computations come with some sort of *guarantee*. In this thesis, we use terminology from program analysis and verification to describe the logical properties of an algorithm which attempts to prove robustness. Following the standard definitions in [133] (page 328), with more in depth treatment in [103], we introduce the concepts of *soundness* and *completeness*. Given a logical property (e.g., satisfying the adversarial robustness formula in Definition 1), an algorithm is *sound* if and only if if it never declares satisfiable formulas to be unsatisfiable; n.b., sound algorithms are permitted to report an input is unsatisfiable even if it is satisfiable. A trivially sound algorithm always returns that the property is unsatisfiable for every input. An algorithm is said to be *complete* if all unsatisfiable formulas are declared to be unsatisfiable; n.b. complete algorithms are permitted to report that a proposition is satisfiable even if it is unsatisfiable. A trivially complete algorithm returns that every property is satisfiable.

For further clarity, we restate these definition in the context of adversarial examples (violations) with respect to local robustness properties. These definitions can be restated as follows:

	Locally Robust	$\exists$ Adversarial Example
Passes Inpection	Accepts Desirable Inputs	Missed Violations
Flagged as Non-Robust	False Alarms	Caught Violations

Table 3.1: Potential outcomes of a robustness analysis.

- **Soundness** - The algorithm only declares a network robust if it proves no adversarial example exists (equivalently, local robustness is satisfied). It can, however, say that an adversarial example exists (equivalently, local robustness is unsatisfiable) even if it does not.
- **Completeness** - The algorithm only declares a network vulnerable if the an adversarial example exists (equivalently, local robustness is unsatisfiable). The algorithm may declare that no adversarial example exists (equivalently, local robustness is satisfied) when one does.

For clarity, we taxonomize the main three types of algorithms considered in this thesis and illustrate their properties using Table 3.1: sound but not complete, complete but not sound, and sound and complete. An algorithm which is sound but not complete will only declare a network robust if it can prove that robustness property (i.e., no adversarial example exists), but may fail to prove robustness. This is captured by allowing for all outcomes in Table 3.1 except it also allows for false alarms to be made. So if a sound but incomplete method does not prove robustness that does not necessarily entail that an adversarial example exists. Similarly, an unsound but complete method (e.g., all adversarial attack algorithms) will declare a network vulnerable if there is an adversarial example, but may not find an adversarial example if they exist. This is captured by allowing for all outcomes in Table 3.1 except it also allows for missed violations. That is, if an attack fails, that does not entail that no adversarial example exists. Finally, a sound and complete method always reports if an adversarial attack exists or if it does not. Regardless of the outcome of a sound and complete algorithm we always know if an adversarial example exists. This is captured by only the green outcomes in Table 3.1.

3.2.2.1 Falsification of Local Robustness

Perhaps the simplest way to gain information about Definition 1 is to attempt to find some $\mathbf{x}' \in T$ such that $f^{\theta}(\mathbf{x}') \notin S$. If such an input is found then we say that $\mathbf{x}'$ *falsifies* the local robustness property of interest. We highlight that an $\mathbf{x}'$ that falsifies a local robustness property is exactly equivalent to an adversarial example

in Definition 2. Given a neural network, finding an adversarial example is often stated as an optimization problem. It is standard practice to set up a loss function $\mathcal{L} := \mathbb{R}^{n_{out} \times n_{out}} \longrightarrow \mathbb{R}$ which takes in a pair of neural network output vectors and returns a score indicating how far away they are with respect to a metric/divergence corresponding to the selection of S. The key idea here is that an input $\mathbf{x}'$ which maximizes $\mathcal{L}(\mathbf{y}, f^{\theta}(\mathbf{x}'))$ will also lead to an input such that $\mathbf{f}^{\theta}(x') \notin S$. By introducing the loss function $\mathcal{L}$, we then pose falsification as an optimization problem:

$$\mathbf{x}^* = \arg\max_{\mathbf{x}'} \mathcal{L}(\mathbf{y}, f^{\theta}(\mathbf{x}')) \tag{3.8}$$

A common choice for the loss function is to minimize the likelihood of the model (discussed further in Section 3.3.2); however, the loss used here and the likelihood need not necessarily be related. We highlight that in [26], the authors survey a wide range of potential loss functions for this problem and provide a thorough treatment of best practices.

Having set up this optimization problem, there are, of course, a myriad of ways that we may go about approximately computing $\mathbf{x}^*$. Given that the goal of falsification algorithms is to produce adversarial examples we call them *adversarial attack* algorithms. Below, we will detail some of the different optimization algorithms which have been successfully employed for approximately solving this problem.

Optimization Methods for Adversarial Attacks It is well known that, due to its non-convex nature, the optimization posed in Equation (3.8) is NP-hard to solve exactly [79]. As such, it is commonplace to attack the problem with standard optimization approaches, namely, first-order methods. No matter the method, unsound algorithms which attempt to approximately solve Equation (3.8) are known as adversarial *attack* algorithms. Because attack algorithms cannot guarantee to find adversarial examples if they exist, they are inherently complete but not sound algorithms.

Any attack which uses first-order optimization is not guaranteed to solve a non-convex problem exactly. Thus, such attack algorithms are rendered *unsound* due to the fact that, if they do not find an adversarial example, it does not mean that no adversarial example exists. They are *complete* because, if an adversarial example is found, then we can necessarily say that the network is not robust. In order to demonstrate this, let us introduce one of the simplest gradient-based attack algorithms, the Fast Gradient Sign Method (FGSM):

Definition 4. *Given a deterministic neural network f^θ, an input $\mathbf{x}$, an input set $T = \mathcal{B}_{\epsilon,\infty}$ and an assigned output $\mathbf{y}$, the fast gradient sign attack at $\mathbf{x}$ is given as*

$$\mathbf{x}_{adv} = \mathbf{x} + \epsilon * sign\big(\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{y}, f^\theta(\mathbf{x})\big) \tag{3.9}$$

We notice that if there is no adversarial example inside of $\mathcal{B}_{\epsilon,\infty}$, that is, if f^θ is locally robust w.r.t. $\mathcal{B}_{\epsilon,\infty}$ and a given S, then certainly $\mathbf{x}_{adv}$ will not be an adversarial example. Similarly, we could arrive at a situation where only $\mathbf{x}' = \mathbf{x} + \epsilon/2$ is an adversarial example for any ϵ vector. Yet, due to the nature of the computation in Definition 4 we can never arrive at such an $\mathbf{x}'$, and thus $\mathbf{x}_{adv}$ will not be an adversarial example. What these two cases demonstrate is the lack of *soundness* of the FGSM attack: if we do not find an adversarial example using Definition 4 then we cannot conclude anything about the local robustness property of interest. Despite the fact that this is true for such a simple attack, it is also true for more complicated attacks such as the Projected Gradient Descent (PGD) attack.

Definition 5. *Given a deterministic neural network f^θ, an input $\mathbf{x}$, an input set $T = \mathcal{B}_{\epsilon,p}$ and an assigned output $\mathbf{y}$, the projected gradient attack at $\mathbf{x}$ is given as*

$$\begin{aligned} \mathbf{x}^0 &= \mathbf{x} \\ \mathbf{x}^{i+1} &= proj_{\mathcal{B}_{\epsilon,p}}\big(\mathbf{x}^i + \epsilon * \nabla_{\mathbf{x}^i} \mathcal{L}(\mathbf{y}, f^\theta(\mathbf{x}^i)\big) \end{aligned} \tag{3.10}$$

where the $proj_{\mathcal{B}_{\epsilon,p}}$ operator projects its input vector onto $\mathcal{B}_{\epsilon,p}$. In this definition, we have overloaded the superscript to be the iteration counter rather than indicating distinct input points.

In principle, PGD is a much stronger and better attack than FGSM. A main disadvantage of FGSM is that, no matter what the size of the norm ball, FGSM will use the direction of input gradient of the loss function at $\mathbf{x}$ and move to the edge of the ball. PGD, on the other hand, will explore the interior of the norm ball by iteratively recomputing the gradients at each step, see Equation (3.10). The cost of PGD, computationally, over FGSM is that each iteration of PGD takes the same computational time as a full FGSM attack, so a PGD attack generated with 100 iterations is 100 times more expensive than computing an FGSM attack. Similarly to FGSM, however, PGD cannot guarantee to find a minimum of the optimization problem posed in Equation (3.8). This becomes clear when we consider that Equation (3.8) is a non-convex optimization problem and, as such, has potentially many local optimum which the PGD procedure could get stuck in: if we imagine that only the global optimum is an adversarial example, then getting stuck in a local optimum

means we will return a non-adversarial example despite the existence of an example in T. Hence, the method is *unsound*.

3.2.2.2 Verification of Local Robustness

The abundance of adversarial examples in practice makes ensuring that they do not exist for a given neural network and input pair a particularly attractive prospect. This is exactly the goal of verification. As we have seen, when an adversarial attack is successful we have concrete proof (in the form of a counter-example) that a local robustness property does not hold. Verification, on the other hand, when successful provides concrete proof that a local robustness property does hold. In essence, verification methods are necessarily *sound* methods. Further, some verification methods are both sound and complete, meaning that if they are successful they have provided a proof that a local robustness property holds and if they fail then they are able to deliver proof that the local robustness property does not hold. In this section, we will start by discussing such complete examples which typically take the form of constraint satisfaction problems and will move on to an exposition of sound but incomplete verification which includes convex relaxation techniques.

Optimization-Based Verification Unlike the general optimization problem posed in Equation (3.8) (i.e., based on a loss function), optimization-based verification seeks to solve a constrained optimization problem. In order to generate sound constraints on a property of interest, one typically needs to place some restrictions on the form of our neural network, f^θ as well as on our local property $\{T, S\}$ [79, 80, 21, 152]. By carefully constraining the kinds of neural networks and properties we consider, we can generate constraint optimization problems which are amenable to mixed integer linear programming algorithms, satisfiability modulo theory (SMT) solvers, and other classical constrained optimization techniques such as the simplex method.

In order to build solvable constraints, current techniques rely on each activation function in f to be piece-wise linear. Similarly, T and S are typically required to consist of a conjunction of linear constraints [60]. Such constraints can be built for a large class of output properties S such that the solution of the constraints corresponds to exactly proving robustness [21]. Once these constraints are built, one inherits the completeness and soundness of a solution from the solver that is chosen.

The primary advantage of using constraint solvers for verification of local robustness properties is that they can be amenable to finding both sound and complete solutions. However, this comes at great computational expense as the number of

constraints that need to be solved grows quickly with the number of neurons in the network and typically cannot handle inputs such as those for large-scale image classification.

Convex Relaxation for Verification Up until now we have focused on methods that *search* for a worst-case adversarial input. When using gradient-based optimizations we found (generally) suboptimal solutions which constitute adversarial examples. However, these processes are unsound. In this section, we will focus on convex relaxation as a away of generating sound but incomplete information about local robustness properties. The most common form of sound but incomplete guarantee for neural network comes by way of bound propagation [54, 60, 150]. In [54], the authors propose an abstract interpretation framework which allows them to propagate an input specification T through the neural network such that the output result is an over-approximation of the reachable outputs from any input in T. More formally, their propagation methodology results in an output interval (or convex shape) $\hat{S}$ such that the following holds: $\forall \mathbf{x} \in T,\ f^{\theta}(\mathbf{x}) \in \hat{S}$. It then suffices to check that the output of the propagation $\hat{S}$ is a subset of the safe set that defines the local robustness property, $\hat{S} \subseteq S$. If this is the case, then we know that the network is necessarily safe with respect to the input interval T.

One drawback of this method is that we can only express certain kinds of specifications T exactly (i.e., those which are exactly expressible by a class of abstract domain). For input specifications which are not easily expressed as one of the abstract domains presented in [54], one must make an over-approximation of T with a larger set $T \subset T'$ such that T' is expressible in an abstract domain. Despite this drawback, it was shown in [60] that the most basic abstract domain, intervals, has very desirable properties in terms of the tightness of $\hat{S}$. Below, we detail the specifics of abstract interpretation when using the *Box* (equivalently, the *interval*) abstract domain. This is known as Interval Bound Propagation (IBP).

The key idea behind IBP is to use intervals over each feature dimension in order to build an over-approximation of T. That is, given an input $\mathbf{x}$ we would like to have an input interval $[\mathbf{x}^L, \mathbf{x}^U]$ such that $\mathbf{x}_i^L \leq \mathbf{x}_i \leq \mathbf{x}_i^U$ for $i \in [n_{in}]$ and $T \subseteq [\mathbf{x}^L, \mathbf{x}^U]$. Given that these two things are true, the goal is to pass the entire interval $[\mathbf{x}^L, \mathbf{x}^U]$ through the neural network.

In order to compute a forward pass with respect to an interval $[\mathbf{x}^L, \mathbf{x}^U]$ we can notice that the pre-activation (the second equation in our iterative equations for a

forward pass) is simply a linear transformation and thus we can straightforwardly push our bounds through by taking an element-wise minimum and maximum:

$$\zeta^L = \mathbf{W}^{(i)}\mathbf{z}^{L,(i-1)} + \mathbf{b}^{(i)}$$

$$\zeta^U = \mathbf{W}^{(i)}\mathbf{z}^{U,(i-1)} + \mathbf{b}^{(i)}$$

$$\boldsymbol{\phi}^{L,(i)} = min\big(\zeta^L, \zeta^U\big)$$

$$\boldsymbol{\phi}^{U,(i)} = max\big(\zeta^L, \zeta^U\big)$$

where we stress that the latter two are element-wise minima and maxima, respectively. Next, we have the application of the activation function σ which, if it is monotonically increasing (note all of those discussed in this thesis are), then we have that:

$$z^{L,(i+1)} = \sigma(\boldsymbol{\phi}^{L,(i)})$$

$$z^{U,(i+1)} = \sigma(\boldsymbol{\phi}^{U,(i)})$$

By setting the base case to be $\mathbf{z}^{L,(0)} = \mathbf{x}^L$ and $\mathbf{z}^{U,(0)} = \mathbf{x}^U$ we can propagate through these equations to get sound upper and lower bounds on the output of the network. One observations about the above equations is that the element-wise maximum and minimum are slower and more inconvenient than linear operations. We can employ the slightly less intuitive, but computationally faster method of interval bound propagation proposed by [60].

For regression tasks, it is clear that the outputs give us the highest and lowest values for each of the target variables. Therefore the worst-case likelihood reachable from a given IBP pass is simply the element-wise maximum of the difference between the target and the output interval $[\mathbf{y}^L, \mathbf{y}^U]$ note $\mathbf{y}^L, \mathbf{y}^U$ are equal to the final $\mathbf{z}^L, \mathbf{z}^U$ in the above equations). For classification, on the other hand, we must take special care to propagate our bounds properly through the softmax function. This can be accomplished by taking the lower bound for the logit of the true class and the upper bound for the logit of all other classes and passing them through the softmax.

3.2.3 Adversarial Training

Nearly as quickly as adversarial examples were discovered, adversarial defences were proposed. In [58], the authors propose to inject their adversarial attacks into the neural network training procedure as a method of making neural networks resistant to adversarial attack. The authors propose to use the following error model:

$$\text{AdvErr}(f^\theta, \mathbf{x}, \mathbf{y}) = \lambda CE(f^\theta(\mathbf{x}), \mathbf{y}) + (1-\lambda)CE(f^\theta(\mathbf{x}_{\text{adv}}), \mathbf{y}) \tag{3.11}$$

where the function CE corresponds to the cross entropy likelihood function, Equation (3.15). The same principle can straightforwardly be applied to other error functions such as the mean squared error that corresponds to a Gaussian likelihood. Following this proposal in [58], Madry et. al. [101] propose a modification of this loss function in conjunction with their development of the PGD attack. Namely, they consider the loss in Equation (3.11) with $\lambda = 0$. Such a setting creates an error which only sees adversarial signal. The authors note the relationship between this kind of adversarial training and robust optimization in that it can be seen as a minimax formulation. This is clear when we reconsider our optimization formulation of adversarial examples given in the above section, e.g. Equation (3.8), and plug it into the second term of Equation (3.11). The major downside of the robust optimization scheme for adversarial training is that it can be very hard to learn an initially good function. That is, if the adversarial noise is strong enough then the adversarial noise will render the learning step intractable. In practice, it is common to place the hyper-parameter λ on a schedule which sees it starting at 0 (i.e., learning on natural data) and ends in the later stages of learning with $\lambda >> 0$. Gradually increasing the magnitude of adversarial noise during training has shown to lead to much better performing (in terms of accuracy) models, which also display heightened adversarial robustness.

In practice, the results from the methodologies outlined above are mixed. In particular, it is found that training against a particular attack can introduce *adversarial overfitting*. This is a phenomenon in which a neural network is trained with $\mathbf{x}_{\text{adv}}$ coming from a specific attack method which causes the network to learn to be robust only with respect to that attack method and not against others. For example, if we consider an FGSM adversary on an $\mathcal{B}_{\epsilon,\infty}$ specification then we also know that our adversarial examples are only coming from the outermost shell of $\mathcal{B}_{\epsilon,\infty}$ as each each feature is maximally perturbed as prescribed by the definition of the attack (Definition 4). This can lead to networks which are robust to the natural point $\mathbf{x}$ and to adversarial examples which lie exactly on the shell of the input specification, but which are non-robust when we consider attacks that return points in the interior of $\mathcal{B}_{\epsilon,\infty}$. It is often observed that networks that are trained with FGSM are not robust against PGD, for example.

One could naively train against many attacks at once in the hope of being robust to all possible attacks; however, new attacks on deep learning models are developed every month and so it is hopeless to try to incorporate every attack possible. Instead, a solution to adversarial overfitting can be found in [60], which trains not on a specific

attack, but on the output of a convex relaxation such as IBP. By training on a likelihood which is based on IBP, one is training against an output which is guaranteed to be worse than anything an adversary could produce and is therefore defending against *any* feasible attack. This method was shown to be extremely promising and gives state-of-the-art defense against adversarial attacks. For classification, this is done practically by taking the upper and lower bounds (as computed by IBP) and computing the minimum of the softmax output of the true class as:

$$\sigma_c^{\min}(f, \mathbf{x}) = \frac{exp(f_c^{\theta,L}(\mathbf{x}))}{exp(f_c^{\theta,L}(\mathbf{x})) + \sum_{l \neq c} exp(f_l^{\theta,U}(\mathbf{x}))} \tag{3.12}$$

where, as before, $f_i^{\theta,L}(x)$ and $f_i^{\theta,U}(x)$ are the lower and upper bounds on the i^{th} logit (final pre-activation of the neural network) as a result of interval bound propagation procedure described in Section 3.2.2.2. This lower bound on the softmax of the true class can then be substituted in for the adversarial term of Eqn. (3.11) to arrive at the training proposed in [60].

3.3 Bayesian Learning for Neural Networks

In our previous sections, we have discussed deep learning, some of the important decisions and pitfalls of developing a neural network for a given application as well as the vast safety and security concerns that are adversarial examples. In an idealised setting, the Bayesian learning paradigm solves many of the vexing problems we have brought up. In particular, architecture choices can be made by way of type II inference [11], overfitting is theoretically a non-issue [99], and more recently it was shown that even adversarial examples can be excluded from the list of worries when using Bayesian neural networks [23]. Unfortunately, the idealised Bayesian deep learning scenario exists in several unrealistic limits, and thus it is rare for these idealised conditions to be realized at a large scale.

Despite this, in this section we describe how Bayesian learning of neural networks differs from frequentist learning. We then describe some of the new modeling choices that need to be made, and the different approximate Bayesian inference methods that can be employed for neural networks. While these approximate inference methods prevent us from reaching the theoretical guarantees which are optimistically stated in [11, 99, 23], we spend the next four chapters of the thesis developing practical (robustness) guarantees for Bayesian neural networks resulting from any inference method.

3.3.1 Bayesian Learning

The difference, philosophically, between Bayesian and frequentist learning (equivalently, maximum likelihood) is well summarized by Bishop in [11] on page 22 (amended to reflect the notation used in this thesis):

> In a frequentist setting, θ is considered to be a fixed parameter, whose value is determined by some form of 'estimator', and error bars on this estimate are obtained by considering the distribution of possible data sets $\mathcal{D}$. By contrast, from the Bayesian viewpoint there is only a single data set $\mathcal{D}$ (namely the one that is actually observed), and the uncertainty in the parameters is expressed through a probability distribution over θ.

This quote prompts us to think about how to arrive at a distribution over θ and exactly how this translates to practical uncertainties. The ingredients necessary to arrive at an appropriate distribution over the parameters of our neural network are prescribed by Bayes' theorem:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)\ \ p(\theta)}{p(\mathcal{D})}$$

Given that this thesis deals exclusively in the supervised learning scenario, in which $\mathcal{D} = (\mathbf{X}, \mathbf{Y})$, we will sometimes want to express Bayes' theorem in terms of these components. Moreover, we will often concern ourselves with the unnormalized form of the posterior in which the evidence term, $p(\mathcal{D})$ above, is omitted. This leaves us with the following form of Bayes' theorem:

$$\underbrace{p(\theta|\mathbf{X}, \mathbf{Y})}_{\text{Bayesian Posterior}} \quad \propto \quad \underbrace{p(\mathbf{Y}|\mathbf{X}, \theta)}_{\text{Likelihood}} \ \underbrace{p(\theta)}_{\text{Prior}} \tag{3.13}$$

In this equation we see that the primary two components of the posterior are the likelihood, which measures the *goodness-of-fit* of a particular parameter setting θ [11], and the prior distribution, which encodes our prior beliefs about the plausible data-generating functions [109]. Both of these are discussed in detail in the following two subsections. By taking the product over these two spaces (which requires marginalization), we arrive at the posterior distribution for our neural network.

Of course, in order to solve the supervised learning problem, we must use this posterior distribution to make predictions about a new, unseen datapoint $\mathbf{x}^*$. In the

Bayesian setting, one uses the posterior predictive distribution in order to express beliefs about unseen data:

$$p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) = \int p(\mathbf{y}^*|\mathbf{x}^*, \theta)p(\theta|\mathbf{X}, \mathbf{Y})d\theta \tag{3.14}$$

we can see that the posterior predictive distribution is made up of the product of our likelihood with our posterior distribution [50].

3.3.2 Likelihoods

The task of the likelihood in both the frequentist (equivalently, maximum likelihood) setting and the Bayesian setting is to measure the *goodness-of-fit* of a particular parameter setting θ [11]. In both, modellers are interesting in finding *good* parameters for our neural network. Of course, this is vague as we have not established any concrete notion of the *goodness* of a given parameter setting. This is the function of the likelihood.

Given a neural network f^θ (architecture and parameters), we are interested in understanding how *likely* it is to observe the true labels of our data. Recall that the goal of supervised learning is to reproduce the label $\mathbf{y}$ for each input $\mathbf{x}$ in the dataset. In order to check how *good* a model is at this task, we use the likelihood to compare the network's output (under some assumed observational noise) against the true or expected label. For regression, the likelihood is probabilistically modelled as:

$$p(\mathbf{y}^{(j)}|\mathbf{x}^{(j)}, \theta) = \mathcal{N}(\mathbf{y}^{(j)}|\mathbf{x}^{(j)}, \theta, \tau\mathbf{I})$$

which is simply a Gaussian distribution centered at the prediction with the identity covariance scaled by a vector τ to model the noise in our observation dimensions. In the maximum likelihood setting, one typically only cares about maximizing the mean of this Gaussian as the covariance term is independent of the observed data [11]. Thus, one seeks to maximize this likelihood by minimizing the mean squared error:

$$\frac{1}{2}\sum_{i=0}^{n_{\text{data}}} ||f^\theta(\mathbf{x}^{(j)}) - \mathbf{y}^{(j)}||^2$$

One can see that the above function has a true minimum when the output of $f^\theta(\mathbf{x}^{(j)})$ is equal to $\mathbf{y}^{(j)}$ for all observed input points, and given that the covariance of the likelihood does not depend on the model output, this maximizes our likelihood. For classification, the likelihood is given by an interpretation of the softmax output as

the parameters of a multinoulli distribution. In this case, we have that the likelihood of observing $\mathbf{y}^{(j)}$, which represents true class c is given as:

$$p(\mathbf{y}^{(j)} = c|\mathbf{x}^{(j)}, \theta) = f_c^{\theta}(\mathbf{x}^{(j)})$$

That is, we take the c^{th} element of the output of the neural network (from the softmax activation function defined in Equation (3.4)) to be the class probability and thus the likelihood. Similarly to what was done for the Gaussian likelihood case, we then minimize error, which in this case is the well-studied cross-entropy error:

$$\sum_{i=0}^{n_{\text{data}}} \sum_{i=0}^{n_{out}} \mathbf{y}_i^{(j)} log(f_i^{\theta}(\mathbf{x}^{(j)})) \tag{3.15}$$

Again, we see that this function has a single minimum when the multinoulli assigns probability of observing the true class as 1 ($f_{j=c}^{\theta}(x^{(j)}) = 1$). Further discussion of these likelihoods and errors can be found in [11, 141].

To finish our discussion of the likelihood, we can consider now that we have our entire dataset $\mathcal{D} = \{\mathbf{X}, \mathbf{Y}\}$ which, as before, is made up of n_{data} input-output pairs, $\{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}_{j=0}^{n_{\text{data}}}$. By assuming that each of these observations are independent and identically distributed given θ, we can measure the likelihood as:

$$p(\mathbf{Y}|\mathbf{X}, \theta) = \prod_{j=0}^{n_{\text{data}}} p(\mathbf{y}^{(j)}|\mathbf{x}^{(j)}, \theta).$$

We can return to a general view of the supervised learning framework is that the n_{data}-many observations we have from the data manifold accurately captures the joint distribution between the data and their labels. If this is true, then, given a new data point i.i.d. from the same distribution as $\mathcal{D}$, and a model which has minimized the error on the n_{data} previously seen points, we expect the model to perform well on this new, unlabeled point. Of course, this is exactly what we discussed as the primary goal of the supervised learning framework and that is a notion of good *generalization* to unseen data. Throughout the thesis we highlight that the notation f^{θ} refers to the output of a neural network with respect to a single parameter setting θ, where $f^{\boldsymbol{\theta}}$ refers to the output over the random variable $\boldsymbol{\theta}$ distributed according to $p(\theta|\mathcal{D})$.

3.3.3 Priors for Bayesian Neural Networks

In the Bayesian learning setting the prior distribution is meant to be chosen according to some practical or aesthetic functional bias. We have already pointed out that, given

a neural network architecture, the parameter setting θ fully determines the input-output mapping of the function. As such, a probability over θ induces a probability over input-output mappings (i.e., functions). The *prior* probability distribution over θ is the distribution over functions *before* we have seen any data. As such, the prior's theoretical role in the Bayesian learning framework is to express our beliefs about the set of functions that we think g (the function we aim to mimic in the supervised learning framework) might come from. For example, if we know some functional property about g (e.g., rotation invariance) before any data has been observed then we would encode that into our prior distribution by placing high probability on functions which exhibit such a property (e.g., rotation invariance).

Unfortunately, in deep learning there is not a establishing a correlation or method for completely understanding how a prior over θ corresponds to a prior over functions is still an open problem [143]. That is, it is difficult to compactly express or approximate all of the functions which may exhibit a desired property (e.g., rotation invariance or even smoothness). This difficulty understanding the precise relationship between a particular weight space setting and the function it corresponds to with a given neural network makes it challenging to make a principled choice of prior in Bayesian deep learning. In practice, one often selects an *uninformative* prior, which is a wide Gaussian prior with diagonal covariance. The intuition for this selection is that a Gaussian prior on each parameter of the network allows us to have some probability density over a wide range of values, and if the network is expressive this directly translates to a wide range of functions.

While in principle we would like to make a better selection for our prior, this is still an active and promising area of research in the Bayesian deep learning community. In [85] the authors use empirical Bayes to learn a weight prior from maximum likelihood estimates. In [156], the authors check weight-space samples for their function space properties and construct a prior based on weight settings with favorable functional properties. For deeper discussion of priors in Bayesian deep learning, we reference interested readers to the Thesis of *Nalisnick*, [109].

3.3.4 Approximate Bayesian Inference

Another major hindrance to the idealised application of Bayes' rule to neural network parameters, outside of the difficulties with the prior, is the reliance on marginalization over the parameters. Directly computing the unnormalized version of Bayes' theorem depends on us being able to compute an integral over the entire parameter space. Given both the size (in terms of number of parameters) and highly non-linear nature

of deep neural networks, makes this marginalization intractable in practice [50, 113]. In order to get around this, one must rely on approximate inference techniques. There are two primary forms of approximation that can be made. The first is to take a sample-based approximation. This involves exploring a Markov chain and sampling a finite number of neural networks proportionally to their probability in the true posterior. The second approximation that can be made is a variational one. This is where the modeller chooses a parametric distribution which one hopes is expressive enough to faithfully approximate the true posterior. In this case, one can approximate Bayes' rule by updating the parameters of this chosen distribution. We detail further specifics and developments below.

Markov Chain Monte Carlo Hamiltonian Monte Carlo (HMC) approximates the posterior by defining a Markov chain whose stationary distribution is $p(\theta|\mathcal{D})$. The primary advantage of such an approach is that in the limit of infinitely many samples, HMC will converge its stationary distribution, in this case, the true Bayesian posterior distribution. HMC proceeds by exploring the Markov Chain using Hamiltonian dynamics to improve efficiency of the exploration. This is achieved by alternating between sampling from the potential energy function, $U(\theta) = -\log(p(\theta))$, and moving around the weight space by following the dynamics described by a kinetic energy function, $K(v) = \sum_{i=1}^{n_{\text{params}}} v_i^2/(2m_i)$, given over auxiliary momentum variable v. The hyper-parameters m_i, the mass of the Hamiltonian system, has the empirical effect of trading exploration with exploitation of the weight space [114].

Despite its scalability issues, HMC is considered to be the gold standard of Bayesian inference for neural networks. This is due to its ability to capture the exact posterior in the limit of samples. Despite this, reliance on computing the gradient of the network's parameters w.r.t. the entire dataset (i.e., the full-data gradient) can become computationally infeasible as the number of input dimensions or the number of datapoints grows. More recently, MCMC methods have been approximated by relaxing the reliance on the computation of the full data gradient instead opting for stochastic gradients, which are used in conjunction with various correction terms to reduce potential errors in inference [32, 147], though these are not studied in this thesis.

Variational Inference (VI) Given that we cannot analytically compute the true posterior probability of our neural network parameters, a common approximation is to fit a variational posterior. This is done by taking a parameterized distribution

(e.g., a Guassian distribution) and attempting to infer the parameters of the distribution rather than the potentially-much-more complex true posterior over neural network parameters. Formally, we take an approximating distribution, $q_\omega(\theta)$, parameterized by ω and try to infer a parameter setting ω such that our posterior and variational approximation are as close as possible. This is the key difference between this approach and MCMC-based methods: variational inference methods replace approximate marginalization with approximate maximization. The measure of closeness of our variational posterior to the true posterior is measured by the Kullback–Leibler (KL) divergence [88]:

$$KL(q_\omega(\theta)||p(\theta|\mathcal{D})) = \int q_\omega(\theta) log\left(\frac{q_\omega(\theta)}{p(\theta|\mathcal{D})}\right) d\theta \tag{3.16}$$

Finding the value of ω which minimizes this divergence is not generally possible, but we can compute a local minimum [50]. This is typically done via the evidence lower-bound objective (ELBO) which is formulated [16]:

$$\int q_\omega(\theta) log\ p(\mathcal{D}|\theta) d\omega - KL(q_\omega(\theta)||p(\theta)) \tag{3.17}$$

$$= \mathbb{E}_{\theta\sim q_\omega(\theta)}[log\ p(\mathcal{D}|\theta)] - KL(q_\omega(\theta)||p(\theta)) \tag{3.18}$$

which is so-named due to the fact that it lower bounds the evidence term $p(\mathcal{D})$. Once we have computed an approximately optimal ω, we are able to use the variational posterior $q_\omega(\theta)$ as a surrogate for the true posteriors in all of our inference equations (e.g., the posterior predictive, Equation (3.14)).

While finding the optimal θ does not guarantee convergence to the true posterior of the BNN, variational methods have significant scalability advantages [115]. Moreover, a number of variational methods have been developed in the literature which have uncertainty and scalability properties that are promising for Bayesian deep learning.

In the following chapters, we will compare and contrast the scalable variational inference methods that are used in this thesis. Blundell et. al. [16] introduced Bayes by Backprop (BBB), a stochastic gradient method to update the parameters of the variational distribution using KL divergence. By computing the gradient of the of both mean and variance terms (by way of the reparameterization trick) with respect to the ELBO objective, BBB derives an update which provided an early and popular method for scalable variational inference for neural networks. In [162] and concurrently in [82], the authors devise a family of even more scalable inference algorithms, from which we focus on the Noisy Adam (NA) implementation given in [162]. In Noisy Adam (and Vadam in [82]), the authors achieve greater scalability updating

both mean and precision (inverse of the variance) by approximately computing the natural-gradient, the gradient of the ELBO scaled by the Fisher information matrix (FIM). A key practical observation of these algorithms was noticing that the popular optimizer Adam [84] was only a reparameterization trick and an approximate Hessian computation away from having updates corresponding to the natural gradient update for the variational posterior. In [115], the authors reintroduce the variational online Gauss-Newton (VOGN) algorithm originally proposed in [82], but make the approximation of the Hessian faster via parallelization. The final algorithm we employ in this thesis is the Stochastic Weight Averaging-Gaussian (SWAG) algorithm from [100]. Unlike the other methods, this algorithm simply collects the final n iterates from stochastic gradient descent and takes their sample mean and variance to be the approximating variational distribution. We note that in principle, this computation is a fairly harsh approximation to Bayesian learning principles.

Chapter 4

Defining Local Robustness for Bayesian Neural Networks

Contents

In this chapter, we formally introduce the problems that this thesis sets out to solve. In doing so, it is pertinent to cover further particulars and background of the Bayesian learning paradigm and its application to neural networks. The ultimate aim of this chapter is to bridge the concepts of local robustness and deep learning covered in the background chapter with Bayesian learning, and in doing so we formulate a theory of local robustness for Bayesian neural networks. We leave practical considerations such as computational methodology, trade-offs, guarantees, and particular parameters to later chapters.

We begin by recalling the definitions of local robustness and Bayesian neural networks and highlight some basic difficulties of straightforward application of local robustness in the Bayesian setting. We then introduce two distinct but related definitions of local robustness for Bayesian neural networks. The first, in Section 4.1, is probabilistic local robustness, which considers both the robustness and inherent

stochasticity of our Bayesian model, and the second, in Section 4.2.1, which considers the local robustness of the optimal Bayes' decision. Throughout, we focus on Examples 1 and 2 in order to develop intuitions and motivations for these notions of robustness.

4.1 Probabilistic Robustness of Bayesian Neural Networks

4.1.1 Defining Probabilistic Robustness

In order to connect Bayesian learning to adversarial robustness, we first recall the key facets of our definition of adversarial robustness for deterministic neural networks. Given a deterministic neural network f^θ and a local robustness property $\{T, S\}$ we have that f^θ is robust to the local robustness property iff:

$$f^\theta(\mathbf{x}') \in S \quad \forall \mathbf{x}' \in T$$

Of course, given that we would like to measure the local robustness of a Bayesian neural network, a natural first step is to try to simply swap the deterministic neural network in the above definition for a probabilistic (namely a Bayesian) neural network. Definition 1 is a binary predicate expressed about the behavior of a single function. In the Bayesian setting, we do not have a single function but instead a probability distribution over functions. Here, we are not concerned with making a binary statement about a distribution over functions. Instead, we can propagate the local robustness property through the probability over functions in order to arrive at what we define as *Probabilistic Local Robustness*:

Definition 6 (Probabilistic Local Robustness)**.** *Given a Bayesian neural network $f^{\boldsymbol{\theta}}$, where $\boldsymbol{\theta}$ is distributed according to $p(\theta|\mathcal{D})$, an input set $T \subset \mathbb{R}^{n_{in}}$ such that $\mathbf{x} \in T$ and a set $S \subseteq \mathbb{R}^{n_{out}}$ of safe outputs, we define probabilistic local robustness as*

$$Prob_{\theta \sim p(\theta|\mathcal{D})}\big(f^\theta(\mathbf{x}') \in S \quad \forall \mathbf{x}' \in T\big) \tag{4.1}$$

While probabilistic local robustness and the deterministic local robustness property are notationally very similar, they are semantically and practically distinct. This distinction is perhaps most obvious when we consider that deterministic local robustness is either 0 or 1 while probabilistic robustness can take any value between 0 and 1, inclusively. Semantically, probabilistic local robustness represents the probability that a function realized from a sample of the Bayesian neural network *model* satisfies

the local robustness property. As such, this definition can be seen as a measure of robustness for the Bayesian posterior that takes into account the stochasticity of the *model*. One of the first obvious drawbacks to this is that it is no longer clear if a network with a probabilistic local robustness of 0.513 is susceptible to adversarial example. The loss of direct correlation between deterministic local robustness and probabilistic local robustness is addressed in the next section. Here, we recall intuitive reasons to consider probabilistic robustness by first examining a pair of examples.

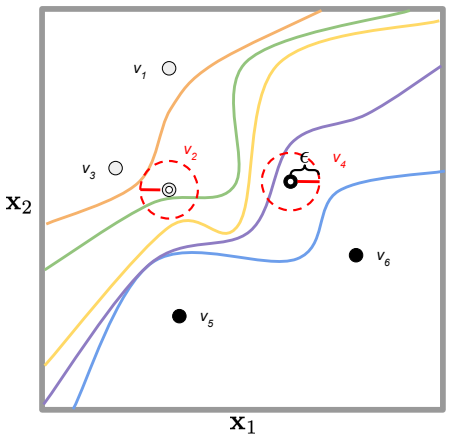
4.1.2 Examples, Intuition, and Motivation

In order to motivate and gain further intuition about probabilistic local robustness, we provide and discuss two running examples, one from classification and the second from regression. In each example we give both a visualization in Figure 4.1 as well as comprehensive breakdowns of how each classifier affects the probabilistic robustness of the Bayesian neural network.

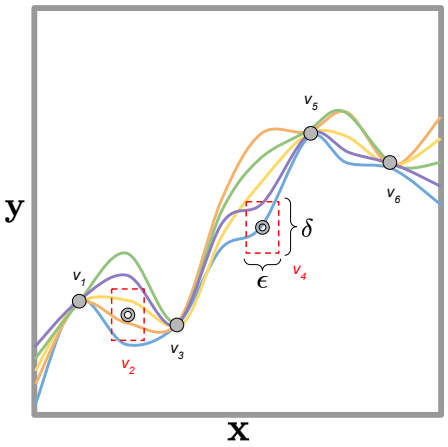
Classification Case In Figure 4.1a, we are given four training data points ($\mathcal{D} := \{v_1, v_3, v_5, v_6\}$) which pose a binary classification problem. The black dots and white dots are taken to be from some arbitrary positive and negative classes, respectively. The Bayesian ensemble is then comprised of the many colored lines which separate the plane into positive and negative halves, with the negative half being above and the positive half being below. We have represented the posterior as having discrete support on classifiers which have high likelihood, that is, they all perfectly split the given four data points.

We also plot two held out test set examples ($\{v_2, v_4\}$), which we may want to use in order to measure the robustness or generalization properties of the given Bayesian neural network. Marked as hollow points colored by their correct class, these two data points fall in a previously sparse region of the input space. As such, classifiers from the posterior given the original data have varying degrees of success classifying these points. We notice the orange classifier has a high bias towards predicting the positive (black) class while the blue classifier has a high bias towards predicting the negative (white) class, and that both classifiers are incorrect on one of the two test set examples. Further, we have the yellow, green, and purple classifiers, which correctly splits both the original data and the held out examples.

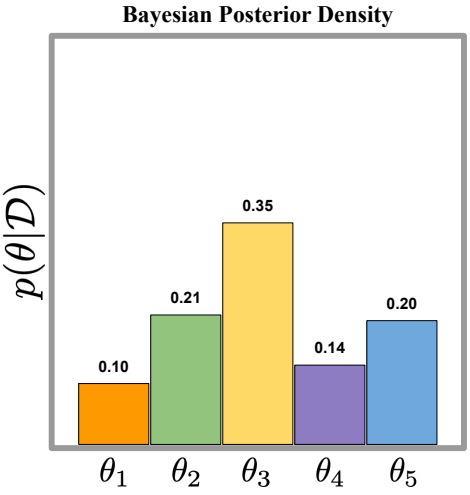
While the models have varying success classifying the point-wise class of the test inputs, we can further consider a local adversarial robustness property of the test



(a) An example of classification in the plane with a Bayesian neural network.



(b) An example of 1 dimensional regression with a Bayesian neural network.



(c) The posterior mass for the Bayesian neural network in our running examples.

Figure 4.1: Probabilistic robustness example figures. Subfigure (a) encapsulates a classification problem where each point is labelled either white or black. Subfigure (b) encapsulates a regression problem in which we are trying to predict a 1D output from 1D inputs. In both cases, unseen inputs are *donut* points where training points are solid. Each local robustness property has its boundaries outlined with dotted red lines.

inputs. In particular, we have visualized an ℓ_2 ball around each test input parameterized by some value ϵ. Because this is a classification problem, it is most natural to consider whether or not the classification is consistent inside of the given ball. One can visually check if the input-output constraints are satisfied by simply checking if the decision boundary intersects the ϵ-ball. If it does, then we know that the classification is not consistent inside of the ball.

Given that (in this example) we have a posterior ensemble with discrete support, we can enumerate the different contributions of each classifier to a points' local robustness. We highlight that for either point we choose the analysis remains the same albeit with different posterior probability values. We arbitrarily choose to compute the probabilistic robustness of the positive (black) class example which is labelled v_4. In order to compute probabilistic local robustness, it is necessary to marginalize over the entire ensemble i.e., probability density over the weights of a neural network (see Figure 4.1c). In this case, the ensemble is represented only in function space and not in weight space and so we can go through the color coded ensemble. Firstly, the blue network consistently assigns the wrong class inside of the ball and therefore does not contribute any probability to the local robustness. Similarly, the purple classifier despite being correct in a majority of the ball's mass is not totally consistent and therefore contributes nothing to the probabilistic robustness. Finally, the yellow, green, and orange classifiers are all consistent and correct inside of the ball and therefore would all be considered locally adversarial robust w.r.t. the black test input and the given specification. The yellow classifier contributes 0.35 probability mass to the probabilistic robustness, the green contributes 0.21 and the orange contributes 0.10, according to the discrete probability mass given in the legend Figure 4.1c. Thus, the probabilistic robustness of the positive (black) test set example w.r.t. the given discrete posterior probability mass is 0.66.

Regression Case In Figure 4.1b we pose a regression problem that is analogous to the classification problem explored in Figure 4.1a. In this problem we have a single feature dimension (labeled $\mathbf{x}$) and a single output dimension (labeled $\mathbf{y}$), and we are tasked with fitting functions to four given training examples (grey points, $\mathcal{D} := \{v_1, v_3, v_5, v_6\}$) and we would like to measure the probabilistic robustness of the model with two held out test set examples (grey donuts $\{v_2, v_4\}$). We have again represented a hypothetical Bayesian posterior with discrete support. The functions in this Bayesian ensemble have again been represented in function space and for

convenience include functions which (nearly) interpolate the training points and thus would have high likelihood on the observed data.

For each test set example, we consider a local robustness property consisting of ℓ_∞ balls in both the input and output spaces. In particular, we have some $\mathcal{B}_{\epsilon,\infty}$ in the input space and $\mathcal{B}_{\delta,\infty}$ in the output space. Thus, given a function f^θ and an input-output pair, $(\mathbf{x}, \mathbf{y})$, we would like to check that $\forall \mathbf{x}' \in \mathcal{B}_{\epsilon,\infty}(\mathbf{x}), f^\theta(\mathbf{x}') \in \mathcal{B}_{\delta,\infty}(\mathbf{y})$. A visual intuition is given for this by the red boxes outlined with dotted lines in Figure 4.1b.

Again, given our function-space visualization and that the Bayesian posterior has discrete support, we can walk through the computation of probabilistic robustness for this ensemble. Given the test input v_2, we can iterate through each function and decide its contribution to the probabilistic robustness. For each checked function, we would like to ensure that the input and output constraints are met and this amounts to ensuring that every portion of the function is within the bounds of the accepted $\mathbf{y}$ values (vertically) for every $\mathbf{x}$ value (horizontally). For example, we can see that both the yellow and orange functions respect the given property and contribute probability 0.35 and 0.10 to the robustness respectively. The blue and purple functions, while they are inside of the accepted output range for some $\mathbf{x}$ values, do not respect the property everywhere and thus contribute nothing to the probabilistic robustness as we must consider the *worst-case* for each function. Finally, the green function does not respect the property anywhere and therefore also contributes nothing to the ultimate probabilistic robustness. This leaves the following ensemble with a robustness of 0.45.

4.1.3 Applications

Connection to Uncertainty As we have established, one of the primary reasons to prefer a Bayesian neural network over a deterministic neural network is the ability to reason about uncertainty through the language of probability. When reasoning about uncertainty, it is common to consider the *variance* of our posterior predictive distribution. Where we take the variance to be the squared deviation between outputs and the expected output, we have that the uncertainty captured by the variance of the posterior predictive is given by:

$$Var_{\theta \sim p(\theta|\mathcal{D})}(f^\theta(\mathbf{x})) := \mathbb{E}_{p(\theta|\mathcal{D})}[f^\theta(\mathbf{x}) - \mathbb{E}_{p(\theta|\mathcal{D})}[f^\theta(\mathbf{x})]]$$

A related quantity is the variation ratio which is defined as:

$$\frac{1}{M}\sum_{i=0}^{M} \mathbb{I}[f^{\theta_i}(\mathbf{x}) - \mathbb{E}_{\theta \sim p(\theta|\mathcal{D})}[f^\theta(\mathbf{x})] > \delta] \xrightarrow[M\to\infty]{} Prob_{\theta \sim p(\theta|\mathcal{D})}(f^\theta(\mathbf{x}) - \mathbb{E}_{\theta \sim p(\theta|\mathcal{D})}[f^\theta(\mathbf{x})] > \delta)$$

where $\mathbb{I}$ is the indicator function returning 1 where the inequality is true and 0 otherwise.

Intuitively, this is the probability that the output of a given function from the posterior deviates from the mean classification by more than a user-selected quantity δ. Interestingly, this property has been used in Bayesian learning (in both Gaussian Processes and Bayesian neural networks) to flag adversarial examples, though with mixed success [62]. Our notion of probabilistic robustness can be related to this quantity with relative ease. Where we take the output property we would like to test to be $S = \mathcal{B}_{\delta,\mathbb{E}[f^\theta(\mathbf{x})]}$, we recover a version of the worst-case variance:

$$Prob_{\theta\sim p(\theta|\mathcal{D})}(f^\theta(\mathbf{x}') \in \mathcal{B}_{\delta,\mathbb{E}_{\theta\sim p(\theta|\mathcal{D})}[f^\theta(\mathbf{x})]} \quad \forall \mathbf{x}' \in T) \tag{4.2}$$

$$= Prob_{\theta\sim p(\theta|\mathcal{D})}(f^\theta(\mathbf{x}') - \mathbb{E}_{\theta\sim p(\theta|\mathcal{D})}[f^\theta(\mathbf{x})] < \delta \quad \forall \mathbf{x}' \in T) \tag{4.3}$$

which is precisely the same as the variation ratio proposed for flagging adversarial examples, except that we have removed the point-wise dependency on a single input $\mathbf{x}$ and replaced it with the input property of interest T. So, under certain conditions, it is intuitive to consider the probabilistic robustness of a Bayesian classifier as a local extension of a point-wise uncertainty measure.

Application to Model Selection We have noted that computation of probabilistic robustness is semantically interpreted as the probability that a given function sampled from the posterior is robust. As such, it can be seen as a metric that allows us to reason about the composition of the posterior (in terms of its robustness properties). Thus, it may be valuable as a way of comparing and selecting preferable posterior distributions. If a the posterior for model A has higher probabilistic robustness than that of model B then we necessarily know that it assigns higher probability to locally robust functions and on that basis we may chose to deploy model A over model B.

4.2 Decision Robustness for Bayesian Neural Networks

In the previous section, we established a straightforward extension of the local robustness property to probabilistic classifiers and showed how such a definition can be useful as an extension of point-wise uncertainty to the adversarial setting. One lingering drawback to this definition, though, is its lack of correlation to adversarial examples as presented in Definition 2. This is as a consequence of the fact that

probabilistic robustness given in Definition 6 is connected to the stochasticity of our posterior beliefs (e.g. the epistemic uncertainty) and not to the ultimate decision of the network. Given posterior beliefs over the parameters of our neural network, irrespective of the methods by which we arrive at such beliefs, the ultimate decision made on an unseen sample is arrived at through reasoning about the loss of outputs according to their posterior predictive probabilities. Thus, decisions are not exclusively reliant on the stochastic properties of the model ensemble representing our beliefs.

In this section, we highlight this difference by drawing our attention to the difference between a *loss* function and a *likelihood* in the Bayesian framework. We then give the definition of decision robustness for a Bayesian neural network which is ultimately a direct analogue to the definition of local robustness for deterministic neural networks. We then cover further examples and motivations for studying this quantity.

4.2.1 Bayesian Decision Theory

When providing exposition on Bayesian learning learning, we focused exclusively on the *likelihood* function as a method for measuring the goodness-of-fit for a particular model. In this section, we make the distinction between the role of the likelihood and the loss function in the Bayesian learning paradigm. An important line of relatively recent work establishes generalized Bayesian posteriors that are the result of connecting information in our observations to updates about our beliefs through a loss function rather than a likelihood [12, 13]. The distinction we draw upon here calls on the fact that, in the Bayesian framework, the methods by which we update our beliefs and the methods by which we reach a decision are *potentially* distinct and must be treated separately.

The *likelihood*, $p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta})$, measures how likely (probabilistically speaking) we are to observe the desired outputs $\mathbf{Y}$ under the stochasticity of our model parameters and observational noise. On the other hand, the *loss* function in a Bayesian setting defines the *badness* of making an incorrect decision. We emphasize that understanding how likely we are to observe a given output value is distinct from quantifying the badness of an incorrect prediction and this is distinction between the *likelihood* and the *loss*. By quantifying how good or bad a given output is w.r.t. a desired output (i.e., defining a *loss* function), we can reason about what the best decision is to make given our uncertainty about our model and our observations. In [12] the authors derive *generalized* posteriors, where they argue that it is reasonable to use a loss function not only in a decision-theoretic capacity, but also as a means of connecting information in the data to the parameters of interest (in our case the distribution over

parameters of a Bayesian neural network). In these cases, a loss function, which is not required to satisfy the properties of a probability distribution, is used not only to measure the goodness or badness of parameters of interest, but also during decision making.

Making this decision in an optimal way is the subject of Bayesian decision theory. Specifically, the problem of making an optimal prediction is to find the best guess $\hat{\mathbf{y}}$ given our beliefs about the output at an unseen input $\mathbf{x}$ (captured by the posterior predictive distribution, Equation (3.14)) as well as a loss function, $\mathcal{L}$, which quantifies the consequences of an incorrect decision. The risk of a given decision, $\mathcal{R}$, is the cumulative loss of a given prediction weighted by the likelihood of that prediction being wrong according to our posterior predictive distribution. In order to make the best decision for an unknown input, we minimize the risk. That is, we would like to find a prediction, $\hat{\mathbf{y}}$, that minimizes the expected loss:

$$\mathcal{R}_{\mathcal{L}}(\hat{\mathbf{y}}|\mathbf{x}) := \int \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) p(\mathbf{y}|\mathcal{D}, \mathbf{x}) d\mathbf{y}. \tag{4.4}$$

We highlight that though the notation for loss, $\mathcal{L}$, is equivalent to that which was used during our discussion of deterministic networks and adversarial examples, the loss here is chosen by the modeller and is not necessarily connected to the loss discussed previously.

Intuitively, optimizing for the expected risk is to find the prediction ($\hat{\mathbf{y}}$) which is quantitative the least bad when we compare it (using $\mathcal{L}$) to every potential true value $\mathbf{y}$ weighted by the probability of $\mathbf{y}$ under our posterior predictive distribution. As noted in [122], the chosen loss function need not have anything to do with the likelihood function. In particular, the loss function may be chosen to suit the needs of a particular application. In order to demonstrate this, we outline some particular loss functions and the decision-theoretic quantity which minimizes the expected risk.

4.2.2 Definition of Bayesian Decision Robustness

In order to compute the robustness of the decision of our Bayesian model and given a loss function, we must propagate our worst-case analysis through the expected loss function as well:

Definition 7 (Bayesian Decision Robustness)**.** *Given a Bayesian neural network $f^{\boldsymbol{\theta}}$ ($\boldsymbol{\theta}$ distributed according to $p(\theta|\mathcal{D})$), a loss function $\mathcal{L}$, an input set $T \subset \mathbb{R}^{n_{in}}$ such*

that $\mathbf{x} \in T$ *and a set* $S \subseteq \mathbb{R}^{n_{out}}$ *of safe outputs, the Bayesian decision is considered to be robust if the following property holds:*

$$\arg\min_{\hat{\mathbf{y}}} \mathcal{R}_{\mathcal{L}}(\hat{\mathbf{y}}|\mathbf{x}', \boldsymbol{\theta}) \in S \quad \forall \mathbf{x}' \in T \tag{4.5}$$

Intuitively, this definition allows us to ensure that, for every point in the input set, the output decision of the Bayesian model, according to the minimizer of the given loss function, is within the safe set S. We now consider how this definitions shifts our perspective on the robustness of the Bayesian ensembles considered in Figure 4.1.

Regression Losses For supervised regression problems, perhaps the most commonly considered loss is the mean squared error or ℓ_2 loss function: $||\mathbf{y} - \hat{\mathbf{y}}||_2$. This encodes the belief that the consequences of an incorrect predictions increase according to the Euclidean distance from the true prediction. It is well known that, in this case, given a posterior predictive distribution $p(\mathbf{y}|\mathbf{x}, \mathbf{X}, \mathbf{Y})$ choosing $\hat{\mathbf{y}}$ to be the mean (equivalently, expected value) of $p(\mathbf{y}|\mathbf{x}, \mathbf{X}, \mathbf{Y})$ minimizes the expected loss (Equation (4.4)). On the other hand, if one believes that incorrect predictions should be penalized according to the ℓ_1 loss function, then it is the median of $p(\mathbf{y}|\mathbf{x}, \mathbf{X}, \mathbf{Y})$ that minimizes the expected loss. For proof that these minimize Equation (4.4) see Section 4.4.2 of [8]. As noted in [122], for symmetric posterior predictive distributions (such as Gaussian or Laplace distributions), the mean and median coincide; however, in the case of Bayesian neural networks we are not guaranteed that the posterior predictive distribution is Gaussian. This highlights that in the case of BNNs it is possible (in fact, likely) that the mean and median of the posterior predictive do *not* coincide and thus the choice of loss function has immediate ramifications regarding the ultimate performance of the model in practice. For a comprehensive treatment of Bayesian decision theory, we reference interested readers to [8].

Classification Losses Perhaps the most common loss function to use in the classification case is the zero-one loss [122]. This simply encodes a loss of 1 for incorrect predictions and a loss of 0 for correct predictions. In classification, we often have that the softmax output of the Bayesian neural network, $f^{\theta}(\mathbf{x})$, is interpreted as the parameters of multinoulli distribution in which each of the entries of the distribution represents the probability that $\mathbf{x}$ belongs to a particular class. As in Equation (3.4), we denote by $f_c^{\theta}(\mathbf{x})$ the probability that $\mathbf{x}$ belongs to class c given the particular parameter setting θ. By marginalizing over the weights, we have, $\mathbb{E}_{\theta \sim p(\theta|\mathcal{D})}[f_c^{\theta}(\mathbf{x})]$, the probability of class c according to the expectation of the Bayesian posterior. Given

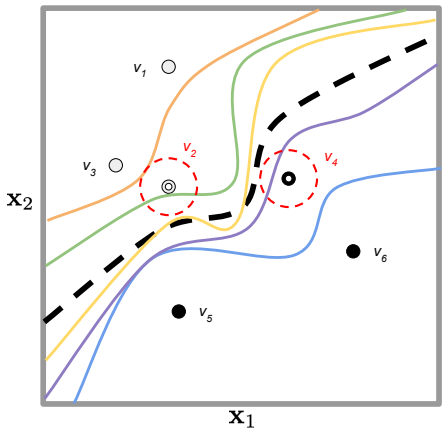
that we pay a unit loss for an incorrect decision, it is intuitive that the decision that minimizes the expected loss (according to our posterior belief) is to label $\mathbf{x}$ as the class c that maximizes $\mathbb{E}_{\theta \sim p(\theta|\mathcal{D})}[f_c^{\theta}(\mathbf{x})]$. See Section 4.4.3 of [8] for proof of this result. This decision criteria induces what is known as the *Bayes Classifier* and, because the Bayes classifier is the result of marginalizing over the weights, we highlight that the Bayes classifier is a deterministic classifier. That is, given a Bayesian neural network $f^{\boldsymbol{\theta}}$ marginalizing over $p(\theta|\mathcal{D})$ for each input, there is a unique, non-stochastic classification c (ties can be broken according to an arbitrary, deterministic rule). Therefore, just as with deterministic neural networks, the Bayes classifier induces a deterministic decision boundary.

4.2.3 Examples, Intuition, and Motivation

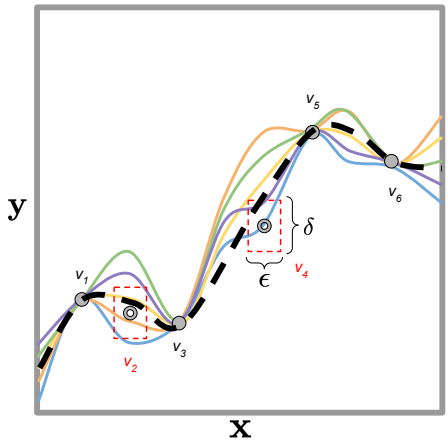
In this section, we augment our visualization of the discrete Bayesian posteriors used to demonstrate probabilistic robustness (Figure 4.1) with the expectation in function space, which allows us to discuss different properties of the decision robustness of a Bayesian neural network in both the classification and regression settings.

Classification Case In Figure 4.2a, we revisit our hypothetical classification in the plane. Again, we give four training points ($\mathcal{D} := \{v_1, v_3, v_5, v_6\}$), two from each class: black (positive) and white (negative). We also give two test set examples, one from each class, which are plotted as donuts ($\{v_2, v_4\}$). We represent our Bayesian posterior ensemble as a series of colored lines. Figure 4.2a differs from Figure 4.1a in that we have added a dashed black line which represents the decision surface of our Bayesian ensemble according to a zero-one loss. This dashed black line separates the plane into what are predicted to be positive (below the line) and negative (above the line).

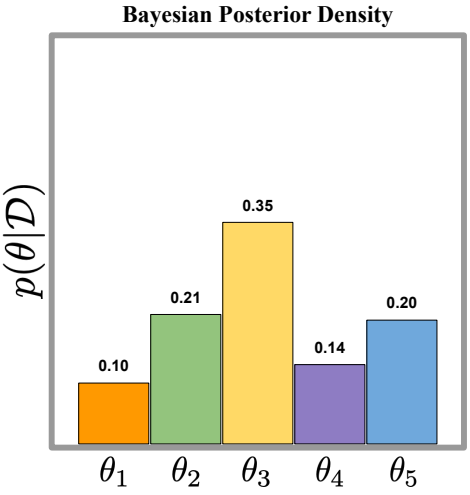
Again, in this figure we consider the same local robustness property in the input space; however, now we are tasked with considering the robustness of our Bayes classifier (the dashed black line). Recall that for v_4, we computed a probabilistic robustness of 0.66 for this Bayesian posterior. For the decision robustness, we only have binary outcomes, either the Bayesian decision surface respects the output constraint everywhere in the input set (decision robustness = 1), or there exists at least one point inside of the input set such that the output constraint is violated (decision robustness = 0). In the case of the positive test set example in Figure 4.2a, we notice that, despite being very close to the decision boundary, no point crosses the decision boundary. Thus, the decision of the Bayes classifier is robust (i.e = 1) with respect



(a) An example of classification in the plane with a Bayesian neural network and the mean of the Bayesian neural network.



(b) An example of 1 dimensional regression with an Bayesian neural network and the mean of the Bayesian neural network.



(c) The posterior mass for the Bayesian neural network in our running examples.

Figure 4.2: Decision robustness example figures. Subfigure (a) encapsulates the classification case where each point is labelled either white or black. Subfigure (b) encapsulates the regression case in which we are trying to predict a 1D output from 1D inputs. In both cases, unseen inputs are *donut* points where training points are solid. In each case we plot the expectation of the model ensemble as a dotted black line. Each local robustness property has its boundaries outlined with dotted red lines.

to this input and local robustness property. The probabilistic robustness being 1 at this point allows us to say with certainty that there does not exist any adversarial examples within the input boundary considered. This is not a statement that can be made (directly) from the probabilistic robustness. Notice that though probabilistic robustness of the model at this point is less than one (0.66), the model is nonetheless robust at this point.

Using Figure 4.2a, we also highlight the apparent protection that comes with using Bayesian model averaging to make a decision. Specifically, we draw attention to the fact that a majority of the deterministic neural networks included in the Bayesian ensemble (i.e., θ_1, θ_2, θ_4, θ_5) are not robust to the local specifications of interest. Yet, the Bayesian model average of these classifiers *is* robust w.r.t. both specifications of interest. Moreover, note that the non-robust classifiers would have captured nearly perfect loss under the training data and therefore would have reasonably been selected under the frequentest learning paradigm despite their poorly calibrated performance (and robustness) in previously sparse regions of the data manifold.

Regression Case In Figure 4.2b, we revisit the one dimensional regression problem explored in Figure 4.1b. We give a function-space view of a hypothetical Bayesian posterior with discrete support where each model, but, in this plot we give the mean of the Bayesian posterior (used for making decisions) as a black dashed line.

We consider the same input property as before, namely, an ℓ_∞ ball with radius ϵ. In the output, however, we are only concerned with whether or not the Bayesian decision varies from the test point by more than δ for any point inside of the input set. It is easy to see that for v_4 that this is not true: the right tail of the line exceeds the output specification. For v_2, however, the decision of the model stays within the specified output range for all points in the input range. Thus, for v_2, we say our model decision is robust (i.e., $= 1$). Given that we have confirmed decision robustness, we can be certain that, for the input and output specification, there are no adversarial examples for this model. Again, like with the classification case, we highlight that the probabilistic robustness at the right most test point was not 1. In fact, it was much less than 1, 0.45. In the next section, we consider when we can conclude decision robustness from high values of probabilistic robustness.

4.3 Summary

In this chapter, we started by considering the notion of a local robustness property for deterministic neural networks. By considering local robustness properties through the lens of Bayesian learning, we arrived at two natural extensions of local robustness properties: probabilistic robustness and Bayesian decision robustness. The former considers the composition of the model and quantifies the probability that a model drawn from a Bayesian posterior respects a given local robustness property. Further, we demonstrated this in a toy setting with a discrete support posterior and discussed its connection to both uncertainty and Bayesian modelling of stochastic processes. Next, we considered verifying Bayesian neural network decisions against adversarial examples. We introduced core concepts in Bayesian decision theory such as the seperation between likelihood and loss, and then discussed the need to propagate local robustness through a decision-theoretic quantity such as the the expectation of the posterior predictive distribution.

In the subsequent two chapters we will move from a toy setting with discrete support to real Bayesian neural networks and will focus on computing these safety-critical quantities with statistical and probabilistic guarantees.

Chapter 5

Statistical Guarantees on Adversarial Robustness of Bayesian Neural Networks

Contents

In this chapter, we are concerned with practical computation of local probabilistic and decision robustness of arbitrary Bayesian neural networks with statistical guarantees. In particular, we begin by defining what we mean by statistical guarantees and the problem of controlling a statistical estimates absolute error and confidence

with *a priori* guarantees. We then discuss the development of statistical estimators for adversarial robustness quantities of interest for Bayesian neural networks. We build on a correspondence between weight-space and function-space probability mass in order to develop our estimators. We then discuss perspectives on computing approximate robustness quantities with the desired guarantees and state concrete and general algorithms for their computation. Finally, we perform empirical analysis of different Bayesian neural networks on several data sets.

5.1 On Statistical Guarantees

In both computing and science at large it is important to understand the accuracy of experimental results. A tremendous amount of effort has been made towards understanding how to efficiently design experiments in order to have some confidence of correctness in our results. One setting which has been studied extensively is computing properties via Monte Carlo simulation of a random variable with support on $[0, 1]$. In this setting, we assume we have a random variable $\boldsymbol{Z}$ and that we would like to estimate the mean. Our information about $\boldsymbol{Z}$ comes from a set of Monte Carlo simulations: $\mathbf{Z}_1, \mathbf{Z}_2 \ldots \mathbf{Z}_n \sim \boldsymbol{Z}$. To understand how accurately these n simulations of the random variable characterize the mean, we consider the standard sample mean estimator:

$$\hat{\mu}_{\boldsymbol{Z}} := \frac{1}{n} \sum_{i=0}^{n} \mathbf{Z}_i$$

. We call $\hat{\mu}_{\boldsymbol{Z}}$ an empirical estimate of the mean, and we are interested in designing some way of understanding how close this empirical estimate is to the true value $\mu_{\boldsymbol{Z}}$. In part, understanding how close these two values are will depend on how we measure closeness. There are two common ways of measuring closeness of an estimate to the true value: the absolute error and the relative error. The absolute error is straightforward:

$$|\hat{\mu}_{\boldsymbol{Z}} - \mu_{\boldsymbol{Z}}|$$

This is, the absolute value of the difference between the values. However, there are some instances in which $\mu_{\boldsymbol{Z}}$ is very small and we would like to control the relative error of our approximation which can be defined as: $\dfrac{\hat{\mu}_{\boldsymbol{Z}}}{\mu_{\boldsymbol{Z}}} - 1$. Whether we are looking to control the relative error or absolute error of our estimate, the statistical guarantees take the same form, which is an *a priori* (ϵ, δ)-guarantee. We say *a priori* as the parameters of the guarantee are user-given values that are established before the Monte Carlo simulations are collected. The value of ϵ provided by the user is a

maximum tolerable error in our estimate (regardless of whether it is the absolute or relative error). The δ provided by the user is a maximum tolerable probability of error (also called the confidence). In other words, δ controls the maximum probability with which a given estimate falls outside of the error range allowed by the user selected ϵ value. We can write down the guarantee as:

$$Prob(||\hat{\mu}_{\boldsymbol{Z}} - \mu_{\boldsymbol{Z}}|| > \epsilon) \leq \delta,$$

and equivalently for the relative error:

$$Prob\left(\frac{\hat{\mu}_{\boldsymbol{Z}}}{\mu_{\boldsymbol{Z}}} - 1 > \epsilon\right) \leq \delta$$

Application to Robustness Estimation In the remainder of this chapter, we will develop statistical estimators for the probabilistic robustness and decision robustness of a Bayesian neural network. By controlling the absolute or relative error and confidence of our estimators we can make concrete statements with (ϵ, δ) statistical guarantees about the robustness of our probabilistic models. When considering decision robustness, these (ϵ, δ) guarantees allow us to make statements about robustness properties with control over how rigorous our estimates are. Such guarantees are of intrinsic value when considering the domains in which adversarial robustness is a concern. For example, in medical diagnosis one may want to consider the worst-case prediction error before issuing a treatment. Similarly, in self-driving cars one may wish to have confidence that the prediction of a Bayesian neural network control unit or perception unit has a level of robustness and certainty before making a decision. Through (ϵ, δ) guarantees on the local robustness (either probabilistic or decision variants), we can enable users of Bayesian neural networks to establish confidence in the adversarial robustness of the outputs of their models.

5.2 Statistical Estimators for Robustness of Bayesian Neural Networks

In this section we start by making a key observation about the nature of samples from Bayesian neural networks. We then recall the definitions of robustness from Chapter 4 and leverage our observation in order to develop statistical estimators for each definition of robustness.

5.2.1 Weight-Space Function-Space Correspondence

An unfortunate facet of evaluating local robustness properties of neural networks is the lack of a convenient relationship between properties of the weights (e.g. norm of the weights) and the local robustness of the function space properties of the network. This issue makes evaluating and verifying the local robustness of both deterministic and Bayesian neural networks difficult. For Bayesian neural networks, the lack of a simple relationship is challenging as probabilities must be measured in the weight space and properties must measured in the function space. In this chapter, we will get around this lack of simple correspondence by reasoning about weights sampled from the posterior. As noted in Section 5.1, computing via Monte Carlo consists of sampling a probability space and updating an estimator for the quantity under consideration. When we consider a Bayesian neural network we know that, by definition, the operation of sampling a parameter vector θ from a given posterior distribution $p(\theta|\mathcal{D})$ returns neural network functions proportionally to their posterior probability. While this statement is straightforward, it is important in the context of statistical estimation of robustness properties. Recall that, given a neural network architecture, the parameter setting θ uniquely determines the input-output mapping, and for a fixed θ we have a deterministic neural network. Combining the previous two observations we have that by sampling parameter vectors we are sampling fixed deterministic neural networks which we observe proportionally to their posterior probability. In the following two subsections we will leverage this to build statistical estimators for Definitions 6 and 7.

5.2.2 Statistical Estimator for Probabilistic Robustness

Recall the formulation of Definition 6 (probabilistic robustness for Bayesian neural networks) wherein we have a posterior $p(\theta|\mathcal{D})$, and a local property, the property of robustness or safety is defined as:

$$Prob_{\text{safe}} = Prob_{\theta \sim p(\theta|\mathcal{D})}(f^{\theta}(\mathbf{x}') \in S \quad \forall \mathbf{x}' \in T)$$

Given that the (unknown) probability of local robustness of our Bayesian neural network posterior is a real value $\phi^{\text{prob}} \in [0, 1]$, we would like to set up an estimator $\hat{\phi}_n^{\text{prob}}$ which approximates ϕ^{prob} with n samples from the Bayesian posterior. We define

the estimator as:

$$\hat{\phi}_n^{\text{prob}} := \frac{1}{n}\sum_{i=0}^{n} \mathbb{I}[f^{\theta_i}(\mathbf{x}') \in S \quad \forall \mathbf{x}' \in T] \tag{5.1}$$

$$\lim_{n\to\infty} \hat{\phi}_n^{\text{prob}} = \phi^{\text{prob}} = Prob_{\text{safe}} \tag{5.2}$$

where the function $\mathbb{I}$ evaluates the Boolean local robustness expression to be in $\{0, 1\}$. The estimator $\hat{\phi}_n^{\text{prob}}$ is simply a Bernoulli random variable whose mean is the probability that a sample from a Bayesian neural network posterior respects the given local robustness constraint.

The advantage of this statistical estimator is that it gives us a clear-cut blueprint for the computation of probabilistic robustness for a Bayesian neural network: sample n different weight parameters from your ensemble, check each one for a violation of the local robustness criteria, and keep track of the proportion of networks that satisfy the criteria. That proportion becomes your robustness probability.

5.2.3 Statistical Estimator for Decision Robustness

In this section, we start by recalling the definition of Bayesian *decision* robustness established in the previous chapter. We then define a statistical estimator for this quantity and give an illustrative example recalling the classification example from Chapter 4. Finally, we discuss some of the advantages and disadvantages of statistical estimation of decision robustness.

In Definition 7, we established that given a posterior $p(\theta|\mathcal{D})$, a local property, and a loss function $\mathcal{L}$, the property of Bayesian decision robustness is defined as:

$$\arg\min_{\hat{\mathbf{y}}} \mathcal{R}_{\mathcal{L}}(\hat{\mathbf{y}}|\mathbf{x}') \in S \quad \forall \mathbf{x}' \in T$$

Unlike the estimator for probabilistic robustness, which sees us constantly checking if a value is in the safe set defined by our local robustness property, decision robustness is a binary predicate which requires us to check the decision for every point in T. As this is a more complicated property to estimate, we present the logic in a step-by-step fashion below.

To compute decision robustness, the definition requires a posterior $p(\theta|\mathbf{X}, \mathbf{Y})$, a loss function $\mathcal{L}$, a test input $\mathbf{x}$, and a local robustness specification $\{T, S\}$. We recall that one computes the risk or expected loss as:

$$\mathcal{R}_{\mathcal{L}}(\hat{\mathbf{y}}|\mathbf{x}) := \int \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) p(\mathbf{y}|\mathbf{x}, \mathbf{X}, \mathbf{Y}) d\mathbf{y}. \tag{5.3}$$

this can be expanded by considering the full form of the posterior predictive:

$$\mathcal{R}_{\mathcal{L}}(\hat{\mathbf{y}}|\mathbf{x}, \boldsymbol{\theta}) := \int\int \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) p(\mathbf{y}|\mathbf{x}, \theta) p(\theta|\mathbf{X}, \mathbf{Y}) d\theta d\mathbf{y}. \tag{5.4}$$

Firstly, notice that the only term affected by an adversarial change to the test point $\mathbf{x}$ in this formulation is the likelihood. Neither the loss nor the posterior distribution depend on $\mathbf{x}$. Now, as Definition 7 is concerned with changes to $\hat{\mathbf{y}}$, we would like to compute is the largest change in $\hat{\mathbf{y}}$. If the largest change in the decision is achieved by $\hat{\mathbf{y}}_{\min}$, and if $\hat{\mathbf{y}}_{\min} \in S$, then we know that decision robustness is satisfied according to Definition 7. To demonstrate how this is computed, we first assume w.l.o.g. that $\mathcal{L}$ is the zero-one loss function, then Bayesian decision theory prescribes that the optimal decision is given by the class which has the largest probability in the multinoulli given by the mean of the posterior predictive distribution $\hat{\mathbf{y}} = \arg\max_{c \in [n_{out}]} \mathbb{E}_{\theta \sim \boldsymbol{\theta}} p(\mathbf{y} = c|\mathbf{x}, \theta)$. This is because of the following fact:

$$\arg\min_{\hat{\mathbf{y}}} \mathcal{R}_{\mathcal{L}}(\hat{\mathbf{y}}|\mathbf{x}) = \arg\max_{c \in [n_{out}]} \mathbb{E}_{\theta \sim p(\theta|\mathcal{D})} p(\mathbf{y} = c|\mathbf{x}, \theta) \tag{5.5}$$

where, again, in classification one would also take the argmax of the right-hand side. For brief discussion of estimators see [122]; specifically, Section 2.4 for regression and Section 3.1.1 for classification. For proofs of these results one can reference [8], specifically Sections 4.4.2 for regression estimators and 4.4.3 for the classification estimators. Given that we would like to compute the largest change in $\hat{\mathbf{y}}$ that we can achieve through modification of the input, we want to compute:

$$\max_{\mathbf{x}' \in T} \left(|| \arg\min_{\hat{\mathbf{y}}} \mathcal{R}_{\mathcal{L}}(\hat{\mathbf{y}}|\mathbf{x}) - \arg\min_{\hat{\mathbf{y}}} \mathcal{R}_{\mathcal{L}}(\hat{\mathbf{y}}|\mathbf{x}') ||_p \right) \tag{5.6}$$

By simply plugging in from Equation (5.5), we get that largest change in the decision can be written as:

$$\max_{\mathbf{x}' \in T} \left(|| \max_{c \in [n_{out}]} \mathbb{E}_{\theta \sim p(\theta|\mathcal{D})} p(\mathbf{y}|\mathbf{x}, \theta) - \max_{c \in [n_{out}]} \mathbb{E}_{\theta \sim p(\theta|\mathcal{D})} p(\mathbf{y}|\mathbf{x}', \theta) ||_p \right) \tag{5.7}$$

We can simplify this optimization further by assuming a classification setting and that we would like to compute the worst-case for a given class c. Then, we have that the likelihood is $p(\mathbf{y}|\mathbf{x}, \theta) = f_c^{\theta}(\mathbf{x})$, and the above optimization problem simplifies to computing:

$$\hat{\mathbf{y}}_{\min} = \min_{\mathbf{x}' \in T} \mathbb{E}_{\theta \sim p(\theta|\mathcal{D})} f_c^{\theta}(\mathbf{x}') \tag{5.8}$$

After computing $\hat{\mathbf{y}}_{\min}$, we must check that $\hat{\mathbf{y}}_{\min}$ is in S. If this is the case then we know that $\forall \mathbf{x} \in T \quad \hat{\mathbf{y}} \in S$. What we have now shown is that by estimating the

expectation $\hat{\mathbf{y}}_{\min}$ with *a priori* statistical guarantees, and by checking $\hat{\mathbf{y}}_{\min} \in S$ we can compute the satisfaction of Definition 7 with statistical guarantees. Notice that $\hat{\mathbf{y}}_{\min}$ is the mean of a random variable for which we can design an estimator. By assuming we have access to $\mathbf{x}_{\text{adv}}$ that minimizes Equation (5.8), we can write out the the estimator from n samples of the posterior as:

$$\hat{\mathbf{y}}_{\min} = \frac{1}{n} \sum_{i=0}^{n} p(\mathbf{y}|\mathbf{x}_{\text{adv}}, \theta_i) \tag{5.9}$$

$$\hat{\phi}_n^{\text{dec}} = \mathbb{I}[\hat{\mathbf{y}}_{\min} \in S] \tag{5.10}$$

Where, again, the function $\mathbb{I}$ evaluates the Boolean local robustness expression to be in $\{0, 1\}$. While this estimator also provides an intuitive blueprint for computing decision robustness, there are a few important practical considerations for this quantity.

5.2.4 Practical Computation of Estimators

Both of the above estimators suggest algorithms for computation of the key robustness properties of Bayesian neural networks based on sampling deterministic functions from the posterior. While these procedures seem straightforward, there are two primary questions which need answering before we can practically compute these values. Namely, the reliance on different *worst-case* output quantities in both estimators as well as the necessity for an infinite number of samples in order to exactly compute the desired robustness value are practically problematic. The latter requirement is dealt with in the following section. In this section, we address how to practically achieve the necessary worst-case quantities to compute valuable approximations of the robustness of Bayesian neural networks. More specifically, we examine how to leverage popular worst-case output approximations in our estimators and the implications for the final value in terms of soundness.

Adversarial Examples In Section 3.2.2.1 we established that, through the use of a differentiable loss function $\mathcal{L}$, one could use gradient-based optimization in order to approximately compute a worst case $\mathbf{x}'$ from a given, typically compact, input set T. By checking the approximate worst-case from such an attack, one arrives at an unsound but complete method of checking robustness (equivalently, a falsification method). Despite the lack of general guarantees provided by attacks, they have some significant advantages. Firstly, they are very efficient to compute, sometimes only requiring the equivalent of two forward passes through the network architecture.

Secondly, their unsound and complete nature means we only get credible information about robustness of a network if the attack is successful, and the lack of robustness of neural networks means this is often the case. The conjunction of these two upsides makes adversarial attacks a fast and cheap way of potentially obtaining a great deal of information about the *lack* of robustness of a given Bayesian neural network posterior.

In the context of probabilistic robustness, using adversarial examples would result in the following procedure: sample a model, compute an adversarial example, check the model's robustness to this attack. This can then be interpreted as the proportion of models which, if attacked, would be robust to unsafe changes in output.

For attacking the decision of a Bayesian neural network, however, we can reason that using an identical procedure would lead to an inappropriately conservative robustness estimate. Consider that the Bayesian decision is a point-wise quantity, and the previous procedure uses a different attack (and thus a different input point) for each network. Tailoring each attack to each sampled network would result in a much more effective attack than one that could be practically achievable. The correct way to estimate the statistical decision robustness of a Bayesian neural network is to sample m different networks which one jointly optimizes over to arrive at a single adversarial input. If we denote this adversarial example $\mathbf{x}^*$ for the Bayesian posterior, and subsequently take $\hat{\mathbf{y}}_{\text{adv}} = \mathbb{E}_{\theta \sim p(\theta|\mathcal{D})} f^{\theta}(\mathbf{x}^*)$ to be the approximate minimization of Equation (5.8), then we have arrived at a statistically sound estimate of the worst-case decision achievable by the adversary.

Convex Relaxation Where adversarial examples are both unsound and incomplete, convex relaxations offer soundness at the cost of over-approximation (and sometimes greater computational expense). Here, we start by recalling the interval bound propagation and introduce formulas for arriving at upper and lower bounds on each dimension of the output. From there, we explain how these sound outputs can be used in our estimators to find the worst-case for different loss functions. As both probabilistic and decision robustness require sampled deterministic networks, we only need to care about performing a convex relaxation of each sample. This means that for statistical guarantees we can rely on known convex relaxations for deterministic neural networks. Recall that, for interval bound propagation, we must first bound our input dimensions with a hyper rectangle $\mathbf{x}^L, \mathbf{x}^U$ where $\mathbf{x}_i^L \leq \mathbf{x}_i \leq \mathbf{x}_i^U \ \forall i$. Then, given these input bounds we can arrive at output bounds by performing the following forward

pass:

$$\mathbf{z}^{(0),L} = \mathbf{x}^L, \qquad \mathbf{z}^{(0),U} = \mathbf{x}^U$$

$$\mathbf{z}^{(i),m} = \frac{\mathbf{z}^{(i),L} + \mathbf{x}^U}{2}, \qquad \mathbf{z}^{(i),r} = \frac{\mathbf{z}^{(i),L} - \mathbf{x}^{(i),U}}{2},$$

$$\boldsymbol{\phi}^{(i),m} = \mathbf{W}^{(i+1)}\mathbf{z}^{(i),m} + \mathbf{b}^{(i)}$$

$$\phi^{(i),L} = \boldsymbol{\phi}^{(i),m} - |W|z^{(i),r}, \qquad \phi^{(i),U} = \boldsymbol{\phi}^{(i),m} + |W|z^{(i),r},$$

$$\mathbf{z}_j^{(i),L} = \sigma^{(i)}\left(\boldsymbol{\phi}^{(i),L}\right) \qquad \mathbf{z}_j^{(i),U} = \sigma^{(i)}\left(\boldsymbol{\phi}^{(i),U}\right)$$

Where the left-hand side equations determine the forward pass with respect to the lower bound and the right-hand side set of equations determine the forward pass with respect to the upper bound. Ultimately, for a single input and sampled neural network, we arrive an an output interval $[\mathbf{y}^L, \mathbf{y}^U]$ which over approximates the set of outputs possible given an input in the given input interval.

For probabilistic robustness, where we are interested in the proportion of robust models, it suffices to check if $[\mathbf{y}^L, \mathbf{y}^U] \subset S$ for each sampled parameter setting. By keeping track of the proportion of networks for which $[\mathbf{y}^L, \mathbf{y}^U] \subset S$, we are keeping track of the proportion of networks from the posterior which we can soundly prove are robust to any attack.

For decision robustness, we would like the worst-case decision achievable by any adversary. Using IBP, we aim to gain a *sound* estimate on the property that $\forall x \in T$ the classification does not change. To do so, we can observe that:

$$\min_{\mathbf{x}\in T}\left(\mathbb{E}_{\theta\sim p(\theta|\mathcal{D})} f_{\mathbf{y}}^{\theta}(\mathbf{x})\right) \geq \mathbb{E}_{\theta\sim p(\theta|\mathcal{D})} \min_{\mathbf{x}\in T} f_{\mathbf{y}}^{\theta}(\mathbf{x})$$

We can define the second quantity in this equation as $\hat{\mathbf{y}}_{\text{IBP}}$. If $\hat{\mathbf{y}}_{\text{IBP}} \in S$, then we know *soundly* that $\hat{\mathbf{y}}_{\min} \in S$. For classification, the statistical estimation of this value can be straightforwardly accomplished by using of Equation (3.12) for each sampled parameter. For regression, one must consider the following $\mathbf{y}$ in order to minimize the likelihood: $\max\{\mathbf{y} - \mathbf{y}^L, \mathbf{y} - \mathbf{y}^U\}$, where max is taken element-wise.

The advantage of computing decision robustness of a Bayesian neural network with convex relaxation is that we have a sound lower-bound on the true decision robustness of the model. The trade-off being that this lower-bound is achieved by performing an optimization for each model which, as we discussed with adversarial examples, is an over-approximation of the worst-case. In the case of adversarial examples, however, this over-approximation was unnecessary as even an over-approximation of decision robustness with adversarial examples can never be sound. In the case of convex relaxation, the over-approximation of the value serves to preserve the soundness of our estimate.

5.3 Statistical Model Checking

At the beginning of this chapter, we introduced the concept of (ϵ, δ)-guarantees and defined statistical estimators for computation of local robustness properties of Bayesian neural networks. In the previous section, we dealt with the practical issue of approximating the worst-case inputs as called for by the statistical estimator. In this chapter, we introduce statistical model checking as a methodology for relaxing the reliance on an infinite number of samples from the Bayesian posterior. In particular, we leverage recent results in statistical model checking in order to derive efficient estimation schemes (in the sense that they require few samples) for our estimators, which also allow us to have sound (ϵ, δ)-guarantees for adversarial robustness properties of Bayesian neural networks.

When estimating the robustness of Bayesian neural networks, we will consider *two-sided* bounds on the *absolute* error of our estimates. Two-sided refers to the quality of bounding of the satisfaction probability from above and below by the user-specified error constant ϵ. We have already made the distinction between relative and absolute error, but for convenience restate the absolute error guarantee that we aim for here:

$$Prob(||\hat{\mu}_{\boldsymbol{Z}} - \mu_{\boldsymbol{Z}}|| > \epsilon) \leq \delta$$

5.3.1 Sample Bounds with Statistical Guarantees

In this section, we concern ourselves with practically achieving the (ϵ, δ)-guarantees established in Chapter 4 with a finite number of samples from the Bayesian neural network posterior. In each case, we assume that we are given some user-specified error and confidence tolerance δ and ϵ, and we would then like to return a number of samples n such that our statistical estimators satisfy the guarantee. We will first observe the simplest two-sided absolute sample bound and then will move to sequential bounds which depend on the probability we seek to estimate.

Chernoff Bounds One of the most widely used bounds for achieving general (ϵ, δ)-guarantees on the estimation of the mean of a random variable is the Chernoff or Okamoto bound. The two-sided, absolute error formulation of the Chernoff bound is as follows:

Theorem 1 (Chernoff Bound)**.** *Given a random variable* $\boldsymbol{Z}$ *whose support is bounded in* $[0, 1]$*, and an error tolerance* $\epsilon \in (0, 1)$ *we have the following inequality:*

$$Prob(|\hat{\mu}_{\boldsymbol{Z},n} - \mu_{\boldsymbol{Z}}| > \epsilon) \leq 2exp(-2n\epsilon^2) \tag{5.11}$$

Chernoff's inequality allows us to to set $\delta = 2\exp(-2n\epsilon^2)$, plug in our user-specified values of ϵ and δ, and solve for n. In this case, n is then guaranteed to be a finite number of Monte Carlo samples from our Bayesian neural network posterior such that the desired statistical guarantees on the absolute error are met. In Figure 5.2 we plot exactly how this bound n changes with different (ϵ, δ)-guarantee parameters. We see that the number of samples required by the Chernoff inequality grows exponentially with the desired tightness in error and confidence. Though we state this bound for support bounded in $[0, 1]$, it can be easily generalized to random variables with support bounded in $[a, b]$ by the general form presented by Hoeffding, $Prob(|\hat{\mu}_{\boldsymbol{Z},n} - \mu_{\boldsymbol{Z}}| > \epsilon) \leq \exp\Big(- \frac{2n^2\epsilon^2}{(b-a)^2}\Big)$.

Sequential Bounds While Chernoff bounds in conjunction with the statistical estimators given in Section 5.2.2 and Section 5.2.3 suffice to compute guarantees on the local robustness of Bayesian neural networks with guarantees, we highlight that in order to achieve a guarantee with $\epsilon = 0.01$, $\delta = 0.05$, the Chernoff bound prescribes a Monte Carlo estimate with 18,445 many samples from the posterior. That is to say, one must sample from the Bayesian neural network posterior *and* check/verify the adversarial robustness in order to make a statement with 1% error and with 95% confidence. In this section, we will introduce two tighter bounds on n, which will allow us to cut the number of samples down to $n = 3,283$ with the same guarantee.

In order to do so, we must first introduce the notion of confidence intervals which is a key building block of statistical model checking algorithms. Given any statistical estimator, one has a "confidence" interval about the estimate.

Definition 8 (Confidence Interval)**.** *We say that an interval with end points* $a, b \in [0, 1], a < b$*, has coverage 1-*δ *for the expectation of a random variable* $\boldsymbol{Z}$ *if:*

$$Prob(\mu_{\boldsymbol{Z}} \in I) = 1 - \delta \tag{5.12}$$

For example, the exact Clopper-Pearson confidence interval guarantees that its coverage of a random variable estimated with m *successes after* n *trials (i.e.* $\hat{\boldsymbol{Z}}_n = \frac{m}{n}$*) is greater than or equal to* $1 - \delta$ *and is given by:*

$$J = \big[\beta^{-1}(\frac{\delta}{2}, m, n - m + 1), \beta^{-1}(1 - \frac{\delta}{2}, m + 1, n - m)\big] \tag{5.13}$$

where $\beta^{-1}(\delta, i, j)$ *is the* δ *quantile of a Beta distribution parameterized by* i *and* j*.*

A confidence interval allows us to gain information about the probabilistic quantity we are trying to estimate *while* we estimate it. By understanding the feasible range of the random variable, we can sharpen our bound with either Hoeffding's or Massart's sequential bound. Hoeffding's bound has been proven in [75].

Theorem 2 (Hoeffding's Bound)**.** *Given a random variable* $\boldsymbol{Z}$ *and an error tolerance* $\epsilon \in (0, 1)$ *we have the following inequality:*

$$Prob(|\hat{\mu}_{\boldsymbol{Z},n} - \mu_{\boldsymbol{Z}}| > \epsilon) \leq 2exp(-h_{hof}(\mu_{\boldsymbol{Z}})n\epsilon^2)$$

where

$$h_{hof}(\mu_{\boldsymbol{Z}}) = \begin{cases} 2 & if\ \mu_{\boldsymbol{Z}} = 1/2 \\ 1/(1-2\mu_{\boldsymbol{Z}})log((1-\mu_{\boldsymbol{Z}})/\mu_{\boldsymbol{Z}}) & if\ \mu_{\boldsymbol{Z}} \neq 1/2 \end{cases} \tag{5.14}$$

While this is a well-known bound, the Massart bound can be used to sharpen h_{hof} to h_{mas} by further using information about ϵ and by assuming that the random variable of interest is a Bernoulli random variable. A proof can be found for this theorem in [76].

Theorem 3 (Massart's Bound)**.** *Given a Bernoulli random variable* $\boldsymbol{Z}$ *and an error tolerance* $\epsilon \in (0, 1)$ *we have the following inequality:*

$$Prob(|\hat{\mu}_{\boldsymbol{Z},n} - \mu_{\boldsymbol{Z}}| > \epsilon) \leq 2exp(-h_{mas}(\mu_{\boldsymbol{Z}}, \epsilon)n\epsilon^2)$$

where,

$$h_{mas}(\mu_{\boldsymbol{Z}}, \epsilon) = \begin{cases} 9/2((3\mu_{\boldsymbol{Z}} + \epsilon)(3(1-\mu_{\boldsymbol{Z}}) - \epsilon))^{-1} & if\ 0 < \mu_{\boldsymbol{Z}} < 1/2 \\ 9/2((3(1-\mu_{\boldsymbol{Z}}) + \epsilon)(3\mu_{\boldsymbol{Z}} + \epsilon)^{-1} & if\ 1/2 \leq \mu_{\boldsymbol{Z}} < 1 \end{cases} \tag{5.15}$$

We demonstrate in Figure 5.2 how much sharper Massart's bounds are compared to Chernoff's bound. We highlight that the Massart and Hoeffding bounds rely on knowledge of the feasible range of values of $\mu_{\boldsymbol{Z}}$. Of course, we do not have access to the value we are trying to estimate, and so instead we must introduce a third parameter of our guarantee, γ. By computing a $1-\gamma$ coverage interval via the exact Clopper-Pearson interval in (5.13) we can get arbitrarily high probability information about the quantity we wish to estimate. By taking the highest number of samples prescribed by the bound given our $1-\gamma$ information about the random variable we maintain our statistical guarantees and greatly reduce the number of samples required.

5.3.2 Algorithms for Estimation of BNN Robustness

In this section we combine the sophisticated statistical estimation schemes implied by Massart and Hoeffding bounds with adversarial robustness information of Bayesian neural network samples in order to arrive at practical algorithms for the computation of probabilistic and decision robustness.

A key line shared by both algorithms is the computation of h_{mas} and h_{hof} based on a Clopper-Pearson exact confidence interval. In order to reduce the number of necessary samples in each algorithm, we keeping track of the $(1-\gamma)$ Clopper-Pearson confidence interval for our estimation on line 8 of Algorithm 1 and on line 10 of Algorithm 2. We then use this to compute the largest possible number of samples we would need according to Theorem 3 and Theorem 2 on the subsequent line of each algorithm. Practically, we compute the highest number of samples necessary by checking if $1/2$ is inside of the confidence interval, if it is, we continue to use the Chernoff bound. If it is not, then we simply take the end-point of the confidence interval closest to $1/2$, compute h_{mas} or h_{hof}, and return the number of samples. As h_{mas} and h_{hof} are convex and have their maximum at $1/2$, we know that the end-point of the interval closest to the maximum will give us the largest possible samples necessary.

Algorithm 1 Probabilistic Robustness Estimation Algorithm

Input: $\mathbf{x}^*$ – Test Point, T – Input Property, S – Output Property, f – Network Architecture, $p(\boldsymbol{\theta}|\mathcal{D})$ – Posterior Distribution, ϵ, δ, γ – Sequential Bound Parameters
Output: $\hat{p}$ – robustness probability estimate which respects statistical guarantees

1: $n^C \leftarrow$ number of samples by Chernoff bounds (5.11)
2: $n_{\max} \leftarrow \lceil n^C \rceil$; $n, k \leftarrow 0, 0$
3: **while** $n < n_{\max}$ **do**
4: $\quad \theta^{(i)} \leftarrow$ sample from $p(\boldsymbol{\theta}|\mathcal{D})$
5: $\quad \mathsf{SAT} \leftarrow \mathbb{I}(f, T, S, \theta^{(i)})$
6: $\quad k \leftarrow k + \mathsf{SAT}$; $n \leftarrow n + 1$
7: $\quad \hat{p} \leftarrow k/n$
8: $\quad I_p \leftarrow$ Clopper-Pearson$(1-\gamma, k, n)$
9: $\quad n^M \leftarrow$ samples num. by Massart bounds Theorem 3 using I_p
10: $\quad n_{\max} \leftarrow \lceil \min(n^M, n^C) \rceil$
11: **end while**
12: return $\hat{p}$

Estimation of Probabilistic Robustness In Algorithm 1, we assemble the previously discussed methodology in order to compute efficient statistical information about the probabilistic robustness of a Bayesian neural network posterior. As inputs, we take the information required by the definition of BNN probabilistic robustness as well as those required to compute the statistical guarantee $(\epsilon, \delta, \gamma)$. On line 1, we make the worst-case assumption for sample complexity of the Massart bound that the robustness is exactly $1/2$ and thus the bound of the algorithm is given by the Chernoff bound (Equation (5.11)). On line 2, we simply set up our counter variables, n the number of samples we have taken and k the number of robust samples. By taking the proportion of these two, k/n, we are computing the empirical estimator $\hat{\phi}_n^{\text{prob}}$ in Equation (5.1). This proportion step can be found on lines 4 and 5 of the algorithm. We then use lines 8 to compute the confidence interval of our current estimate using the exact Clopper-Pearson interval defined in Definition 8. From this estimate, we can compute the function h_{mas} from Theorem 3. Due to Theorem 3, we know that the returned result of our algorithm is an estimate of Definition 6 such that the statistical guarantees are respected.

Estimation of Decision Robustness In Algorithm 2, we assemble the previously discussed methodology in order to compute efficient statistical information about the *decision* robustness of a Bayesian neural network posterior. The algorithm presented assumes we would like a sound estimate (e.g., via IBP) and that we are in the classification setting. We highlight that this algorithm would also return sound results for regression whose co-domain for which each output dimension of the codomain is bounded in the interval $[0, 1]$. This bound can be generalized to random variables with bounded support on $[a, b]$ where both a and b are non-negative constants. Further, if the random variable of interest (e.g., output of the BNN) is multidimensional we can bound each dimension of the output by an interval $[a_i, b_i]$ and arrive at a very similar bound which can be found and is proved in [68]. As in the probabilistic robustness estimation, we take as input the parameters required by the definition of decision robustness, Definition 7. We also require the user-defined guarantee parameters $(\epsilon, \delta, \gamma)$.

The procedure also follows very similar control flow to our estimation of probabilistic robustness with the primary distinctions being in the treatment of our empirical estimate. On lines 1 and 2 we set up our empirical estimators for the mean of the output of our Bayesian posterior predictive distribution. On line 5, we compute an output over-approximation $\mathbb{O}$ function w.r.t. the provided input interval T, which

allows us to then compute the worst-case for our likelihood. Minimizing the likelihood on line 7 is done in order to satisfy the form of our estimator in Equation (5.10) for *any* adversary. This is distinct from the previous algorithm in which we combine the worst-case approximation and safety check into one line (line 5 of Algorithm 1). Again, we compute the Clopper-Pearson intervals on line 10, but this time we use the Hoeffding inequality h_{hof} rather than the Massart bound as the Massart bound only applies to Bernoulli random variables. After the termination of the `while` loop on line 3, we know from Theorem 2 that our estimate of the mean of the Bayesian neural network posterior predictive is at most ϵ away from the true mean with probability $1-\delta$. One can then either simply check that this estimate is in S and make a statement with statistical guarantees, or by taking into account the entire interval $[\hat{\mathbf{y}}-\epsilon, \hat{\mathbf{y}}+\epsilon]$ we can know that with probability $1-\delta$ the mean of the decision is in the safe set S. Correctness of this procedure comes from the application of Theorem 2 to our estimator in Equation (5.10). We also highlight that this algorithm may require few samples if one follows the procedure of [35].

The formulation of Algorithm 2 and the explanation throughout this section has so far dealt with the mean exclusively. In order to observe that our procedure can also produce sound median estimates wrt our (ϵ, δ)-guarantees, consider the definition of the median of a random variable $\boldsymbol{Z}$ which is distributed according to p_Z is given as $m(\boldsymbol{Z}) := x \iff \int_{-\infty}^{x} p_Z(v)dv = 0.5$. That is, when the cumulative density or mass is 0.5. We then consider a median estimate from n samples as the sample median $\hat{m}_n(\boldsymbol{Z})$ and can see that the Hoeffding (and, in fact, Massart) (ϵ, δ)-guarantees carry over as by considering estimation of $Prob(\boldsymbol{Z} > \hat{m}_n(\boldsymbol{Z}))$. That is, we pose estimation of the probability that a value sampled from $\boldsymbol{Z}$ is less than our sample median estimate. By taking n to be the result of the Hoeffding or Massart bound, we arrive at a sample median estimate which approximately satisfies the definition of a median. The one caveat is that our error in this case is w.r.t. the probability of deviating from the definition of a median, not the true median itself. Thus, our guarantees in this case are weaker as we can only claim that the (ϵ, δ)-estimated median is safe w.r.t. the local robustness property.

Relationship to Randomized Smoothing Before discussing the results of our framework for computing statistical guarantees for Bayesian neural networks, we quickly compare and contrast with a related methodology for statistical guarantees of arbitrary classifiers: randomized smoothing [33]. Randomized smoothing takes an arbitrary classifier f and "smooths" its outputs by assuming a local distribution over

each input (e.g., a Gaussian distribution). That is, we say a function h is a smoothed version of f if the output of h is the mean of outputs of f on points sampled from the local smoothing distribution (i.e., a Gaussian). By estimating the class-wise probabilities (softmax outputs) of the smoothed classifier and through the application Neyman-Pearson Lemma, one is able to arrive at a robust radius for the given input. The metric that the radius is guaranteed for depends on the kind of smoothing distribution used and general frameworks have been set up to this end [40].

While the form of the guarantees between the presented methodology and randomized smoothing is similar, there are several key distinctions that ought to be made. Firstly the guarentees from randomized smoothing rely on three key components: the magnitude and form of the smoothing distribution (larger magnitude gives larger radius), the probability of the true class under the smoothing distribution and the probability of all other classes under the smoothing distribution. These lend themselves to a few downsides: firstly, if the smoothing distribution uses harsh noise the smoothed version of the classifier, h may see degradation in performance. Though, one can train in particular ways to alleviate this [161]. Further, the robust radius from smoothing depends on the certainty of the classier: a more certain classifier leads to a larger robust radius. Thus, the robust radius produced by randomized smoothing inherently penalizes uncertainty in the posterior predictive. In contrast the methodology presented here does not require any modification of the Bayesian neural network, nor does it penalize uncertainty.

Despite the few downsides, smoothing has some important strengths: it can be readily applied to even the largest domains without it's bounds becoming vacuous and it's statistically sound bounds apply to arbitrary classifiers whereas the methods investigated here are particularly for probabilistic neural networks.

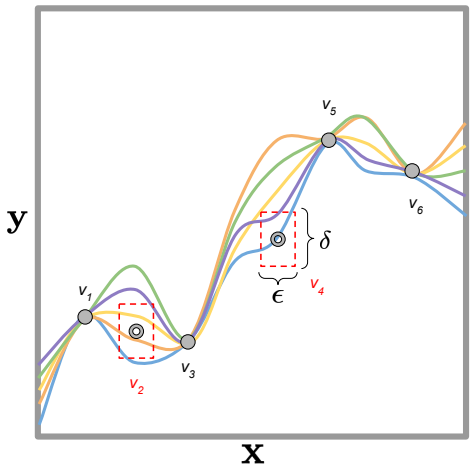
Algorithm 2 Bayesian Decision Robustness Estimation Algorithm

Input: $\mathbf{x}^*$ – Test Point, T – Input Property, S – Output Property, f – Network Architecture, $p(\boldsymbol{\theta}|\mathcal{D})$ – Posterior Distribution, ϵ, δ, γ – Sequential Bound Parameters
Output: $\hat{p}$ – decision robustness estimate which respects statistical guarantees

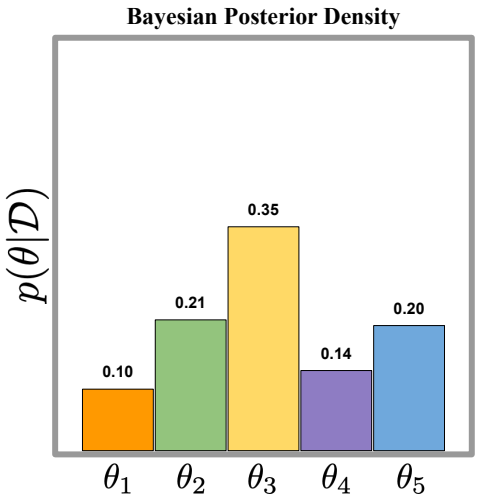
$n^C \leftarrow$ number of samples by Chernoff bounds (5.11)
$n_{\max} \leftarrow \lceil n^C \rceil$; $n, k \leftarrow 0, 0$
while $n < n_{\max}$ **do**
  $\theta^{(i)} \leftarrow$ sample from $p(\boldsymbol{\theta}|\mathcal{D})$
  $[\mathbf{y}^L, \mathbf{y}^U] \leftarrow \mathbb{O}(f, T, \theta^{(i)})$ # $\mathbb{O}$ *returns over-approximation e.g., via IBP*
  $n \leftarrow n + 1$
  $\mathbf{y}_{\min} \leftarrow \arg\min_{\mathbf{y} \in [\mathbf{y}^L, \mathbf{y}^U]} p(\mathbf{y}|\mathbf{x}, \theta^i)$ # *Computed according to Equation* (3.12)
  $k \leftarrow k + \mathbf{y}_{\min}$
  $\hat{\mathbf{y}} \leftarrow k/n$
  $I_p \leftarrow$ Clopper-Pearson$(1 - \gamma, k, n)$
  $n^M \leftarrow$ samples num. by Hoeffding bounds Theorem 2 using I_p
  $n_{\max} \leftarrow \lceil \min(n^M, n^C) \rceil$
end while
return $\hat{\mathbf{y}} \in S$

5.4 Experiments

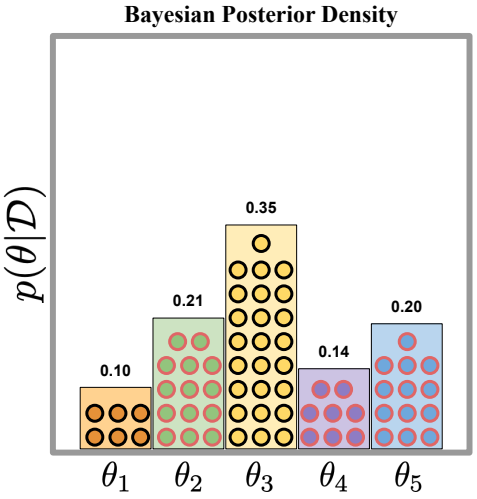
In this section, we start by visualizing the workings of Algorithm 1 as well as running tests on simulated random variables in order to demonstrate the nature of our guarantees. We then study how this extends to real datasets with an array of regression tasks taken from the University of California Irvine machine learning repository [38] (henceforth, UCI datasets). We choose datasets following the lead of [67, 52, 82] who study approximate Bayesian inference. Following this we study how our methodology scales to different image recognition challenges starting with MNIST [93] and scaling up to the German Traffic Sign Recognition Benchmark (GTSRB) [70]. For every dataset, we will use our methods to measure and visualize the average local robustness of each Bayesian neural network. The average local robustness (for both probabilistic and decision robustness) is computed as the mean robustness value of random test-set examples. This is a useful metric as it provides a picture of the worst-case robustness performance of the posterior distribution under consideration (in the same way that the test-set loss or test negative loglikelihood is a good measure of average case performance). In some applications, inference on particular test-set instances maybe of higher safety concern than others (e.g. classifying a red traffic light versus classifying



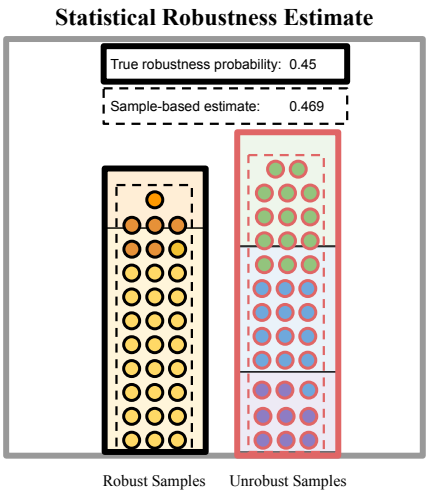
(a) Our regression running example where training points are solid dots and test points are hollow dots. We consider computing the probabilistic robustness of this BNN wrt specification v_2.



(b) We recall the posterior mass that corresponds to the discrete support of the Bayesian posterior in subfigure (a).

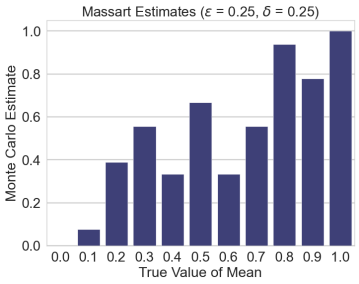


(c) Here we show a number of samples from the posterior such that each sample is outlined in black if it is robust and red if it is not robust wrt the v_2 specification in subfigure (a).

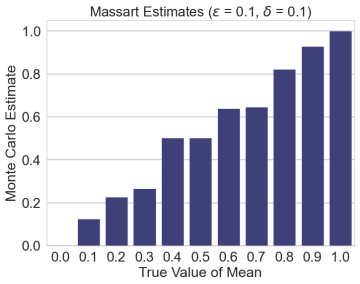


(d) By tallying the parameters that were robust and non-robust in subfigure (c), we arrive at an accurate estimate of the true probabilistic robustness.

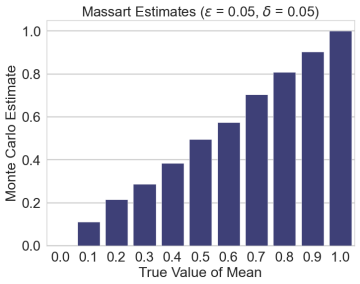
Figure 5.1: Computing the probabilistic robustness for point v_2 in our regression running example.



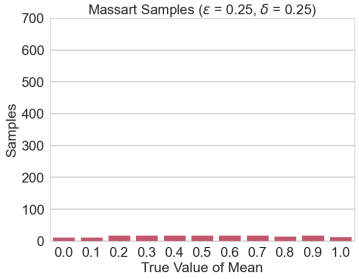
(a) Final estimates with $\epsilon = 0.25, \delta = 0.25$



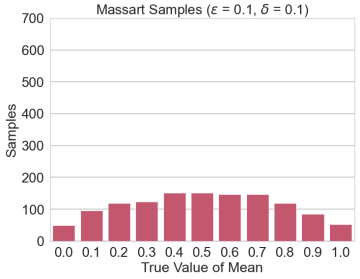
(b) Final estimates with $\epsilon = 0.1, \delta = 0.1$



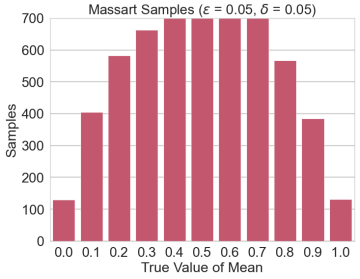
(c) Final estimates with $\epsilon = 0.05, \delta = 0.05$



(d) Number of Massart samples with $\epsilon = 0.25, \delta = 0.25$



(e) Number of Massart samples with $\epsilon = 0.1, \delta = 0.1$



(f) Number of Massart samples with $\epsilon = 0.05, \delta = 0.05$

Figure 5.2: **Top Row:** On the x-axis we plot the true value of the random variable. On the y-axis we plot the final estimate which satisfied the Massart bound. **Bottom Row:** On the x-axis we again lot the true value of the Bernoulli random variable, and on the y-axis the number of samples needed before we reached the final estimate. We see that when estimating a mean that is around 0.5 we need the most samples (equivalent to the Chernoff bound).

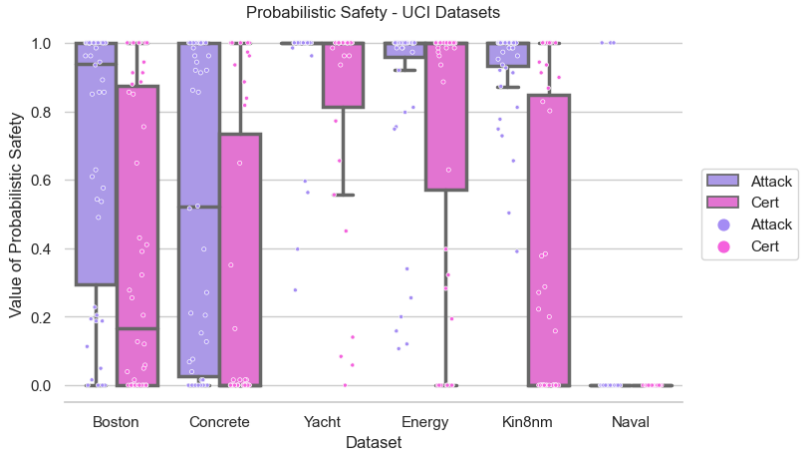
a house number), in which case a weighted average or particular analysis should be conducted. An example of this kind of decomposition is done in our analysis of the MNIST dataset.

5.4.1 Intuitive Examples

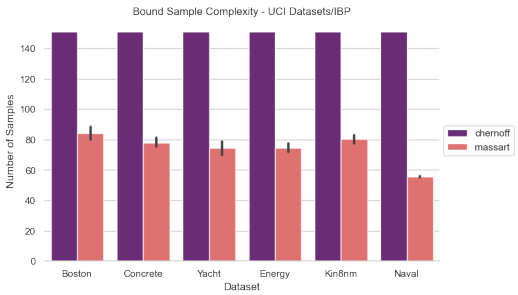
Algorithmic Visualization In Figure 5.1, we provide a visual intuition for the key steps of Algorithm 1. We recall that the problem we considered was that of estimating the probabilistic robustness of test input v_2 as visualized by Figure 5.1a. In this example, we assume that we do not have access to the tractable and convenient discrete posterior distribution visualized in Figure 5.1b and must sample from it as an unknown distribution. In Figure 5.1c, we visualize the process of approximately sampling from the posterior distribution by marking each sample with a dot. Further, we outline each sample from the posterior in Figure 5.1c with black if it is robust and in red if it is non-robust. The last step of the algorithm is to see what proportion of the collected samples is robust. In Figure 5.1d we do exactly this and find that the proportion of samples which were robust is 0.469. Of course, given this simple example, we can see that this is 0.019 away from the true probability estimate of 0.45, which we computed in our previous section.

Sample Complexity Visualization In Figure 5.2, we demonstrate the practical sample complexity of using the Massart bound in a simulated scenario. In order to do so, we create a Bernoulli random variable $\boldsymbol{Z}_p$ with mean p. We then use the sequential Massart bound (Theorem 3) in order to estimate the mean p. At the end of this procedure, we plot both the estimate of p (top row of Figure 5.2) as well as the number of samples necessary (bottom row of Figure 5.2). We perform this estimation for different values of p, plotted along the x-axis, as well as for different values of ϵ and δ. As expected, our estimates become tighter as we decrease ϵ and δ at the cost of increased sample complexity (according to the concentration inequalities in the previous sections). Moreover, we notice that when we are estimating values around 0.5, our sample complexity is the worst and is in fact equivalent to the Chernoff sample bound which is exactly what is to be expected according to the Massart bound.

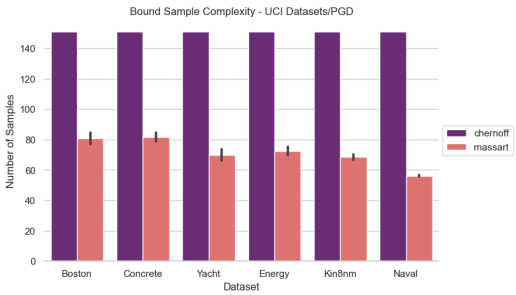
The key take away from this visualization is how tight our empirical estimates of p can be, especially in the case of Figure 5.2c. Further, for small values of ϵ and δ we give a visual intuition for exactly how much more efficient the Massart bound is than the Chernoff bound. We see in Figure 5.2f that the Massart bound can require as few as 1/7 of the samples required by the Chernoff bound.



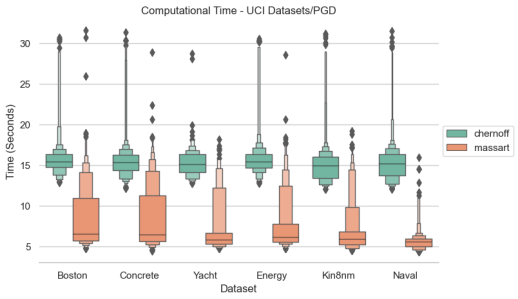
(a) We visualize the estimated probabilistic safety for each of the studied UCI datasets. For some network (e.g. Yacht) the posterior is very robust. For others (e.g. Naval) the posterior is completely vulnerable.



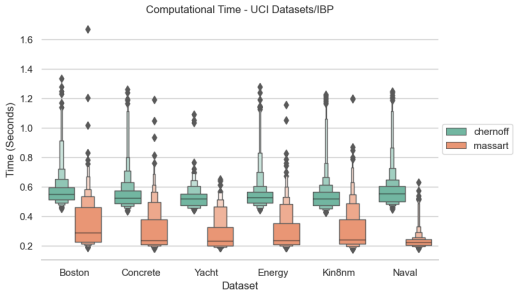
(b) Sample complexity of achieving the required (ϵ, δ)-guarantee given that we test each UCI dataset with PGD.



(c) Sample complexity of achieving the required (ϵ, δ)-guarantee given that we test each UCI dataset with IBP.



(d) We find that decreased sample complexity leads to improved computational time in practice, especially when using an expensive attack such as PGD.



(e) Even using IBP we observe an improvement in computational time. Due to the efficiency of IBP, we see that our time saving is less. This is due to a larger proportion of the process being algorithm-independent overhead.

Figure 5.3: UCI Dataset analysis shows us that we can understand the robustness profiles of real-world applications while realizing the benefits of our efficient sampling method.

5.4.2 UCI Datasets

We move from contrived examples to real-world analysis with a set of simple but realistic regression tasks taken from the UCI machine learning repository [38]. In particular, we focus on datasets which have been previously studied with Bayesian deep learning [67, 82, 52]. We start with a subsection detailing the meaning of the datasets, training/inference methodology, as well as the attacks and certification methods used as subroutines. Using this context, we then provide a thorough discussion of the results obtained in these settings.

5.4.2.1 Experimental Setting

Datasets In order to go beyond toy examples, we study eight regression datasets which are commonly used to benchmark Bayesian deep learning. In particular, we adopt the 1D regression benchmarks studied by [67, 52] and [82] to test approximate inference methods. Though each of the datasets is a 1D regression problem they cover a range of input dimensionalities as well as a varying degree of non-linearity. In particular, we study the following datasets each of which is from [38]: **Boston Housing** A regression task which asks us to predict housing prices based on 14 different real-valued statistics about the local geographic area; the **Concrete** dataset asks us to predict the compressive strength of concrete based on 8 key factors including its ingredients and age; the **Yacht** dataset involves predicting the residuary resistance of a yacht design based on structural properties of the design; the **Energy** efficiency dataset is interested in predicting the cooling/heating load of a proposed building based on key facets of the structure; **Kin8nm** is thousands of state-space readings of the the forward dynamics of an 8 link robot arm; the **Naval** propulsion contains the results from numerical simulation of a naval vessel.

Inference For each dataset, the we train a corresponding Bayesian neural network using the Variational Online Gauss Newton (VOGN) method proposed in [82]. In particular, we set up their network, a single hidden layer neural network with 50 hidden units and rectified linear unit (ReLU) activation functions. We find that we are able to roughly reproduce the same performance as the authors of [82] before turning our attention to the robustness of the resulting posteriors. For each dataset there are several cross-validation splits provided. We provide analysis for Bayesian neural networks trained on the first cross-validation split. Further parameters are in the Appendix for the validation splits we refer readers to the corresponding code for this thesis.

Attacks and Certifications As is noted in Section 5.2.4, the choice of attack or certification method used is of paramount importance to the interpretation of the resulting bounds. When we perform attacks on sampled deterministic neural networks from the Bayesian posterior we rely on projected gradient descent (PGD) which is run for 25 iterations with one restart. The PGD attack is aimed at maximizing the training loss (in this case the mean squared error) between the true value and the one predicted by the network. When performing certification, we use the standard IBP formulation.

5.4.2.2 Robustness Analysis

Probabilistic Robustness Estimation In Figure 5.3a we visualize the computation of probabilistic robustness for 250 test set inputs for each dataset. While each test input has an independent probabilistic robustness, we plot all of the samples together in order to gain insight into the robustness profiles of the Bayesian posteriors under investigation.

For a given test input, we run Algorithm 1. We build the input region T as an ℓ_∞ ball with radius 0.01 (all training inputs are scaled to $[0, 1]$ and then these scaling values are imposed onto the test set inputs as is standard). The output specification, S, is then taken to be an ℓ_∞ ball with radius 0.05 which is centered at the mean output of the Bayesian neural network (where again the space is mapped to the unit interval). Thus, the specification can be interpreted as *a 1% change in the input should not correspond to more than a 5% difference in the neural network output.* The mean output of the Bayesian neural network for the center of the output specification is estimated via Monte Carlo sampling with 100 sampled neural networks. One could also test with an output specification which is centered at the true value in which case one is measuring at once the sensitivity of the learned function as well as the error. In this case, by centering the output specification at the mean prediction of the BNN we have not accounted for the error with respect to the ground truth when measuring robustness properties. We compute all estimates with $(0.1, 0.1)$ (ϵ, δ)-guarantees using the sequential Massart bound to ensure that we have collected enough samples.

The resulting bounds are illustrated in Figure 5.3a. For some datasets, we find that we are able to gain meaningful statistical certification. For the Yacht dataset using IBP we found that we are able to certify that, on average, the given robustness specification was satisfied by sampled networks with probability 0.883. Moreover, when we test the posterior trained on the Yacht dataset with PGD we are able to say that networks are robust, on average, with probability 0.995. These numbers provide

strong indication that the posterior trained on the Yacht dataset is robust to small peturbations. One potential reason that Yacht is particularly verifiable is that it is a very low dimensional dataset (with each feature vector being described by only 6 dimensions). The Energy dataset is tied for the second smallest and we were also able to get some strong guarantees for may of its test set examples. Further, the Naval dataset, for which we did not certify any examples, has the largest number of features, 16.

In a similar vein, we are able to show that PGD is successful on less than 5% of the tested posteriors (on average) for the Kin8nm dataset. Despite this mark of robustness, when testing with IBP, we are only able to certify an average probabilistic robustness of 0.471. This indicates that, while the Kin8nm posterior would be robust at test time to PGD adversaries, we cannot make strong statistical claims about its robustness to *any* adversary. It would then be up to those deploying the network to decide if this is satisfactory performance

Finally, we turn our attention to the right-most dataset in Figure 5.3a, the Naval dataset. For this dataset, we were unable to certify any probabilistic robustness properties. Moreover, we were able to successfully attack 100% of the posteriors sampled for 100% of the test inputs tested. This immediately indicates that the Bayesian neural network is unfit for deployment if the specification stated above is of critical importance. It is not possible to know from this analysis alone what causes such a lack of robustness. It could be that the data distribution is degenerate and that the chosen specification considers too much of the input space to give us meaningful robustness analysis. What we do know is that, if the specification is taken to be meaningful, then this posterior is not fit for deployment. In Chapter 7, we will see ways of potentially remedying this scenario by incorporating robustness specifications into Bayesian inference for neural networks.

Sample Complexity and Computational Time Improvements The primary advantages of using a more sophisticated sequential sampling algorithm for statistical model checking is the reduced number of samples and therefore the reduced computational time. In Figures 5.3b and 5.3c we investigate if these improvements are realized in a practical regression setting. In each subfigure, we visualize the number of samples required in practice for both the Chernoff bound as well as the Massart bound. Of course, we find that the Chernoff bound is fixed at roughly 150. The Massart bound on the other hand requires as few as 60 samples in many cases. This

reduced sample complexity allows us to achieve the desired *a priori* statistical guarantee with roughly 1/3 of the samples of standard Chernoff bounds. We highlight that the sample complexity improvement is best for the Naval dataset which has robustness 0 everywhere. As we observed in Figure 5.2, the further the probability we seek to estimate is from 0.0 the better our performance.

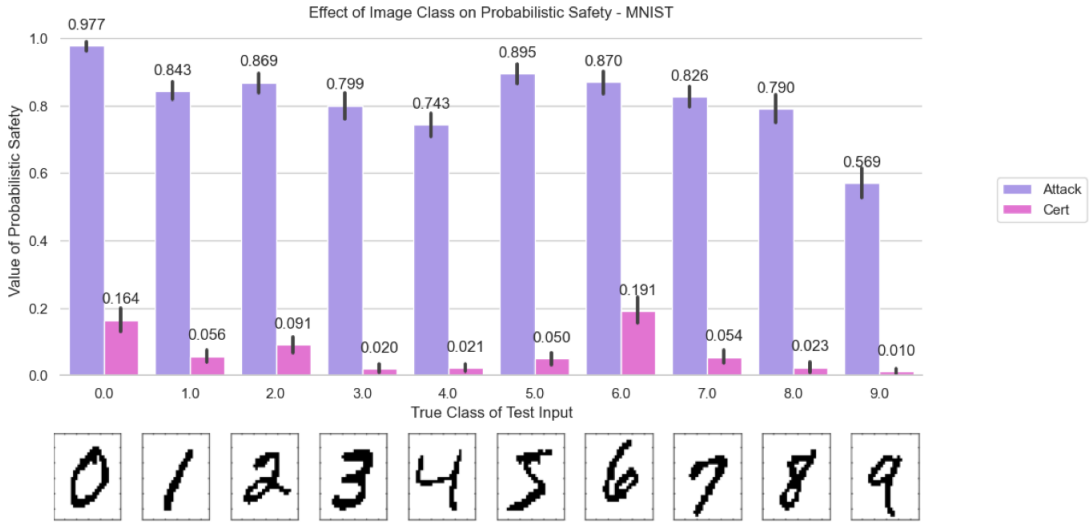
In Figures 5.3d and 5.3e, we highlight the computational time improvement corresponding to the sample improvement. We find that the sample complexity improvement directly translates to improved statistical certification times. Of course, we can see that the Massart bound does not *strictly* run at 1/3 of the computational time. This is likely due to the fact that some inputs will require us to estimate a probabilistic robustness close to 0.5 in which case the computational time between the Massart and Chernoff bounds should be roughly equivalent. Computational overhead may also be a factor in the time estimates being slightly closer than we may expect from the sample complexity improvement. Ultimately, Figures 5.3d and 5.3e confirm that we are indeed able to realize the improvements from our efficient sampling framework on real-world data.

5.4.3 MNIST

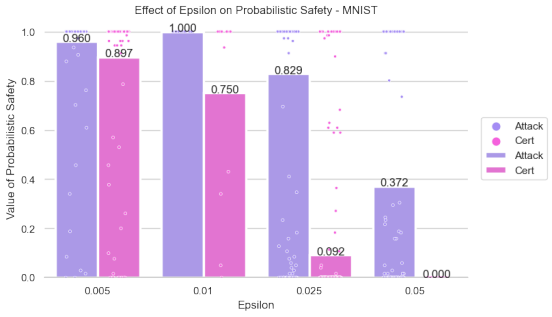
In this section, we turn our attention to a much larger dimensionality example than those considered in our UCI analysis. MNIST is a well-studied, small-scale image recognition datast. The dataset poses the problem of handwritten digit recognition. Each input is a 28 by 28 (784 dimensional) black and white image which encodes the pen stroke corresponding to a digit 0 through 9. Per-class examples are represented at the bottom of Figure 5.4a. In our case study of the UCI regression datasets, we focused on how to practically interpret the output of our algorithm as well as ensuring that the theoretically possible speed-ups warrant the proposed methodology. In this section, we will dive further into components of the BNN that affect probabilistic robustness and the relationship between decision robustness and probabilistic robustness.

5.4.3.1 Experimental Setting

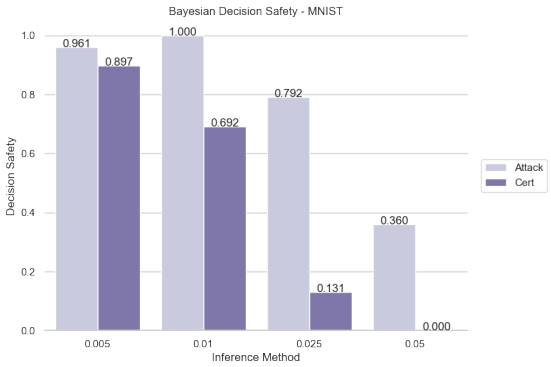
In order to learn on MNIST we use the standard 50,000/10,000 test/train split that is provided [93]. For each tested posterior, we again train using the VOGN algorithm from [82]. Our learning rate parameters is chosen based on a randomly selected validation set which is separated from the training set randomly at the beginning of each run. In this section, we will study several different architecture configurations; we



(a) We break down the class-wise probabilistic robustness of a posterior distribution in order to understand where we might need more data. As an example, class '9' and '4' both have high probability of being susceptible to attack.



(b) We visualize the effect of the attack magnitude on the the probabilistic robustness estimate. We find that IBP causes a much steeper decline to the robustness estimate.



(c) We visualize the effect of the attack magnitude on the the decision robustness estimate. We find that the values of the decision robustness of the network are similar to those of probabilistic robustness.

Figure 5.4: We use our notions of probabilistic and decision robustness to analyze the adversarial properties of Bayesian neural network posteriors on MNIST. Each dot in the above figures represents the estimate for a single image.

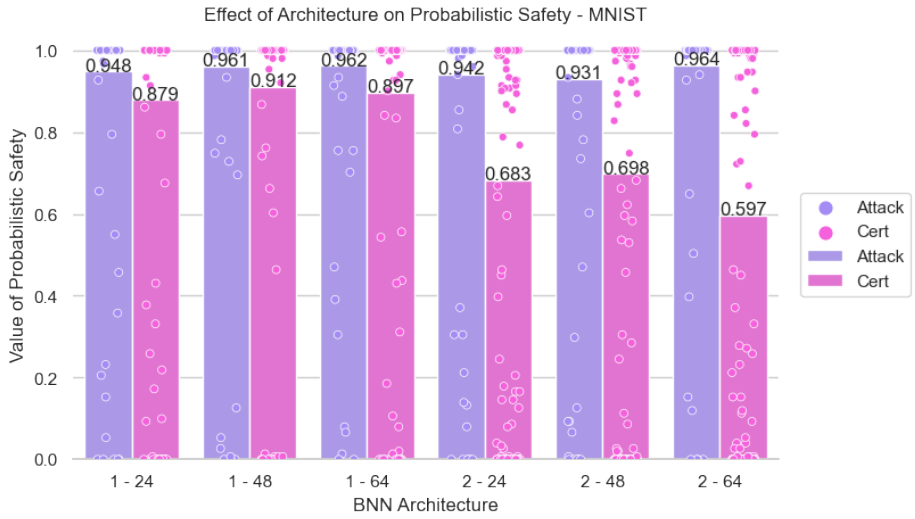


Figure 5.5: We visualize the effect of larger and more complicated architectures, listed as (`depth - width` on the x-axis). We find little correlation between the size of the posterior and its robustness, save for the decrease in certifiable robustness with IBP for two layer networks.

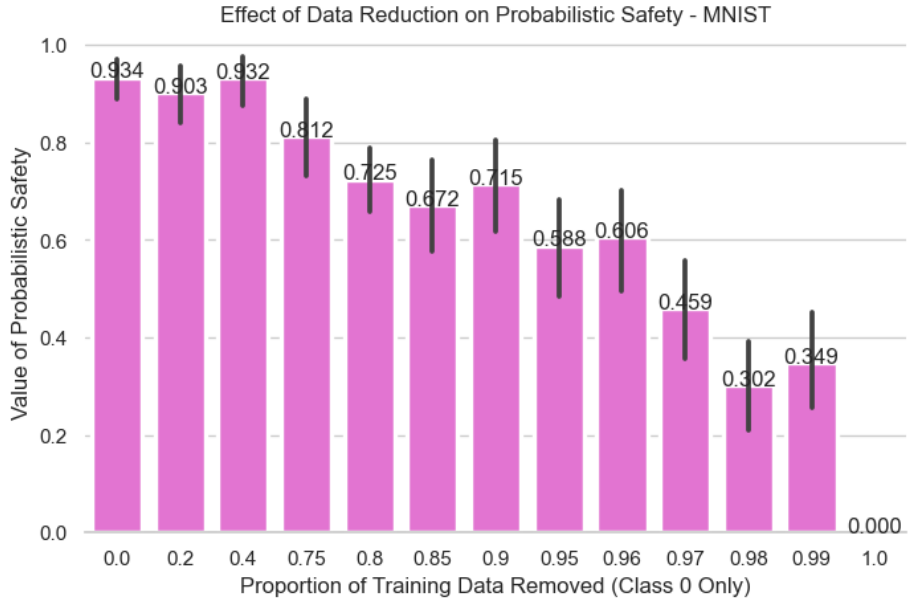


Figure 5.6: By artificially reducing the number of training instances for Class 0, we can see that there is a strong correlation between the amount of data trained on and certifiable robustness. Specifically, we find that introducing only 592 images (10% of the original amount) is enough to achieve certifiable robustness on 71.5% of test set instances.

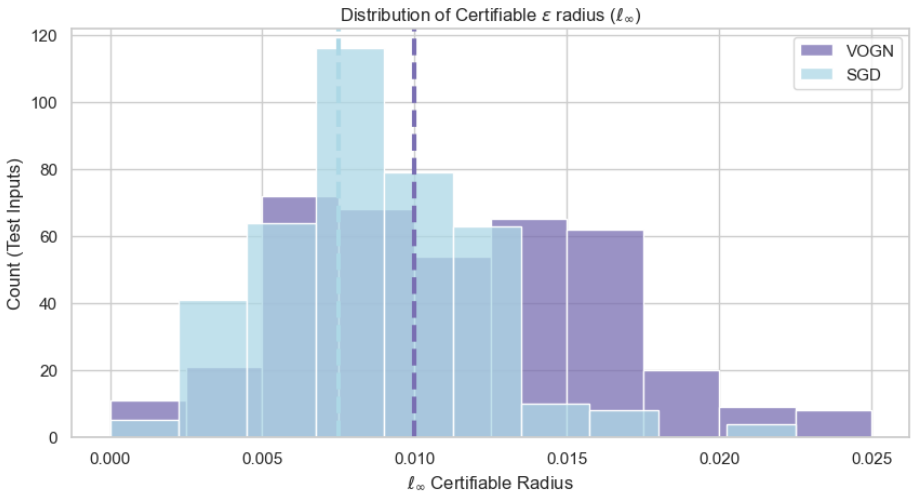


Figure 5.7: We test the certifiable decision robustness of BNNs versus corresponding DNNs. We find the certifiable robustness of the BNN inferred approximately with VOGN to have significantly stronger robustness performance than that of the same network trained with SGD. The dashed line indicates the median certifiable radius for each method.

tune the learning rate parameters for each architecture individually. Each architecture uses rectified linear units as activation functions and a softmax as its final layer. We take the likelihood of the model to be the sparse categorical cross-entropy.

5.4.3.2 Robustness Analysis

In this section, we will describe the probabilistic robustness tests carried out on the Bayesian neural network posteriors which were trained on MNIST. Our results are visualized in Figure 5.4 and Figure 5.5.

Probabilistic Robustness In order to measure probabilistic robustness of our MNIST trained posteriors, we build our input property as an ℓ_∞ ball centered at the test input with a variety of radii specified per analysis. Given that this is a classification dataset, S is taken to be the set of outputs such that the classification is equivalent to the true class. In this way, we consider natural errors to count against the robustness of the network as an input which is misclassified prior to the introduction of adversarial noise is certainly not robust for any selected input property. In order to understand the probabilistic robustness profiles of each network, we again take 250 test set images and compute their probabilistic and decision adversarial robustness using both PGD and IBP. We again consider $(0.1, 0.1)$ (ϵ, δ)-guarantees using the sequential Massart bound to ensure that we have considered enough samples.

In Figure 5.4a, we analyze a Bayesian neural network with one hidden layer that has 64 hidden neurons. The input property under consideration is an ℓ_∞ ball with radius 0.025. Along the x-axis of the plot we separate all of the MNIST classes 0-9 and provide a representative image for the class. The y-value for each bar is then the average probabilistic local robustness for images from that particular class. This kind of analysis is helpful in that it shows us where in particular our classifier is vulnerable. This can help us to design improved networks. We notice in Figure 5.4a that the class associated with the digit 9 is particularly vulnerable to attacks and was unable to be certified in any way. As such, we may seek to collect more data which represents the digit 9 or augment the current collection of examples with some noise which may improve the robustness. The correlation between training data and certifiable robustness can be found in Figure 5.6. To produce this figure we study class 0 with ℓ_∞ radius 0.005 (for which we can certify nearly every test set input). We then reduce the amount of training data artificially and re-measure the certifiable robustness. Of course, when no training set instances are found we observe 0 certifiable robustness. Interestingly, we see that adding back only 10% of the training data for this class (592 images) is able to recover more than 70 percent of the certifiable robustness.

A theoretically principled approach to improving robustness is given in Chapter 7. Further, we highlight that despite displaying relatively high robustness to adversarial examples, we are unable to certify significant proportion of the posterior samples for any given class. This highlights that, while we appear to be robust to PGD adversarial examples, there may exist an attack which greatly reduces our robustness probability.

In Figure 5.5 we continue to study the probabilistic robustness of our Bayesian posteriors on MNIST, this time focusing on different network architectures. We take the input property of interest to be an ℓ_∞ ball with radius 0.01. We hypothesize that without any adversarial training, we would see a large decrease in robustness for larger networks which has been observed for non-defended models [101]. In fact, we notice that there is no correlation between model size and robustness to PGD attack for the models tested. This lack of correlation is in line with the findings of [23] who theoretically show that, in the limit of the size of a BNN, the robustness to attacks should be maximized. They further find a positive correlation between the size (in neurons) and robustness to gradient-based attacks. This is due to the gradient cancellation affect discussed in Section 2.2.1. For IBP, on the other hand, we notice a 0.2 drop in probabilistic robustness for networks with more than one hidden layer. It seems straightforward to attribute this to the increased output approximation that comes with interval bound propagation through more computation. That is, as a

network gets deeper, the approximating bound propagation pass becomes more over-approximate.

Decision Robustness In Figures 5.4c and 5.7 we study the effect of the size of the ℓ_∞ ball radius on decision robustness of a Bayesian posterior with one hidden layer and 64 hidden units. We know that probabilistic robustness is a monotonically decreasing function of the radius of the input ℓ_∞ ball. This can be observed by considering that when widening the considered radius by some value η the worst-case example for each element of the posterior either gets worst (i.e. is in the outer η shell) or remains the same worst-case example from before we widened the radius. As such, the probabilistic robustness can only decrease.

It is not entirely clear what the relationship between probabilistic robustness and decision robustness will be *a priori*. In Figures 5.4b and 5.4c we see that the notions of probabilistic robustness and decision robustness are tightly linked, only differing by a couple hundredths. In Figure 5.7, we again study the effect of the ℓ_∞ radius, this time by computing the maximal radius for which each test input can be certified to be robust (w.r.t. decision robustness). We can then compare this directly with what is obtained for SGD. We find that the certifiable radii of the Bayesian neural network decision are significantly larger than those of the DNN. Specifically, we find the median of the DNN certifiable radii to be 25% smaller compared to the BNN.

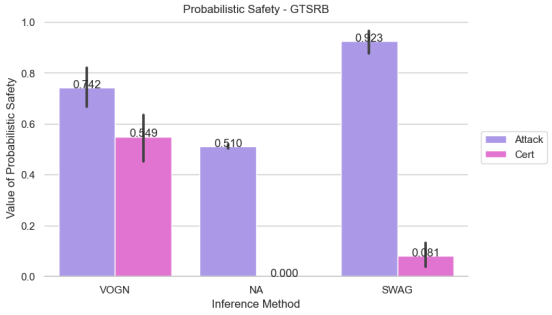
5.4.4 GTSRB

For our final case study, we analyze different approximate Bayesian posteriors on a larger-scale and more safety-critical image classification task. We use the German Traffic Sign Recognition Benchmark (GTSRB) which asks us to distinguish each of the German traffic signs from 28 by 28 RGB images. These images are three times the size of the MNIST images considered in the previous section, which brings us to an input dimensionality of 2353. In this section, we perform the same analyses as in the previous sections and focus our discussion primarily on any contrasts that exist. We start by covering the particulars of the experimental setting and then focus on how our analysis on GTSRB differs from what was observed on MNIST.

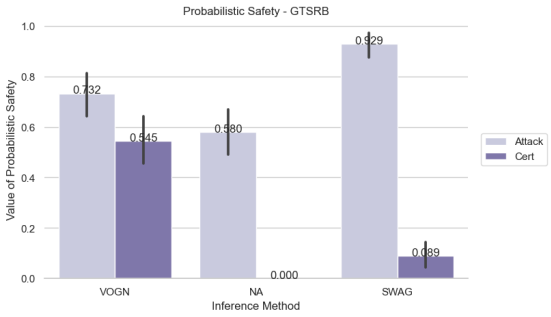
5.4.4.1 Experimental Setting

The GTSRB dataset has roughly 40,000 training set images and 10,000 test set images which are unevenly distributed over more than 40 classes. In our set up, we take a two class subset from this forty and perform random data augmentation to come up

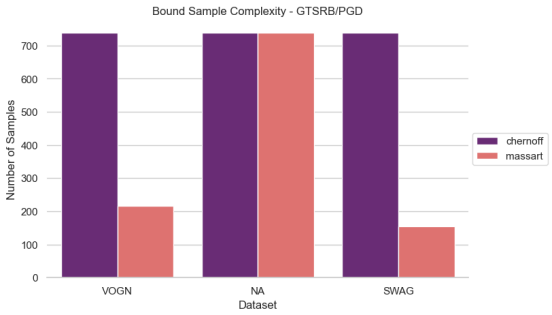
(a) We visualize two example images from different classes from the GTSRB dataset.



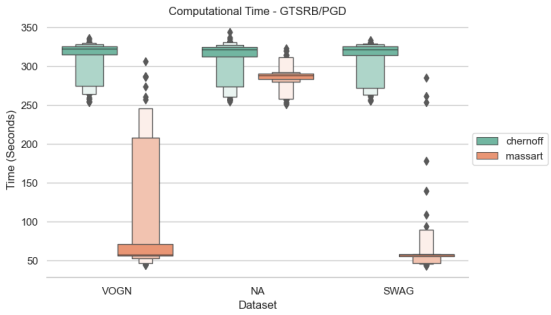
(b) We investigate how different posterior approximations perform in terms of their probabilistic robustness profiles. We find diverse profiles for different posteriors.



(c) We investigate how different posterior approximations perform in terms of their decision robustness profiles. Again, we find decision robustness is strongly linked to probabilistic robustness.



(d) Analyzing the sample complexity in a more realistic case study shows confirms that we get consistent improvements in efficiency.



(e) Inspecting the time complexity shows us that for posteriors whose robustness is near 0.5 our improvement is the most negligible.

Figure 5.8: We analyze different posterior approximation methods when studying Bayesian neural networks on a large-scale, realistic image classification dataset.

with a dataset that has 7,500 training examples and 1,000 test set examples. We then perform approximate Bayesian inference with some of the most scalable methods. In particular we choose the Variational Online Gauss Newton (VOGN), Noisy Adam (NA), and Stochastic Weight Averaging - Gaussian (SWAG). We randomly pick a validation set of 500 images in order to do our hyper-parameter optimization and tune the learning rates of our optimizer to maximize performance on these images. We use a four hidden layer neural network architecture with two convolutional layers followed by two fully connected layers. Each of the inference methods is able to achieve a test set accuracy greater than 96% on the two class problem. Given that the accuracy is so high, one may consider using such a classifier in a safety-critical scenario and thus the analysis represented in this subsection is similar to analysis one might perform if employing a Bayesian neural network in a such a safety-critical context.

5.4.4.2 Robustness Analysis

In Figure 5.8a we visualize the two class subset of the GTSRB dataset that we consider throughout this section. Given that this dataset is aimed at autonomous driving, which is inherently a more safety-critical task than digit recognition, we consider a stricter statistical guarantee. In particular, we compute $(0.05, 0.05)$ (ϵ, δ)-guarantees which we do in order to understand how our guarantees scale not only with larger and more complex neural network architectures, but also in an attempt to gauge just how computationally costly it is to perform robustness evaluation of a realistically scaled Bayesian neural network. The result of this tightening of our bounds results in an increase in the Chernoff sample bound from 150 to 750. For more on the sample complexity and computational time see the paragraph below.

Probabilistic Robustness When testing the robustness of each approximate inference method we consider an ℓ_∞ ball with radius $4/255$. Similarly to the analysis performed in the last section, we consider a prediction safe if the worst-case input inside of the input property is unable to change the predicted class of the sampled neural network or of the Bayesian decision. In Figures 5.8b and Figures 5.8c we highlight the probabilistic and decision robustness of the trained posteriors. Interestingly, we notice that VOGN has a smaller gap between its robustness to adversarial attacks and its certified robustness. An exact explanation for this phenomena is difficult to pin down, yet one hypothesis is that VOGN is performing more faithful Bayesian inference than the other methods. Thus, it is gaining an advantage through more

careful regularization (through the prior) and has better uncertainty which may also be correlated with higher robustness.

Sample Complexity and Computational Time The most noticeable difference between Figures 5.8d, 5.8e and the identical analysis on the UCI dataset (Figures 5.3b and 5.3d) is the scale of the y-axes. In particular, our analysis of the UCI dataset took only 150 samples from the posterior which required at most 30 seconds. In contrast, analysis of a larger CNN with input dimension several hundred times larger takes up to 360 seconds (6 minutes). It is not just the larger input dimensionality and architecture that causes this 10x jump in computational time, it is also the tightened (ϵ, δ) guarantees causes an 5x increase in the number of samples. From this, we can approximately infer that the sampling and attacking procedure takes twice as long as that of the small UCI networks that we have previously considered.

Despite six minutes being a relatively long time compared to UCI, it is not an unreasonable amount of time to require for safety testing a perception unit for an autonomous vehicle. We also highlight that the independence of test set examples means that we can parallelize our analysis, which in turn means we could feasibly test dozens of images in a six minute duration.

5.5 Summary

In this chapter, we showed how one can compute local robustness properties of Bayesian neural networks. The key contribution of this chapter was the introduction of algorithms which produce estimates of different notions of robustness and that these can be computed in practice with Monte Carlo integration over the posterior predictive distribution. Secondly, noticing the safety-critical nature of the domains in which local robustness is a concern, we provide exposition and adaptation of different statistical estimation schemes to our notions of robustness such that guarantees about the correctness of our estimate are satisfied. Specifically, we introduce statistical (ϵ, δ)-guarantees which allow us to make statements of the following kind: "*We are 95% certain that the given Bayesian neural network is robust to any attack algorithm with probability between 90% and 95% for the given input-output specification.*"

Methodologically, we provide several insights into the computation of quantitative statements like this including the sample complexity of being able to make statements with tighter statistical guarantees. We introduced the Chernoff bound which requires exponentially many samples with decrease in the (ϵ, δ) parameters. We further

provided insights on well-known (Hoeffding) and lesser-known (Massart) sequential sample bounds which have been used in the field of statistical model checking. The major advantage of applying the Massart or Hoeffding bounds to this problem is a polynomial reduction in the number of samples needed to reach the desired bounds on confidence and error of our estimates. The primary downside of these algorithms is their *sequential* nature. That is, we must keep a running empirical estimate that is checked before each interval. This additional logical check does not, in theory, produce a significant computational overhead, but in practice, this logical check does prevent the use of parallelization which can harm the potential theoretical speed up. Despite this, one can still use parallelization at the input level, as inputs are as assumed to be independent and identically distributed. Moreover, one can perform the logical check of the empirical estimate in a batch-wise capacity, which introduces the possibility of sampling more networks than necessary.

Finally, in this chapter we benchmarked our methodology on several different datasets and Bayesian neural networks of varying complexity. We started by demonstrating our framework in a contrived setting where we are able to visualize the weight-space function space correlation that is being leveraged. From this intuition, we began by analyzing a benchmark of six different datasets which have been previously used to study approximate Bayesian inference. We find, as we might expect that the robustness of the Bayesian neural network posteriors are strongly dependent on the dataset that they were trained on, with each dataset being conducive to different *smoothness* properties of the resulting posterior. We also observe that, in practice, we are able to achieve strong improvements in the sample complexity required by leveraging statistical model checking. This sample complexity improvement also directly translates to an improvement in the computational time for each bound. From this regression dataset we moved to the MNIST hand-written digit recognition benchmark. In our analysis of posteriors trained on MNIST, we showed how our algorithm is able to provide insights into specific robustness qualities of the posterior such as the class-wise robustness of images from the test set. Moreover, we studied the correlation between decision robustness and probabilistic robustness. We found that the two notions were strongly correlated on the MNIST dataset. We also studied the effect of architecture size on our statistical estimates of robustness and found that the deeper the architecture the more approximate our IBP robustness estimate became. Finally, we empirically studied a more realistic setting in the German Traffic Sign Recognition Benchmark. Containing real-world, color images of traffic signs, we train a Bayesian convolutional neural network and analyze its robustness properties.

In this setting, we required much tighter statistical guarantees and still found that, by and large, we preserve a significant computational advantage when employing statistical model checking to the problem of robustness estimation for Bayesian neural networks, even for large-scale networks and inputs.

In the next chapter, we will consider further algorithms for the computation of adversarial robustness properties of Bayesian neural networks and will re-analyze many of these settings to gain a better understanding of their adversarial robustness.

Chapter 6

Probabilistic Guarantees on Adversarial Robustness of Bayesian Neural Networks

Contents

6.1 On Probabilistic Guarantees

In this section, we consider a tighter class of guarantees than the statistical (ϵ, δ)-guarantees considered in the previous chapter. Namely, we consider computing sound lower and upper bounds on the probability for each of our robustness definitions. With the previous efficient framework for computing statistical (ϵ, δ)-guarantees there is no

possibility of computing a guarantee with $\delta = 0$ without infinite samples from the posterior distribution. Thus, any statistical guarantee comes with some admitted probability of failure which can be unsatisfactory in highly safety-critical environments such as airborne collision avoidance (explored in Figure 6.3). Instead, in this section we consider probabilistic guarantees which completely remove reliance on δ and compute upper and lower bounds whose function is similar to that of the ϵ parameter in the statistical guarantees in Chapter 5. Given a Bayesian neural network f^{θ} and a local property $\{T, S\}$ with probabilistic robustness $\mu_{\boldsymbol{Z}}$ we consider computing probabilities $\overline{\mu_{\boldsymbol{Z}}}$ and $\underline{\mu_{\boldsymbol{Z}}}$ such that:

$$\underline{\mu_{\boldsymbol{Z}}} \leq \mu_{\boldsymbol{Z}} \leq \overline{\mu_{\boldsymbol{Z}}}$$

A primary appeal of such a probabilistic guarantee is that in the case that $\underline{\mu_{\boldsymbol{Z}}} > 0.99$ we know that with absolute certainty (i.e., $\delta = 0$) there is a less than a 1% chance of failure for the given posterior and local property. Further, if $\overline{\mu_{\boldsymbol{Z}}} < 0.5$ we know with certainty that the 50% of the models in our Bayesian posterior do not satisfy the property under consideration. With the δ term removed from the equation, one can consider the bounds we compute in this chapter to be related to the statistical error, ϵ, in the following way: $\epsilon = \overline{\mu_{\boldsymbol{Z}}} - \underline{\mu_{\boldsymbol{Z}}}$. Further, these bounds on robustness allow us to have the same level of rigour as has been developed for deterministic neural networks, thus enabling the deployment of Bayesian neural networks with strong probabilistic guarantees in all fields where deterministic neural networks may be currently deployed.

In this section, we first outline a series of propositions which theoretically allow one to achieve sound probabilistic upper and lower bounds on probabilistic properties of Bayesian neural networks. We then recall the formulation of probabilistic robustness and formally state the necessary steps for its upper and lower bounding. After a discussion of strengths and limitations, we finish the chapter with another comprehensive empirical investigation and find that we can compute probabilistic bounds for networks in safety-critical fields such as airborne collision avoidance.

6.2 Computing Probabilistic Guarantees

6.2.1 Exact Probabilistic Safety from Maximal Safe Weight Sets

When computing statistical guarantees, we highlighted the correspondence between sampling in the weight-space and sampling in the function space. Namely, we pointed

out that by sampling weights $\theta \sim p(\theta|\mathcal{D})$ we are also sampling functions proportionally to their posterior probability. Throughout this chapter, we will assume w.l.o.g. that our parameters are distributed according to an arbitrary distribution $q(\theta)$ which can, of course, be the posterior. In this section, we begin tackling the computation of probabilistic bounds from the same observation. This time, rather than reasoning about a single weight θ, we reason about an set of infinite samples (and therefore functions) in order to theoretically build probabilistic bounds. Given a local robustness property $\{T, S\}$, we will define $Prob_{\text{safe}}(T, S, \boldsymbol{\theta})$ to be the real value probability corresponding to the model robustness of a Bayesian neural network $f^{\boldsymbol{\theta}}$. In what follows and without loss of generality, we will formulate the theoretical framework to evaluate probabilistic safety for Bayesian neural networks. To start we show the exact computation of $Prob_{\text{safe}}(T, S, \boldsymbol{\theta})$ reduces to computing the maximal set of weights for which the corresponding deterministic neural network is safe (or unsafe). To formalize this concept, consider the following definition.

Definition 9. *We say that $H \subseteq \mathbb{R}^{n_{params}}$ is the maximal safe set of weights from T to S, or simply the maximal safe set of weights, iff $H = \{\theta \in \mathbb{R}^{n_{params}} \,|\, \forall \mathbf{x} \in T, f^{\theta}(\mathbf{x}) \in S\}$. Similarly, we say that $K \subseteq \mathbb{R}^{n_{params}}$ is a maximal unsafe safe set of weights from T to S, or simply the maximal unsafe set of weights, iff $K = \{\theta \in \mathbb{R}^{n_{params}} \,|\, \exists \mathbf{x} \in T, f^{\theta}(\mathbf{x}) \notin S\}$.*

From the definition of a maximal safe weight set, we present a proposition which allows us to directly relate the maximal safe set to the probability of safety. First, we highlight that, given H is a safe set of weights, Definition 9 implies that for any $\theta \in H$ the corresponding neural network is safe, i.e., $\forall \mathbf{x} \in T, f^{\theta}(\mathbf{x}) \in S$. Then, a trivial consequence of this in conjunction with the definition of probabilistic safety is the following proposition.

Proposition 1. *Let H and K be the maximal safe and unsafe set of weights from T to S. Assume that $\theta \sim q(\theta)$. Then, it holds that*

$$\int_H q(\theta)d\theta = Prob_{safe}(T, S, \boldsymbol{\theta}) = 1 - \int_K q(\theta)d\theta. \tag{6.1}$$

Proposition 1 simply translates the safety property from the function space to an integral computation on the weight space. The intuitive interpretation of this proposition is that, if we know the maximal safe (or unsafe) set of weights, then by computing the probability of the set under the posterior we arrive exactly at the probability we desired.

The technique of reducing probabilistic safety computation to computing a maximal safe or unsafe set gives us a perspective on how we might theoretically go about computing the true probability of safety; however, for practical neural networks with support on real intervals (or any continuous support) the computation of H or K would require marginalization over an infinite space of weights. In the following section, we state practical methods for computing bounds on the true value of robustness offered by approximating H and K.

6.2.2 Bounding Probabilistic Robustness

In this section, we start from the observation that computation of the maximal safe and unsafe sets is unfeasible due to marginalization over weights being intractable. Given this, we make the following relaxation of the safe and unsafe weight sets:

Definition 10. *Given a maximal safe set H or a maximal unsafe set of weights, we say that $\hat{H}$ and $\hat{K}$ are a safe and unsafe set of weights from T to S, iff $\hat{H} \subseteq H$ and $\hat{K} \subseteq K$.*

Here, we allow $\hat{H}$ and $\hat{K}$ to be safe and unsafe sets (respectively) without requiring that they be *maximal* as before. Essentially, $\hat{H}$ and $\hat{K}$ can consist of *any* known safe and unsafe weights respectively. Without maximality, we no longer have strict equality in Proposition 1, but instead we have arrived at bounds on the value of probabilistic robustness:

Proposition 2. *Let H and K be the maximal safe and unsafe set of weights from T to S. Let $\hat{H} \subseteq H$ and $\hat{K} \subseteq K$ be non-maximal safe and unsafe weights. Assume that $\theta \sim q(\theta)$. Then, it holds that*

$$\int_{\hat{H}} q(\theta)d\theta \leq Prob_{safe}(T, S, \boldsymbol{\theta}) \leq 1 - \int_{\hat{K}} q(\theta)d\theta. \tag{6.2}$$

We highlight that the inequalities in this proposition become strict when we disallow subset equality (i.e., $\hat{H} \subset H$ and $\hat{K} \subset K$). Further, we notice that this guarantee is exactly the kind of guarantee we desired at the outset of this chapter. Now that we have relaxed the requirement on maximality of the safe and unsafe sets, we can begin to reason about building a safe or unsafe set of weights by sampling and checking safety, as in algorithms from Chapter 5, Algorithm 1 and Algorithm 2. However, the problem with simply building a finite set of point-wise safe weights is that they will inevitably have measure 0 (when the posterior has support over a continuous space) and therefore will give us vacuous bounds on the desired safety, which is discussed in more detail below.

6.2.2.1 Computing the Probability of Weight Sets

We now know that if we can build an under and over-approximation to the safe weight set then we can arrive at lower and upper bounds on probabilistic safety. Before continuing, we briefly discuss computation of the integrals involving these parameter sets. Given a parameter set (or a disjoint set of parameter sets), H, we are interested in knowing the following quantity: $\int_H p(\theta|\mathcal{D})d\theta$. In practice, we leverage the approximate posterior form arrising from various approximate inference techniques in order to compute this value. As we have discussed in Chapter 2 and 3, there are two primary forms of approximate inference for Bayesian neural networks: Markov chain Monte Carlo methods and variational inference methods. For Markov chain Monte Carlo methods, the resulting posterior is represented as a series of samples $\{\theta^{(i)}\}_{i=0}^{m}$ where m is the number of samples reached when the chain either terminated or converged. Where we assume each sample has probability $1/m$ of being drawn from the posterior, we can compute the desired probability as follows:

$$\sum_{i=0}^{m} \mathbb{I}(\theta^{(i)} \in H)/m \tag{6.3}$$

where $\mathbb{I}(\theta_i \in H)$ returns 1 if the sample is inside of the safe interval(s) and 0 otherwise.

For variational posteriors, the procedure is slightly different. Firstly, let us assume that our variational posterior takes the form of a Gaussian distribution with diagonal covariance (this is a very common assumption in Bayesian deep learning). If this is the case, then we have the following proposition to compute the integral of interest:

Proposition 3. *Assume that Σ, the covariance matrix of the posterior distribution of the weights, is diagonal with diagonal elements $\Sigma_1, ..., \Sigma_{n_w}$. Let $\hat{H}_1, ..., \hat{H}_M$ be M disjoint safe sets of weights such that, for $i \in \{1, .., M\}$, $\hat{H}_i = [l_1^i, u_1^i] \times ... \times [l_{n_w}^i, u_{n_w}^i]$ and $\hat{H}_i \cap \hat{H}_j = \emptyset$ for $i \neq j$. Then, it holds that*

$$\int_{\hat{H}_1,...,\hat{H}_M} p(\theta|\mathcal{D})d\theta \geq \sum_{i=1}^{M} \prod_{j=1}^{n_w} \frac{1}{2} \left(erf\left(\frac{\mu_j - l_j^i}{\sqrt{2\Sigma_j}} \right) - erf\left(\frac{\mu_j - u_j^i}{\sqrt{2\Sigma_j}} \right) \right).$$

where erf is the Gaussian error function.

Given that we now have a methodology for computing probability of our safe parameter sets, in conjunction with Proposition 1 and 2, we now have a methodology for computing or bounding probabilistic safety given that we have a set of safe or unsafe parameters. In the next section, we will show how to compute *weight intervals* and determine their safety for a given Bayesian neural network and local robustness specification.

6.2.2.2 Building $\hat{H}$ and $\hat{K}$ from Intervals

Here we address the problem of sequentially considering single parameter samples to build safe and unsafe sets. We introduce the notion of a *weight interval*, defined as $[\theta^L, \theta^U]$ such that $\forall i \;\; \theta_i^L < \theta_i^U$. We then show how, given a local robustness property, $\{T, S\}$, one can prove that for all $\theta \in [\theta^L, \theta^U]$, f^θ respects the local robustness property. In order to do so, we notice that the interval $[\theta^L, \theta^U]$ can be treated identically to the input intervals we have considered in interval bound propagation. That is, for a single input, we could perform interval computations with the weight parameters instead of the input property to arrive at over-approximations of the output set. Yet, in order to verify that the functions $f^{\theta'}(\mathbf{x}')$ is safe for all $\mathbf{x}' \in T$ and for all $\theta' \in [\theta^L, \theta^U]$ we need to consider jointly the worst-case input and weight sample. We highlight that computation of a point-wise worst-case for either the weight space or input space is a non-convex optimization problem and therefore cannot be efficiently solved. Instead, by leveraging convex relaxation of both of these problems, we can compute a sound over-approximation of the worst-case effect from input and weight space (jointly). In order to illustrate how this is done, we consider intervals both in the input and weight space and show how one can perform interval arithmetic to propagate worst-case errors through the network.

Before discussing our interval bound propagation technique in detail, we first recall our notation for fully-connected networks. We highlight that at this point it is convient to split θ into sets for the weights, $\{W^{(i)}\}_{i=0}^{n_{\text{layers}}}$ and biases $\{b^{(i)}\}_{i=0}^{n_{\text{layers}}}$. The formulation we give here for fully-connected networks can be extended to convolutional neural networks through the method of [17].

$$\mathbf{z}_j^{(0)} = \mathbf{x}_j, \quad \boldsymbol{\phi}_j^{(i)} = \sum_{k=1}^{n_{i-1}} \mathbf{W}_{jk}^{(i)} \mathbf{z}_k^{(i-1)} + \mathbf{b}_j^{(i)}, \quad \mathbf{z}_j^{(i)} = \sigma^{(i)}\left(\phi_j^{(i)}\right), \quad j = 1, \ldots, n_i.$$

for $k = 1, \ldots, n_{\text{layers}}$, where n_{layers} is the number of hidden layers. We also recall that $\sigma(\cdot)$ is an activation function, $W^{(k)} \in \mathbb{R}^{n_k \times n_{k-1}}$ and $b^{(k)} \in \mathbb{R}^{n_k}$ are the matrix of weights and vector of biases that correspond to the kth layer of the network and n_k is the number of neurons in the kth hidden layer. We are now interested in propagating two sets of bounds (one in the input space and one in the weight-space) through these equations in order to check if an output constraint is met. In what follows, we consider an extension of interval bound propagation (IBP) to accomplish this task.

Interval Bound Propagation for Bayesian Neural Networks In previous sections, we have discussed interval bound propagation as a means of over-approximating the worst-case output of a neural network f^θ for a given input interval $T := [\mathbf{x}^L, \mathbf{x}^U]$. The key in this propagation was *over-approximation*, i.e., that the output $[\mathbf{y}^L, \mathbf{y}^U]$ was worse than anything that could be achieved by an input $\mathbf{x}^* \in [\mathbf{x}^L, \mathbf{x}^U]$. This was completed by taking upper and lower bounds on the pre-activations by considering both $\mathbf{x}^L$ and $\mathbf{x}^U$ prior to the application of a monotonically increasing activation function. Indeed, the same can be done for a weight-space interval, $[\theta^L, \theta^U]$. One can compute the upper and lower bounds on the pre-activations in exactly the same way to achieve an output interval $[\mathbf{y}^L, \mathbf{y}^U]$ which accounts for the worst-case output achievable by manipulating the weights inside of the given weight interval.

Despite this, simply propagating with respect to $[\theta^L, \theta^U]$ or $[\mathbf{x}^L, \mathbf{x}^U]$ independently does not allow us to safely include $[\theta^L, \theta^U]$ in the set of safe weights. In order to soundly conclude that a given weight interval $[\theta^L, \theta^U]$ is safe we must over-approximate the worst-case outputs achievable from the joint application of the input and weight interval. To see this, assume we have both the worst-case weight $\theta^* \in [\theta^L, \theta^U]$ and worst-case input $\mathbf{x}^* \in [\mathbf{x}^L, \mathbf{x}^U]$. If $f^{\theta^*}(\mathbf{x}^*) \in S$, then necessarily all of the parameter settings in $[\theta^L, \theta^U]$ map inside of S as well.

Below, we show how to jointly propagate a weight interval $[\theta^L, \theta^U]$ and input interval $[\mathbf{x}^L, \mathbf{x}^U]$. This propagation takes the form of a modified formulation of a forward pass through the neural network architecture such that the resulting output bounds are an over-approximation of the outputs achievable through simultaneous modification of the input and weights (within the given intervals). Both of the propagation techniques shown below use intervals in the input and weight space. Other relaxation techniques such as linear bounds on inputs and weights can be used at the cost of greater computational complexity. Another difficulty with the extension to more complex relaxation techniques is the computation of non-overlapping safe weight sets. Dealing with these complications is a valuable direction of research for future work.

Proposition 4. *Let $f^\theta(\mathbf{x})$ be the network defined by the set of Equations (3.2)–(3.3) with K hidden layers, let for $k = 0, \ldots, K$. Further, let $W_{:j}$ represent the j^{th} column of the matrix W.*

$$t_{ij}^{(k),L} = \min\{W_{ij}^{(k),L} z_j^{(k),L}, W_{ij}^{(k),U} z_j^{(k),L}, W_{ij}^{(k),L} z_j^{(k),U}, W_{ij}^{(k),U} z_j^{(k),U}\}$$
$$t_{ij}^{(k),U} = \max\{W_{ij}^{(k),L} z_j^{(k),L}, W_{ij}^{(k),U} z_j^{(k),L}, W_{ij}^{(k),L} z_j^{(k),U}, W_{ij}^{(k),U} z_j^{(k),U}\}$$

where $i = 1, \ldots, n_{k+1}$, $j = 1, \ldots, n_k$, *and* $z^{(k),L} = \sigma(\zeta^{(k),L})$, $z^{(k),U} = \sigma(\zeta^{(k),U})$ *and:*

$$\zeta^{(k+1),L} = \sum_j t_{:j}^{(k),L} + b^{(k),L}$$
$$\zeta^{(k+1),U} = \sum_j t_{:j}^{(k),U} + b^{(k),U}.$$

Then we have that $\forall x \in T$ *and* $\forall w \in \hat{H}$*:*

$$f^w(x) = \zeta^{(K+1)} \in \left[\zeta^{(K+1),L}, \zeta^{(K+1),U}\right].$$

The proposition above, whose proof is in the Appendix (this can also be found in [150]), yields a bounding box for the output of the neural network in T and $\hat{H}$. We notice that this bound makes use of element-wise minimum and maximum computations (for each of the bound combinations). Doing so allows us to achieve tighter bounds on the output interval than naively combining the previous two notions of IBP. However, element-wise minimum and maximum are slightly more expensive on modern computational hardware and can sometimes be difficult to implement correctly in auto-differentiation software. As such, we also provide a looser but computationally faster method, which may be easier to implement in modern auto-differentiation software. *We employ the center point method that is used for deterministic certification but with a center point over weights as well. The intuition for this is that that the over-approximation holds in one case and so ought to hold in the second. A formal proof that this is true can be found in the Appendix for this chapter.* :

Proposition 5. *Let* $f^\theta(x)$ *be the network defined by the set of Equations* (3.2)–(3.3) *with* K *hidden layers, let for* $k = 0, \ldots, K$*:*

$$z^{(k),c} = \frac{z^{(k),U} + z^{(k),L}}{2} \qquad W^{(k),c} = \frac{W^{(k),U} + W^{(k),L}}{2}$$
$$z^{(k),r} = \frac{z^{(k),U} - z^{(k),L}}{2} \qquad (6.4) \qquad W^{(k),r} = \frac{W^{(k),U} - W^{(k),L}}{2} \qquad (6.5)$$

$$t^{(k),c} = W^{(k),c} \cdot z^{(k),c} \qquad (6.6)$$

$$\begin{aligned}\zeta^{(k+1),U} &= t^{(k),c} + |W^{(k),c}| \cdot z^{(k),r} + W^{(k),r} \cdot |z^{(k),c}| + |W^{(k),r}| \cdot |z^{(k),r}| + b^{(k),U} \\ \zeta^{(k+1),L} &= t^{(k),c} - |W^{(k),c}| \cdot z^{(k),r} - W^{(k),r} \cdot |z^{(k),c}| - |W^{(k),r}| \cdot |z^{(k),r}| + b^{(k),L}\end{aligned} \qquad (6.7)$$

Then we have that $\forall x \in T$ *and* $\forall w \in \hat{H}$*:*

$$f^w(x) = \zeta^{(K+1)} \in \left[\zeta^{(K+1),L}, \zeta^{(K+1),U}\right].$$

The proof of this proposition can also be found in the Appendix.

6.3 Bounds on Probabilistic Robustness

In this section we present our methodology for computing upper and lower bounds on different notions of robustness algorithmically. For each algorithm, we first analytically express the bound on the desired robustness quantity and then we proceed to discuss exactly how each algorithm proceeds in computing the bound as well as any practical parameter choices that need to be made in order for the algorithm to function properly.

Algorithm 3 Lower Bounding Probabilistic Safety for BNNs

Input: T – Compact Input Region, S – Safe Output Set, f^{θ} – Bayesian Neural Network, $p(\theta|\mathcal{D})$ – Posterior Distribution, N – Number of Samples, γ – Weight margin.
Output: Safe lower bound on probabilistic robustness.

$\hat{H} \leftarrow \emptyset$ # $\hat{H}$ is a set of known safe weight intervals
for $i \leftarrow 0$ to N **do**
  $\theta^{(i)} \sim p(\theta|\mathcal{D})$
  $[\theta^{(i),L}, \theta^{(i),U}] \leftarrow [\theta_i - \gamma, \theta_i + \gamma]$
  $\mathbf{y}^L, \mathbf{y}^U \leftarrow \texttt{Method}(f, T, [\theta^{(i),L}, \theta^{(i),U}])$ *# Interval Bound Propagation*
  if $\forall \mathbf{y} \in [y^L, y^U] \mathbf{y} \in S)$ **then**
    $\hat{H} \leftarrow \hat{H} \bigcup \{[\theta^{(i),L}, \theta^{(i),U}]\}$
  end if
end for
$\hat{H} \leftarrow$ MergeOverlappingRectangles$(\hat{H})$ *# Ensure disjoint intervals Prop. 3*
$p \leftarrow 0.0$
if $\hat{H} \neq \emptyset$ **then**
  for $[\theta^{(i),L}, \theta^{(i),U}] \in \hat{H}$ **do**
    $p \leftarrow p + Prob_{p(\theta|\mathcal{D})}([\theta^{(i),L}, \theta^{(i),U}])$ *# See Section 6.2.2.1*
  end for
end if
return p

6.3.1 Sound Lower Bounds on Probabilistic Robustness

Throughout this chapter we have discussed the case of lower-bounding probabilistic robustness. We established that if we knew all of the weight settings for which the local robustness property holds, then we would know the exact probabilistic robustness. Given that we cannot know all such safe weights, we turned our attention to subsets of the safe weights. In particular, we highlighted that if we knew a strict subset of

the safe-weights, then we would have a lower bound on the probabilistic robustness. Given such a subset of safe weights $\hat{H}$, we have the following bound on probabilistic safety from the first half of Proposition 2:

$$\int_{\theta \in \hat{H}} q(\theta) d\theta \leq Prob_{\text{safe}}(T, S, \boldsymbol{\theta})$$

In our discussion, what remained was to compute such an $\hat{H}$ which we showed was possible through bound propagation in the weight and input space simultaneously. In Algorithm 3 we perform each of these steps sequentially in order to arrive at a safe set of weights $\hat{H}$ as well as the corresponding lower bound on $P_{\text{safe}}(T, S, \boldsymbol{\theta})$.

The parameters in Algorithm 3 are the local property of interest $\{T, S\}$, a Bayesian neural network $f^{\boldsymbol{\theta}}$, and a number of samples to take from the posterior, N. For posteriors with continuous support, we introduce a weight margin (γ) in order to define a interval around the sampled weight. A key insight for the proper functioning of the algorithm is to scale γ by the covariance matrix of the variational posterior (in the case of MCMC posteriors no weight margin is used). This allows for the weight margin to be specified in number of standard deviations above and below the sample which in turn allows us to know approximately what probability density each sampled interval will cover.

On line 1, we initialize the set of safe weights to the empty set, corresponding to setting the lower bound at probability 0. We then proceed to sample and check N different weight intervals. For each sampled weight, we build the interval on line 4 (assuming the support is continuous) and then propagate bounds through the network in order to get an over-approximation of the output of the neural network. If the entire output is within the safe set, then we are able to soundly conclude that the weight interval is in the safe set and take the union of the already computed safe set with the current weights. After we have checked all of the sampled weight intervals we proceed to *merge overlapping interval*. Merging overlapping intervals simply enforces that any θ^i belongs to only one member of the set $\hat{H}$ which is a precondition of Proposition 3. Once we have arrived at a subset of the safe weight set, we must compute the probability of seeing these weights under the Bayesian posterior. This is done as described in Section 6.2.2.1.

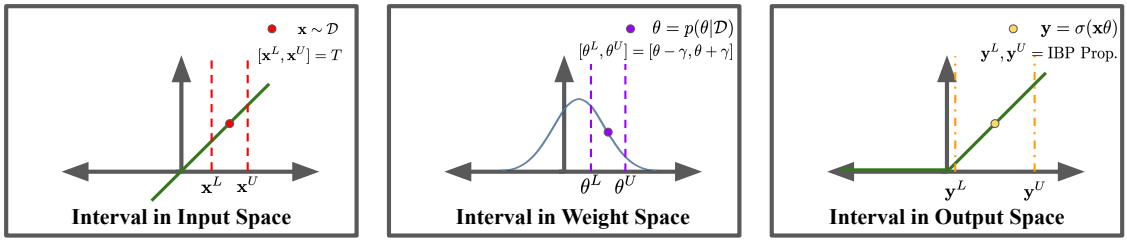
6.3.2 Sound Upper Bounds on Probabilistic Robustness

Throughout our exposition of computing bounds on $Prob_{\text{safe}}(T, S, \boldsymbol{\theta})$, though we focused on computing safe weights $\hat{H}$, we also stated our theorem for unsafe weights

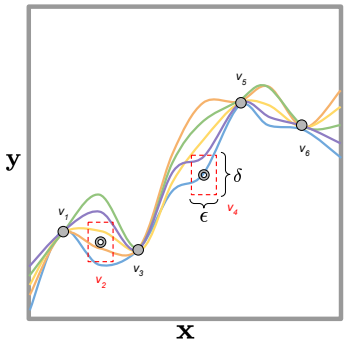
Algorithm 4 Upper Bounding Probabilistic Safety for BNNs

Input: T – Input Set, S – Safe Set, f^{θ} – Bayesian Neural Network, $\boldsymbol{\theta}$ – Posterior Distribution, N – Number of Samples, γ – Weight Margin.
Output: Safe upper bound on probabilistic robustness.

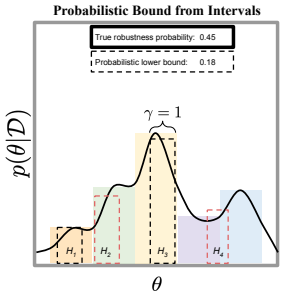
$\hat{K} \leftarrow \emptyset$ *# $\hat{K}$ is a set of known unsafe weight intervals*
for $i \leftarrow 0$ to N **do**
  $\theta^{(i)} \sim \boldsymbol{\theta}$
  $\mathbf{x}_{\text{adv}} \leftarrow \texttt{Attack}(f, \theta_i, T)$ *# FGSM/PGD*
  $[\theta^{(i),L}, \theta^{(i),U}] \leftarrow [\theta^{(i)} - \gamma, \theta^{(i)} + \gamma]$
  $\mathbf{y}^L, \mathbf{y}^U \leftarrow \texttt{Method}(f, \mathbf{x}_{\text{adv}}, [\theta^{(i),L}, \theta^{(i),U}])$ *# Interval Bound Propagation*
  if $\forall \mathbf{y} \in [y^L, y^U] \mathbf{y} \notin S)$ **then**
    $\hat{K} \leftarrow \hat{K} \bigcup \{[\theta^{(i),L}, \theta^{(i),U}]\}$
  end if
end for
$\hat{H} \leftarrow \text{MergeOverlappingRectangles}(\hat{H})$ *# Ensure disjoint intervals Prop. 3*
$p \leftarrow 0.0$
if $\hat{H} \neq \emptyset$ **then**
  for $[\theta^{(i),L}, \theta^{(i),U}] \in \hat{H}$ **do**
    $p \leftarrow p + Prob_{p(\theta|\mathcal{D})}([\theta^{(i),L}, \theta^{(i),U}])$ *# See Section 6.2.2.1*
  end for
end if
return p



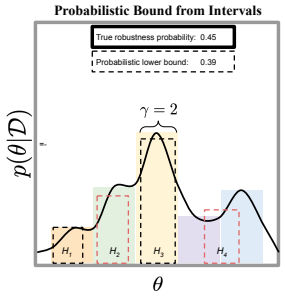
(a) We plot an example of a 1D input, single-neuron example of our algorithm for one iteration. **Left:** We take any input $\mathbf{x}$ and its input interval as specified by the user , **Center:** After sampling the posterior, we take an interval in the weight space based on a margin γ the effect of which is further explored below. **Right:** By using proposition 4 we are able to arrive at bounds on the output. Checking the safety of this interval will allow us to either include or exclude the interval $[\theta^L, \theta^U]$.



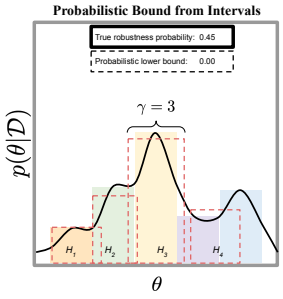
(b) We again consider our regression running example from the previous chapters, this time allowing for the visualized samples to be from a posterior with continuous support (visualized in subfigures b-d). We consider $\hat{H} = \{H_i\}_{i=0}^4$ as visualized by the disjoint intervals above. Red indicates not robust weight intervals, black indicates robust intervals.



(c) For small weight interval ($\gamma = 1$) we recover an overly conservative probability estimate.



(d) For medium weight margin ($\gamma = 2$), we would recover a better lower bound than for smaller weight margin.



(e) For large weight margin ($\gamma = 3$), we are considering too large of a range in the parameter space and achieve only a vacuous lower bound of 0.

Figure 6.1: We demonstrate the workings of our algorithm in (a), and the effect of of γ, the weight margin, by using our regression running example in (b) and (c-e).

$\hat{K}$. Given a local robustness property $\{T, S\}$ and the induced safety probability, $Prob_{\text{safe}}(T, S, \boldsymbol{\theta})$, one can consider an upper bound to the probabilistic safety by first considering a sound bound on the probability of *unsafety*. To do so, we imagine we have access to every weight such that the adversarial property does not hold. Of course, we could then compute the probability of such weights (i.e., $Prob_{\text{unsafe}}(T, S, \boldsymbol{\theta})$) and this would give us an exact bound on the safety of the network. This is a straightforward implication of the fact that a particular weight is either safe or unsafe. Thus, the maximal safe weight set and the maximal unsafe weight set are disjoint and we know that $Prob_{\text{unsafe}}(T, S, \boldsymbol{\theta}) + Prob_{\text{safe}}(T, S, \boldsymbol{\theta}) = 1$, and therefore $Prob_{\text{safe}}(T, S, \boldsymbol{\theta}) = 1 - Prob_{\text{unsafe}}(T, S, \boldsymbol{\theta})$. To observe how an upper bound is established consider a subset of the maximal unsafe set weights, $\hat{K}$, then we have the following bound directly from the second half of Proposition 2:

$$\int_{\theta \in \hat{K}} q(\theta) d\theta \leq Prob_{\text{unsafe}}(T, S, \boldsymbol{\theta}),$$

and correspondingly via simple application of the relations given above we have,

$$Prob_{\text{safe}}(T, S, \boldsymbol{\theta}) \leq 1 - \int_{\theta \in \hat{K}} q(\theta) d\theta.$$

To compute this bound, we can proceed exactly as we did for computing a safe lower bound to probabilistic safety save for the way in which we check if our output interval is unsafe. In order to determine that a weight is unsafe, we must check that for at least one input in the input set the output is unsafe. This is the logical inverse of our safety property: $\neg(f^{\theta}(\mathbf{x}) \in S \ \forall \mathbf{x} \in T) = (\exists \mathbf{x} \ s.t. \ f^{\theta}(\mathbf{x}) \notin S)$. Notice that, unlike the procedure for computing safety, here we do not need to jointly propagate a weight-space interval as we only need to find an $\mathbf{x}$ which causes the entire weight interval to be mapped outside of S. Luckily, computing an input such that the output property is violated is identical to the formulation for adversarial examples. Thus, in order to test if there exists a single input that causes the weight interval to be unsafe, we leverage the developments in adversarial attacks in order to attack each sampled weight θ_i (see line 4 of Algorithm 3). We highlight that our algorithm is transparent to the methods of computing adversarial examples as well as the method for performing bound propagation. As such, users can select an attack or propagation method corresponding to the desired computational complexity. Moreover, using fast methods such as FGSM and IBP require the computation equivalent of only four forward passes through the neural network architecture, which can be done efficiently with modern deep learning software.

6.4 Empirical Investigation

6.4.1 Intuitive Examples

In this section, we start by extending our running example set up in order to provide visual aids for descriptions of Algorithms 3 and 4. Following this discussion, take a very similar regression task and learn a small Bayesian neural network posterior. We use both of these simple settings to set up and then confirm intuitions about the proposed methodologies.

Running Example In Figure 6.1(a-e) we give a visual intuition for the working of our proposed methodology. In Figure 6.1a, we briefly demonstrate how interval bound propagation works for the simplest Bayesian neural network. Consider a single hidden unit, one input, one output Bayesian neural network with no bias. This can be expressed as $\mathbf{y} = \sigma(\boldsymbol{\theta}\mathbf{x})$. By sampling, $\theta' \sim p(\theta|\mathcal{D})$, we get a sampled function $\mathbf{y} = \sigma(\theta\mathbf{x})$ which is visualised as the solid dots in In Figure 6.1a. In the first panel of Figure 6.1a, we consider an interval in the input space which is represented as an interval over the identity function. In the second panel of Figure 6.1a we represent a sample from the posterior distribution over our parameter and its corresponding interval which we arrive at by taking $\pm\gamma$. Recall that, in practice, γ is typically scaled by the variance of the posterior (where available). In the right-most panel, we have the resulting bounds in output space. This comes from Proposition 4. We can provide a simple visual intuition here for how Proposition 4 works. Consider that we only have four possible combinations of inputs to the function σ these inputs are: $\{\mathbf{x}^L\theta^L, \mathbf{x}^L\theta^U, \mathbf{x}^U\theta^L, \mathbf{x}^U\theta^U\}$. Given that σ is monotonically increasing, we know that the max and min of these options will maximize and minimize σ, respectively.

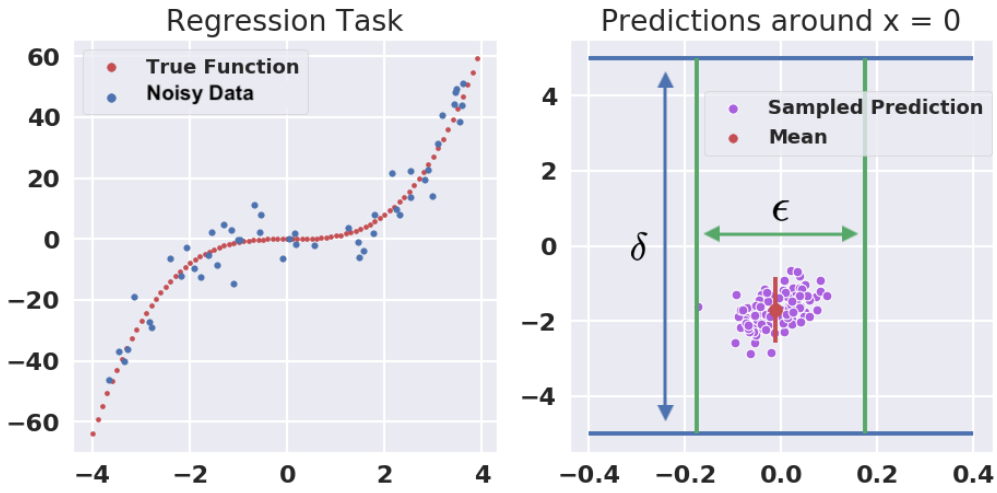
While $\mathbf{x}^L$ and $\mathbf{x}^U$ are chosen according to some desired robustness specification, θ^L and θ^U (or more simply just the margin γ) are selected arbitrarily. In Figure 6.1(c-e) we briefly reason about how this selection of γ matters in practice. We consider computing the robustness of test input v_2 from our running example which is recalled in Figure 6.1b. In previous examples, we knew from having access to a clean posterior mass function that the probabilistic model safety was 0.45. In this example, we consider approximating the previous probability mass function with a smooth and continuous (but not analytical) probability density function which can be seen in black in Figures 6.1(c-e). We continue our assumption that the true probabilistic safety is 0.45 and that samples from the colored region of the density are arbitrarily similar (functionally) to the corresponding colored function in the running example,

Figure 6.1b. Given this set up we reason about selecting different values of γ. In Figure 6.1c, we see that for small γ we are able to sample safe regions, but the density considered inside of the range of the weight margin is too small to build up a tight lower bound. For example these regions my only cover 0.18 of the total robust region which has cumulative density 0.45. One solution to this is to simply take more sampled weight regions. In the next panel, Figure 6.1d, we consider a larger weight margin which allows us to both verify the safety of the sampled weight intervals and build reasonable probability mass. In this case, we showed that we were able to build up a lower bound of 0.39 which is considerably closer to the true value of the probabilistic model robustness. In order to potentially increase the bound even further, in Figure 6.1e, we represent what happens when we take γ to be even larger. The result is that all of the sampled intervals are too large to verify (in this case, no matter where we place them). This yields a vacuous lower bound of 0 as no weight was found to be safe.

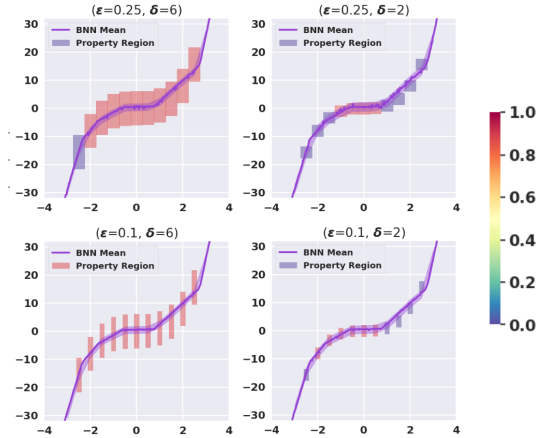
One might observe that we could have solved the problem that arose in Figure 6.1c by taking more samples. Perhaps, taking twice as many samples would have yielded an equivalent lower bound to adjusting margin (as in Figure 6.1d). However it would have been twice as computationally expensive to check all of the new intervals. While increasing the margin can lead to efficient computation of probabilistic robustness, we observed in Figure 6.1e that making the margin too wide results in vacuous bounds. Thus, selecting the margin and number of samples is a balancing act. It is more computationally expensive to check more weight intervals compared to checking fewer with a larger weight margin, but having a margin too large makes verification vacuous.

Polynomial Regression Example To confirm our hypotheses that we established in the above, we adopt a simple experimental setting that that was studied in [90, 67]. Namely, we consider a regression task where we learn a Bayesian neural network from noisy data centred around the function $y = x^3$, as illustrated in Figure 6.2. To generate our dataset, we draw 50 values (taken as inputs $\mathbf{x}$) uniformly from the interval $[-4, 4]$ and assign them a $\mathbf{y}$ label according to $\mathbf{x}^3 + \mathcal{N}(0, 4)$. Given this dataset, we train a Bayesian neural network with 128 hidden neurons for 10 epochs.

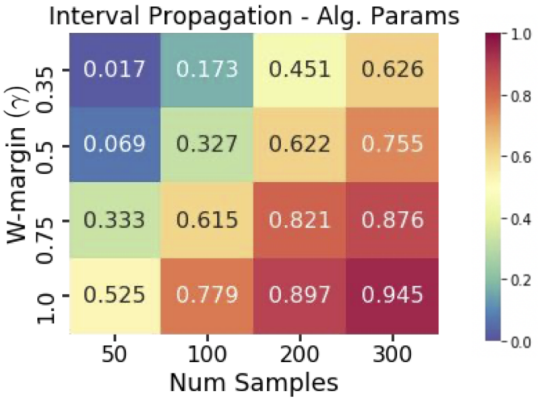
We illustrate what a potential property of interest looks like in this scenario by setting $T = [-\epsilon, \epsilon]$ and $S = [-\delta, \delta]$, with $\epsilon = 0.2$ and $\delta = 5$ as the input and output specification. This can be seen in the right half of Figure 6.2a. We use Algorithm 3 to compute $P_{\text{safe}}(T, S)$, that is, the probability that for all $x \in [-\epsilon, \epsilon]$, $f^{\theta}(x) \in [-\delta, \delta]$.



(a) **Left:** An illustration of the regression problem with true function in red and the noisy data in blue. **Right:** Sampled BNN predictions which are all within in property region defined by ϵ and δ.



(b) Lower bound on the probabilistic safety of the BNN given different input output properties. Boxes represent the properties and are colored by their lower bound.



(c) We plot the effect of the primary hyper-parameters for our bounds: γ the weight margin, and the number of samples. As both increase we are considering more of the posterior and get a tighter lower bound.

Figure 6.2: Simple robustness tests and illustrations for a toy regression example.

We highlight that this specification is identical to the kind of specification that was considered in our running example above.

In the rightmost plot of Figure 6.2c, we analyze performance of Algorithm 3 under different parameter settings. In particular, we try a range of values for N (number of samples) and γ (the weight interval/margin). The general trend hypothesized in our previous section is realized: we see an increase in the lower bound for increased samples (N) and increased weight margin (γ). Further, we notice that our discussion on increasing margin or number of samples was fairly accurate. Given the smallest margin tested (0.35) one could either evaluate more samples, making the algorithm 4 to 6 times more computationally expensive, or one can simply increase the margin. Increasing the margin in this case does not increase the computational load, but increases the lower bound considerably. Increasing both, taking 300 samples with margin 1.0, we are able to get a lower bound probability of 0.945. Not only is this bound both sound and tight, but it is tighter than any bound achievable by the Massart bound with that many samples. We highlight in Figure 6.2c that we never ran into an issue with a margin too large as in Figure 6.1e likely due to the fact that the property was true everywhere and thus it would have been difficult to arrive at a weight margin that causes vacuous bounds in this case.

In Figure 6.2b, we continue our analysis of this setting by expanding the properties we consider. In particular, we center 11 properties at $\mathbf{x}^{(k)} = (k-5)/2$ for $k \in \{0 \dots 10\}$. For each point, the input specification is taken to an ℓ_∞ ball centered at the point with radius ϵ: $T_\epsilon^{(k)} := [\frac{(k-5)}{2} - \epsilon, \frac{(k-5)}{2} + \epsilon]$. The output specification is taken to be an ℓ_∞ ball centered at the true output and parameterized by δ: $S := [\big(\frac{(k-5)}{2}\big)^3 - \delta, \big(\frac{(k-5)}{2}\big)^3 + \delta]$. In conjunction, these properties constitute a Lipschitz continuity property of the Bayesian neural network samples. We visualize these properties for four different property specifications for the combination of $\epsilon \in \{0.1, 0.25\}$ and $\delta \in \{2, 6\}$.

In the upper-left quadrant of Figure 6.2b we visualize the property that is visualized in Figure 6.2c but shifted around to different $\mathbf{x}$ values. We see that for most selected values of $\mathbf{x}$ we are able to prove the property holds for the given Bayesian posterior. However, for the value $\mathbf{x} = \langle -2.5 \rangle$ we get a lower bound of 0. Upon closer inspection, we can see that this is not actually a cause for concern as it is not clear that the true function (drawn in purple) respects the given property. By tightening δ (upper right plot of Figure 6.2b) we see that fewer of the properties are specifiable as we might expect. Conversely, increasing ϵ (lower left plot of Figure 6.2b) we are able to prove the property holds everywhere that it is tested. Finally, we decrease ϵ and δ

Method	Property	P_{safe}	Time (s)	Num. Samples
IBP	ϕ_1	0.9739	136	10500
	ϕ_2	0.9701	117	9000
	ϕ_3	0.9999	26	2000
	ϕ_4	0.9999	26	2000

Table 6.1: VCAS probabilistic lower bound. ϕ_1, ϕ_2 check consistency of DES1500 and CLI1500, respectively. ϕ_3, ϕ_4 check for the lack of dangerous DES1500 and CLI1500 predictions, respectively.

(lower right plot of Figure 6.2b) and find that this tighter property is only verifiable in flat portions of the function.

In summary, in this section we realized the hypotheses generated by our intuitive visualization of our algorithm In the following sections we will turn our attention to realistic scenarios to assess the practical applicability of our algorithms.

6.4.2 Aircraft Collision Avoidance

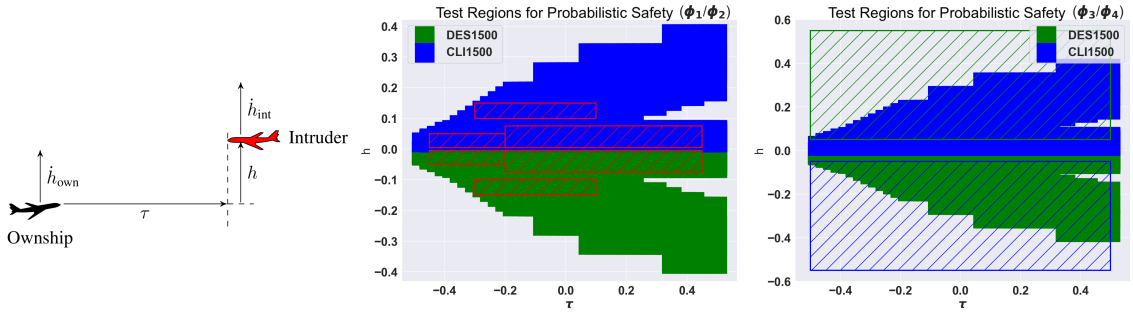


Figure 6.3: VCAS encounter geometry and properties under consideration. Left: Taken from [77], a visualization of the encounter geometry and the four variables that describe it (distance τ, ownship heading $\dot{h}$ own, intruder heading $\dot{h}_{int}$, and vertical separation h). Center: Visualization of ground truth labels (in color); red boxes indicate hyper-rectangles that make up the input areas for property ϕ_1 (red boxes in the blue area) and ϕ_2 (red boxes in the green area). Right: Hyper-rectangle for visualization of properties ϕ_3 and ϕ_4: we ensure that DES1500 is not predicted in the green striped box and CLI1500 is not predicted in the blue striped box.

We empirically evaluate probabilistic safety for the vertical collision avoidance system dataset (VCAS) [77]. The task of the original neural network is to take as input the information about the geometric layout (heading, location, and speed) of the ownship and intruder, and return a warning if the ownship's current heading puts it on course for a near midair collision (NMAC). The VCAS dataset consists of 36.4

million examples of intruder scenarios, each of which is described using four variables (displayed in the leftmost component of Figure 6.3, taken from [77]). There are four input variables describing the scenario (Figure 6.3) and nine possible advisories corresponding to nine output dimensions. Each output is assigned a real-valued *reward*. The maximum reward advisory indicates the safest warning given the current intruder geometry. The three most prominent advisories are clear of conflict (COC), descend at a rate $\geq$ 1500 ft/min (DES1500), and climb at a rate $\geq$ 1500 ft/min (CLI1500). We train a BNN with one hidden layer with 512 hidden neurons that focuses on the situation in which the ownship's previous warning was COC, where we would like to predict if the intruder has moved into a position which requires action. This scenario is represented by roughly 5 million entries in the VCAS dataset and training our BNN with VI results in test accuracy of 91%. We use probabilistic local robustness to evaluate whether the network is robust to four properties, referred to as ϕ_1, ϕ_2, ϕ_3 and ϕ_4 which comprise a probabilistic extension of those considered for NNs in [79, 145]. Properties ϕ_1 and ϕ_2 test the consistency of DES1500 and CLI1500 advisories: given a region in the input space, ϕ_1 and ϕ_2 ensure that the output is constrained such that DES1500 and CLI1500 are the maximal advisories for all points in the region, respectively. On the other hand, ϕ_3 and ϕ_4 test that, given a hyper-rectangle in the input space, no point in the hyper-rectangle causes DES1500 or CLI1500 to be the maximal advisory. The properties we test are depicted in the centre and right plot of Figure 6.3.

In Table 6.1 we report the results of the verification of the above properties, along with their computational times and the number of weights sampled for the verification. Our implementation of Algorithm 3 with IBP is able to compute a tight lower bound for probabilistic safety with these properties in a few hundreds of seconds.[1]

This case study and the results obtained validate the necessity and value of Algorithm 3. The benefits of calibrated uncertainty in the case of airborne collision avoidance are clear: if there is a case which is not captured by the training data, we would not like for a confident advisory to be issued. However, calibrated uncertainty is typically not achievable with deterministic neural networks and so one would seek to use Bayesian neural networks. Despite the value of Bayesian neural networks in this setting, it was previously impossible to verify the correctness of their behavior. Thus, it was impossible to deploy BNNs in this setting. Through the guarantees

[1]Note that in the case of ϕ_1 and ϕ_2 the input set T is composed of three disjoint boxes. Our framework can be used on such sets by computing probabilistic safety on each box and then combining the results together via the union bound.

of Algorithm 3, we can now employ Bayesian neural networks in airborne collision avoidance and a host of other tasks with similar requirements.

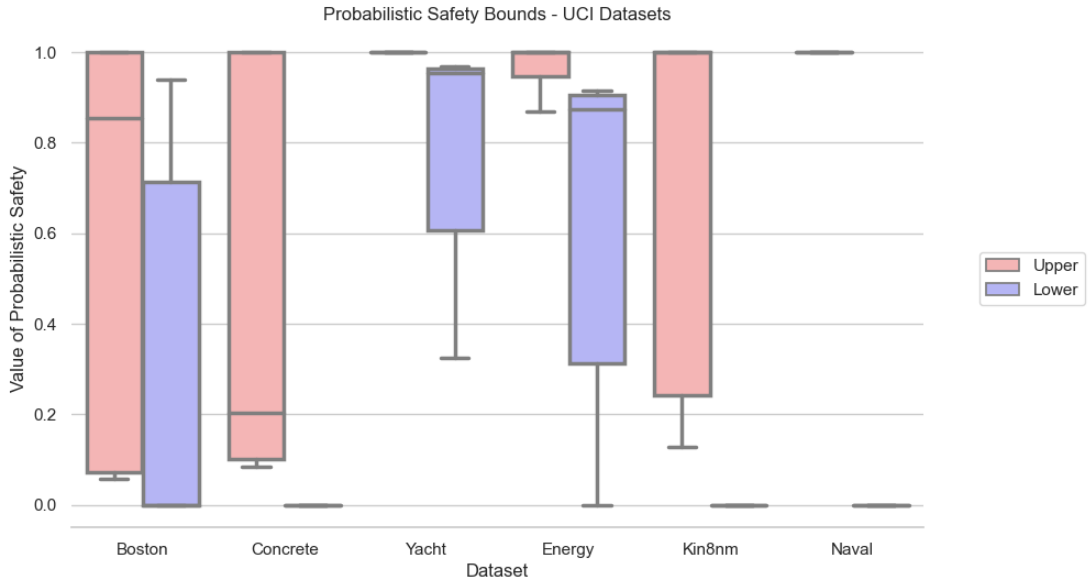


Figure 6.4: We visualize the probabilistic upper bounds (red) and lower bounds (blue) for each of the UCI datasets that was inspected. With the exception of the Naval dataset we find that we are able to compute tight bounds.

6.4.3 UCI Datasets

We again study the UCI Regression Benchmark [38] which was studied in [52] and [82]. In this section, we adopt the same small networks tested in the previous chapter. This allows us to directly compare the upper and lower bounding algorithms' in this chapter to the empirical estimates obtained in the Chapter 4. We begin by more briefly recalling the meaning of each dataset with emphasis on *why* one might want to verify these settings in order to further highlight the importance of sound verification. We then discuss our algorithms' performance on these datasets and reason about the performance of our bounds through various experiments.

	Boston	Concrete	Yacht	Energy	Kin8nm	Naval
$\overline{\mu_{\boldsymbol{Z}}}$ (Alg. 4)	0.610	0.493	0.999	0.834	0.855	0.989
$\hat{\mu_{\boldsymbol{Z}}}$ (Alg. 1)	0.568	0.388	0.883	0.827	0.471	0.0
$\underline{\mu_{\boldsymbol{Z}}}$ (Alg. 3)	0.258	0.111	0.718	0.600	0.108	0.0

Table 6.2: This table, where we take $\mu_{\boldsymbol{Z}}$ to be the probabilistic robustness, represents the average upper bounds (top row), empirical estimate (middle row), and lower bounds (bottom row) for a few hundred test set samples from the UCI dataset. Further visualization of the bounds can be found in Figure 5.3a for the middle row and Figure 6.4 for the top and bottom rows.

6.4.3.1 Experimental Setting

Datasets In this section we briefly discuss *why* one may want to verify the UCI datasets that are studied in this thesis. We emphasize that for some applications the safety-critical nature likely does not warrant the expensive verification procedure proposed here; however, we put forward these reasons in order to have context for the practical interpretation of our bounds. **Boston Housing** predicting property values may be of direct interest to financial institutions who want to ensure that appraised value of a property (and subsequently a loan given) does not result in the bank being over-leveraged should a borrower default. Ensuring that we correctly predict the compressive strength for the **Concrete** dataset is of paramount importance to infrastructure projects where catastrophic failure can result in fatalities. Predicting the residuary resistance of a yacht design **Yacht** may be of interest to regulatory bodies who want to ensure that such designs meet energy efficiency standards. Similarly, ensuring that the cooling/heating load of a proposed building predicted by the **Energy** efficiency dataset can allow building planners to have confidence that they are within regulatory limits prior to commencing construction. Finally, the **Kin8nm** and the the **Naval** dataset are both the results of modelling dynamics of different non-linear systems. Accurately capturing theese models can be of direct interest to control engineers who seek to ensure that controllers in safety-critical scenarios (i.e., autonomous robots and naval navigation) are sound, see [151] for further examples of this.

6.4.3.2 Analysis

In this section we compare lower and upper bounds from Algorithms 3 and 4 respectively. In order to do so, we compute upper and lower bounds for each posterior on 250 test set inputs. We use 2500 posterior samples with a margin of 1.5 for the upper bounds and a margin of 2.0 for the lower bound as these were empirically found to give consistent bounds across all applications tested.

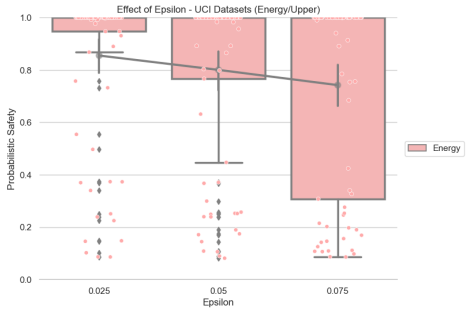
Probabilistic Bounds In Figure 6.4 as well as Table 6.2 we present the results of our analysis of each dataset with probabilistic bounds on local model robustness. In particular, we take the same properties that were studied in Figure 5.3a so that we can compare the upper and lower bounds to the empirical estimates that were taken. We recall that, for each test set example, we build the input region T as an ℓ_∞ ball with radius 0.01 (all training inputs are scaled to $[0, 1]$ and then these scaling values are imposed onto the test set inputs as is standard). The output specification, S, is then taken to be an ℓ_∞ ball with radius 0.05, which is centered at the mean output of the Bayesian neural network (where again the space is mapped to the unit interval). Thus, the specification can be interpreted as *a 1% change in the input should not correspond to more than a 5% difference in the neural network output.*

In Figure 6.4 we utilize box plots to visualize the distribution of the upper and lower bounds for the tested inputs for each posterior. We notice that the tightness of the bounds computed are very dataset dependent. For example, for inputs drawn from the Yacht dataset, we find that our upper bounding algorithm (Algorithm 4) is unable to come up with tight bounds for many inputs. This is not cause for concern, as Algorithm 3 is able to readily verify that the property of interest holds with high probability for many inputs. On the Kin8nm dataset, on the other hand, we find the opposite is true. Algorithm 4 is able to find some very useful upper bounds which prove that the property of interest does not hold, while Algorithm 3 is unable to compute non-trivial bounds for many inputs. In both of these cases, we emphasize that the failure of our algorithms to find tight bounds is likely not due to an insufficiency in the algorithm, but in the fact that the property of interest is too tight or loose to verify upper bounds or lower bounds, respectively. In the case of the Boston dataset, for example, we are able to verify tight lower bounds for some inputs (> 0.9) and tight upper bounds for other inputs (< 0.1). This is not the case for the Naval dataset. For the Naval dataset, neither Algorithm 4 nor Algorithm 3 is able to arrive at non-trivial bounds for the inputs tested. Without further inspection, it is

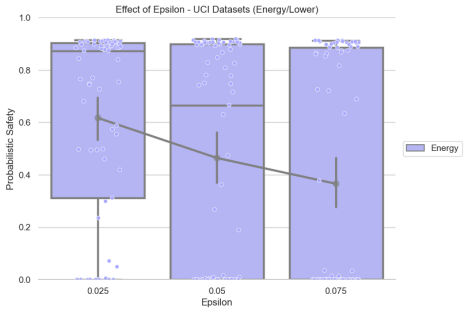
difficult to know why these algorithms fail in this case. It does present an interesting case study for future works as our algorithm is clearly not able to handle this case.

In Table 6.2, we list the empirical estimate of the statistical and probabilistic estimates for probabilistic safety arrived at by the presented algorithms. Each row is labelled and color coordinated according to the estimated value and its plotted color from previous sections. This table sheds further light on the results presented in Figure 6.4. In particular, we see that for some datasets the upper and lower bounds are very tight, which gives us useful information about the posterior robustness. For example, we can see that, on average, test set inputs from the Yacht dataset our posterior distribution contains networks which with at least probability 0.718 can be verified to satisfy the given property. This gives us a strong reason to rely on the posterior distribution learned for the Yacht dataset. For the Concrete dataset, on the other hand, we prove that on average, less than half of the networks in the posterior satisfy the property of interest. If the property considered is of great interest then from this we could conclude that the given posterior is not fit for deployment in settings where we need to understand the compressive strength of concrete. Another benefit of Table 6.2 is it allows us to compare our bounds to the empirical estimates we arrived at. This allows us to understand when the upper bound and lower bounds are tight. We see that for the Yacht and Energy datasets the lower bounding algorithm, Algorithm 3 is tight, whereas for the Concrete and Boston datasets Algorithm 4 is tight. Unfortunately, we see that the Kin8nm represents a case in which neither algorithm presents a particularly tight bound and this could be up to a poor selection of the algorithm parameters. For the Naval dataset, however, we see that Algorithm 4 has failed and that Algorithm 3 actually has reasonable performance given that no sampled parameter (even without a weight margin) was able to be proven to satisfy the property of interest. In future works, the Naval dataset will prove a useful resource to develop tighter upper bounding algorithms for probabilistic local robustness of Bayesian neural network posteriors.

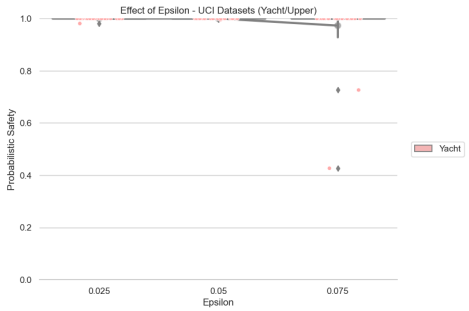
Effect of Input Specification Size We again turn our attention to the size of the input specification. As before, we expect for the probabilistic robustness to naturally decrease with an increase in the size of the input specification as probabilistic robustness is a monotonically decreasing function of the input specification size (when we consider ℓ_p norm balls). This intuition was confirmed though many experiments in the previous chapter. In this experiment, however, we seek to also understand if algorithmically our upper and lower bounds continue to perform well (i.e., return



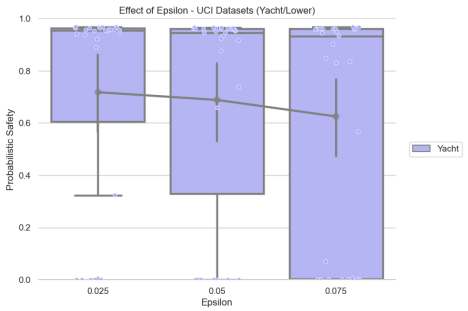
(a) Effect of input property magnitude on our upper bound for the Energy dataset.



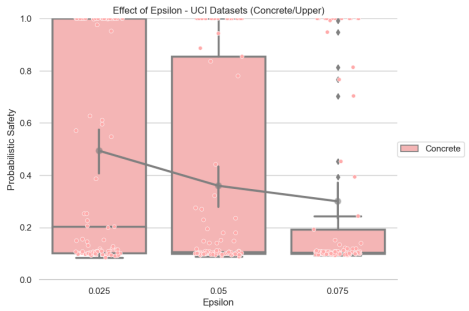
(b) Effect of input property magnitude on our lower bound for the Energy dataset.



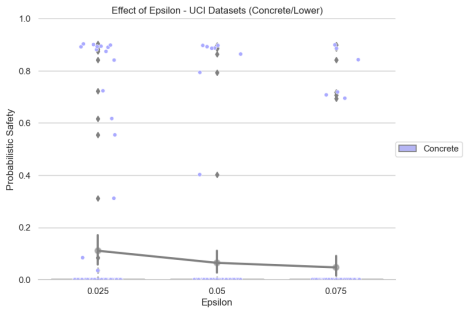
(c) Effect of input property magnitude on our upper bound for the Yacht dataset.



(d) Effect of input property magnitude on our lower bound for the Yacht dataset.



(e) Effect of input property magnitude on our upper bound for the Concrete dataset.



(f) Effect of input property magnitude on our lower bound for the Concrete dataset.

Figure 6.5: Computing probabilistic bounds on the UCI regression datasets allow us to make concrete statements about their robustness.

tight bounds) even with larger input properties or if the bounds will rapidly become vacuous. In particular, this could be the case if the approximation from our bound propagation explodes with the size of the input ball considered. To test this hypothesis we selected three values of ϵ (0.025, 0.05, 0.075) and re-run the procedure for estimating probabilistic safety used in Figure 5.3a, albeit with only 100 input points. Using different levels of adversarial input noise in this case (where we are measuring a Lipshitz-like property) we can gain a higher-level understanding of the smoothness properties of each of the learned posteriors.

In Figure 6.5, we further explore the Energy, Yacht, and Concrete datasets. These were selected as these are tests where the upper bound or lower bound performed well in our previous analysis with fixed epsilon. Given that probabilistic safety is a monotonically decreasing function of ϵ for ℓ_p norm balls, we expect that the lower bound will decrease and the upper bound will decrease as well. For the upper bound, we notice a strong decreasing trend in the case of both the Energy and Concrete datasets (Figures 6.5a and 6.5e). For the Yacht dataset (Figure 6.5c) we highlight that there is only a slight decrease in the upper bound for $\epsilon = 0.075$ and that is because we were able to compute upper bounds for only two of the 100 tested inputs. For the Lower bound on the Yacht dataset, Figure 6.5d, we also notice a less pronounced decrease in the probabilistic robustness lower bound (when compared, for example, to the lower bound on the Energy dataset). This leads us to believe that the posterior is relatively robust to the levels of ϵ that are tested here. On the other hand, for the Concrete dataset, we notice that the upper bounds became tight (average upper bound around 0.2) and the lower bounds became vacuous (average lower bound around 0.04) from this we can distinctly conclude that the Bayesian neural network trained on the Concrete dataset is provably not robust to the specification of interest.

The final observation that we make about these tests is that, with the exception of the Concrete dataset, the gap between upper and lower bound became slightly wider as we increased ϵ. This provides us with some preliminary evidence that the larger the input radius, the greater the approximation from the bound propagation method.

6.4.4 MNIST

Previously studied datasets have had up to 40 input dimensions. In this section, we again study MNIST which is almost 20 times larger. With 784 input dimensions, the MNIST input space was originally a challenge for verification of deterministic neural network methods (e.g., Reluplex [79]). In testing our approach, here we focus on lower bounds from Algorithm 3. We hope to see that our method is able to realize

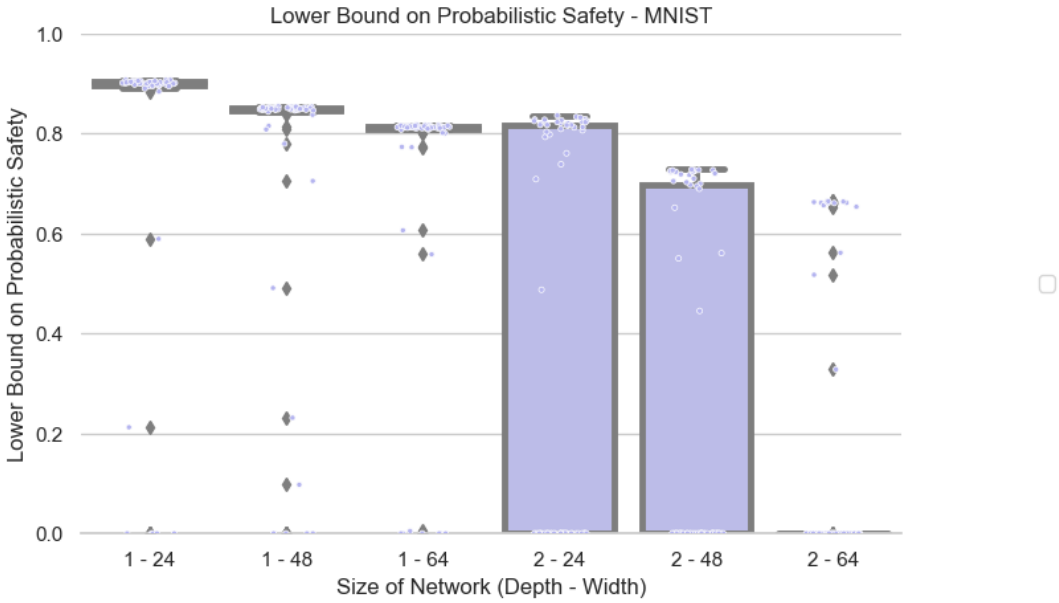
tight bounds on probabilistic model robustness even for image classification. In order to do so, we re-study the same Bayesian neural network posteriors that were used in the previous section. For experimental setting details, please refer back to the MNIST section of Chapter 5. We only perform the analysis below.

Arch. (Depth - Width)	1 - 24	1 - 48	1 - 64	2 - 24	2 - 48	2 - 64
$\hat{\mu_{\boldsymbol{Z}}}$	0.879	0.912	0.897	0.683	0.698	0.597
$\underline{\mu_{\boldsymbol{Z}}}$	0.792	0.749	0.712	0.365	0.243	0.126
Gap ($\hat{\mu_{\boldsymbol{Z}}} - \underline{\mu_{\boldsymbol{Z}}}$)	0.087	0.163	0.185	0.318	0.455	0.471

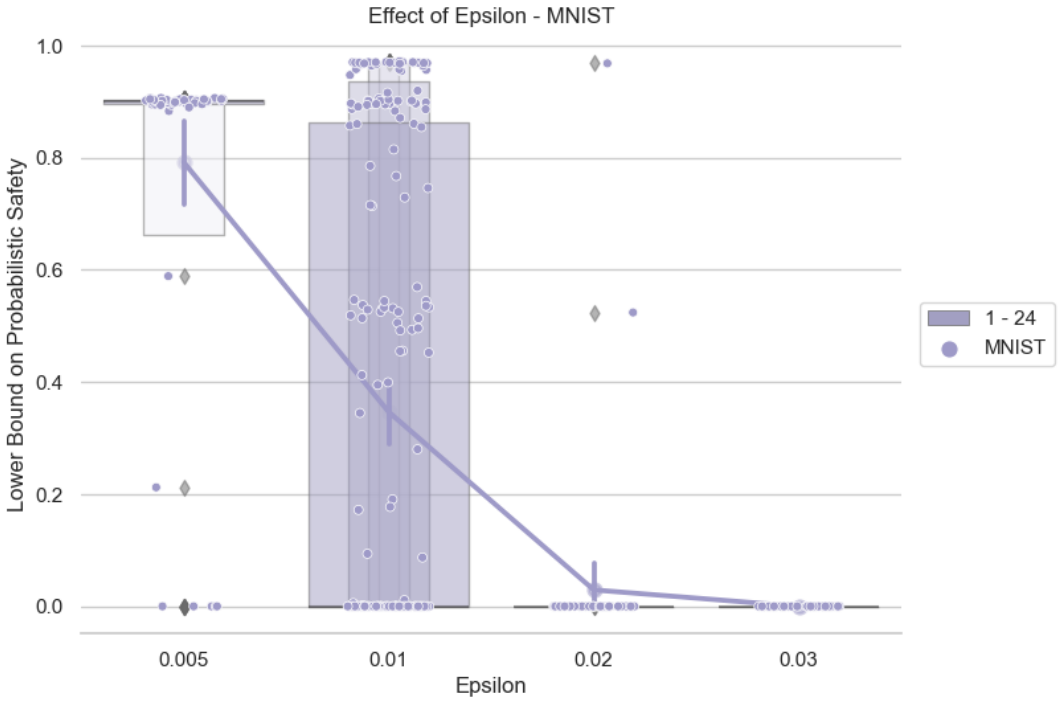
Table 6.3: Where we take $\mu_{\boldsymbol{Z}}$ to be the probabilistic model robustness, this table represents the average empirical estimate (top row), and lower bounds (bottom row) for a few hundred test set samples from the MNIST dataset. Further visualization of the bounds can be found in Figure 5.5 for the middle row and Figure 6.6a for the top and bottom rows.

Architecture Complexity We take T to be an ℓ_∞ ball centered at the test point of interest with radius 0.005. We take S to be the set of softmax vectors such that the true class (i.e., we ensure that the classification is correct and robust). We begin our analysis of Algorithm 3 on MNIST by considering how our bounds are affected by the architecture complexity. Given that the MNIST dataset has roughly 40 times more dimensions than the datasets previously examined in this chapter, it is natural to attempt to push the limits of our method in terms of scalability. We evaluate our methodology on the same six architectures that were studied in Chapter 5, starting from a one hidden layer 24 hidden units neural network. For this network we find that almost all of the tested inputs are robust with probability > 0.9. Owing to our observations in Chapter 5 that an increased input radius leads to a more approximate output set even when we were not considering a weight interval, for this algorithm, we hypothesize that the effect will be even more pronounced.

As we increase the width of the neural network (keeping all else equal, including the parameters of the algorithm) we see a slight decrease in the estimated lower bound. For neural networks with only one layer, we highlight that most inputs generally have lower bounds greater than 0.9 (see Figure 6.6). When we add a hidden layer, however, we see a stark drop off in the estimated robustness probability. This indicates a much



(a) We compute sound lower bounds on the probabilistic safety of MNIST posteriors for different architecture sizes. We find that as the size increases, our bound becomes worse with a steep drop-off resulting from adding a hidden layer.



(b) We study the effect of increasing the size of the input specification. As expected, we find that the bound quickly decreases with an increased input specification size.

Figure 6.6: We analyze how our probabilistic bounds scale to a high-dimensional image dataset.

stronger approximation for networks which are deep compared to those which are wide, and is exactly in line with what we observed in Chapter 5.

We continue our analysis of the phenomena in Table 6.3, where we report the mean of the statistical estimate from Chapter 5, the average verified lower bound from this Algorithm 3, as well the gap between these two. We notice that, for the smallest network architecture, the bound is relatively tight with a gap of less than 10%. However, when we add a layer to the network the increases up to more than 30%. This indicates that the approximation involved in increasing ϵ is indeed more pronounced for Algorithm 3 than it is for Algorithm 1.

Effect of Input Specification Size Similarly to what has been explored in previous chapters and for previous datasets in this section, we again study the effect of increasing the size of the input specification under consideration. Given the high dimensionality of the input space, we hypothesize that there should be a steeper drop off in the computed probability than what we have previously seen for other datasets. Indeed, in Figure 6.6b we see that despite having nearly perfect, verifiable robustness for $\epsilon = 0.005$, our bounds are completely vacuous by the time $\epsilon = 0.03$. Thus, Figure 6.6b indicates that while we are able to certify larger dimensional problems, our method may not be suitable for deep, large scale image classifiers. In the Chapter 7, we will consider incorporating the verifiability of the posterior into the inference problem. This allows for much tighter bounds on posteriors for larger scale problems.

6.5 Summary

In this chapter, we introduce methodologies to compute probabilistic guarantees on local robustness properties of Bayesian neural networks. We first establish that a probabilistic guarantees allow us to make statements such as "*We verify that the probability that the given Bayesian neural network is robust to any attacker is strictly between 0.75 and 0.90.*" The key contribution of this chapter is that such statements enable practitioners to employ Bayesian neural networks in safety-critical contexts with provable guarantees of correct performance. The desire to deploy networks in these contexts comes from the inherent usefulness of calibrated uncertainty in safety-critical domains, e.g., knowing if a self-driving car is certain about its prediction that an image contains a red light.

Theoretically, we build up notions of maximal safe and unsafe weights in order to demonstrate that one can convert the exact computation of model and decision

robustness to an integral in the parameter space of our Bayesian neural network. The computation of this integral is hindered by size and non-linear nature of Bayesian neural networks. Thus, we provide a refinement which allows us to use non-maximal safe and unsafe weight sets in order to bound the probabilistic robustness of Bayesian neural networks from above and below. Next, we showed how, via sampling weight space intervals and performing a specialized convex relaxation, we are able to compute non-maximal safe and unsafe sets in practice. Having access to these weight sets, we briefly discuss how to compute the cumulative density or mass of these weight sets which in turn leads to our sound probabilistic bounds on the property of interest. Assembling each of these components (the Bayesian adaptation of IBP and estimation of the cumulative probability density), we arrive at Algorithm 3 and Algorithm 4. These provide a full outline of our our methodology works in order to verify the probabilistic robustness of Bayesian neural networks.

To study the effectiveness of our methodology we provide empirical analysis in several different domains. We start, as before, by providing some empirical analysis in a contrived setting (i.e., revisiting our running examples). Using this example, we provide an intuition into the workings of our methodology, specifically the parameters γ (the weight margin) and N (the number of samples). We verify this intuition in a polynomial regression benchmark taken from [67]. In our analysis of the polynomial regression benchmark, we break down the effect of each algorithm parameters as well as the effect of the specifications parameters. Having gained an appreciation for how the probabilistic verification of Bayesian neural networks proceeds, we then analyze an airborne collision avoidance system. Being a common benchmark application for verification of deterministic neural networks, we show that we are able to verify a Bayesian neural network in this setting in order to demonstrate how our method concretely advances Bayesian neural networks in a way that enables their deployment in safety-critical scenarios. Next, we revisited the Bayesian neural network posteriors that were trained on six different regression benchmarks. We compute both upper and lower bounds on their probabilistic safety and compare them to the statistical estimates which were derived in the previous chapter. Here, we find that our algorithm is particularly tight for some posteriors, but we also found that our upper bounding algorithm failed on the Naval dataset. The final experimental setting of this chapter was revisiting the MNIST digit recognition benchmark. Again, we use the same posteriors trained in the previous section and found that for small networks our lower bound on probabilistic robustness is remarkably tight. Further studies on larger architectures and ℓ_∞ balls with larger radii indicated that the use of highly

approximate convex relaxations may be a significant bottleneck for our methodology to scale to larger dimensions.

In the next chapter, we move from studying the quantification of robustness properties of Bayesian neural networks to the incorporation of robustness properties at inference time. In particular, we study extensions of the likelihood term in Bayes theorem to appropriately capture known robustness and symmetry properties inherent in the data which may be insufficiently outlined by the training data.

Chapter 7

Adversarially Robust Bayesian Inference for Neural Networks

Contents

In Chapters 4, 5, and 6 we considered *a posteriori* verification of Bayesian neural networks with statistical and probabilistic guarantees. In this chapter, rather than quantifying robustness of a learned (equivalently, inferred) posterior distribution, we will consider Bayesian learning where adversarial robustness of the resulting posterior distribution is a primary desideratum. To this end, we provide a perspective and methodology for incorporating adversarial robustness into the Bayesian learning

framework. We begin by considering the different modelling decisions that are made when employing Bayesian deep learning and briefly consider how each decision might be changed in order to incorporate adversarial robustness. Ultimately, we establish that modification of the likelihood can allow for principled incorporation of adversarial robustness. We conclude the methodological contribution of the chapter by deriving the error models (and subsequently the likelihoods) for robustness specifications, which in principle can allow for heightened local robustness. The remainder of the chapter is left for discussion and comparison of our methodology with those that have been developed for deterministic adversarial training as well as an comprehensive set of experiments which elucidate the method's effectiveness.

7.1 On Robust Bayesian Learning

We have established that we can quantify probabilistic and decision robustness for local robustness properties of Bayesian neural networks. In our evaluation, we took these quantities to be fixed values for a given dataset and posterior. Here, we will take a look at the Bayesian learning framework (as it relates to neural networks) and will pose the question of how to increase the adversarial robustness of the resulting posterior distribution. To start with, let us first enumerate some potential candidates which one may feasibly modify in order to arrive at a more adversarial robust posterior distribution. The modelling choices made in the development of a Bayesian neural network are: the choice of prior, the choice of architecture, and the choice of a likelihood. Further, the choice of approximate inference will vary based on the size and complexity of the application at hand, and as such we do not consider it below and instead explore a wide variety of possibilities experimentally.

A natural starting point for considering the incorporation of robustness into a Bayesian neural network is the prior distribution. Recall that Bayesian neural networks incorporate priors over their hidden parameters in order to induce a prior over functions. Indeed, we have used the probabilistic correspondence between the weight space and the function space throughout this thesis. In this case, we would like to choose a prior over weights, which corresponds to a function space prior with high probability mass on *smooth* or *robust* functions. Ultimately, accomplishing this is not without some difficulties. Firstly, we would like the functions to be locally adversarially robust (e.g., smooth) within a small radius of our input points, yet if the function is smooth everywhere then we will lose expressiveness and subsequently accuracy. By way of example, consider a prior which contains only functions which are constant

classifiers, e.g., assign the same class to every point in the domain. This classifier will certainly be robustly correct on all data points which are from that particular class, but is a trivially bad (equivalently, poorly performing) classifier. One might address this problem by incorporating knowledge of the data distribution into the prior in order to choose functions which are just locally robust, but not constant in areas of little data. But, even if one knew *a priori* a class of functions which were locally adversarially robust on the data manifold, it is non-trivial to convert this class of functions into a weight-space distribution for a given Bayesian neural network architecture. This is again due to the fact that there is no strong connection between properties of the weight space and properties of their induced function space. Converting a function space prior to a weight-space prior, though difficult, is still an active and important area of research for the Bayesian deep learning community, so while our current lack of knowledge in this area (in addition to the other concerns listed in this section) prevents us from using priors to bias our Bayesian models towards more robust functions, there is still hope that this may become a feasible approach in the future. We turn interested readers to [109, 156, 83] for current research in this direction.

Another key modeling choice in Bayesian deep learning, which from a theoretical standpoint is not wholly different from choice of the prior, is the choice of model architecture. In Chapter 3, we highlighted that the architecture (computation graph) and parameter setting for that architecture uniquely determines the function, i.e., the input-output mapping. Whereas the prior controls the weight setting, we now turn our attention to the architecture. It is widely regarded that the architecture choice in deep learning encodes prior information. This is because the architecture is chosen on grounds of either experience (i.e., this architecture has worked well in similar settings) or in principle (i.e., this kind of model encodes a desirable invariance/equivariance). In either case, we may be interested in selecting the *best* architecture for capturing an adversarially robust posterior distribution. Doing so would require us to have a principled way of comparing architectures. There are well established methods for doing so in the Bayesian inference literature known as Type-II inference. The distinction between the kind of inference we have been discussing throughout this thesis and Type-II inference is succinctly summarized in David MacKay's book on inference [99]:

> Two levels of inference can often be distinguished in the process of data modelling. At the first level of inference, we assume that a particular

> model is true, and we fit that model to the data, i.e., we infer what values its free parameters should plausibly take, given the data. [...] This analysis is repeated for each model. The second level of inference is the task of model comparison. Here we wish to compare the models in the light of the data, and assign some sort of preference or ranking to the alternatives.

While it is true that performing Type-II inference may allow us to identify deep learning architectures which we prefer on the basis of their emergent adversarial robustness properties, it comes with two major downsides. Firstly, performing Type-II inference requires us to systematically explore the space of Bayesian deep neural networks in order to compare their performance and robustness. This is a downside as it is can be computationally prohibitive; both approximate Bayesian inference and quantification of local robustness properties can be very costly to evaluate and doing so over the space of interesting models may be practically infeasible. Secondly, performing Type-II inference, while it may result in the discovery of a model with heightened local robustness properties, does not truly strike at the heart of the problem we wish to solve. That is, even if we arrived at such an architecture, we could still ask the question "can this model be made more robust by means of modification of the type one inference."

Finally, we consider the likelihood. Where the prior and architecture selection involve the incorporation of knowledge in parameter space or preference in the model space, respectively, the likelihood allows us to model assumptions on the level of the data. In a Bayesian inference scenario, the computation of the likelihood, $p(\mathcal{D}|\boldsymbol{\theta})$, assumes we have accurately captured the data generating distribution with the dataset, $(\mathbf{X}, \mathbf{Y})$. However, this assumption may not hold in practice. Rather, the fixed set of samples which we hope is representative of the full data distribution may not contain information about relevant inputs. For applications in which Bayesian deep learning is employed we usually have access to a very large number of data points.[1] Yet, in spite of typically having many data points, it is clear from the lack of adversarial robustness of most deep learning systems that these samples do not accurately capture noise and symmetries that we would like to tolerate at deployment time. As such, modification of the likelihood allows us to directly incorporate assumptions about the (potentially adversarial) noise that we would encounter at deployment time. If a model has high likelihood against a local robustness specification, then by definition it is a robust model.

[1] In the sparse data regime it is almost always preferable to use a Gaussian process.

Before continuing to derive our methodology, we highlight that employing a robust likelihood in no way precludes the additional application of the other two methods described. Should the development of robust priors prove fruitful, and the computational burden of type 2 inference is negligible (perhaps due to Moore's Law), then one could, in principle, use all three methods in conjunction to greatly heighten robustness.

7.2 Deriving Robust Likelihoods

In this section we present a methodology for incorporating local robustness properties into Bayesian inference for neural networks. In particular, we recall the general supervised learning scenario for Bayesian inference. We then consider the cross-entropy error model under the standard assumptions and again under the assumption that our data distribution does not appropriately account for the local robustness property of interest. With local properties in mind, we show how to properly modify the likelihood in order to arrive at a more robust posterior distribution. Finally, we repeat this analysis but for the mean squared error loss which allows this technique to be used in regression settings as well.

Recall that in the general supervised learning scenario we have a data set of feature vectors and their corresponding labels $\{(\mathbf{x}^{(j)}, \mathbf{y}^{(j)})\}_{j=0}^{n_{\text{data}}}$. Next, given a Bayesian neural network consisting of an architecture f and a prior distribution $\theta \sim p(\theta)$ we would like to approximately infer the posterior distribution by using Bayes theorem $p(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta)p(\theta)$. Our proposed modification from the likelihood is built on the assumption that this finitely many samples from the data manifold is insufficient to instill our Bayesian posterior with the desired level of adversarial robustness. This assumption has been shown to hold in settings even in which we have direct access to the data-generating distribution. In [7, 56], the authors study adversarial robustness properties of Bayesian and non-Bayesian (respectively) neural networks in a setting in which they are able to have as many data points as they desired, and yet they found that even in this setting adversarial examples still exist.

7.2.1 Probabilistic Local Robustness Properties

In this section, we define probabilistic *local* robustness properties. We will build the definition of a probabilistic property starting from the perspective of attempting to capture multiple local robustness properties at one time. When attempting to model adversarial robustness at training time (i.e., during inference) we may be interested

in maximizing our robustness to different kinds of attacks simultaneously. That is, we might be interested in being robust to both ℓ_∞ norm attacks as well as ℓ_2 norm attacks.

In previous sections, we have considered a local robustness properties to be a two tuple $\{T, S\}$ with T being an arbitrary input set and S being the set of safe output sets. This was particularly useful it gives us a universal but elementary way of stating what it means for a network to be locally robust. A limitation of this semantic definition, which did not particularly matter until this chapter, was that in practice we might want to simultaneously gain information about two different robustness properties, e.g., $T^{(1)}, T^2$ and $S^{(1)}, S^{(2)}$. A naive solution to this would be to take $T^* = T^{(1)} \cup T^{(2)}$ and $S^* = S^{(1)} \cap S^{(2)}$. While, proving robustness w.r.t. the property $\{T^*, S^*\}$ does give us any sound information about *both* components of the safety property. It will be highly conservative as if our properties are even marginally different, we will be computing the probability that for each input set, we simultaneously satisfy $S^{(1)}$ and $S^{(2)}$. In the worst case, where $S^{(1)} \cap S^{(2)} = \emptyset$ robustness will necessarily be 0 everywhere.

Another potential solution to considering the union of multiple local robustness properties is to use the aptly named union bound. This was used in Chapter 6 to get a lower bound on the satisfaction of a VCAS property. In this instance, we simply say that our robustness of $\{T^{(1)}, S^{(1)}\}$ and $\{T^{(2)}, S^{(2)}\}$ is defined as $P_{\text{safe}}(T^{(1)}, S^{(1)}, \boldsymbol{\theta}) \cdot P_{\text{safe}}(T^{(2)}, S^{(2)}, \boldsymbol{\theta})$. This side-steps the problem of the previous attempt in that the union bound is only equal to 1 if both component properties are equal to 1. However, it suffers the opposite problem: it is overly conservative for the purposes of inference. During robust inference, we may have a series of specifications $\{T^{(j)}, S^{(j)}\}_{j=0}^k$ to which we would like to be robust. Yet, if we use the union bound to combine these definitions, and even one specification has robustness 0 then we lose all robustness information completely. This loss of robustness information would make it very difficult to learn in the presence of even mildly strong adversarial robustness properties.

The proposed solution is a probabilistic local property. Again, we consider a series of input specifications, $\{T^{(j)}, S^{(j)}\}_{j=0}^k$, and we introduce a user-defined probability density or mass over different specifications. We can start by considering a discrete distribution over these properties: $\{w_j\}_{i=0}^k \; s.t. \; \sum_{j=0}^k w_j = 1$ and each $w_i \geq 0$. The vector $\mathbf{w} = \langle w_j \rangle$ is then the multinoulli mass function. The probabilistic property then has robustness is then defined as $\sum_{j=0}^k Prob_{\text{safe}}(T^{(j)}, S^{(j)}, \boldsymbol{\theta}) w_j$. Given this definition, we have the desired property that the robustness is 1 if and only if each of

the components are 1, and we do not lose information about all of the properties if one property happens to induce robustness 0. Further, this allows us to express a the relative importance of a series of local adversarial robustness specifications through the probability mass defined by the sequence of w_j's (discussed in more detail later).

The final step in realizing the usefulness of this definition is not to consider it alongside probabilistic safety, but rather fusing it with our likelihood. Combining the probability mass function from above with the general form of a likelihood we get:

$$p(\mathbf{y}|\mathbf{x},\theta) = \sum_{j=0}^{n} w_j \min_{\mathbf{x}' \in T^{(j)}} p(\mathbf{y}|\mathbf{x}',\theta) \tag{7.1}$$

This likelihood reflects the worst-case probability of observing the true data ($\mathbf{y}$) given *all* of our local properties. Moreover, we need not have a discrete number of properties. For example we may care about an infinite number of ℓ_∞ balls: $\mathcal{B}_{\epsilon,\infty}(\mathbf{x})$ for $\epsilon \in [0, 0.1]$. In this case, we can introduce a probability density over the values of ϵ, $p(\epsilon)$ in which case the above likelihood naturally generalizes to an integration problem:

$$p(\mathbf{y}|\mathbf{x},\theta) = \int_{\mathbb{R}_{\geq 0}} \min_{\mathbf{x}' \in T_\epsilon} p(\mathbf{y}|\mathbf{x}',\theta) p(\epsilon) d\epsilon \tag{7.2}$$

In both of these formulations, we allow the modeller to use the language of probability to express an importance of each of a potentially infinite number of local robustness properties. By maximizing this worst-case likelihood (or, more tractably the log-likelihood) we are in effect maximizing the likelihood of being robust to the given specifications.

7.2.2 Adjusted Error Models and Likelihoods

Here, we consider explicit forms of $p(\mathbf{y}|\mathbf{x},\theta)$ and then discuss how to properly modify the error model in order to incorporate the distribution over worst-case local robustness properties discussed above. We do this explicitly for two of the most common error models: the categorical cross-entropy error model for classification tasks and the mean squared error model for regression tasks.

Classification Error Model We define $f_{\min}^{\theta,\epsilon}(\mathbf{x})$ to be the vector of logits corresponding to the minimizer of the softmax of the true class $\mathbf{y}$ (the $\mathbf{y}^{th}$ dimension of the softmax being denoted $\sigma_{\mathbf{y}}$) for any input point in an $\epsilon-$ball around $\mathbf{x}$:

$$\sigma_{\mathbf{y}}(f_{\min}^{\theta,\epsilon}(\mathbf{x})) = \min_{\mathbf{x}' \in T} \sigma_{\mathbf{y}}(f^{\theta}(\mathbf{x}')), \tag{7.3}$$

where $\sigma_{\mathbf{y}}(\cdot)$ is the softmax value relative to the true or expected class $\mathbf{y}$. This particular quantity is what is required for the integration problem above. Now that we have assumed access to the minimizer, $f_{\min}^{\theta,\epsilon}(\mathbf{x})$, we can can straightforwardly plug this into the integration above. By marginalizing over p_ϵ, we obtain the following likelihood function, which we call the *robust likelihood*:

$$\begin{aligned} p(\mathbf{y}|\theta,\mathbf{x}) = &\int_{\mathbb{R}_{\geq 0}} \sigma_{\mathbf{y}}(f_{\min}^{\theta,\epsilon}(\mathbf{x}))p_\epsilon(\epsilon)d\epsilon \\ =&\mathbb{E}_{\epsilon\sim p_\epsilon}[\sigma_{\mathbf{y}}(f_{\min}^{\theta,\epsilon}(\mathbf{x}))]. \end{aligned} \tag{7.4}$$

In this particular case, we have stated the robust likelihood as the worst-case of the standard-likelihood for points in an ϵ-ball around $\mathbf{x}$ for each ϵ (note that, as before, $\sigma_{\mathbf{y}}(f_{\min}^{\theta,\epsilon}(\mathbf{x}))$ is a monotonically decreasing function of ϵ). We then compute the average with respect to p_ϵ. We should stress that Eqn. (7.4) defines a marginal probability, and hence is a well defined probability. Furthermore, we also note that, for $p_\epsilon = \delta_0$, the delta function centered in the origin, we recover the standard cross-entropy likelihood model.

In order to clearly demonstrate the implications of our robust likelihood, and to cleanly write out its corresponding error model, we can assume a very simple mass function for p_ϵ. For $0 \leq \lambda \leq 1$ and $\eta > 0$, we consider the following discrete distribution for ϵ:

$$p_\epsilon(\epsilon) = \begin{cases} \lambda & \text{if } \epsilon = 0 \\ 1-\lambda & \text{if } \epsilon = \eta \end{cases}. \tag{7.5}$$

The intuitive reason behind choosing such a p_ϵ density is that this allows the user to simply select a relative weighting of the clean data (when $\epsilon = 0$) and the adversarial data $\epsilon = \eta$. This is a very common scenario in adversarial training of deterministic neural networks [58, 90]. This density leads to the following simplified form of the robust likelihood:

$$p(\mathbf{y}|\theta,\mathbf{x}) = \lambda \cdot \sigma_{\mathbf{y}}(f^\theta(\mathbf{x})) + (1-\lambda)\cdot\sigma_{\mathbf{y}}(f_{\min}^{\theta,\eta}(\mathbf{x})),$$

which is simply a weighted sum of two softmax functions, one given by the standard likelihood and the other accounting for adversarial robustness.

By assuming the statistical independence of the training labels given input and weights (which is the standard assumption for classification [11]), we obtain the following negative log-likelihood error for our model:

$$E = -\sum_{i=1}^{n_{\mathcal{D}}} log\big(\mathbb{E}_{\epsilon\sim p_\epsilon}[\sigma_{\mathbf{y}^{(i)}}(f_{\min}^{\theta,\epsilon}(\mathbf{x}^{(i)}))]\big). \tag{7.6}$$

and in the case of the simple density from above this turns into:

$$E = -\sum_{i=1}^{n_{\mathcal{D}}} log\big(\lambda \cdot \sigma_{\mathbf{y}}(f^{\theta}(\mathbf{x})) + (1-\lambda)\cdot \sigma_{\mathbf{y}}(f^{\theta,\eta}_{\min}(\mathbf{x}))\big). \tag{7.7}$$

Notice that both of these have a trivial absolute minimum when $\mathbb{E}_{\epsilon\sim p_\epsilon}[\sigma_{\mathbf{y}^i}(f^{\theta,\epsilon}_{\min}(\mathbf{x}^{(i)}))] = 1$ for all $(\mathbf{x}^i, \mathbf{y}^i) \in \mathcal{D}$. Hence, the absolute minimum of the negative log-likelihood (which would correspond to the maximum likelihood estimation) is reached for the set of weights θ^*, if it exists, such that for any $(\mathbf{x}^i, \mathbf{y}^i) \in \mathcal{D}$ almost surely $f^{\theta^*}(\mathbf{x}^i)$ has no adversarial examples in an $\epsilon-$ball around $\mathbf{x}^i$, for any ϵ in the support of p_ϵ.

We highlight that computation of Equation (7.6) can be computed easily by assuming the minimizer comes from either an adversarial example or from convex relaxation. Consequences of these assumptions (i.e., how we choose to do the minimization) are noted in the next section.

Regression Error Model The above analysis concerns a classification framework. For a regression problem everything follows similarly except that the likelihood is a Gaussian distribution with variance Σ [11]. In particular, assuming for simplicity and without lost of generality that $n_{\text{out}} = 1$ – i.e., single output regression – call $f^{\theta,\epsilon}_{\max}(\mathbf{x}) = \max_{\mathbf{x}'\in T} f^{\theta}(\mathbf{x}')$ and $f^{\theta,\epsilon}_{\min}(\mathbf{x}) = \min_{\mathbf{x}'\in T} f^{\theta}(\mathbf{x}')$. That is, $f^{\theta,\epsilon}_{\max}(\mathbf{x})$ and $f^{\theta,\epsilon}_{\min}(\mathbf{x})$ are the maximum and minimum of f^{θ} for all the points in an $\epsilon-$ball centered around $\mathbf{x}$. Then, in the regression case, for $\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}$ the robust likelihood is:

$$p(\mathbf{y}|\mathbf{x},\theta) = \frac{1}{\sqrt{2\pi\Sigma}} exp\big(-\frac{1}{2\Sigma}\max\{(\mathbb{E}_{\epsilon\sim p_\epsilon}[f^{\theta,\epsilon}_{\max}(\mathbf{x})] - \mathbf{y})^2, (\mathbb{E}_{\epsilon\sim p_\epsilon}[f^{\theta,\epsilon}_{\min}(x)] - \mathbf{y})^2\}\big).$$

Of course, given the independence of Σ from the data itself the error model we would like to minimize is simply the *robust mean squarred error*:

$$E = \big(-\max\{(\mathbb{E}_{\epsilon\sim p_\epsilon}[f^{\theta,\epsilon}_{\max}(\mathbf{x})] - \mathbf{y})^2, (\mathbb{E}_{\epsilon\sim p_\epsilon}[f^{\theta,\epsilon}_{\min}(\mathbf{x})] - \mathbf{y})^2\}\big).$$

Here we highlight that we have modified the mean of the likelihood rather than the variance. Heavy-tailed likelihoods or employing a larger variance will intuitively not have the same affect given the pervasive lack of robustness for neural networks. That is, given an input and a neural network it is empirically the case that we can induce *any* prediction we would like. Hence, we can make the mean of an undefended and poorly-trained network any arbitrary value. Thus, even with extremely heavy tails an adversary would be able to reduce the likelihood. If we can stabilize the mean against adversaries, however, then we have "defended" the network.

7.3 Practical Computation of Robust Likelihoods

Above, we discussed the error models corresponding to classification and regression tasks. In both cases, we rely on minimizing the likelihood assigned by the model to the true class. In this section, we discuss the practicalities of doing so with both adversarial examples as well as with bound propagation. We finish our discussion of practical computation by formally stating the algorithms for performing approximate inference with worst-case likelihoods. Throughout this section we will consider the simple adversarial density defined in Equation (7.5).

7.3.1 Adversarial Examples

In Chapter 3, we described that adversarial attack algorithms are complete but not sound. While there has been theoretical arguments made for the universality of a PGD adversary in the limit of the number of optimization steps [101], this theoretical guarantee offers no practical soundness. Despite the lack of soundness of adversarial attacks, they can be a very useful kind of noise to consider at training time. It has been empirically shown, in [27], that taking into account first order adversaries during training can lead to heightened provable robustness guarantees (which must be shown *a posteriori*). In addition to this, if one is concerned about a particular threat vector, i.e., one has prior knowledge about how an adversary might craft attacks for their particular model, then it makes sense to consider the exact adversary one expects at deployment during inference. In this case, one could even consider the optimization used to generate the attack as a parameter of the local property. For example, $T_{\epsilon,\text{PGD}}$, $T_{\epsilon,\text{FGSM}}$, where each of these properties are constrained by both a magnitude and attack method. Then, using the presented framework above one could encode in p_ϵ the relative probabilities of seeing each adversary in order to properly account for our beliefs about attackers in the posterior distribution.

We can illustrate an effect of our framework by considering the expected robust likelihood of a classifier compared to the standard likelihood when measured over our observed data. We estimate the likelihood of our data as follows:

$$p(\mathbf{y} = c|\mathbf{x}, \theta) = f^\theta_c(\mathbf{x})$$

For regression, one can simply swap the $f^\theta(\mathbf{x})_c$ term for the Gaussian likelihood, $\mathcal{N}(f^\theta(\mathbf{x}), \Sigma)$. Now, if we consider the robust likelihood (with Equation (7.5) as our $p(\epsilon)$ density) then we have:

$$p_{\text{adv}}(\mathbf{y} = c|\mathbf{x}, \theta)) = \lambda f^\theta_c(\mathbf{x}) + (1 - \lambda) f^\theta_c(\mathbf{x}^{\text{adv}})$$

Again, corresponding changes can be made to accommodate the regression likelihood. A clear observation from these two equations is the following relationship that $p_{\text{adv}}(\mathbf{y} = c|\mathbf{x}, \theta)) \leq p(\mathbf{y} = c|\mathbf{x}, \theta)$ and therefore that:

$$p_{\text{adv}}(\mathcal{D}|\theta) \leq p(\mathcal{D}|\theta)$$

Which is clear if we make the very mild assumption that $\mathbf{x}^{(j),\text{adv}}$ does increase the likelihood. While the fact that the robust likelihood is a lower bound on the standard likelihood is both standard and straightforward, it highlights two key facts. Firstly, we can reason that given θ has lower robust likelihood than standard likelihood does, there are fewer values of θ which have high likelihood and therefore it may be more difficult to learn in an adversarial setting. This has been empirically shown to be the case for deterministic neural networks [101]. Secondly, we know that if a model has high robust likelihood then it necessarily has high standard likelihood as well.

7.3.2 Bound Propagation

While developing a likelihood based on the relative probabilities of seeing different attack vectors is possible, it is usually preferable to consider the *worst-case* attack that any adversary could cause. The over-approximations offered by bound propagation (and more generally, abstract interpretation) allows us to consider all possible adversaries at once. This comes at the cost of over-approximation which can lead to very conservative negative log-likelihoods. Despite the conservative nature of bound propagation techniques, it has been shown for deterministic neural networks that training with these methods leads to state-of-the art adversarial robustness [60]. While these techniques have largely been studied for adversarial robustness there are also recent developments allowing for richer specifications to be considered (e.g., fairness).

We theoretically examine the prospect of integrating interval bound propagation (or similar convex relaxation) into our framework by considering the effect on the average likelihood. Consider the fact that even for an optimal adversary i.e., one that performs the NP-hard computation of finding the best adversary for every input point, we have the following:

$$p(\mathbf{y}|f^{\theta}_{\text{IBP}}(\mathbf{x})) < p(\mathbf{y}|f^{\theta}(\mathbf{x}^{\text{adv}}))$$

where f^{θ}_{IBP} is the forward pass through the BNN wrt the property specified. In practice, what this tells us is that, by performing inference with IBP, if we perform well, then we necessarily perform well against even an optimal adversary. In the

previous subsection, we showed that though it will be necessarily harder to achieve the same likelihood in the presence of an adversary, good performance in the presence of an adversary implies good performance on natural data. In this section we took things one step further and showed that good performance on IBP necessarily entails good performance against even the worst adversary.

7.3.3 Complete Algorithm for Robust Inference

In the previous sections, we motivated and demonstrated the ability of the likelihood to capture relevant adversarial robustness properties during Bayesian inference for neural networks. Further, we discussed the potential emergent properties of using either bound propagation or adversarial attacks in order to bound the robust likelihood. In this section, we will formally state and discuss the incorporation of this robust likelihood into two common approximate Bayesian inference algorithms. We focus on a popular variational approximation of the posterior, natural gradient variation inference, as well as perhaps the most popular sample-based algorithm, Hamiltonian Monte Carlo (HMC).

Robust Variational Inference We remark that, by changing the parameter update on line 10 with approximations to the Hessian, computing the gradient wrt μ, s, or by introducing momentum parameters, this algorithm can be converted to any of the gradient and natural gradient variational inference methods which have been proposed in recent years, including those of [61, 16, 82] and [115].

Algorithm 5 Robust Natural Grad. Variational Inference

Input: Prior Mean and Precision: $\mu_{\text{prior}}, s_{\text{prior}}$, BNN Architecture: f, Dataset: $\mathcal{D}$, Learning Rate: α, Iterations: K, Mini-Batch Size: m, ϵ and λ parameters of p_ϵ.
Output: Mean and precision of Gaussian approximate posterior.

1: $s \leftarrow s_{\text{prior}}$; $\mu \leftarrow \mu_{\text{prior}}$
2: **for** $t = 1, \ldots, K$ **do**
3: $\quad \{\mathbf{X}, \mathbf{Y}\} \leftarrow \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}_{j=0}^{m}$ {Sample Batch}
4: $\quad \theta = \mu + ((n_{\mathcal{D}} s)^{-1/2} \mathcal{N}(0, I))$
5: $\quad \mathbf{Y}_{\text{clean}} \leftarrow \sigma(f^{\theta}(\mathbf{X}))$
6: $\quad f^{\theta,L,\epsilon}(\mathbf{X}), f^{\theta,U,\epsilon}(\mathbf{X}) \leftarrow \text{IBP}(f, \theta, \mathbf{X}, \epsilon)$
7: $\quad \mathbf{Y}_{\text{worst}} \leftarrow \sigma(f_{LB}^{\theta,\epsilon}(\mathbf{x}))$ $\quad \#$ Eqn. (3.12) wrt. $f^{\theta,U,\epsilon}(\mathbf{X}), f^{\theta,L,\epsilon}(\mathbf{X})$
8: $\quad l \leftarrow -\mathbf{Y} log(\lambda \mathbf{Y}_{\text{clean}} + (1-\lambda)\mathbf{Y}_{\text{worst}})$
9: $\qquad\qquad + \mathbb{D}_{\text{KL}}(\mathcal{N}(\mu_{\text{prior}}, 1/s_{\text{prior}}) \mid \mathcal{N}(\mu, 1/s))$
10: $\quad s \leftarrow (1-\alpha)s + \alpha \nabla_w^2 l$; $\quad \mu \leftarrow \mu - \alpha s^{-1} \nabla_w l$
11: **end for**
12: return (μ, s)

Robust Hamiltonian Monte Carlo A similar modification needs to be made in the case of Hamiltonian Monte Carlo inference. When computing the potential energy function the same procedure outlined in lines 6-9 is employed.

Here, we present the modified updates for the HMC algorithm following the conventions of [114]. In particular, learning with HMC involves updating the parameters of our weight parameters according to Hamiltonian dynamics followed by a Rosenbluth-Metropolis-Hastings acceptance criteria [114]. The Hamiltonian dynamics are decomposed into two function: the potential energy $U(\theta)$ and the kinetic energy $K(\mathbf{v})$. The kinetic energy term in this system only concerns the momentum, $\mathbf{v}$, which is randomly drawn from a diagonal covariance matrix. The formulation for this is $K(\mathbf{v}) = \sum_{i=0}^{n_{\text{params}}-1} \frac{v_i^2}{2m_i}$, where $\mathbf{m}$ is the *mass* associated with each parameter in the network. Both $\mathbf{m}$ and $\mathbf{v}$ are user-selected hyperparameters. The second term of interest is the potential energy, which is a function of our parameters θ. This can be expressed, with our likelihood as: $U(\theta) = -log(p(\theta)) - log(\mathbb{E}_{p_\epsilon}[f_{\text{min}}^{\theta,\epsilon}(\mathbf{x})])$. Both of these functions, in conjunction with a numerical integrator (e.g., the leap frog integrator) and Rosenbluth-Metropolis-Hastings acceptance criteria completes the HMC algorithm for approximately sampling from the posterior distribution w.r.t. a prior and our robust likelihood.

7.4 Discussion

Having stated our general framework for robust Bayesian inference of neural network parameters, we conclude by comparing the methodology presented here to other common perspectives on adversarial training.

For deterministic neural networks, the standard adversarial training likelihood, originally proposed in [58], was presented in Equation (3.11). When formulated specifically for classification with the sparse categorical cross-entropy, this likelihood is:

$$L(\mathbf{x}, \mathbf{x}_{\text{adv}}, c) = \lambda\big(log(f_c^\theta(\mathbf{x}))\big) + (1-\lambda)\big(log(f_c^\theta(\mathbf{x}_{\text{adv}}))\big) \tag{7.8}$$

where c is the correct class and $\mathbf{x}_{\text{adv}}$ is an arbitrary adversary. The formulation above has been adopted widely in the adversarial training algorithms [91, 58, 60]. Interestingly, the formulation used in these methods is different from the sparse categorical-cross entropy formulation for our method:

$$L(\mathbf{x}, \mathbf{x}_{\text{adv}}, c) = log\big(\lambda f_c^\theta(\mathbf{x})) + (1-\lambda) f_c^\theta(\mathbf{x}_{\text{adv}}))\big) \tag{7.9}$$

The placement of the sum (the expectation in the general case) inside of the *log* is a direct consequence of starting from a probabilistically principled perspective on adversarial perturbations. It is yet to be seen how meaningful this change is in practice. In theory, we highlight that these two likelihoods are *not* proportional to one another and thus, according to the likelihood principle may contain different information which could result in rather different posteriors [8]. Interestingly, the only place in which these two perspectives overlap is that of robust optimization [101] in which $\lambda = 0$, which consistently shows the best adversarial robustness.

Before continuing, it is also worth noting that this is not the first attempt at performing adversarial training from a Bayesian perspective. In a similar development, [98] have developed a method for robust optimization of BNNs trained with Gaussian variational inference. However, this method cannot be directly extended to other approximate inference algorithms and relies on gradient-based attacks (i.e., PGD [101]) to approximate worst-case perturbations.

7.5 Experimental Evaluation

In order to study the proposed methodology, we infer approximate posteriors with a host of different inference methods. We combine each inference method with the standard likelihood, the robust likelihood using adversarial examples, and the robust likelihood using interval bound propagation. The chosen approximate inference methods include: SWAG [100], NoisyAdam (NA) [162], Variational Online Gauss Newton (VOGN) [82], Bayes by Backprop (BBB) [16], and Hamiltonian Monte Carlo (HMC) [114].

We open the chapter, in the same fashion as all other primary chapters of this thesis, with our running example. In this case, we simply show an intuitive effect of the robust likelihood on the classification posterior. Following this, we move into practical use cases. Not only do we study a variety of approximate inference methods, but we also focus on larger scale datasets than we have considered in previous chapters. We study MNIST [93], FashionMNIST [154], and CIFAR-10 [86]. As in our other chapters, we break down our analysis by dataset. However, for ease of comparison, we visualize the analysis done for each dataset side by side in the same figure. Following our analysis of the performance per dataset, we study how our analysis is affected by different parameters of our algorithm. In particular, we study the effect of choosing different densities for the probabilistic property under consideration.

In order to evaluate each dataset, we consider an ℓ_∞ norm ball centered at the test input $\mathbf{x}$. In order to measure the robustness of the posterior, we consider the *robust accuracy* of each model. The robust accuracy is simply the empirical mean of the decision robustness (as computed in Chapter 5 by Algorithm 2) measured not at one point but averaged over an entire test set of inputs. That is, given a set of test points $\{\mathbf{x}^{i,*}\}_{i=1,\ldots,m}$, we define the *robust accuracy* (denoted $\mathcal{R}_\epsilon$) for $\epsilon > 0$ as the ratio between the number of points $x^{i,*}$ for which no adversarial example exists within ϵ radius according to Definition 7, and the total number of test points, that is:

$$\mathcal{R}_\epsilon = \frac{1}{m}\sum_{j=1}^{m} \mathbb{I}\left[\forall \bar{\mathbf{x}}\ s.t.\ |\mathbf{x}^{(j),*} - \bar{\mathbf{x}}| \leq \epsilon, \arg\max_{c\in\{1,\ldots,C\}} \hat{\mathbb{E}}_{\theta\sim p(\theta|\mathcal{D})}(f^\theta(\bar{\mathbf{x}})) = c_i^*\right], \tag{7.10}$$

where $\mathbb{I}[\cdot]$ evaluates to 1 if the expression inside the brackets is true, and to zero otherwise. When this is the measured value, we will plot it as simply the *robustness* of the network the y-axes below. As we showed in Chapter 5, specifically Figure 5.4c and Figure 5.8c, we can measure this value by either approximating the worst-case with either adversarial examples, in which case we refer to it as $\mathcal{R}_\epsilon^{\text{PGD}}$ or with convex relaxation in, which case we refer to it as $\mathcal{R}_\epsilon^{\text{IBP}}$.

The other experimental analysis we perform in this chapter is an estimation maximal ϵ radius. Given an input $\mathbf{x}$ we estimate the maximum radius by taking the set of safe inputs S to be the set of softmax vectors such that the true class is maximal. We then consider maximizing the value of ϵ for our input property $\mathcal{B}_{\infty,\epsilon}$ such that the decision robustness is still 1. This can be stated formally as:

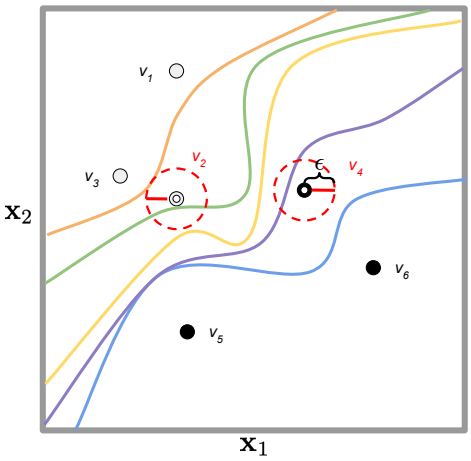
$$\max_\epsilon \epsilon D_{\text{safe}}\big(\{\mathcal{B}_{\infty,\epsilon}, S\}\big) \tag{7.11}$$

where, as before, $D_{\text{safe}}(\{\mathcal{B}_{\infty,\epsilon}, S\})$ evaluates to 1 if the decision safety holds and 0 otherwise. Thus, the function is maximized when ϵ is as large as possible without causing $D_{\text{safe}}(\{\mathcal{B}_{\infty,\epsilon}, S\}$ to evaluate to 0. In practice, this is done by performing a line search where we start with $\epsilon = 0.1$ and we halve the value if $D_{\text{safe}}(\{\mathcal{B}_{\infty,\epsilon}, S\} = 0$, and we increase by 25% if the property holds. In the limit of iterations, we compute the exact maximal ϵ, but in practice we only use 16 iterations.

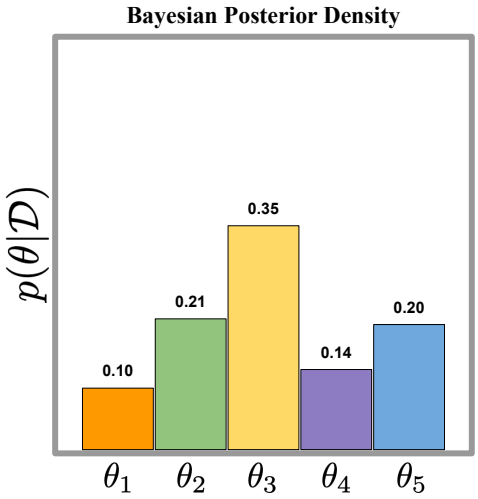
Finally, an important note is, that for HMC, we set the initial weight to be a sample from the prior when performing standard training and set the initial weight to a pre-trained SGD iterate when performing inference with robust likelihood; this is to enforce that the starting point of the algorithm is closer to the target distribution.

$$p(\mathbf{y}|\theta, \mathbf{x}) = \sigma_{\mathbf{y}}(f^{\theta}(\mathbf{x}))$$

(a) The standard classification likelihood we have used up to this chapter.



(b) Our example discrete Bayesian posterior for classifying the white and black dots. Each colored line represents a different parameter setting of a Bayesian neural network. We highlight that this would arise from the standard inference paradigm.

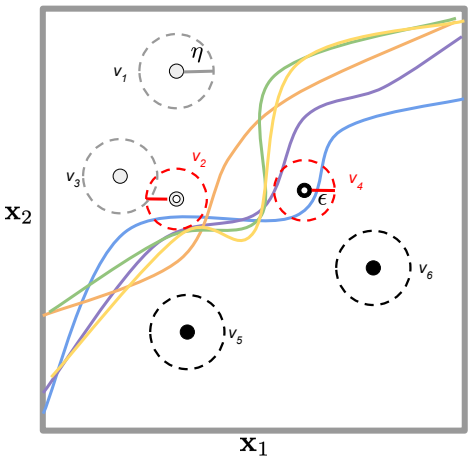


(c) The posterior mass function for the given for the posterior. We highlight that this would arise from the standard inference paradigm.

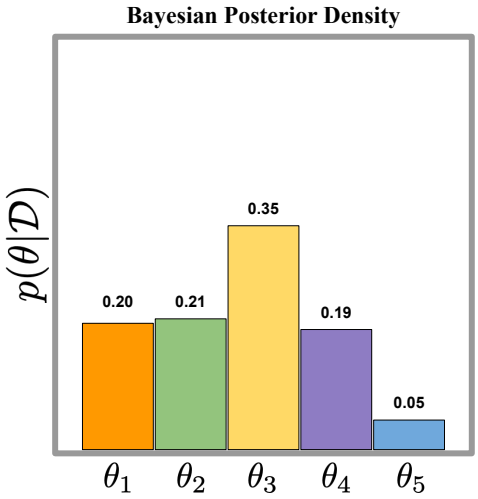
Figure 7.1: We recall the hypothetical result of the standard (i.e., non-robust) Bayesian inference paradigm on our running classification example.

$$p(\mathbf{y}|\theta, \mathbf{x}) = \sigma_{\mathbf{y}}(f^{\theta,\eta}_{\min}(\mathbf{x}))$$

(a) The robust classification likelihood where we take the p_ϵ mass function to be the δ_η function which is a Dirac distribution with mass only on the value η. This corresponds to robust optimization settings in other adversarial training procedures.



(b) A posterior visualization that takes into account not only the training examples (black and white points) but also a local robustness property which is specified as: $\mathcal{B}_{2,\eta}(v_i)$.



(c) The posterior mass function for the given for the posterior. We highlight that it is not only the shape and placement of the parameters in the posterior ensemble, but also their relative weighting that changes as a result of robust inference.

Figure 7.2: We provide an intuitive visualization of changes that our classification running example would incur as the result of application of the robust Bayesian inference paradigm.

7.5.1 Intuitive Example

In Figure 7.1 and Figure 7.2 we highlight how the use of a robust likelihood may affect the composition of a posterior distribution. In Figure 7.1, we recall the classification example that we have used throughout this thesis. In Chapters 5 and 6 we showed how we could compute the probabilistic adversarial robustness of point v_4 to be 0.45. Here, we simply recall how the posterior mass is distributed over the five parameter settings in function space (Figure 7.1b) as well as how the posterior mass is distributed (Figure 7.1c). In Figure 7.2, we no longer represent the standard classification inference problem that we have used for the past 3 chapters and instead opt for a robust version of the inference. For this, we consider a local robustness property that is defined as $\mathcal{B}_{2,\eta}(v_i)$ for each point v_i in our training dataset. These ℓ_2

norm balls are visualized as dotted lines in Figure 7.2b. In Figure 7.2a we highlight that in this setting we will consider the likelihood to be derived exclusively from the worst case likelihood in $\mathcal{B}_{2,\eta}(v_i)$. With this change in likelihood, we observe that the shape and positions of the Bayesian ensemble have changed (see the function space representation in Figure 7.2b). Further, it is not only the shape and placement of the functions that change, but also their posterior mass function could shift dramatically; this is represented in Figure 7.2c. The result of this change is ultimately that the posterior distribution is more robust, which can be seen by re-computing the robustness values for test input v_2.

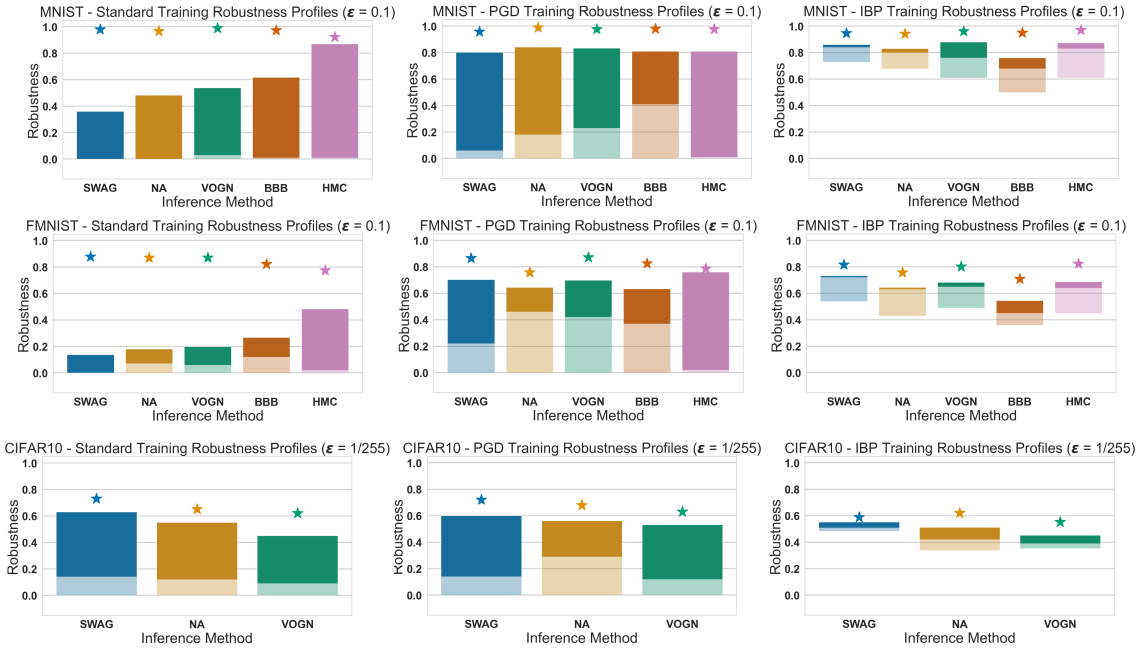


Figure 7.3: Accuracy (plotted as star points), an empirical estimation of $\mathcal{R}_\epsilon$ obtained using PGD (upper bound of each bar), $\mathcal{R}_\epsilon^{LBP}$ (lower bound of each bar), and $\mathcal{R}_\epsilon^{IBP}$ (shaded lower bound of each bar) obtained for $\epsilon = 0.1$ on the MNIST dataset (top row) and FMNIST (middle row) as well as for $\epsilon = 1/255$ on the CIFAR-10 dataset (bottom row). Each bar refers to a different approximate Bayesian inference technique. **Left Column:** results for the standard likelihood. **Centre Column:** results for approximation of robust likelihood using PGD. **Right Column:** results for training with formal IBP lower bound of robust likelihood (Eq (7.6)). With our method we obtain up to 75% certified robust accuracy on MNIST and up to 50% on CIFAR-10.

7.5.2 MNIST

In this section we investigate how our methodology improves the decision robustness of various posterior distributions trained on MNIST. In order to perform robust training on these networks we use p_ϵ introduced in Eqn (7.5) with $\eta = 0.1$ and $\lambda = 0.25$. We train a single hidden layer BNN with 512 neurons on the full MNIST dataset. In this case, we leave further training parameters to the Appendix.

In Figure 7.3 (top row) we analyze how different training methods affect the accuracy, robustness to PGD attacks ($\mathcal{R}_{0.1}^{PGD}$), as well as the certified lower bounds using IBP ($\mathcal{R}_{0.1}^{IBP}$) and linear bound propagation (LBP) ($\mathcal{R}_{0.1}^{LBP}$). Linear bound propagation is very similar to IBP save for that we are no longer propagating intervals through the network, but linear inequalities. LBP tends to be give a sharper bound than IBP. We can see this in Figure 7.3, where the IBP lower bound is the shaded bottom of the bar whereas the LBP lower bound is the solid bottom of the bar. We use both IBP as well as the more computationally expensive but tighter LBP in order to study the effect of training with our robust likelihood without the bias of training and evaluating with the same certification method. As we discussed in Chapter 2 and Chapter 3, this can result in adversarial overfitting. However, this is not typically a problem for certification methods as we are minimizing a provable lower bound. We find that, while all BNNs trained with the standard likelihood (left plot) perform comparably well in terms of accuracy, there is a marked difference in their robustness against PGD. This is in line with what was observed by [23], where the more fidelity an inference method has to the true Bayesian posterior, the greater is its robustness to gradient-based attacks. Further, we highlight that the robustness to ϵ for these posteriors is markedly higher than those studied in the previous two chapters. This is again in line with the observations of [23], who notice a strong positive correlation between the width of a network and its adversarial robustness. Despite the heightened robustness to PGD, the certified robust accuracy obtained using standard likelihood is identically zero, that is, we obtain no certification for any posterior in these settings. This implies, for example, that although HMC is resistant to PGD attacks, we cannot guarantee that a different, successful attack method does not exist.

We make similar observations for networks trained using the robust likelihood where PGD is used in order to approximate teh worst case. This can be seen in the middle of the top row of Figure 7.3. For training with the PGD approximation of the robust likelihood, we highlight that the accuracy on clean data (marked with a star for each inference method) is comaprable or in fact increased for each network. Meanwhile, the most noticeable difference the robustness against PGD of each model

is now around 80%, which represents more than 100% increase for methods such as SWAG. Despite the heightened robustness to PGD for each model, the robust accuracy when using IBP is still at 0% when measured with IBP. This underscores the phenomena of adversarial overfitting: our posterior is robust to PGD, but cannot be proven to be robust to *any* adversary. Thus, we cannot rule out the possibility that their is another more powerful attack which greatly degrades the models robustness. This observation is not unique to our study; similar behaviour has also been observed for adversarial training with gradient-based attacks for deterministic neural networks [60]. Theoretically, this happens because PGD provides an upper bound to $\mathcal{R}_\epsilon$, and the maximisation of an upper bound does not provide any guarantees on the final results on the actual quantity of interest.

By using IBP during training to lower-bound the robust likelihood (right plot in the figure), we find that, not only do we obtain similar levels of accuracy and PGD robustness as before, but we are also able to provide non-trivial certification on the robust accuracy, $\mathcal{R}_\epsilon$, of the networks, that is, against *any* possible adversarial perturbation of magnitude up to $\epsilon = 0.1$. For example, using SWAG we obtain $\mathcal{R}_{0.1} \approx 75\%$. This can directly be interpreted as the Bayesian neural network posterior inferred with SWAG and robust likelihood is provably adversarially robust wrt 75% of the points included in the MNIST dataset.

We continue our analysis by approximately computing the maximization posed in Eqn. (7.11). In Figure 7.4, we analyze the maximum certifiable radius using the 16 iteration procedure described at the start of this section. In order to check the safety of one of the local robustness properties, $\{\mathcal{B}_{\infty,\epsilon}, S\}$, we use the certification method provided in [17] which has been shown to be a tight linear bound propagation method. Interestingly, we see that, unlike what is shown in Figure 7.3, Figure 7.4 shows us a non-trivial increase in the average maximum verifiable radius for networks which are trained with PGD. This gain is relatively marginal compared to the gain achieved through our likelihood combined with IBP which is able to realize a 2 to 3 times improvement in the certifiable radius without losing considerable performance on clean data.

7.5.3 FashionMNIST

In the center row of Figure 7.3, we use the same networks, p_ϵ distribution (with $\eta = 0.1$ and $\lambda = 0.25$) and evaluation methods stated for MNIST, but applied to the FashionMNIST dataset [154]. The FashionMNIST dataset poses an image classification problem which is composed of 28 by 28 black and white images (the same

format as MNIST). It comprises 10 classes including various clothing items such as shirts, tennis shoes. It is widely held that the FashionMNIST dataset is a harder dataset than the MNIST digit classification dataset. In fact, this is why the dataset was originally proposed in [154]. The increased difficulty comes from the fact that it is intuitively harder to separate classes which are less black and white (i.e., literally contain more grey tones) and classes which encode more semantically rich images.

Despite being a harder dataset than MNIST, as evidenced by the reduced accuracy of the approximate posteriors that are shown in the middle row of Figure 7.3, we find the robustness trends to be qualitatively similar to those on MNIST and CIFAR10. We do note, however, that PGD training was much more effective at increasing the certified bound than when the bound is computed with LBP. One potential hypothesis we can make about the decrease in robustness from MNIST to FashionMNIST for posteriors trained with the standard likelihood comes from [23], which states that networks which are less accurate are also potentially less robust to gradient-based attacks. Thus, PGD is able to find strong adversarial examples which can increase the robustness of the posterior.

Finally, we again turn to the results in Figure 7.4 which computes the maximum verifiable radius for each image and then we plot the average. In this case, we notice an almost identical trend to that which is observed for MNIST. Namely, that we get a small but noticeable increase in certifiable robustness from training with our likelihood and PGD, but this pails in comparison to the impressive gains we are able to achieve with the robust likelihood in conjunction with IBP.

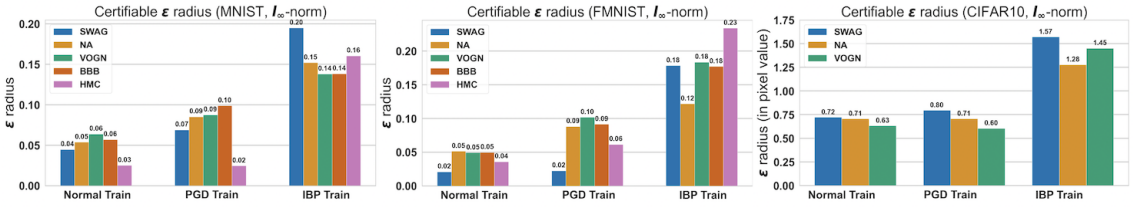


Figure 7.4: We plot the average certified radius for images from MNIST (right), FashionMNIST (middle), and CIFAR-10 (left) using the methods of [17]. We observe that robust training with IBP roughly doubles the maximum verifiable radius compared with standard training and that obtained by training on PGD adversarial examples.

7.5.4 CIFAR-10

Finally, evaluate the effect of the robust likelihood on BNNs trained on the CIFAR-10 dataset. The CIFAR-10 is much more challenging compared to MNIST and Fashion-

MNIST, and hence not all the training methods considered for MNIST can be used to train reasonably accurate BNNs on this dataset [15]. CIFAR-10 is more challenging in many ways. For one, it is 3072 dimensional as its inputs are 32 by 32 RBG images, making it roughly 4 times larger than the MNIST and FashionMNIST images. In addition to being full color images, the images are real-world images which have just been scaled down; this is perhaps one of the hardest contexts for image classification. Consequently, for CIFAR-10 we provide results only for SWAG, NA and VOGN. In particular, we train a Bayesian convolutional neural network (CNN) with 2 convolutional layers (each with 32, 4 by 4 filters) followed by a max pooling layer and 2 fully-connected layers (one with 512 hidden neurons and the other with 10). For the robust likelihood, we consider Eqn (7.5) with $\eta = 1/255$ and $\lambda = 0.25$. Finally, we introduce the standard exponential decay on the learning rate to ensure stable convergence.

We perform a similar evaluation to that discussed for MNIST and FashionMNIST, the results of which are plotted in the bottom row of Figure 7.3. Consistently with what we observed for MNIST, we obtain that BNNs trained by using the standard likelihood (left plot) and PGD attacks (central plot) do not allow for the computation of certified guarantees (the lower bound of the bars is close to zero for all the inference methods). In contrast, for the BNN trained with our robust likelihood and IBP we find that, even for CIFAR-10, we are able to compute non-trivial lower bounds on $\mathcal{R}_\epsilon$. For instance, on SWAG we obtain $\mathcal{R}_\epsilon^{IBP} \approx 50\%$, which is comparable to state-of-the-art results with adversarial training of deterministic NNs on CIFAR-10 [17].

Again, we provide further anlaysis for the effect of each inference scenario on robustness by studying the maximal verifiable radius. Results are given in Figure 7.4, where we report the average maximum adversarially safe radius over 100 test CIFAR-10 images. Unlike our observations for MNIST and FashionMNIST, we find that, PGD training does not increase the robustness in a noticable or statistically significant way compared to standard training. Robust training with IBP, on the other hand, is able to roughly double the robustness for the three training methods explored here. The one minor caviat to this result is that there is a slight decrease in the clean performance of models trained with our robust likelihood. However, it is not a significant decrease in clean accuracy.

7.5.5 Effect of Probability Density or Mass Function

In previous sections we focused on how our robust likelihood performs when paired with various approximate inference methods and with various ways of approximating

the worst-case input for a probabilistic local robustness property. In this section, we turn our attention to the other *key* aspect of our proposed methodology which was held fixed in our previous analysis and that is the probability mass or density that corresponds to the probabilistic local robustness property. In this section, we start by modifying the various parameters of the simple mass function used in our previous analysis, and also consider the performance of different density functions under various parameterizations.

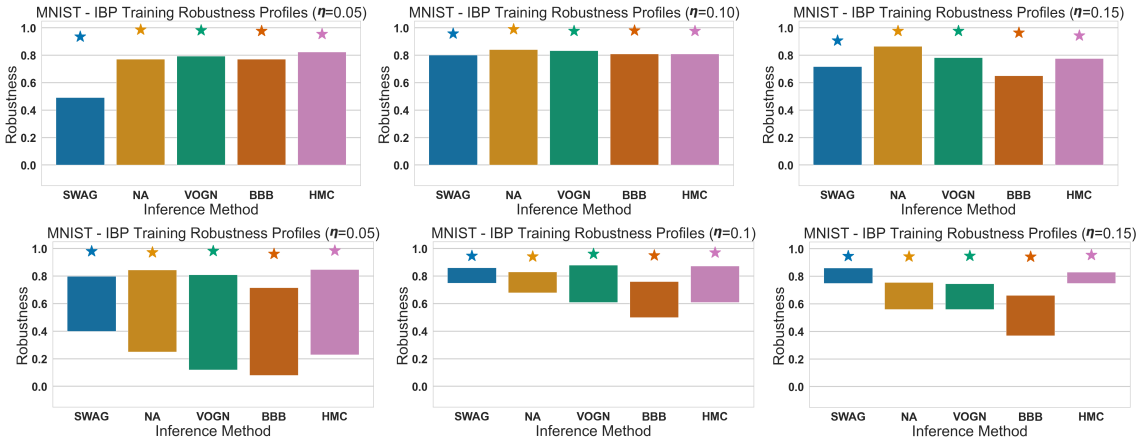


Figure 7.5: **Left to Right:** Effect of varying (increasing) values of η on the robustness profile of resulting approximate posteriors. **Top Row:** Robustness profiles of networks using the robust likelihood with PGD as an approximate worst-case adversary. **Bottom Row:** Robustness profiles of networks using the robust likelihood with IBP as an approximate worst-case adversary. Accuracy (plotted as star points), an empirical estimation of $\mathcal{R}_\epsilon$ obtained using PGD (upper bound of each bar), and $\mathcal{R}_\epsilon^{IBP}$ (lower bound of each bar), obtained for $\epsilon = 0.1$ on the MNIST dataset.

7.5.5.1 Probability Mass Functions for the Robust Likelihood

The Effect of η in Equation (7.5) In this section we analyse the choice of p_ϵ, that is, the distribution that controls the adversarial perturbation strength at inference time. Specifically, we reconsider the probability mass function given in Eqn. (7.5). We first study the affect of changing the λ parameter in Eqn (7.5) which parameterizes the relative penalty between accuracy and robustness during inference. Next, we study the effect of changing η in Eqn (7.5), which sets a the maximum allowable manipulation magnitude during inference. Finally, we study the effect of changing the form of the ϵ probability density function to two different continuous, non-negative distribution.

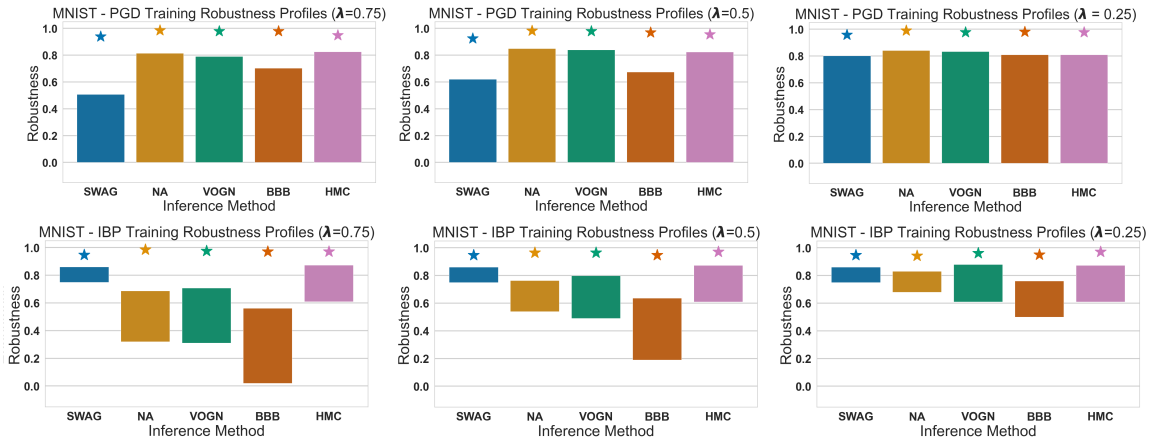


Figure 7.6: **Left to Right:** Effect of varying (decreasing) values of λ on the robustness profile of resulting approximate posteriors. **Top Row:** Robustness profiles of networks using the robust likelihood with PGD as an approximate worst-case adversary. **Bottom Row:** Robustness profiles of networks using the robust likelihood with IBP as an approximate worst-case adversary. Accuracy (plotted as star points), an empirical estimation of $\mathcal{R}_\epsilon$ obtained using PGD (upper bound of each bar), and $\mathcal{R}_\epsilon^{IBP}$ (lower bound of each bar), obtained for $\epsilon = 0.1$ on the MNIST dataset.

In each figure, we maintain the plotting conventions used in Figure 7.3 of the main text. For each posterior: accuracy is plotted as a star point, an empirical estimation of $\mathcal{R}_\epsilon$ obtained using PGD (upper bound of each bar), and $\mathcal{R}_\epsilon^{IBP}$ (lower bound of each bar), obtained for $\epsilon = 0.1$ on the MNIST dataset. For the following analysis we only report the lower bound based on IBP.

The Effect of λ in Equation (7.5) In Eqn (7.5) the parameter λ effectively controls the relative weighting of accuracy-error and robust-error during the inference procedure. Specifically, we note the cases $\lambda = 1.0$ which results in the standard likelihood (a.k.a. the categorical cross-entropy in the case of classification), and $\lambda = 0.0$ results in a framework in which give importance solely to robustness. In Figure 7.6 we report the change in robustness profiles for $\lambda \in \{0.75, 0.5, 0.25\}$ for training with the worst-case approximated by PGD (top row) and IBP (bottom row).

When approximating the robust likelihood with PGD, we find that HMC and natural gradient methods (VOGN, NA) are not strongly affected by the choice of λ, whereas we see the most pronounced difference with SWAG which is greatly affected by the choice of λ. In particular, we highlight roughly a 20% raw increase in the robustness to gradient based attacks for each 0.25 decrease in λ. On the other hand, when training with IBP there is a large shift in the resulting robustness profiles for parameter and natural gradient VI methods (BBB, VOGN, NA). Notably, we see a

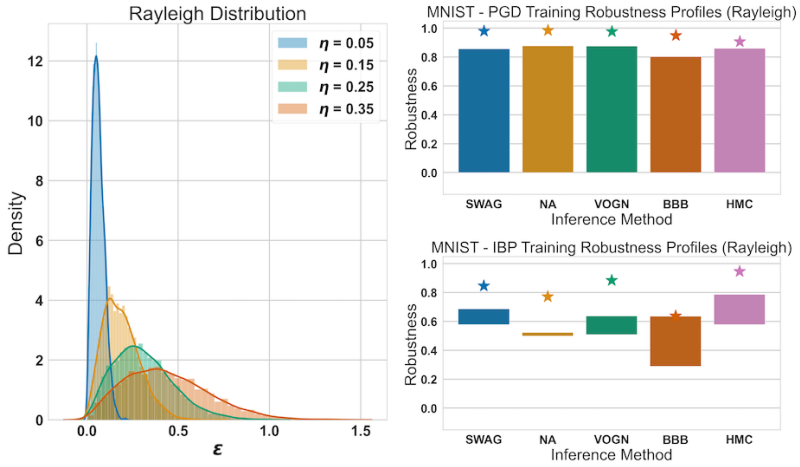


Figure 7.7: **Left:** Effect of varying the scale η of the Rayleigh distribution on the density p_ϵ when training we use $\eta = 0.1$. **Right, Top Row:** Robustness profiles of networks using the robust likelihood with PGD as an approximate worst-case adversary. **Right, Bottom Row:** Robustness profiles of networks using the robust likelihood with IBP as an approximate worst-case adversary. Accuracy (plotted as star points), an empirical estimation of $\mathcal{R}_\epsilon$ obtained using PGD (upper bound of each bar), and $\mathcal{R}_\epsilon^{IBP}$ (lower bound of each bar), obtained for $\epsilon = 0.1$ on the MNIST dataset.

large (50% raw) increase in the lower bound for BBB as the value for λ varies between 0.75 and 0.25.

7.5.5.2 Probability Density Functions for the Robust Likelihood

In Figure 7.7 and Figure 7.8, we study changing the form of p_ϵ from the density given in Eqn (7.5) to a Rayleigh distribution and an exponential distribution, respectively. We have chosen these distributions in particular because they have non-negative support and a single controlling variable. In principle, however, any distribution (with a positive support) can be chosen for the form of p_ϵ. As noted in the main text, during the computation of the loss function, one must marginalize over the selected p_ϵ distribution, which in this case is done via Monte Carlo with only 10 samples from p_ϵ per batch. Consistent with the study presented in Figure 7.3, we evaluate robustness profiles with ϵ set to 0.1.

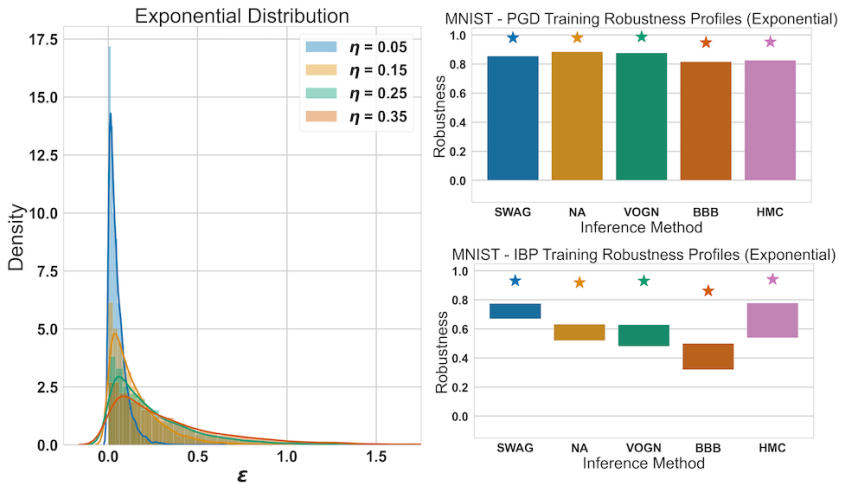


Figure 7.8: **Left:** Effect of varying the scale η of the exponential distribution on the density p_ϵ when training we use $\eta = 0.1$. **Right, Top Row:** Robustness profiles of networks using the robust likelihood with PGD as an approximate worst-case adversary. **Right, Bottom Row:** Robustness profiles of networks using the robust likelihood with IBP as an approximate worst-case adversary. Accuracy (plotted as star points), an empirical estimation of $\mathcal{R}_\epsilon$ obtained using PGD (upper bound of each bar), and $\mathcal{R}_\epsilon^{IBP}$ (lower bound of each bar), obtained for $\epsilon = 0.1$ on the MNIST dataset.

7.5.5.3 Using a Rayleigh Distribution

In Figure 7.7, we plot the case in which training is done by using an Rayleigh distribution with the scale set to η for p_ϵ as follows:

$$p_\epsilon(\epsilon) = \frac{\epsilon}{\eta^2} \exp\left(\frac{-\epsilon^2}{2\eta^2}\right) \tag{7.12}$$

In our experiments, we find that using a Rayleigh distribution for p_ϵ only marginally improves the robustness ($\mathcal{R}_\epsilon$) when training against a PGD adversary ($\approx 4\%$ on average). We highlight the continued observation that training with robust likelihood in conjunction with IBP is the only method that gives non-trivial lower bounds on robustness. However, we find that the use of the Rayleigh distribution has an adverse affect on the overall robustness profile compared to training with Equation (7.5). This is likely due to the fact that, when using the probability mass function from Equation (7.5), we have a considerable amount of mass on the adversarial portion (when $\epsilon = 0.1$) of the likelihood with mass $= 0.85$. In contrast, the cumulative density of the Rayleigh distribution which is as strong or stronger than $\epsilon = 0.1$ in Equation (7.5) is much lower. Thus, we expect that at inference time the Rayleigh

trained network saw less challenging adversaries and thus exhibits lower test-time robustness.

7.5.5.4 Using an Exponential Distribution

In Figure 7.8, we give the results when p_ϵ is selected as an exponential distribution with the rate set to η^{-1}:

$$p_\epsilon(\epsilon) = \frac{1}{\eta}\exp\left(\frac{-\epsilon}{\eta}\right) \tag{7.13}$$

When training against a PGD adversary, we found that using an exponential distribution for p_ϵ also leads to small increases in robustness against adversarial attacks, with an average increase of $\approx 5\%$. Consistent with the results for the Rayleigh distribution and those of the probability mass function in Equation (7.5), we again highlight that training with the robust likelihood and IBP is the only method that gives non-trivial lower bounds on robustness. In fact, we continue to find that the use of the exponential distribution, has an adverse affect on the overall robustness profile compared to training with Equation (7.5). Our hypothesis stated for the Rayleigh distribution, that it on average saw weaker adversaries at inference time, remains a plausible hypothesis in this case as well. Given that this observation is made for both of these continuous densities compared to the mass function, we conjecture that if one has a particular ℓ_∞ radius that they would like their Bayesian posterior to be robust to, then it is best to use either Equation (7.5) with $\eta = \epsilon$ or a continuous density with parameter slightly larger than the target in order to ensure that the cumulative probability of seeing an adversary at the target strength or stronger is sufficiently high.

7.6 Summary

In this chapter, we started by considering different components of the Bayesian inference framework which may be good candidates for inclusion of local robustness properties. After discussing the prior, architecture, and likelihood, we establish that the likelihood provides a principled avenue for the incorporation of adversarial noise into the Bayesian inference framework due to its role modelling the data distribution.

Methodologically, we began by modifying our notion of local robustness in order to appropriately combine multiple local robustness properties. By introducing a probability density or mass over the properties of interest, we establish that an expectation

over this probability allows us to combine multiple, potentially diverse local robustness criteria without losing soundness and without losing too much information in the case that one of the properties is too difficult to satisfy. From our development of probabilistic properties, we establish how we can incorporate them into the Bayesian inference framework. In order to do so, we re-derive the likelihoods and error models for classification and regression. Finally, we provide a discussion of the insights that the resulting likelihoods offers relative to the standard notions of adversarial training commonly found in the literature.

One key advantage of our proposed methodology is its ability to naturally incorporate into any approximate inference method. In order to study the emergent adversarial robustness properties of our method, we take five different, commonly used approximate inference methods and study them under different forms of our likelihood. In particular, we train different architectures on MNIST, FashionMNIST, and CIFAR10. For each dataset, we also perform approximate inference using the standard likelihood, as well as the robust likelihood whose value is approximated by either PGD or IBP. For every trained model we perform two kinds of robustness analysis. Firstly, we study the robust accuracy which we define as the proportion of test set inputs for which the Bayesian neural network maintains both accurate and robust decisions (according to Definition 7). Secondly, we study the effect of our robust likelihood on the average maximum safe radius. This is defined as the largest value of ϵ such that the Bayesian decision remains correct inside of an $\mathcal{B}_{\epsilon,\infty}$ ball. In each case, our analysis yielded consistent and conclusive evidence. We found that robust training with PGD improves the robustness of the posterior to attacks and in some cases improves the verifiable accuracy. Further, we found that training with IBP greatly improves both attack robustness and verifiable robustness of each tested posterior. Finally, we tested how our robust likelihood performs under different choices of p_ϵ. We first studied the effect of changing parameter values for the simple probability mass function defined in Eqn. (7.5), and then studied different continuous densities with positive support.

The primary conclusion we draw from this section is that, in cases in which one would like to deploy Bayesian neural networks in safety-critical domains, it is advisable to carefully consider desirable local robustness properties and to probabilistically incorporate them into the inference procedure. Doing so can result in certifiable Bayesian neural networks on tasks as large and complex as CIFAR10.

Chapter 8

Conclusion

Contents

We opened this thesis with a wide view of classical computer programming and classical verification. Classical computer programming transforms semantic relationships between inputs and outputs into algorithmic relationships which can be tractably analysed via their computational complexity, space complexity, and their correctness properties. As the field of computer science has matured we have turned our attention to problems for which the semantic input-output relationship can no longer be transformed neatly into an algorithmic relationship. Where this is the case, programs are *learned* rather than crafted by hand. Over the past decade, these learned solutions have provided astonishing solutions to complex problems such as image classification, medical diagnosis, and natural language processing. The primary downside of these approaches is their lack of interpretability. In particular, we introduced and heavily studied adversarial examples as primary evidence of this phenomenon. Where adversarial examples provide concrete proof of a learned algorithm's failures, verification offers concrete proof of a learned algorithm's correctness. The lack of interpretability of learned solutions not only manifests itself by way of adversarial examples, but also in the over-confident, wrong predictions that it makes on natural examples. By introducing calibrated uncertainty into predictions one can also address this issue. Both the problem of robustness and calibrated uncertainty are of paramount importance to the safe deployment of learned solutions in safety-critical applications.

Motivated by the development of algorithms which are both robust and uncertain, in this thesis we take a step toward verification of an important class of learned programs: Bayesian neural networks. In order to do so, we first establish novel notions of robustness which are distinct from those which have previously been studied for deterministic neural networks. We then develop algorithms for verifying the correctness of Bayesian neural networks with different levels of guarantees, and finally, we provide a principled probabilistic framework for the incorporation of local robustness properties into Bayesian inference for neural network parameters.

In the final chapter of this thesis, we provide concluding remarks regarding the strengths and weaknesses of the methodology provided in this thesis. We reflect on the major empirical take-away points from the thesis. With an eye on the weaknesses of our methodology, we use the final section to discuss future works which would strengthen the content of this thesis and which would be of further interest to those who would like to study the robustness of Bayesian neural networks.

8.1 Summary of Contributions

In this section, we revisit the contributions from Chapter 1 and provide a slightly more technical statement of contribution for each item given that formal definitions have now been established. After restating our contributions, in the final subsections, we provide a discussion of strengths and weaknesses of each contribution which draws on the empirical studies performed in each chapter.

- We show how probabilistic parameters prohibit direct translation of adversarial robustness to Bayesian neural networks. By first defining adversarial robustness properties as arbitrary input-output specifications, we show how to propagate this specification through the probabilistic parameter space to develop a notion of probabilistic robustness. This can intuitively be interpreted as the probability that a model drawn from a given (potentially approximate) Bayesian posterior is robust to a provided specification.

- Drawing on the distinction between a loss function and a likelihood in the context of Bayesian learning, we highlight the distinction between adversarial examples in a frequentist context versus a Bayesian one. Given this, we define decision robustness for Bayesian neural networks as a direct analog to the notions of robustness that are commonly used to provide assurances of correctness for deterministic neural networks. Proving the robustness of the Bayesian

decision-theoretic quantity directly translates to proving the absence of adversarial examples for the provided Bayesian posterior.

- In order to compute the established definitions of robustness, we start by highlighting a correspondence between probabilities in weight space and the emergent robustness properties of our Bayesian neural network in function-space. Building on this, we provide theoretical and practical perspectives on estimating both the probabilistic robustness and the Bayesian decision robustness with *a priori* statistical guarantees on the error and confidence. Through the use of concentration inequalities and by leveraging advancements in statistical model checking we are able to efficiently guarantee that the computed values corresponding to both definitions are statistically tight for a given Bayesian neural network and robustness specification. Empirically, we show how this can scale to computer vision applications in autonomous vehicles, and how statistical measures can be used to perform different analysis (e.g., class-wise) of the posterior.

- By again leveraging the weight-space function-space correspondence mentioned above, we establish a theory of probabilistic verification for Bayesian neural networks which frames the established definitions from previous chapters as integrals in the parameter space of a Bayesian neural network. After some development of this perspective, we establish an algorithm which provides sound bounds on both probabilistic robustness of Bayesian neural networks. In particular, we leverage advances in convex relaxation of neural network computations in order to arrive at sound lower and upper bounds on the result of marginalization over the posterior predictive distribution. This in turn allows us to provide concrete proofs of correctness (e.g., the absence of adversarial examples) for Bayesian neural networks. We validate this experimentally performing a detailed study of our algorithms parameters, as well as studies on real-world safety critical applications such as airborne collision avoidance. We also show how our methodology scales to small image classification tasks.

- We provide a principled probabilistic perspective on the incorporation of adversarial signals into the inference or approximate inference procedure of Bayesian neural networks. We first extend notions of local robustness via the introduction of a probability distribution over a set of robustness specifications. We then show how to incorporate this probabilistic perspective into a novel likelihood function. The result is an inference procedure which takes into account

a probabilistic – but still worst-case – adversary. Given that our methodology involves the development of a novel adversarial likelihood (and corresponding error model) our approach is compatible with all forms of approximate Bayesian inference for neural networks. We show empirically how our training method, in conjunction with five approximate inference methods, is the only one which is able to reliably instill verifiable robustness properties into the posterior distribution.

8.1.1 Strengths

This thesis considers and puts forward methodologies and definitions for local adversarial robustness of Bayesian neural networks. A key contribution in doing so is the development of guarantees that allow for deployment of Bayesian neural networks in safety critical applications. The definitions established in this thesis owe a great deal to the robustness literature for Gaussian processes and probabilistic verification; however, the methodology for Gaussian processes in no way translates to providing sound bounds on robustness of Bayesian neural networks. This also means that the methodology for computing robustness presented in this thesis is novel.

A further strength of this thesis is the comprehensive nature of the guarantees established. Many applications vary in their need for verification of adversarial robustness. Some settings such as airborne collision avoidance necessitate strong and certain assurances of correctness; for this, one can use the methods presented in Chapter 6. Yacht design, on the other hand, may only require some loose notion of robustness. For these applications we provide statistical guarantees with user-defined tightness criteria. In short, for each application, we have defined a suitable procedure for robustness estimation.

For each estimation algorithm, we also provide empirical evidence of effectiveness. Our experimental analysis provided evidence that our methodology is indeed effective at estimating statistical properties of larger convolutional Bayesian neural networks as well as proving sound properties of smaller networks on safety critical tasks such as airborne collision avoidance. Throughout the thesis we highlighted and empirically showed how probabilistic and decision robustness are monotonically decreasing functions of ϵ for any $\mathcal{B}_{\epsilon,p}$ input property. Similarly, we studied the effect of architecture size on the guarantees produced for various properties. This kind of analysis allows us to gain further insight into the robustness properties of our Bayesian posterior distributions and can help in model selection for those interested in deploying robust and uncertain neural networks. In the case of Chapter 7, we show through the

study of dozens of Bayesian neural networks that our methodology of robust inference (through the modification of the likelihood) is the best likelihood to achieve sound robustness guarantees for Bayesian neural networks.

8.1.2 Weaknesses

Though the methodology presented in this thesis is novel and to a large extent is empirically very effective, there is still room for improvement as well as various frontiers which require further attention. In Chapter 5, we present a study of both decision and probabilistic robustness. While we can discuss theoretical inequalities that link these two quantities, it remains to be seen exactly how these two quantities compare under various loss functions and specifications. Specifically, we have shown here that the two are very strongly correlated, but is there a scenario in which this correlation is loosened?

In Chapter 6, we provide strong evidence that our bounds are empirically effective in safety-critical scenarios and can scale to problems with large input dimension; however, it remains to be seen how much of an improvement can be made by relying on a tighter relaxation method than IBP. For example, using a MILP or branch and bound formulation to compute sound properties would be more expensive, but would allow a much better and more precise look at the robustness of Bayesian neural network posteriors. We also only study relatively small posteriors in this work. From our published work, we understand that we can scale to larger networks than what is presented in this thesis; however, the extent of scalability of these algorithms remains to be analysed both theoretically and empirically.

Finally, in Chapter 7 we give a perspective of incorporation of robustness into the likelihood. While we provide coverage of all of the modeling choices made wrt the likelihood (e.g., IBP or PGD usage and form of the probabilistic property density) we do not study how these choices are affected by the prior of our Bayesian neural network. This is a general question that is left unanswered by the contributions of this thesis: how does the selection of a prior influence the emergent robustness properties. Moreover, works published related to this thesis (e.g., [23]) make theoretical arguments about emergent robustness phenomena for Bayesian neural networks and these could have been studied under this framework.

8.2 Future Works

Given the above strengths and weaknesses, we put forward the following future avenues of research based on this thesis:

- Scalable probabilistic verification. Extending the methods presented in this thesis in order to get strong and sound bounds in realms such as medical diagnosis remains an important avenue of research for verified Bayesian machine learning.

- Stronger robustness properties. In this thesis we model general input-output properties $\{T, S\}$; however, we only study adversarial robustness. While this is an important first step, continuing to time-based properties such as those presented in [151] is needed. Or, an important potential direction for this research is proving that a network has a level of uncertainty everywhere that it is incorrect similarly to that which is investigated in [9].

- Effect of the prior. There are, in fact, several works which attempt to build informed priors for Bayesian deep learning [156]. Some of these works even allow for heightened robustness. It remains to be seen how the work presented in this thesis can be even further improved by the incorporation of a robust prior.

- Phenomenological study. Many claims have been made about Bayesian neural networks and their robustness properties. The techniques developed in this thesis provide us with the tools to empirically validate and explore these properties further. In particular, understanding how the kind of approximate inference chosen affects the emergent robustness properties of the posterior remains of great interest.

Bibliography

[1] Naveed Akhtar and Ajmal Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *Ieee Access*, 6:14410–14430, 2018.

[2] Moustafa Alzantot, Yash Sharma, Supriyo Chakraborty, Huan Zhang, Cho-Jui Hsieh, and Mani B Srivastava. Genattack: Practical black-box attacks with gradient-free optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1111–1119, 2019.

[3] David F Andrews. A robust method for multiple linear regression. *Technometrics*, 16(4):523–531, 1974.

[4] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International conference on machine learning*, pages 274–283. PMLR, 2018.

[5] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. In *International conference on machine learning*, pages 284–293. PMLR, 2018.

[6] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of model checking*. MIT Press, 2008.

[7] Arturs Bekasovs and Iain Murray. Bayesian adversarial spheres: Bayesian inference and adversarial examples in a noiseless setting. pages 1–6, November 2018. Third workshop on Bayesian Deep Learning 2018, NIPS 2018 Workshop ; Conference date: 07-12-2018 Through 07-12-2018.

[8] James O Berger. *Statistical decision theory and Bayesian analysis*. Springer Science & Business Media, 2013.

[9] Leonard Berrada, Sumanth Dathathri, Robert Stanforth, Rudy Bunel, Jonathan Uesato, Sven Gowal, M Pawan Kumar, et al. Verifying probabilistic specifications with functional lagrangians. *arXiv preprint arXiv:2102.09479*, 2021.

[10] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, 2018.

[11] C.M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, USA, 1995.

[12] Pier Giovanni Bissiri, Chris C Holmes, and Stephen G Walker. A general framework for updating belief distributions. *Journal of the Royal Statistical Society. Series B, Statistical methodology*, 78(5):1103, 2016.

[13] Pier Giovanni Bissiri and Stephen G Walker. On general bayesian inference using loss functions. *Statistics & Probability Letters*, 152:89–91, 2019.

[14] Arno Blaas, Luca Laurenti, Andrea Patane, Luca Cardelli, Marta Kwiatkowska, and Stephen Roberts. Adversarial robustness guarantees for classification with gaussian processes. *AISTATS*, 2020.

[15] Léonard Blier and Yann Ollivier. The description length of deep learning models. In *NeurIPS*, pages 2220–2230, 2018.

[16] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *ICML*, 2015.

[17] Akhilan Boopathy, Tsui-Wei Weng, Pin-Yu Chen, Sijia Liu, and Luca Daniel. Cnn-cert: An efficient framework for certifying robustness of convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3240–3247, 2019.

[18] George EP Box and George C Tiao. A further look at robustness via bayes's theorem. *Biometrika*, 49(3/4):419–432, 1962.

[19] Tom B Brown, Nicholas Carlini, Chiyuan Zhang, Catherine Olsson, Paul Christiano, and Ian Goodfellow. Unrestricted adversarial examples. *arXiv preprint arXiv:1809.08352*, 2018.

[20] Tom B Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch. *arXiv preprint arXiv:1712.09665*, 2017.

[21] Rudy Bunel, P Mudigonda, Ilker Turkaslan, P Torr, Jingyue Lu, and Pushmeet Kohli. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(2020), 2020.

[22] Rudy R Bunel, Ilker Turkaslan, Philip Torr, Pushmeet Kohli, and Pawan K Mudigonda. A unified view of piecewise linear neural network verification. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[23] Ginevra Carbone, Matthew Wicker, Luca Laurenti, Andrea Patane, Luca Bortolussi, and Guido Sanguinetti. Robustness of bayesian neural networks to gradient-based attacks. *35th Conference on Neural Information Processing Systems*, 2020.

[24] Luca Cardelli, Marta Kwiatkowska, Luca Laurenti, Nicola Paoletti, Andrea Patane, and Matthew Wicker. Statistical guarantees for the robustness of bayesian neural networks. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 5693–5700. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[25] Luca Cardelli, Marta Kwiatkowska, Luca Laurenti, and Andrea Patane. Robustness guarantees for bayesian inference with gaussian processes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7759–7768, 2019.

[26] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57, Los Alamitos, CA, USA, may 2017. IEEE Computer Society.

[27] Nicholas Carlini, Guy Katz, Clark Barrett, and David L Dill. Provably minimally-distorted adversarial examples. *arXiv preprint arXiv:1709.10207*, 2017.

[28] Nicholas Carlini, Pratyush Mishra, Tavish Vaidya, Yuankai Zhang, Micah Sherr, Clay Shields, David Wagner, and Wenchao Zhou. Hidden voice commands. In

25th {USENIX} Security Symposium ({USENIX} Security 16), pages 513–530, 2016.

[29] Nicholas Carlini and David Wagner. Defensive distillation is not robust to adversarial examples. *arXiv preprint arXiv:1607.04311*, 2016.

[30] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Adversarial attacks and defences: A survey. *arXiv preprint arXiv:1810.00069*, 2018.

[31] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pages 15–26, 2017.

[32] Tianqi Chen, Emily Fox, and Carlos Guestrin. Stochastic gradient Hamiltonian Monte Carlo. In *International Conference on Machine Learning*, pages 1683–1691, 2014.

[33] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning*, pages 1310–1320. PMLR, 2019.

[34] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[35] Paul Dagum, Richard Karp, Michael Luby, and Sheldon Ross. An optimal algorithm for monte carlo estimation. *SIAM Journal on computing*, 29(5):1484–1496, 2000.

[36] Elvis Dohmatob. Generalized no free lunch theorem for adversarial robustness. In *International Conference on Machine Learning*, pages 1646–1654. PMLR, 2019.

[37] Harris Drucker and Yann Le Cun. Improving generalization performance using double backpropagation. *IEEE Transactions on Neural Networks*, 3(6):991–997, 1992.

[38] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[39] Krishnamurthy Dvijotham, Marta Garnelo, Alhussein Fawzi, and Pushmeet Kohli. Verification of deep probabilistic models. *arXiv preprint arXiv:1812.02795*, 2018.

[40] Krishnamurthy (Dj) Dvijotham, Jamie Hayes, Borja Balle, Zico Kolter, Chongli Qin, Andras Gyorgy, Kai Xiao, Sven Gowal, and Pushmeet Kohli. A framework for robustness certification of smoothed classifiers using f-divergences. In *International Conference on Learning Representations*, 2020.

[41] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. Hotflip: White-box adversarial examples for text classification. In *ACL (2)*, pages 31–36, 2018.

[42] Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pages 269–286. Springer, 2017.

[43] Ruediger Ehlers. Planet. In *https://github.com/progirep/planet*. GitHub, 2017.

[44] Mohammad Emtiyaz Khan, Alexander Immer, Ehsan Abedi, and Maciej Korzepa. Approximate inference turns deep networks into Gaussian processes. *NeurIPS*, 2019.

[45] Logan Engstrom, Andrew Ilyas, and Anish Athalye. Evaluating and understanding the robustness of adversarial logit pairing. *SecML 2018*, 2018.

[46] Alhussein Fawzi, Hamza Fawzi, and Omar Fawzi. Adversarial vulnerability for any classifier. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 1186–1195, Red Hook, NY, USA, 2018. Curran Associates Inc.

[47] Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Analysis of classifiers' robustness to adversarial perturbations. *Machine Learning*, 107(3):481–508, 2018.

[48] Alhussein Fawzi, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. Robustness of classifiers: From adversarial to random noise. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 1632–1640, Red Hook, NY, USA, 2016. Curran Associates Inc.

[49] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting adversarial samples from artifacts. *arXiv:1703.00410*, 2017.

[50] Yarin Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

[51] Yarin Gal and Zoubin Ghahramani. Bayesian convolutional neural networks with bernoulli approximate variational inference. *ICLR 2016 workshop track, https://openreview.net/pdf?id=3QxqXoJEyfp7y9wltP11*, 2015.

[52] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *ICML*, pages 1050–1059, 2016.

[53] Yarin Gal and Lewis Smith. Sufficient conditions for idealised models to have no adversarial examples: a theoretical and empirical study with bayesian neural networks. *arXiv preprint arXiv:1806.00667*, 2018.

[54] 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 S&P*, pages 3–18. IEEE, 2018.

[55] Justin Gilmer, Ryan P Adams, Ian Goodfellow, David Andersen, and George E Dahl. Motivating the rules of the game for adversarial example research. *arXiv preprint arXiv:1807.06732*, 2018.

[56] Justin Gilmer, Luke Metz, Fartash Faghri, Sam Schoenholz, Maithra Raghu, Martin Wattenberg, and Ian Goodfellow. Adversarial spheres, 2018.

[57] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[58] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

[59] Pascale Gourdeau, Varun Kanade, Marta Kwiatkowska, and James Worrell. On the hardness of robust classification. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[60] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *SecML 2018*, 2018.

[61] Alex Graves. Practical variational inference for neural networks. *Advances in neural information processing systems*, 24, 2011.

[62] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial examples for malware detection. In *European symposium on research in computer security*, pages 62–79. Springer, 2017.

[63] Kathrin Grosse, David Pfaff, Michael T Smith, and Michael Backes. The limitations of model uncertainty in adversarial settings. *4th workshop on Bayesian Deep Learning (NeurIPS 2019), arXiv:1812.02606*, 2018.

[64] Peter Grünwald. The safe bayesian. In *International Conference on Algorithmic Learning Theory*, pages 169–183. Springer, 2012.

[65] Peter Grünwald and Thijs Van Ommen. Inconsistency of bayesian inference for misspecified linear models, and a proposal for repairing it. *Bayesian Analysis*, 12(4):1069–1103, 2017.

[66] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *International Conference on Learning Representations*, 2019.

[67] José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International conference on machine learning*, pages 1861–1869. PMLR, 2015.

[68] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. In *The collected works of Wassily Hoeffding*, pages 409–426. Springer, 1994.

[69] Hossein Hosseini, Baicen Xiao, and Radha Poovendran. Google's cloud vision api is not robust to noise. In *2017 16th IEEE international conference on machine learning and applications (ICMLA)*, pages 101–105. IEEE, 2017.

[70] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks*, 2013.

[71] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *CAV*, pages 3–29. Springer, 2017.

[72] Peter J Huber. A robust version of the probability ratio test. *The Annals of Mathematical Statistics*, pages 1753–1758, 1965.

[73] Peter J Huber. *Robust statistics*, volume 523. John Wiley & Sons, 2004.

[74] Todd Huster, Cho-Yu Jason Chiang, and Ritu Chadha. Limitations of the lipschitz constant as a defense against adversarial examples. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 16–29. Springer, 2018.

[75] Cyrille Jegourel, Jun Sun, and Jin Song Dong. Sequential schemes for frequentist estimation of properties in statistical model checking. In *International Conference on Quantitative Evaluation of Systems*, pages 333–350. Springer, 2017.

[76] Cyrille Jegourel, Jun Sun, and Jin Song Dong. On the sequential massart algorithm for statistical model checking. In *International Symposium on Leveraging Applications of Formal Methods*, pages 287–304. Springer, 2018.

[77] Kyle D Julian and Mykel J Kochenderfer. Guaranteeing safety for neural network-based aircraft collision avoidance systems. *DASC*, 2019.

[78] Harini Kannan, Alexey Kurakin, and Ian Goodfellow. Adversarial logit pairing. *arXiv preprint arXiv:1803.06373*, 2018.

[79] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *CAV*, 2017.

[80] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. The marabou framework for verification and analysis of deep neural

networks. In *International Conference on Computer Aided Verification*, pages 443–452. Springer, 2019.

[81] Alex Kendall and Yarin Gal. What uncertainties do we need in Bayesian deep learning for computer vision? In *NeurIPS*, 2017.

[82] Mohammad Khan, Didrik Nielsen, Voot Tangkaratt, Wu Lin, Yarin Gal, and Akash Srivastava. Fast and scalable bayesian deep learning by weight-perturbation in adam. In *International Conference on Machine Learning*, pages 2611–2620. PMLR, 2018.

[83] Mohammad Emtiyaz Khan and Siddharth Swaroop. Knowledge-adaptation priors. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[84] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *3rd International Conference for Learning Representations, arXiv arXiv:1412.6980*, 2014.

[85] Ranganath Krishnan, Mahesh Subedar, and Omesh Tickoo. Specifying weight priors in bayesian deep neural networks with empirical bayes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4477–4484, 2020.

[86] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Technical Report*, 2009.

[87] William Kruskal, Thomas S Ferguson, John W Tukey, and EJ Gumbel. Discussion of the papers of messrs. anscombe and daniel. *Technometrics*, 2(2):157–166, 1960.

[88] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[89] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale. 2017.

[90] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *NeurIPS*, pages 6402–6413, 2017.

[91] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. NIPS'17, page 6405–6416, Red Hook, NY, USA, 2017. Curran Associates Inc.

[92] Michael Truong Le, Frederik Diehl, Thomas Brunner, and Alois Knol. Uncertainty estimation for deep neural object detectors in safety-critical applications. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3873–3878. IEEE, 2018.

[93] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[94] Klas Leino, Zifan Wang, and Matt Fredrikson. Globally-robust neural networks. *arXiv preprint arXiv:2102.08452*, 2021.

[95] Jianlin Li, Jiangchao Liu, Pengfei Yang, Liqian Chen, Xiaowei Huang, and Lijun Zhang. Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification. In *International Static Analysis Symposium*, pages 296–319. Springer, 2019.

[96] Yingzhen Li and Yarin Gal. Dropout inference in bayesian neural networks with alpha-divergences. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 2052–2061. JMLR.org, 2017.

[97] Wu Lin, Mark Schmidt, and Mohammad Emtiyaz Khan. Handling the positive-definite constraint in the bayesian learning rule. In *International Conference on Machine Learning*, pages 6116–6126. PMLR, 2020.

[98] Xuanqing Liu, Yao Li, Chongruo Wu, and Cho-Jui Hsieh. Adv-bnn: Improved adversarial defense through robust Bayesian neural network. *ICLR*, 2019.

[99] David JC MacKay. A practical Bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

[100] Wesley J Maddox, Pavel Izmailov, Timur Garipov, Dmitry P Vetrov, and Andrew Gordon Wilson. A simple baseline for Bayesian uncertainty in deep learning. In *Advances in Neural Information Processing Systems*, pages 13132–13143, 2019.

[101] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. *arXiv e-prints*, June 2017.

[102] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[103] Bertrand Meyer. Soundness and completeness: With precision, Apr 2019.

[104] R. Michelmore, M. Wicker, L. Laurenti, L. Cardelli, Y. Gal, and M. Kwiatkowska. Uncertainty quantification with statistical guarantees in end-to-end autonomous driving control. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7344–7350, 2020.

[105] Rhiannon Michelmore, Matthew Wicker, Luca Laurenti, Luca Cardelli, Yarin Gal, and Marta Kwiatkowska. Uncertainty quantification with statistical guarantees in end-to-end autonomous driving control. *ICRA*, 2019.

[106] Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, pages 3578–3586. PMLR, 2018.

[107] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1765–1773, 2017.

[108] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.

[109] Eric Thomas Nalisnick. *On priors for bayesian neural networks*. University of California, Irvine, 2018.

[110] Vedant Nanda, Samuel Dooley, Sahil Singla, Soheil Feizi, and John P Dickerson. Fairness through robustness: Investigating robustness disparity in deep learning. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pages 466–477, 2021.

[111] Nina Narodytska. Formal analysis of deep binarized neural networks. In *IJCAI*, pages 5692–5696, 2018.

[112] Nina Narodytska, Shiva Kasiviswanathan, Leonid Ryzhyk, Mooly Sagiv, and Toby Walsh. Verifying properties of binarized deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[113] Radford M Neal. *Bayesian learning for neural networks*. Springer Science & Business Media, 2012.

[114] Radford M Neal et al. Mcmc using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

[115] Kazuki Osawa, Siddharth Swaroop, Mohammad Emtiyaz Khan, Anirudh Jain, Runa Eschenhagen, Richard E. Turner, and Rio Yokota. Practical deep learning with bayesian principles. In *NeurIPS*, pages 4289–4301, 2019.

[116] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P Wellman. Sok: Security and privacy in machine learning. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 399–414. IEEE, 2018.

[117] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE symposium on security and privacy (SP)*, pages 582–597. IEEE, 2016.

[118] Nicolas Papernot, Patrick D McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. corr abs/1511.07528 (2015). *arXiv preprint arXiv:1511.07528*, 2015.

[119] Kyriakos Polymenakos, Luca Laurenti, Andrea Patane, Jan-Peter Calliess, Luca Cardelli, Marta Kwiatkowska, Alessandro Abate, and Stephen Roberts. Safety guarantees for planning based on iterative gaussian processes. *arXiv preprint arXiv:1912.00071*, 2019.

[120] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *International Conference on Learning Representations*, 2018.

[121] Aditi Raghunathan, Jacob Steinhardt, and Percy S. Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In *NeurIPS*, pages 10900–10910, 2018.

[122] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer, 2003.

[123] Ambrish Rawat, Martin Wistuba, and Maria-Irina Nicolae. Adversarial phenomenon in the eyes of Bayesian deep learning. *arXiv preprint arXiv:1711.08244*, 2017.

[124] Stefano Rosa, Andrea Patane, Chris Xiaoxuan Lu, and Niki Trigoni. Semantic place understanding for human–robot coexistence—toward intelligent workplaces. *IEEE Transactions on Human-Machine Systems*, 49(2):160–170, 2018.

[125] Stefano Rosa, Andrea Patanè, Xiaoxuan Lu, and Niki Trigoni. Commonsense: Collaborative learning of scene semantics by robots and humans. In *Proceedings of the 1st International Workshop on Internet of People, Assistive Robots and Things*, pages 1–6, 2018.

[126] Ishai Rosenberg, Asaf Shabtai, Lior Rokach, and Yuval Elovici. Generic black-box end-to-end attack against state of the art api call based malware classifiers. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 490–510. Springer, 2018.

[127] Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. *IJCAI*, 2018.

[128] Ali Shafahi, W. Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 6106–6116, Red Hook, NY, USA, 2018. Curran Associates Inc.

[129] Ali Shafahi, W. Ronny Huang, Christoph Studer, Soheil Feizi, and Tom Goldstein. Are adversarial examples inevitable? In *International Conference on Learning Representations*, 2019.

[130] Yash Sharma and Pin-Yu Chen. Bypassing feature squeezing by increasing adversary strength, 2018.

[131] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin T Vechev. Fast and effective robustness certification. *NeurIPS*, 1(4):6, 2018.

[132] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–30, 2019.

[133] Michael Sipser. Introduction to the theory of computation. *ACM Sigact News*, 27(1):27–29, 1996.

[134] Lewis Smith and Yarin Gal. Understanding measures of uncertainty for adversarial example detection. *UAI*, 2018.

[135] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

[136] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *ICLR*, 2014.

[137] Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations*, 2019.

[138] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. In *International Conference on Learning Representations*, 2019.

[139] Yusuke Tsuzuku, Issei Sato, and Masashi Sugiyama. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 6542–6551, Red Hook, NY, USA, 2018. Curran Associates Inc.

[140] John W Tukey. A survey of sampling from contaminated distributions. *Contributions to probability and statistics*, pages 448–485, 1960.

[141] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[142] Vladimir N Vapnik. An overview of statistical learning theory. *IEEE transactions on neural networks*, 10(5):988–999, 1999.

[143] Mariia Vladimirova, Jakob Verbeek, Pablo Mesejo, and Julyan Arbel. Understanding priors in bayesian neural networks at the unit level. In *International Conference on Machine Learning*, pages 6458–6467. PMLR, 2019.

[144] Benjie Wang, Stefan Webb, and Tom Rainforth. Statistically robust neural network classification. *arXiv preprint arXiv:1912.04884*, 2019.

[145] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *USENIX Security 18*, 2018.

[146] Siyue Wang, Xiao Wang, Pu Zhao, Wujie Wen, David Kaeli, Peter Chin, and Xue Lin. Defensive dropout for hardening deep neural networks under adversarial attacks. In *Proceedings of the International Conference on Computer-Aided Design*, page 71. ACM, 2018.

[147] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688. Citeseer, 2011.

[148] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. Feature-guided black-box safety testing of deep neural networks. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 408–426. Springer, 2018.

[149] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. Feature-guided black-box safety testing of deep neural networks. In *TACAS*, pages 408–426. Springer, 2018.

[150] Matthew Wicker, Luca Laurenti, Andrea Patane, and Marta Kwiatkowska. Probabilistic safety for bayesian neural networks. In *Conference on Uncertainty in Artificial Intelligence*, pages 1198–1207. PMLR, 2020.

[151] Matthew Wicker, Luca Laurenti, Andrea Patane, Nicola Paoletti, Alessandro Abate, and Marta Kwiatkowska. Certification of iterative predictions in bayesian neural networks. *arXiv preprint arXiv:2105.10134*, 2021.

[152] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pages 5286–5295. PMLR, 2018.

[153] Min Wu, Matthew Wicker, Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. A game-based approximate verification of deep neural networks with provable guarantees. *Theoretical Computer Science*, 807:298–329, 2020.

[154] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[155] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017.

[156] Wanqian Yang, Lars Lorch, Moritz A Graule, Srivatsan Srinivasan, Anirudh Suresh, Jiayu Yao, Melanie F Pradier, and Finale Doshi-Velez. Output-constrained bayesian neural networks. *arXiv preprint arXiv:1905.06287*, 2019.

[157] Nanyang Ye and Zhanxing Zhu. Bayesian adversarial learning. In *Proceedings of the 32nd international conference on neural information processing systems*, pages 6892–6901, 2018.

[158] Matthew Yuan, Matthew Robert Wicker, and Luca Laurenti. Gradient-free adversarial attacks for bayesian neural networks. In *Third Symposium on Advances in Approximate Bayesian Inference*, 2021.

[159] Mikhail Yurochkin, Amanda Bower, and Yuekai Sun. Training individually fair ml models with sensitive subspace robustness. *arXiv preprint arXiv:1907.00020*, 2019.

[160] Radosiaw R Zakrzewski. Verification of a trained neural network accuracy. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)*, volume 3, pages 1657–1662. IEEE, 2001.

[161] Runtian Zhai, Chen Dan, Di He, Huan Zhang, Boqing Gong, Pradeep Ravikumar, Cho-Jui Hsieh, and Liwei Wang. Macer: Attack-free and scalable robust training via maximizing certified radius. In *International Conference on Learning Representations*, 2020.

[162] Guodong Zhang, Shengyang Sun, David Duvenaud, and Roger Grosse. Noisy natural gradient as variational inference. In *International Conference on Machine Learning*, pages 5852–5861. PMLR, 2018.

[163] Guoming Zhang, Chen Yan, Xiaoyu Ji, Tianchen Zhang, Taimin Zhang, and Wenyuan Xu. Dolphinattack: Inaudible voice commands. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 103–117, 2017.

[164] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric Xing, Laurent El Ghaoui, and Michael Jordan. Theoretically principled trade-off between robustness and accuracy. In *International Conference on Machine Learning*, pages 7472–7482. PMLR, 2019.

[165] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In *NeurIPS*, pages 4939–4948, 2018.

[166] Yijie Zhang and Eric Nalisnick. On the inconsistency of bayesian inference for misspecified neural networks. In *Third Symposium on Advances in Approximate Bayesian Inference*, 2020.

[167] Stephan Zheng, Yang Song, Thomas Leung, and Ian Goodfellow. Improving the robustness of deep neural networks via stability training. In *Proceedings of the ieee conference on computer vision and pattern recognition*, pages 4480–4488, 2016.

[168] Roland S Zimmermann. Comment on" adv-bnn: Improved adversarial defense through robust bayesian neural network". *arXiv preprint arXiv:1907.00895*, 2019.

Appendix A

Appendix for Chapter 5

In this section of the Appendix we give implementation details and further particulars regrading the experiments for Chapter 5 of this thesis. Upon final submission of the thesis all of the source code to reproduce our results will be made public along with this manuscript. Below, we spend one section on each dataset tested. We open this section with a brief discussion of a few empirical tricks which are outside of the scope of our methodological contribution, but can make implementation of our algorithms more efficient and practical.

A.1 General Hardware/Software Setup

We highlight that for all computational times reported in the thesis a single CPU was used and the code was run without access to a GPU for sake of consistency.

Hardware

- CPU: 2x 20-core Intel Core Xeon 6230 Gold
- GPU: 6x NVIDIA 2080Ti

Software

- Programming Language: Python 3.8
- Inference Algorithms: pre-release version of DeepBayes framework
- Auto-differentiation: Tensorflow 2.0/Tensorflow Probability
- Further Dependencies: Numpy, Scikit learn, Matplotlib, Seaborn

A.2 UCI Experiment Details

In the main text we list statistical bounds using PGD and IBP for approximate Bayesian posteriors trained on a commonly used subset of regression tasks from the UCI Machine learning repository [38]. For these tasks, we train a neural network with a single layer and 25 hidden units. The hidden layer is equipped with rectified linear unit (Relu) activation functions. The output layer comprises a single unit with a linear activation function (i.e. no activation function). For these networks we use both a mean squared error likelihood and loss function. We approximately infer each posterior with the modification of the Variational Online Gauss Newton (VOGN) algorithm described in [97] and opt for our variational posterior to be a Gaussian with diagonal covariance. In Table A.1, we provide the learning rate, epochs, prior constant, and batch size used for each of the presented datasets. We highlight the meaning of the *prior* constant row which sees us tuning the variance of our Gaussian prior. The framework that we have created to perform approximate Bayesian inference automatically determines the variance for each network based on the variance used to initialize the corresponding deterministic neural network [135]. This kind of initialization has been well-studied, and performs well in practice for a large number of networks. We find that it also works well as a baseline for general selection of Bayesian neural network priors, but adding a scaling factor can improve results. As such, we pick a scaling factor that works well according to a randomly held validation set before observing our results on the test set. Of course, it should be noted that this kind of treatment of our prior distribution does correspond to an empirical Bayesian approach (or potentially a hierarchical Bayesian approach).

	Boston	Concrete	Yacht	Energy	Kin8nm	Naval
Learning Rate	0.225	0.15	0.45	0.125	0.4	0.5
Epochs	3000	2000	2750	2000	1500	1500
Prior Constant	0.25	1.00	0.01	1.00	0.01	0.01
Batch Size	256	256	64	256	256	256

Table A.1: UCI regression benchmark training hyperparameters for each dataset.

A.3 MNIST Experiment Details

	MNIST
Learning Rate	0.35
Epochs	20
Prior Constant	2.0
Batch Size	128

Table A.2: MNIST training hyperparameters used consistently for each architecture.

In Chapter 5, we train a series of six fully-connected neural networks. We train three single-layer and three two-layer fully-connected networks. For both layer configurations we train three neural networks with 24, 48, and 64 hidden nodes per layer, respectively. For each network the parameters of the Variational Online Gauss-Newton (VOGN) algorithm are listed in Table A.2 are used. While having a single set of parameters likely results in sub-optimal performance for some of the posteriors we still find empirically that each of the approximate posteriors inferred with these parameters achieves greater than 95% test-set accuracy.

A.4 GTSRB Experiment Details

Layer	Layer Params	Activation
2D Convolution	6 x (5, 5)	Relu
2D Convolution	6 x (5, 5)	Relu
2D Max Pooling	(2, 2)	Relu
Flatten	N/A	N/A
Dense	50	Relu
Dense	2	Softmax

Table A.3: The four hidden layer convolutional neural network used for training on the GTSRB benchmark. Optimization parameters for each approximate inference method can be found in Table A.4.

	VOGN	NoisyAdam	SWAG
Learning Rate	0.25	0.025	0.015
LR Decay	0.025	0.025	0.00
Epochs	25	25	25
Prior Constant	1.0	1.0	N/A
Batch Size	128	128	128

Table A.4: GTSRB Training hyper-parameters for the convolutional neural network reported in Table A.3.

For the GTSRB dataset we pair down the data to roughly 6000 examples which represent the two classes visualized in Figure 5.8a. Namely, speed limit 50 km/h advisory and a road work ahead advisory signs. We start by holding out 1000 images as a test set. We then augment the training data with a further 5000 images which are the result of randomized cropping, vertical shifts, horizontal shifts, and rotation of images in the training set. The robustness results reported in the chapter are from images randomly selected from the held out test set.

Appendix B

Appendix for Chapter 6

B.1 Hardware/Software Setup

With the exception of the VCAS analysis, the hardware and software setup remains consistent with what is reported in Appendix A.1. The VCAS dataset analysis was completed originally for [150] and was computed on a 2013 Macbook Pro with an Intel i5 processor and 16GB or RAM. It was also completed with Python 2.7.

B.2 Further Experimental Details

B.2.1 Experiment Details for VCAS

Training of a Bayesian neural network for VCAS is done with Bayes by Backprop [16]. The network architecture is a single-layer fully-connected neural network with 512 hidden nodes. The network employs Relu activation functions on the hidden layer and a linear activation on ouput layer. The learning rate is set to 0.005 for this network and it is trained for 20 epochs with a batch size of 512.

Full code for training and evaluation of the VCAS benchmark reported in this thesis can be found in `https://github.com/matthewwicker/ProbabilisticSafetyforBNNs`.

B.2.2 Experiment Details for UCI Datasets

Chapter 6 employs the same approximate posteriors reported for Chapter 5. For the training parameters used in the Variational Online Gauss-Newton (VOGN) inference method for each network we refer readers to Table A.1. For full discussion we refer readers to Appendix A.2.

B.2.3 Experiment Details for MNIST

The same approximate posteriors tested in Chapter 5 are tested in Chapter 6. We refer readers to Appendix A.3 for the full training details used.

B.3 Proofs

In this section of the Appendix, we provide proofs for the main propositions stated in the thesis. The proof of Proposition 4 is due to Andrea Patane and was originally stated in [150].

B.3.1 Proof of Proposition 4

The bounding box can be computed iteratively in the number of hidden layers of the network, K. We show how to compute the lower bound of the bounding box; the computation for the maximum is analogous.

Consider the k-th network layer, for $k = 0, \dots, K$, we want to find for $i = 1, \dots n_{k+1}$:

$$\min_{\substack{W_{i:}^{(k)} \in [W_{i:}^{(k),L}, W_{i:}^{(k),U}] \\ z^{(k)} \in [z^{(k),L}, z^{(k),U}] \\ b_i^{(k)} \in [b_i^{(k),L}, b_i^{(k),U}]}} z_i^{(k+1)} = \sigma\left(\sum_{j=1}^{n_k} W_{ij}^{(k)} z_j^{(k)} + b_i^{(k)}\right).$$

As the activation function σ is monotonic, it suffice to find the minimum of: $\sum_{j=1}^{n_k} W_{ij}^{(k)} z_j^{(k)} + b_i^{(k)}$. Since $W_{ij}^{(k)} z_j^{(k)}$ is a bi-linear form defined on an hyper-rectangle, it follows that it obtains its minimum in one of the four corners of the rectangle $[W_{ij}^{(k),L}, W_{ij}^{(k),U}] \times [z_j^{(k),L}, z_j^{(k),U}]$.

Let $t_{ij}^{(k),L} = \min\{W_{ij}^{(k),L} z_j^{(k),L}, W_{ij}^{(k),U} z_j^{(k),L}, W_{ij}^{(k),L} z_j^{(k),U}, W_{ij}^{(k),U} z_j^{(k),U}\}$ we hence have:

$$\sum_{j=1}^{n_k} W_{ij}^{(k)} z_j^{(k)} + b_i^{(k)} \geq \sum_{j=1}^{n_k} t_{ij}^{(k),L} + b_i^{(k),L} =: \zeta_i^{(k+1),L}.$$

Thus for every $W_{i:}^{(k)} \in [W_{i:}^{(k),L}, W_{i:}^{(k),U}]$, $z^{(k)} \in [z^{(k),L}, z^{(k),U}]$ and $b_i^{(k)} \in [b_i^{(k),L}, b_i^{(k),U}]$ we have:

$$\sigma\left(\sum_{j=1}^{n_k} W_{ij}^{(k)} z_j^{(k)} + b_i^{(k)}\right) \geq \sigma\left(\zeta_i^{(k+1),L}\right)$$

that is $z_i^{(k+1),L} = \sigma\left(\zeta_i^{(k+1),L}\right)$ is a lower bound to the solution of the minimisation problem posed above.

□

B.3.2 Proof of Proposition 5

Here, we prove that the outputs of Proposition 5 over-approximates the worst-case output of a BNN. As in Appendix B.3.1, we use the fact that the activation function σ is monotonic, therefore it suffices to find the minimum of: $\sum_{j=1}^{n_k} W_{ij}^{(k)} z_j^{(k)} + b_i^{(k)}$. Since $W_{ij}^{(k)} z_j^{(k)}$ is a bi-linear form defined on an hyper-rectangle, it follows that it obtains its minimum in one of the four corners of the rectangle $[W_{ij}^{(k),L}, W_{ij}^{(k),U}] \times [z_j^{(k),L}, z_j^{(k),U}]$. Moreover, we know that simply selecting $b^{(k),L}$ must minimize the objective so we omit it from consideration below.

Now, to prove Proposition 5, w.l.o.g. it suffices to prove that the result of Equation (6.7) bounds the worst-case combination from a rectangle $[W_{ij}^{(k),L}, W_{ij}^{(k),U}] \times [z_j^{(k),L}, z_j^{(k),U}]$ from below before we consider the bias. We override our notation to give us the bound before the application of the bias:

$$\zeta_j^{(k+1)} = W_{ij}^{(k),c} z_j^{(k),c} - |W_{ij}^{(k),r}| z_j^{(k),c} - W_{ij}^{(k),c} | z_j^{(k),r}| - |W_{ij}^{(k),r}||z_j^{(k),r}| \tag{B.1}$$

We use an exhaustive argument to show that the result of Equation (B.1) bounds the lowest value of the rectangle $[W_{ij}^{(k),L}, W_{ij}^{(k),U}] \times [z_j^{(k),L}, z_j^{(k),U}]$ from below.

The first case is one in which both intervals have no width (i.e. $z_j^{(k),L} = z_j^{(k),U}$ and $W_{ij}^{(k),L} = W_{ij}^{(k),U}$). In this case, we know that $z_j^{(k),L} = z_j^{(k),U} = z_j^{(k),c}$ and $W_{ij}^{(k),L} = W_{ij}^{(k),U} = W_{ij}^{(k),c}$. Thus, the first term in Equation (B.1) gives us the exact lowest value in the rectangle. Moreover, because both $|W_{ij}^{(k),r}| = 0$ and $|z_j^{(k),r}| = 0$ all of the other terms in Equation (B.1) are 0. Thus the result of the lower bound gives us exactly the value of the lower bound of the rectangle.

Next we have the case where only one of these conditions is true. That is, $z_j^{(k),L} = z_j^{(k),U}$ or $W_{ij}^{(k),L} = W_{ij}^{(k),U}$. W.l.o.g., consider that only $W_{ij}^{(k),L} = W_{ij}^{(k),U}$. As before this implies $W_{ij}^{(k),L} = W_{ij}^{(k),U} = W_{ij}^{(k),c}$. Further we know that $z_j^{(k),L} < z_j^{(k),U}$. We know that the lowest value of the rectangle in this case is at $W_{ij}^{(k),c} z_j^{(k),L}$. Observing our bound, the first term $W_{ij}^{(k),c} z_j^{(k),c}$ gives us how the center of the input interval, $z_j^{(k),c}$ is shifted by the weight value. The next term in the lower bound is $-|W_{ij}^{(k),c}| z_j^{(k),r}$ which accounts for the negative contribution of the width of the input interval scaled by the value of $|W_{ij}^{(k),c}|$. The rest of the terms in the lower bound in this case are 0. Therefore, if,

$$W_{ij}^{(k),c} z_j^{(k),L} \leq W_{ij}^{(k),c} z_j^{(k),c} - |W_{ij}^{(k),c}| z_j^{(k),r}$$

then we know that the lower bound is sound in this case. This can be observed by simply plugging in the definitions of the terms for the right-hand side of the inequality

and divide both sides by $W_{ij}^{(k),c}$ to get:

$$
\begin{aligned}
z_j^{(k),L} &\leq \left(\frac{z_j^{(k),U} + z_j^{(k),L}}{2}\right) - \left(\frac{z_j^{(k),U} - z_j^{(k),L}}{2}\right) \\
2z_j^{(k),L} &\leq z_j^{(k),U} + z_j^{(k),L} - z_j^{(k),U} + z_j^{(k),L} \\
2z_j^{(k),L} &\leq 2z_j^{(k),L}
\end{aligned}
$$

Because this inequality holds, we know that this case also holds.

Finally, we have the case that neither $z_j^{(k),L} = z_j^{(k),U}$ nor $W_{ij}^{(k),L} = W_{ij}^{(k),U}$. This implies that $z_j^{(k),L} < z_j^{(k),U}$ and $W_{ij}^{(k),L} < W_{ij}^{(k),U}$. It is straightforward in this case to see that the first term, $W_{ij}^{(k),c} z_j^{(k),c}$, is again the center of the interval. The second term properly accounts for the width of the input interval scaled by the strictly positive value of the center of the weight $|W_{ij}^{(k),c}| z_j^{(k),r}$ (see the second case for proof). The third term accounts for the $W_{ij}^{(k),r} |z_j^{(k),c}|$ width of the weight interval scaled by center of the input interval (see the second case for proof). And the final term $|W_{ij}^{(k),r}||z_j^{(k),r}|$ accounts for the possibility that both or either of the intervals is centered at 0. When one or both of the intervals are not centered at 0 the term $|W_{ij}^{(k),r}||z_j^{(k),r}|$ introduces an over-approximation of the lower bound of the value of the rectangle. □

Appendix C

Appendix for Chapter 7

In this Appendix, we provide the details to aid in the reproducibility of our results . For the code to reproduce both the experiments found in the main text and in these extended materials see the github code repository released upon completion of this thesis

C.1 Approximate Inference Parameters

In this section, we list the training parameters that we used for the training of each of the networks discussed in the main text.

C.1.1 MNIST and FashionMNIST Parameters

	SWAG	NoisyAdam	VOGN	BBB	HMC
Learning Rate	0.1	0.001	0.35	0.45	0.075
Prior Scaling	N/A	10	10	20	500
Batch Size	128	128	128	128	60k
Epochs/Samples	20/250	20/(N/A)	20/(N/A)	20/(N/A)	(N/A)/25
PGD Iterations	10	10	10	10	10

Each network trained on MNIST is a single hidden layer fully-connected architecture with 512 neurons in the hidden layer. The parameters used for the 5 training methods are listed in the table above. Prior scaling refers to a multiplicative constant w.r.t. the initialisation parameters described in [135]. In fact, we often find the initial variance described in the later to be too small for retrieving good uncertainty estimates, and, thus, we further multiply it by the values reported in the table. Further parameters that are specific to HMC, and not included in the table, are: 3 iterations of burn-in, with 20 steps of the leapfrog numerical integrator followed by the

reported 25 samples from the posterior each which explore the chain for 25 steps with the leapfrog integrator. We again note that when we perform approximate inference with HMC and the robust likelihood that we choose the initial network parameters to be the result of 10 epochs of stochastic gradient descent rather than the full-data gradient descent used during normal burn-in. Finally, we note that we follow the empirically optimal procedure stated by [60]. In particular, we train with an η linearly increasing to its target value at every epoch. Again as in [60], we set the target η value 10% larger than the 'desired' robustness value.

C.1.2 CIFAR10 Parameters

Layer	Layer Params	Activation
2D Convolution	16 x (4, 4)	Relu
2D Convolution	32 x (4, 4)	Relu
2D Max Pooling	(2, 2)	Relu
Flatten	N/A	N/A
Dense	100	Relu
Dense	10	Softmax

Table C.1: The four hidden layer convolutional neural network used for training on the CIFAR benchmark. Optimization parameters for each approximate inference method can be found in Table C.2.

	SWAG	NoisyAdam	VOGN
Learning Rate	0.015	0.00025	0.25
LR Decay	0.0	0.025	0.025
Prior Scaling	N/A	5	5
Batch Size	128	128	128
Epochs/Samples	45/500	45/(N/A)	45/(N/A)
PGD Iterations	10	10	10

Table C.2: CIFAR-10 Training hyper-parameters for the convolutional neural network reported in Table C.1.

.

For CIFAR10, prior to inference we perform data augmentation which involves horizontal flipping as well as random translations by up to 4 pixels. We randomly select an image from the train set with uniform probability and then select a transformation (translation or horizontal flipping) until we have augmented the data size

from 60k to 100k images Finally, the network architecture is made of two convolutional layers, respectively with 16 and 32 four by four filters, followed by a 2 by 2 max pooling layer, and a fully connected layer with 100 hidden neurons.