On the Role of Explainability and Uncertainty in Ensuring Safety of AI Applications

Rhiannon Falconmore

Balliol College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2022

To my wonderful husband Ben,

I couldn't have done this without you

Acknowledgements

A massive thank you to my supervisor, Professor Marta Kwiatkowska, who has guided me throughout this project and provided invaluable support when I wasn't sure what to do next. Another massive thank you to everyone in our research group, especially Pascale, Max, Matthew, Andrea, Emanuele and Luca, who have been both excellent collaborators and excellent friends. Speaking of friends, I very much value the support of my close friends and family: Jamie, Rayhaan, David, Will, everyone in Jeff's server, and my entire family (including in-laws!). Last but not least, I want to say thank you to my husband Ben, who knows how hard it is to do a PhD and made sure to encourage and support me the entire way through, and had faith in me even when I didn't.

This work received partial support from the FUN2MODEL project (grant agreement No. 834115) led by my supervisor Marta Kwiatkowska, funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme. This work also received support from the Engineering and Physical Science Research Council (grant reference 1894767). Finally, this work received support from Balliol College Oxford, where I was awarded the Dervorguilla scholarship.

Abstract

Deep learning, and in particular neural networks (NNs), have seen a surge in popularity over the past decade. Their use has increased in, often safety-critical, decision-making systems such as self-driving, medical diagnosis and natural language processing. Thus, there is an urgent need for methodologies to aid the development of AI-based systems. In this thesis, we investigate the role that explainability and uncertainty can play in providing safety assurance for AI applications based on neural networks. Our first contribution, studied primarily for decisions based on neural network models, is a method to derive local explanations with provable robustness and optimality guarantees called Optimal Robust Explanations (OREs). OREs imply the model prediction and thus provide sufficient reason for the model decision. We develop an algorithm to extract OREs that uses a neural network verification tool Marabou or Neurify as a black-box solver. We demonstrate the usefulness of OREs in model development and safety assurance tasks such as model debugging, bias evaluation and repair of explanations provided by non-formal explainers such as Anchors. Our second contribution focuses on an autonomous driving scenario enabled by an end-to-end Bayesian neural network (BNN) controller trained on data from the Carla simulator. BNNs have the ability to capture the uncertainty within the learning model, while retaining the main advantages intrinsic to neural networks. We propose two methods to evaluate the safety of decisions of BNN controllers in the presence of uncertainty in offline and online settings. We develop techniques to approximate bounds on the safety of the entire system with respect to given criteria, with

high probability and *a priori* statistical guarantees. Our final contribution a collection of methods that combine the uncertainty information available from Bayesian neural networks with local explanation methods. We show how to formulate Bayesian versions of existing feature scoring explanation methods, as well as a Bayesian version of our OREs, called Bayes-optimal robust explanations (B-OREs). We define a covering explanation, which condenses the information produced from a number of BNN posterior samples into a single explanation, with a probability of the likelihood that this explanation is an explanation of a random sample. In the case of Bayes-optimal robust covering explanations, we obtain a probability for how likely the explanation is to imply the prediction. We combine Bayesian covering explanations with a notion of feature uncertainty, to give an ordering of importance to each feature that appears in the covering explanation, and we show that feature uncertainty can be used to provide a global overview of the input features that the model most associates with each class.

Contents

List of Figures

Chapter 1

Introduction

Deep learning, and in particular neural networks (NNs), have seen a surge in popularity over the past decade. NNs are becoming more prevalent in real-world, often safety-critical, decision-making systems such as self-driving, natural language processing and medical diagnosis. As a result of this, AI safety has become a central problem, as we seek to complement such highly-accurate but opaque models with comprehensible explanations and rigorous safety guarantees.

Currently, since few safety guarantees are available, we have seen erroneous edge-case behaviours already. A notable example of such behaviour occurred in the application of AI systems to autonomous driving. In 2016, a Tesla Model S (a brand and model of car well known for its autonomous driving modes) failed to distinguish a white trailer against the bright sky, leading to a fatal crash [132]. Thus, there is an urgent need for methods that are capable of accurately detecting, analysing and diagnosing such erroneous behaviours.

Explaining the reason behind a decision is equally important for ensuring AI safety, and it is even required by law in some regions of the world [24]. Consider, for example, the traffic sign shown in Figure 1.1. By examining which parts of the image influence a decision by a model that classifies such signs, human interpreters can verify that the appropriate and relevant information is used (for example, we might require that the model classifies based on the sign itself and not the background which could indicate a spurious decision). Model developers also benefit from explanations, as they facilitate model debugging and bias evaluation. For example, consider the following Tweet from a sentiment analysis dataset extracted from Twitter [43]: *"i really want to listen to some taylor swift i cant find the CD & i dont feel like playing it on the computer it sounds better on the cd"*. This Tweet has a negative sentiment, but a simple neural network developed as part of this research classifies it as positive. By examining the *explanation* behind the classification, we can determine whether the positive classification can be solely attributed to the presence of the name of popular singer-songwriter Taylor Swift or not, i.e whether it is a biased or unjustified decision.

A variant of neural networks called Bayesian neural networks (BNNs) have the ability to capture the uncertainty within the learning model, while retaining the main advantages intrinsic to neural networks [83]. As a consequence, they are particularly appealing for safety-critical applications, such as autonomous driving, where uncertainty estimates can be propagated through the decision pipeline to enable safe decision making [85]. Intuitively, if the uncertainty of the model is high, we may want to check its prediction more rigorously, or intervene. Consider, for example, a self-driving car that, while driving, finds an obstacle in the middle of the road. At each timestep, the controller may be uncertain on the steering angle to apply and, in order to avoid the obstacle, may choose angles which turn the car either right or left, with equal probability. A BNN controller is able to indicate such uncertainty, unlike an NN controller which may end up wiggling along a straight line towards the obstacle (the result of picking left, right, left and so on). Having precise quantitative measures of the BNN uncertainty facilitates the detection and resolution of such ambiguous situations.

Additionally, it is important that explanations remain robust and useful when faced with the uncertainty within the decision making process, as highly variable, complex explanations for very similar decisions do not provide good or useful feedback for the human interpreter. Guaranteeing robustness to adversarial perturbations

Figure 1.1: A picture of a traffic sign from the GTSRB dataset.

ensures stability of the explanation[1], and useful explanations can be gathered using user-defined cost functions (to ensure the resulting explanation is relevant and useful). Currently, there is a lack of techniques exploring the interplay between uncertainty information and explanations in ensuring AI safety.

1.1 Contributions

In this thesis, we investigate the role that explainability and uncertainty can play in ensuring the safety of AI applications. We show this in the setting of decision functions of the form $f : X \to Y$. Here, X is a set of inputs represented as feature vectors, Y is an output set (a class label for classification problems, or a continuous output for regression) and decision function f is a neural network that makes a prediction that is required to be safe. Feature vectors can encompass anything from pixel values to words (via the use of embedding functions), and so this decision function captures a wide range of applications not limited to those discussed in this thesis.

In our first setting, setting A, we consider deterministic decision functions and feature vectors in the form of word embeddings representing text. We develop a collection of methods to extract robust and optimal explanations for such decision

[1]While stability is often defined as achieving the same result over each subsequent test, we note that explanations are not necessarily unique and we therefore define stability as being locally-robust to adversarial perturbations i.e. one (of potentially several) explanation of input x is still an explanation of an adversely perturbed input x'.

functions. These methods can be used for model debugging and bias detection, among others, which are important in improving AI safety. Here, we work with sentiment analysis as a proxy application for scalability reasons: it is a task with a lower dimensionality than, for example, image classification, and the solvers we use struggle with anything more. Despite this, we stress that these methods are applicable to many other domains and feature vector types. This setting and corresponding work is described in Chapter 4.

We then extend setting A into setting B by adding uncertainty into the decision functions, and we now consider feature vectors in the form of images. We explore ways of extracting and quantifying uncertainty information from such decision functions, and how this information can be used in improving safety. As our proxy application, we use an end-to-end autonomous driving controller represented as a Bayesian neural network, and we study how uncertainty can be used to ensure safety in both offline and online settings. This setting and corresponding work is described in Chapter 5.

In setting C, we investigate a combination of settings A and B: we consider decision functions that incorporate uncertainty and input feature vectors in the form of both word embeddings and images. We develop a collection of methods to provide feedback to modellers and human interpreters that integrate explanations with uncertainty information in a principled manner. We show how existing explanation methods can be formulated in a Bayesian manner, including those covered in setting A, and we explore how to extract and use individual feature uncertainty to ensure safety. Our methods are demonstrated on both image classification and sentiment analysis as proxy applications, and this setting and corresponding work is described in Chapter 6.

The main contributions of this thesis can be summarised as follows:

1. We present a method to derive local explanations, primarily for Natual Language Processing (NLP) models, with provable robustness and optimality guarantees. We call such explanations Optimal Robust Explanations (OREs). This method shares similarities with abduction based explanations (ABEs) [58] in that the computed ORE implies the decision, but is better suited to any model (ín particular NLP models) where the unbounded nature of ABEs may result

in trivial explanations, of no use to the human interpreter.

2. We demonstrate that our local explainability approach can provide useful explanations for non-trivial fully-connected and convolutional neural networks on three widely used sentiment analysis benchmarks, and we provide a framework for computing our OREs with a choice of solution algorithms based on either hitting sets or minimum satisfying assignments, in conjunction with neural network verification tools Marabou and Neurify.

3. We compare OREs with a state-of-the-art explanation method, called Anchors, and we show that Anchors often lacks prediction robustness in our benchmarks. We demonstrate the usefulness of our framework on tasks such as model debugging, bias evaluation and repair of non-formal explainers like Anchors.

4. We present a statistical method for evaluating the safety of end-to-end BNN controllers for applications in self-driving. This method allows one to obtain and quantify the quality of uncertainty estimates for the controller's decisions, and provides bounds on the safety of the entire system with respect to a given criteria, with high probability and *a priori* statistical guarantees.

5. We show that this statistical framework can be used to evaluate controller's model robustness to changes in weather, location and observation noise, and we empirically demonstrate how our real-time statistical estimates can be used to avoid a high percentage of collisions.

6. We present a collection of methods that combine the uncertainty information drawn from Bayesian neural networks with local explanation methods. We show how to formulate Bayesian versions of feature score-based explanation methods such as Layerwise Relevance Propagation (LRP), as well as a Bayesian version of our OREs, called Bayes-optimal robust explanations (B-OREs).

7. We define a covering explanation, which condenses the information produced from a number of BNN posterior samples into a single explanation, yielding the

probability that the explanation is an explanation of a random sample. In the case of Bayes-optimal robust covering explanations, we obtain a probability for how likely the explanation is to imply the prediction.

8. We combine Bayesian covering explanations with a notion of feature uncertainty, to give an ordering of importance for each feature that appears in the covering explanation.

9. We show that feature uncertainty can be used to give a global overview of the input features that the model most associates with each class.

1.2 Publications

Some of the work in this thesis has been published in papers with joint authorship. Here I clarify my personal contribution to each work.

Uncertainty Quantification with Statistical Guarantees in End-to-End Autonomous Driving Control [86] I am the primary author on this paper. I jointly devised the concept of the framework presented in this work, and the definitions of probabilistic safety and real-time decision confidence. I programmed the framework myself and performed all the experiments. I also wrote the majority of the paper itself.

On Guaranteed Optimal Robust Explanations for NLP Models [74] In this paper, I jointly contributed to the development of the ORE concept. In addition, I defined and implemented the approach to introduce non-trivial (i.e. not simply the length of the explanation) cost functions to be optimised. I programmed and ran all of the experiments with these non-trivial cost functions, and I implemented the Neurify solver integration that allowed us to produce OREs for networks that were an order of magnitude larger than the ones possible with the Marabou solver, and allowed us to use L_1 and L_2 distance metrics (as opposed to only L_∞ with Marabou). I contributed to the paper writing.

1.3 Thesis Outline

This thesis is organised as follows. In Chapter 2 we introduce the technical background needed for the remainder of this thesis and in Chapter 3 we review the related work in the literature. Chapters 4, 5 and 6 cover the three settings A, B and C, respectively. We conclude with Chapter 7 by summarising our work, highlighting useful and interesting observations made along the way, and discussing future directions.

Chapter 2

Background

This chapter introduces the technical background needed for the remainder of this thesis. First, we discuss the basics of topics including probability, Bayesian statistics, vector embeddings and distance metrics. Next, we cover neural networks and Bayesian neural networks, then explainability, and finally neural network verification.

2.1 Basics

This section introduces the basic concepts from computer science and mathematics that are used in this thesis. Topics covered include probability (both frequentist and Bayesian), vector embeddings and distance metrics.

2.1.1 Probability Basics

Probability describes how likely something is to occur in a random experiment: an experiment whose outcome cannot be predicted until it is observed. Probability of an event A is denoted as $P(A)$ and takes a value between 0 and 1, where 0 essentially means impossibility and 1 essentially means certainty.

A sample space Ω is the set of all outcomes of a random experiment, and a random variable is a variable whose possible values are numerical outcomes of a random experiment, that is, it maps the sample space to a measurable space (e.g the real numbers $\mathbb{R}$). Discrete random variables can only have a countable number of discrete values, whilst continuous random variables can have an uncountable number. In the example below, $X : \Omega \to \mathbb{R}$ is a discrete random variable for a coin toss experiment:

$$X = \begin{cases} 1, & \text{if heads} \\ 0, & \text{if tails} \end{cases} \tag{2.1}$$

The expectation of random variable X, written as $E[X]$ is approximately equal to the mean of N random observations of X. For a discrete random variable:

$$E[X] = \sum_x x F_X(x) \tag{2.2}$$

Here, $F_X(x)$ is the the probability mass function. The probability mass function is a function (for discrete random variables) $P(x)$ that satisfies three properties: (1) $P[X = x] = P(x)$, (2) $P(x)$ is non-negative for all real x, (3) the sum of $P(x)$ over all values of x is 1.

The expectation of a continuous random variable, as $N \to \infty$, is defined as:

$$E[X] = \int_{-\infty}^{\infty} x F_X(x) dx \tag{2.3}$$

Here, $F_X(x)$ is the probability density function. The probability density function is a function (for continuous random variables) $f(x)$ that satisfies three properties: (1) that $P[a \leq x \leq b] = \int_a^b f(x)dx$, (2) that $f(x)$ is non-negative for all real x, and (3) that the integral over the reals $\int_{-\infty}^{+\infty} f(x)dx = 1$.

The variance of a random variable X is a measure of how concentrated the distribution of X is around its mean, defined as follows.

$$\text{Var}[X] = E[[X - E[X]]^2] = E[X^2] - [E[X]]^2 \tag{2.4}$$

A joint probability distribution of discrete random variables X and Y defines their simultaneous behaviour in random experiments and is defined as $F_{XY}(x, y) = P(X =$

$x, Y = y)$. Finally, if another random variable Z is dependent on X and Y, then the distribution $P(Z|X, Y)$ is the conditional probability distribution of Z with respect to X and Y.

Common Probability Distributions Some important probability distributions that are commonly found in machine learning are described here.

The Bernoulli distribution is a discrete distribution characterised by probability parameter p. Probability $P[X = 1] = p$ and $P[X = 0] = (1 - p)$, expectation $\mathrm{E}[X] = p$ and variance $\mathrm{Var}[X] = P(1 - p)$. A graph of the distribution can be found in Figure 2.1.

The normal distribution, also known as the Gaussian distribution, is a continuous distribution characterised by two parameters μ, the mean, and σ, the standard deviation. It is usually denoted as $\mathcal{N}(\mu, \sigma^2)$, where σ^2 is the variance, and its probability density function is given as:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{2.5}$$

A graph of the distribution can be found in Figure 2.2.

Bernoulli Random Variables A Bernoulli random variable is the simplest type of discrete random variable and can take only two values: 0 and 1. If an experiment with probability p resulted in a success it is a 1, else 0. A random variable X with the distribution of a Bernoulli distribution is denoted as:

$$X \sim \mathrm{Ber}(p) \tag{2.6}$$

where $\mathrm{Ber}(p)$ is the Bernoulli distribution with probability parameter p.

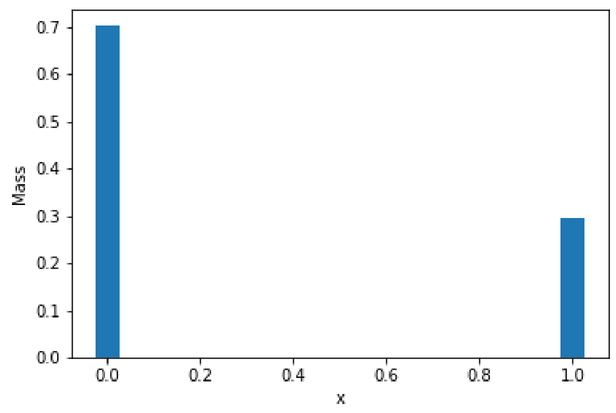


Figure 2.1: The Bernoulli distribution mass function, with parameter $p = 0.3$.

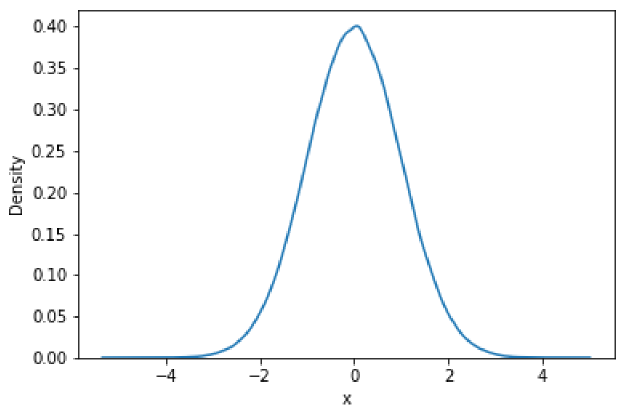


Figure 2.2: The normal distribution density function, with mean 0 and standard deviation 1.

2.1.2 Bayesian Statistics

Bayesian statistics is based on the Bayesian interpretation of probability, in which probability expresses a degree of belief in an event, rather than frequency. Initial beliefs about an event are specified in a *prior* probability, which, when presented with new data, is updated to a *posterior* probability that better represents all the evidence.

Bayesian methods are a set of machine learning tools that use Bayesian statistics, explained above, to provide a way of integrating prior beliefs about a model into the training process. Once a prior distribution (belief) of the model has been selected, the set of training data is used to compute the likelihood function and then, from these, the posterior distribution can be calculated using Bayes' rule, see Equation 2.7. In the case where a single output is needed, the posterior distribution can be used to select point estimates.

$$P(H|\mathcal{D}) = \frac{P(\mathcal{D}|H)P(H)}{P(\mathcal{D})} = \frac{P(\mathcal{D}|H)P(H)}{\int_H P(\mathcal{D}|H)P(H)dH} \tag{2.7}$$

Here, H is a hypothesis, for example neural network parameters, and $\mathcal{D}$ is some data. $P(\mathcal{D}|H)$ is the likelihood, $P(H)$ is the prior, $P(\mathcal{D})$ is the evidence and $P(\mathcal{D}) = \int_H P(\mathcal{D}|H)P(H)dH$.

2.1.3 Vector Embeddings and Distance Metrics

In machine learning, *inputs* for neural networks must be real-valued vectors in one or more dimensions, depending on the model. Additionally, *features* are the elements of an input vector. This section describes how text and image data can be represented as real vectors, and how the distance between two input vectors can be calculated for different applications.

Vector Embeddings In the case of images as inputs to neural networks, these can be treated as 2D (for grayscale) or 3D (for colour) real-valued vectors, where entries correspond to the pixel colour value at that location. Pixel values range between $0 - 255$ in raw image data; however, in most applications, the values are normalised between $0 - 1$. Additionally, 2D and 3D inputs are sometimes flattened to become one large 1D vector, depending on network architecture.

Text as input is a more difficult case, as there are no inherent numbers associated with text as there are with pixel values (colours). In this case, we use *word embeddings*, which are a learned representation for text where words that have a similar semantic meaning have a similar representation, for example based on co-occurrence of words. Each word is mapped to a single real-valued vector in a predefined space, and the representation is learned based on the usage of the words.

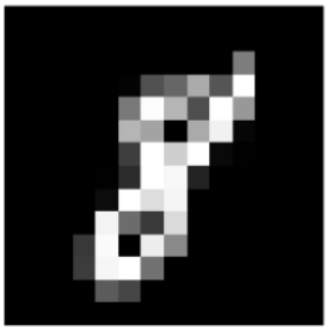
The learning process of the word embedding can be done in several ways, but the simplest is by learning as a side-effect of another learning method (for example, a neural network) on a specific natural language processing task (e.g document classification, or sentiment analysis). This requires that each word is one-hot encoded (a vector of binary variables for every possible word, where each entry indicates the presence or absence of a word), and the size of the vector space is fixed (for example, 5 dimensions). Then, the vectors are initialised with small random numbers and are learned during the same learning process that the given learning method utilises (for example, using backpropagation for neural networks).

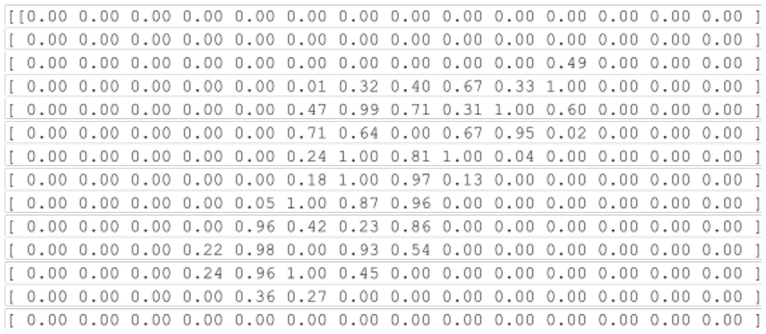
Other word embedding learning processes include Word2Vec [87], which uses a statistical method for efficiently learning a standalone word embedding from a text corpus, and GloVe [97], which extends Word2Vec by adding in classic matrix factorisation techniques.

Examples of image and text inputs, along with their equivalent representations, can be seen in Figures 2.3 and 2.4.

Distance Metrics In machine learning, there is often a need to calculate the distance between two neural network inputs. A distance function, or metric, δ should take the form $\delta : X \times X \to [0, \infty)$ and should satisfy the following axioms for all $x, y, z \in X$:

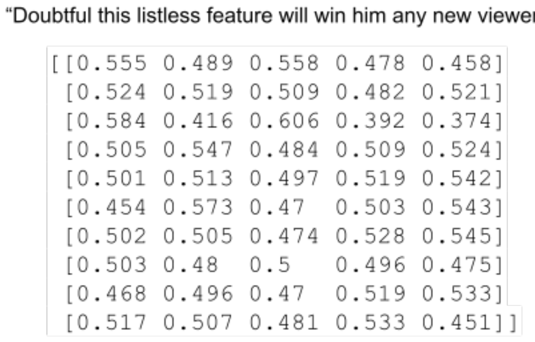
- Identity: $\delta(x, y) = 0 \iff x = y$





```
[[0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 ]
 [ 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 ]
 [ 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.49 0.00 0.00 0.00 ]
 [ 0.00 0.00 0.00 0.00 0.00 0.01 0.32 0.40 0.67 0.33 1.00 0.00 0.00 0.00 ]
 [ 0.00 0.00 0.00 0.00 0.00 0.47 0.99 0.71 0.31 1.00 0.60 0.00 0.00 0.00 ]
 [ 0.00 0.00 0.00 0.00 0.00 0.71 0.64 0.00 0.67 0.95 0.02 0.00 0.00 0.00 ]
 [ 0.00 0.00 0.00 0.00 0.00 0.24 1.00 0.81 1.00 0.04 0.00 0.00 0.00 0.00 ]
 [ 0.00 0.00 0.00 0.00 0.00 0.18 1.00 0.97 0.13 0.00 0.00 0.00 0.00 0.00 ]
 [ 0.00 0.00 0.00 0.00 0.05 1.00 0.87 0.96 0.00 0.00 0.00 0.00 0.00 0.00 ]
 [ 0.00 0.00 0.00 0.00 0.96 0.42 0.23 0.86 0.00 0.00 0.00 0.00 0.00 0.00 ]
 [ 0.00 0.00 0.00 0.22 0.98 0.00 0.93 0.54 0.00 0.00 0.00 0.00 0.00 0.00 ]
 [ 0.00 0.00 0.00 0.24 0.96 1.00 0.45 0.00 0.00 0.00 0.00 0.00 0.00 0.00 ]
 [ 0.00 0.00 0.00 0.00 0.36 0.27 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 ]
 [ 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 ]
```

Figure 2.3: An image from the MNIST dataset, scaled down to 14×14 pixels, along with the 2D real-valued vector representation.



"Doubtful this listless feature will win him any new viewers"

```
[[0.555 0.489 0.558 0.478 0.458]
 [0.524 0.519 0.509 0.482 0.521]
 [0.584 0.416 0.606 0.392 0.374]
 [0.505 0.547 0.484 0.509 0.524]
 [0.501 0.513 0.497 0.519 0.542]
 [0.454 0.573 0.47  0.503 0.543]
 [0.502 0.505 0.474 0.528 0.545]
 [0.503 0.48  0.5   0.496 0.475]
 [0.468 0.496 0.47  0.519 0.533]
 [0.517 0.507 0.481 0.533 0.451]]
```

Figure 2.4: A movie review from the SST dataset, along with its corresponding word embeddings, with one 5-dimensional embedding for each word.

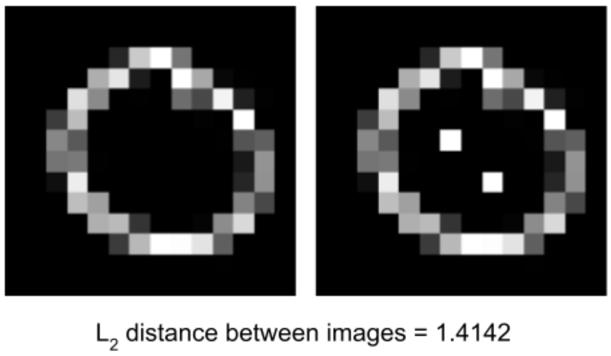


Figure 2.5: An image from the MNIST dataset, and a copy of that image with two pixels changed from a value of 0 to 1. The L_2 distance between the two images is given below.

- Non-negativity: $\delta(x, y) \geq 0$
- Symmetry: $\delta(x, y) = \delta(y, x)$
- Triangle Inequality: $\delta(x, y) \leq \delta(x, z) + \delta(z, y)$

In practice, and in this thesis, L_p norms are often used to construct distance metrics. The L_p distance between two inputs x, y is denoted as $||x - y||_p$.

The L_1 distance, also called the Manhattan distance, is the sum of the absolute difference between two vectors, formally defined in Equation 2.8. Here, n is the dimension of the input.

$$||x - y||_1 = \sum_{i=1}^{n} |x_i - y_i| \tag{2.8}$$

The L_2 distance, also known as the Euclidean distance, is the straight-line distance between two points, formally defined in Equation 2.9.

$$||x - y||_2 = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{2.9}$$

An example of the L_2 distance between two images, viewed as 1D vectors, can be found in Figure 2.5.

Finally, the L_∞ distance is the maximum deviation along any one coordinate,

Word	**Word Embedding**
‘good’	[0.477, 0.541, 0.469, 0.522, 0.584]
‘excellent’	[0.413, 0.580, 0.421, 0.558, 0.543]
‘car’	[0.51, 0.469,0.549, 0.479, 0.499]

Distance (Cosine, L_2, L_∞)	‘good’	‘excellent’	‘car’
‘good’	0.0, 0.0, 0.0	0.0038, 0.103, 0.063	0.0078, 0.148, 0.085
‘excellent’	0.0038, 0.103, 0.063	0.0, 0.0, 0.0	0.0181, 0.2153, 0.1281
‘car’	0.0078, 0.148, 0.085	0.0181, 0.2153, 0.1281	0.0, 0.0, 0.0

Table 2.1: The top table displays several words and their associated 5D self-trained embeddings, and the bottom table displays the cosine, L_2 and L_∞ distances between pairs of words.

formally defined in Equation 2.10.

$$||x - y||_\infty = \max(|x_1 - y_1|, ..., |x_n - y_n|) \tag{2.10}$$

Another type of distance, not based on an L_p norm, is the cosine distance. This metric is often used to measure similarity of texts in natural language processing applications. Cosine distance (Equation 2.11) is the angular distance between two input vectors, where magnitude information is not used. The second term in the equation is called the *cosine similarity*.

$$\text{cosine\_}\delta(x, y) = 1 - \frac{x \cdot y}{||x||_2 ||y||_2} \tag{2.11}$$

Table 2.1 shows some examples of the cosine distance between words, compared to the L_2 and L_∞ distances. We see that the cosine distance between the semantically similar words ‘good’ and ‘excellent’ is a lot smaller than between the semantically different word ‘car’.

2.2 Neural Networks

This section introduces machine learning and neural networks, including what they are, how they work and why they are useful.

Machine learning is the practice of using algorithms to parse data, detect patterns in, and learn from data, in order to make a prediction or determination about the world [23]. Learning from data in this way can be categorised into three topics: unsupervised learning, supervised learning and reinforcement learning, although this thesis focuses on supervised learning only. Supervised learning is the task of learning a decision function $f : X \to Y$ that maps from an input in the space of inputs X to an output in the space of outputs Y, based on a set of labeled training data.

Neural networks are a machine learning tool to solve the above problem and find the decision function f. Based loosely on the human brain, they consist of layers of connected units called neurons and were first proposed in the 1950s [102]. Over the past two decades, neural networks have seen an explosion in popularity and have become one of the most researched topics in machine learning.

2.2.1 Formulation

Given a set of N training examples in the form $\{(\mathbf{x_1}, y_1), ..., (\mathbf{x_N}, y_N)\}$, a neural network attempts to approximate the function $f : X \to Y$. X is the input space, Y is the output space, $\mathbf{x_i} \in X$ is a real-valued feature vector of the i^{th} training example and $y_i \in Y$ is the matching ground truth label of the i^{th} training example. If Y is finite, the task is known as a classification task, or a regression task otherwise.

2.2.2 Architecture Types

In this thesis, we focus on fully-connected and convolutional neural networks only. The diagram in Figure 2.6 shows a typical feed-forward fully-connected (FC) neural network layout. To illustrate how effective these networks are, the network shown is fairly accurate at classifying handwritten digits, using an image as input. Each white circle represents a neuron and the connections between sets of neurons are represented by arrows. Blue circles are biases, which act as constants to the activation function and are analogous to the intercept c in the straight line equation $y = mx + c$. The

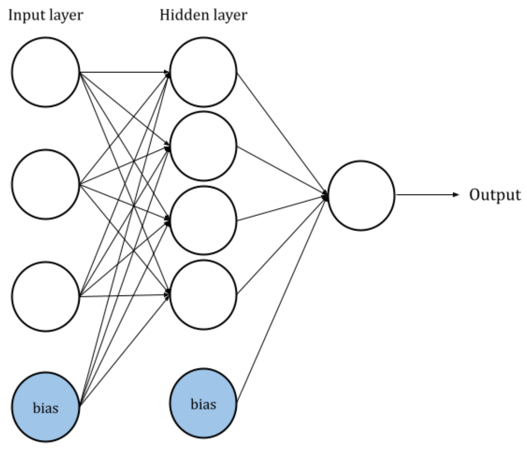


Figure 2.6: A feed-forward neural network with a single hidden layer.

neurons are arranged into layers that have no inter-connected neurons. The first layer, with three neurons in this figure, is the input layer. The second layer is a hidden layer and the third is the output layer.

Each neuron multiplies its inputs with its weight, sums the outcome and adds the bias, and finally applies a non-linear activation function before feeding forward to the next layer of neurons. If the input to the l^{th} layer of the network is:

$$x^l = (x^l_1, ..., x^l_n) \tag{2.12}$$

then the output of that layer is:

$$x^{l+1}_j = a(b^l_j + \sum_{k=1}^{n} W^l_{jk} x^l_k) \tag{2.13}$$

where b^l is the bias vector, W^l is the weight matrix and a is the activation function. The activation function is usually non-linear (although can be the identity function $a(x) = x$), which is desirable as it has been proven that networks as small as two layers, which include this non-linearity, are universal function approximators [26]. Common activation functions include the sigmoid function [26], $f(x) = \frac{1}{1-e^{-x}}$, and ReLU (rectified linear unit) [92], $f(x) = \text{if } x < 0 : 0 \text{ else } x$.

For classification tasks, the output layer typically has as many output neurons as

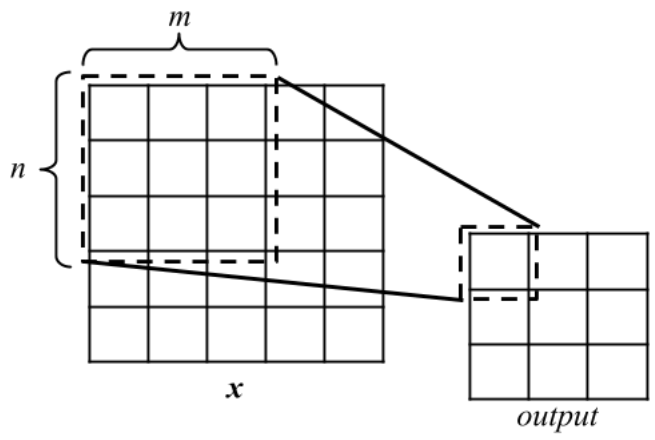


Figure 2.7: A visual example of the convolution operation.

there are classes, and the final layer's activation function is a softmax function (see Equation 2.14), where the index of the maximum value in this vector selects the most likely class.

$$\text{softmax}(\mathbf{z}) := [\frac{e^{z_1}}{\sum_{k=1}^{N} e^{z_k}}, ..., \frac{e^{z_N}}{\sum_{k=1}^{N} e^{z_k}}] \tag{2.14}$$

Convolutional neural networks (CNNs) are another commonly used network type, particularly for image-based tasks such as object detection and lane following. Their adoption in these tasks can be attributed to their reduction of training parameters and training time, and their robustness against overfitting [44]. CNNs differ from FCs in that they include convolution layers that apply the convolution operation to the input and pass the result to the next layer. The convolution operation over a layer, visualised in Figure 2.7, applies a convolution kernel of size m x n repeatedly over the input image with a step size s (called stride). The application of the kernel is as follows:

$$y(i,j) = \sum_{u=-\frac{m}{2}}^{\frac{m}{2}} \sum_{v=-\frac{n}{2}}^{\frac{n}{2}} H(u,v)x(i+u,j+v) \tag{2.15}$$

where H is the convolution kernel.

The structure of CNNs closely resembles the human visual system, and they are translation invariant, which suggests that CNNs are well suited for vision tasks [44].

2.2.3 Training

Training a neural network amounts to using an iterative optimisation algorithm to find a set of weights that best maps training data inputs to outputs. This problem is difficult as the error surface is highly multidimensional and non-convex, with plenty of local minima and flat spots. To quantify the difference between the expected outcome and the outcome produced by the neural network, a loss function is used, given below.

$$L_S = (y - \hat{y})^2 \tag{2.16}$$

Networks intended for regression tasks typically have a single, continuously valued output and a common loss function for optimisation is the squared loss, where y is the ground truth label, and $\hat{y}$ is the predicted value (see Equation 2.16). In classification problems, the cross-entropy loss is a common choice.

$$L_{CE} = -\sum_{i=1}^{c} T_i \log(S_i) \tag{2.17}$$

Equation 2.17 defines the cross-entropy loss, where c is the number of classes, T is the one-hot encoded vector for ground truth label y (for example, if there are 3 classes and $y = 2$, then $T = [0, 1, 0]$) and S is the output of the application of the softmax function on the final layer of the neural network (as in Equation 2.14).

During the learning process, the loss gradient (the derivative of a loss function with respect to the weights) of the network is used to adjust the current weights. Obtaining the loss gradient is done through backpropagation [106], while updating the weight values using that gradient is done through an optimisation algorithm.

A simple choice of optimisation algorithm is gradient descent, which corrects weights by applying the equation $W' = W - a \times$ gradient. Here, a is constant called the learning rate that controls the speed of training and gradient is the derivative of the given loss function with respect to the weights. When choosing a, there is a trade-off between non-convergence and time consumption. The update process is then repeated, usually for a given number of iterations or until a target error has been reached.

2.2.4 Inference

Inference applies knowledge from a trained neural network model to infer a result. Inference for a deterministic neural network involves running new data points through the network, with the previously learned and subsequently fixed weights, to make a prediction.

For more information on neural networks, we refer the interested reader to [44].

2.3 Bayesian Neural Networks

This section begins with a discussion of the building blocks of Bayesian neural networks: uncertainty information and the Gaussian process. Then, we formulate a Bayesian neural network, and cover multiple ways of performing posterior inference (training) on these networks, along with a discussion of different measures of uncertainty and how they can be extracted.

2.3.1 Uncertainty

In many fields, uncertainty is used to determine the dependability of a model. In Bayesian modelling, there are two main types of uncertainty that can be modelled: aleatoric uncertainty and epistemic uncertainty. Aleatoric uncertainty captures noise inherent in the observations, for example sensor noise (variations of sensor output unrelated to the input), and this uncertainty cannot be reduced even if more data was collected. Epistemic uncertainty accounts for the uncertainty in model parameters: it captures our ignorance about which model generated our data and it can be explained away given enough data (it is also known as model uncertainty).

In classification problems, the softmax probabilities are not enough to indicate whether the model is confident in its prediction, as a standard model would pass

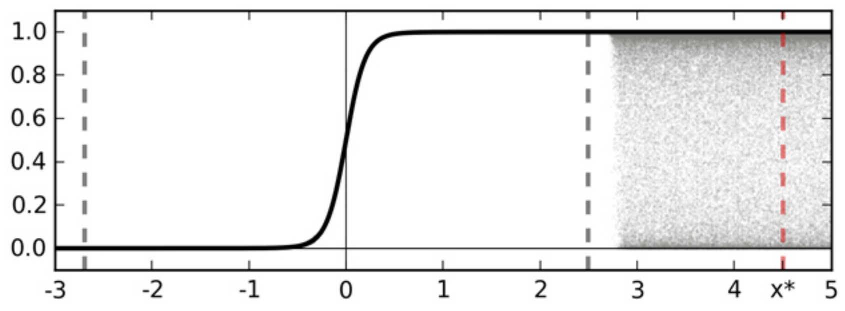


Figure 2.8: Softmax output as a function of data, taken from [37].

the predictive mean (a point estimate) through the softmax rather than the whole predictive distribution [37]. This leads to high softmax probabilities (confidence) on points far from the training data. As an example, consider the binary classification problem. Figure 2.8 shows the softmax output of the classifier as a function of the data (solid line), where the training data is between the two grey dashed lines. The point x^* is far from the training data, but without taking uncertainty into account, it is classified as a 1 with probability 1. The grey shading indicates the distribution achieved by passing the entire distribution, rather than a point estimate, through the softmax function. This gives us a more applicable result, as taking the mean of this distribution gives us a prediction of class 1, but with probability 0.5.

The above was an example of out-of-distribution test data. Other examples of sources of uncertainty include noisy data, and situations where many models explain the same dataset equally (model parameter uncertainty). Noisy data is an example of aleatoric uncertainty, whereas model parameter uncertainty is an example of model uncertainty (or epistemic uncertainty) [38], the confidence the model has in its prediction.

2.3.2 Gaussian Processes

One of the building blocks of Bayesian methods is the Gaussian process. The Gaussian process is a tool for modelling distributions over functions that can be applied to both regression and classification tasks. Similarly to neural networks, given a training set $x_1, ..., x_n$ and corresponding outputs $y_1, ..., y_n$, the function $y = f(x)$ that is likely to have generated the data should be estimated. The Gaussian process also offers properties such as robustness to overfitting, principled hyper-parameter tuning, and uncertainty estimates over the function values, which is what this research is focused on.

In practice, for regression tasks, Gaussian processes place a joint Gaussian distribution over all the function values $\mathbf{F}$, seen in Equation 2.18, where $\mathbf{X}$ and $\mathbf{Y}$ denote the inputs and outputs respectively. By modelling the distribution over the space of functions like this, the corresponding posterior can be evaluated analytically (or estimated in the case of classification).

$$\begin{aligned} \mathbf{Y}|\mathbf{F} &\sim \mathcal{N}(\mathbf{F}, \tau^{-1}\mathbf{I}_N) \\ \mathbf{F}|\mathbf{X} &\sim \mathcal{N}(\mathbf{0}, \mathbf{K}(\mathbf{X}, \mathbf{X})) \end{aligned} \tag{2.18}$$

The covariance function $\mathbf{K}(\mathbf{X}, \mathbf{X})$ must be selected to model the data. This function defines similarity between every pair of input points and represents our prior belief of what the form of the function we are modelling should be. Certain covariance functions correspond to certain non-linearities, for example the hyperbolic tangent or ReLU functions. The parameter τ is a precision hyper-parameter of the Gaussian process, and $\mathbf{I}_N$ is the identity matrix. For further details, please refer to [40].

Sometimes, the Gaussian process posterior evaluation requires inversion of a matrix as large as the dataset, which is intractable to compute (for reasonably large datasets). A more efficient method to calculate this posterior is so called variational inference, which provides an approximation of the posterior. This is done by defining a “simpler” approximating variational distribution $q_\theta(\omega)$, parameterised by some variational parameters θ, and ω is a finite set of random variables that the model is first conditioned on. Then minimising the KL (Kullback-Leibler) divergence between that and the original posterior distribution $P(\omega|\mathbf{X}, \mathbf{Y})$ (the one that could not be evaluated due to the large matrix inverse operation).

Due to the difficulty in training and scaling outlined above, we require an easier way of obtaining uncertainty information from a Bayesian model. A Bayesian neural network (BNN) is, in essence, an approximate Gaussian process, as with infinitely many weights in the BNN, we can recover a Gaussian process [40].

2.3.3 Formulation

A Bayesian neural network (BNN), as opposed to a deterministic network, allows for the extraction and quantification of model and data uncertainty. BNNs follow the Bayesian interpretation of probability, by replacing the singular valued weights and biases of a deterministic network with weight and bias probability distributions (one for each model parameter). For a test input $x \in \mathbb{R}^m$ a BNN with C output units and an unspecified number (and type) of hidden layers is denoted as $f^{\mathbf{w}}(x) = [f_1^{\mathbf{w}}(x), \dots, f_C^{\mathbf{w}}(x)]$, where $\mathbf{w}$ denotes the weight vector random variable. Given w, a weight sampled from the distribution of $\mathbf{w}$, we denote with $f^w(x)$ the corresponding deterministic neural network with weights fixed to w and with $P(f^{\mathbf{w}}(x))$ the resulting distribution of $f^{\mathbf{w}}(x)$.

2.3.4 Training

In the Bayesian setting, standard network training (i.e., how deterministic networks are trained) via optimisation is equivalent to maximum likelihood estimates (MLE) for the model parameters. However, this ignores the additional data that using a probability distribution introduces (i.e MLE uses point estimates) and overfitting (modelling the training data too closely) is common [39].

As with the Gaussian process, the correct way to train is to perform posterior inference. To exemplify this in the case of classification, consider classification with a softmax likelihood model. Let $\mathcal{D} = \{(x, y) \,|\, x \in \mathbb{R}^m,\ y \in \{1, ..., C\}\}$ be the training set. We then define the likelihood function σ_c for observing class y_c for a given input x and given weight matrix w as:

$$\sigma_c(f^w(x)) = \frac{e^{f_c^w(x)}}{\sum_{j=1}^{C} e^{f_j^w(x)}} \tag{2.19}$$

The combined vector of class likelihoods σ is defined as:

$$\sigma(f^w(x)) = [\sigma_1(f^w(x)), ..., \sigma_C(f^w(x))] \tag{2.20}$$

Then, a prior distribution is assumed over the weights, i.e. $\mathbf{w} \sim P(w)$, and the associated random variable induced by this distribution is denoted $\sigma(f^{\mathbf{w}}(x))$. Learning for the BNN amounts to computing the posterior distribution over the weights, $P(w|\mathcal{D})$, via the application of Bayes' rule, seen in Equation 2.21.

$$P(w|\mathcal{D}) = \frac{P(\mathcal{D}|w)P(w)}{P(\mathcal{D})} = \frac{P(\mathcal{D}|w)P(w)}{\int_w P(\mathcal{D}|w)P(w)dw} \tag{2.21}$$

Here, $P(\mathcal{D}|w)$ is the likelihood, which indicates how compatible the data is with the given weights. $P(w)$ is the prior probability, which is chosen based on intuition about the data.

The same problems as with Gaussian processes arise here, in that the exact posterior inference calculation is computationally infeasible, so approximation methods have been developed. Three of these methods, Monte Carlo dropout (MCD), Hamiltonian Monte Carlo (HMC) inference and mean field variational inference (VI) (mentioned briefly in the Gaussian process example above), are now described.

Monte Carlo Dropout A recent technique, called Monte Carlo Dropout (MCD), from Gal and Ghahramani [40], allows the gathering of approximate uncertainty information from neural networks without changing the architecture (given that some stochastic regularisation technique such as dropout has been used). To summarise, MCD finds an approximating Gaussian distribution $q(w) \approx P(w|\mathcal{D})$ that takes the form of the product between Bernoulli random variables and the corresponding weights. Hence, sampling from $q(w)$ reduces to sampling Bernoulli variables.

Dropout is a regularisation technique that sets 1-p proportion of the dropout layers' input to be 0, where p is the dropout probability [51]. The dropped weights are often scaled by $1/p$ to maintain constant output magnitude.

It has been shown that a network that uses dropout is an approximation to a Gaussian process [40]. Equation 2.22 shows how a prediction can be performed with a Gaussian process, where $\mathbf{f}$ is the space of functions, $\mathbf{X}$ is the training data and $\mathbf{Y}$ is the training outputs. The expectation of $\mathbf{y^*}$ for input $\mathbf{x^*}$ is called the predictive mean of the model and the variance is the predictive uncertainty.

$$P(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) = \int P(\mathbf{y}^*|\mathbf{f}^*)P(\mathbf{f}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y})\mathrm{d}\mathbf{f}^* \tag{2.22}$$

For a regression network, Equations 2.23 and 2.24 are respectively used to obtain an approximation of the predictive mean and variance of the Gaussian process that the network is an approximation of.

$$\mathbb{E}(\mathbf{y}^*) \approx \frac{1}{K}\sum_{k=1}^{K}\widehat{\mathbf{y}}_k^*(\mathbf{x}^*) \tag{2.23}$$

$$\mathrm{Var}(\mathbf{y}^*) \approx \tau^{-1}\mathbf{I}_D + \frac{1}{K}\sum_{k=1}^{K}\widehat{\mathbf{y}}_k^*(\mathbf{x}^*)^T\widehat{\mathbf{y}}_k^*(\mathbf{x}^*) - \mathbb{E}(\mathbf{y}^*)^T\mathbb{E}(\mathbf{y}^*) \tag{2.24}$$

The set $\{\widehat{\mathbf{y}}_k^*(\mathbf{x}^*)\}$ of size K comprises the results from K stochastic forward passes through the network. It is important that the non-determinism from dropout is retained at prediction time to ensure different units will be dropped per pass through. Relating back to the Gaussian process, these are empirical samples from the approximate predictive distribution seen in Equation 2.22. τ relates to the precision of the Gaussian process model, and is used in the calculation of the predictive variance. τ can be calculated as seen in Equation 2.25, where l is a user-defined length scale, p is the probability of units *not* being dropped, N is the number of training samples and λ is the multiplier used in the L_2 regularisation of the network.

$$\tau = \frac{l^2 p}{2N\lambda} \tag{2.25}$$

A small length-scale (corresponding to high frequency data) with high τ (corresponding to small observation noise) will lead to a small weight decay, which might mean the model fits the data well but generalises badly. Conversely, a large length-scale and low τ will lead to strong regularisation. There is a trade-off between length-scale and model precision. In practice, the model precision τ is often found by grid searching over the weight decay λ to minimise validation error, choosing a length-scale that correctly describes the data, and then putting the values into Equation 2.25. It can also be found by grid searching over τ values directly.

Hamiltonian Monte Carlo Hamiltonian Monte Carlo (HMC) is a Markov chain Monte Carlo (MCMC) method that proceeds by defining a Hamiltonian function $H(q,p) = U(q) + K(p)$ [13]. In the context of Bayesian neural networks, q denotes the weights of the network, p is an artificially introduced momentum variable, $U(q)$ is the potential energy which, in this setting, is equal to minus the log probability density of the posterior distribution (i.e $-log(P(w|\mathcal{D})))$, and $K(p)$ is the kinetic energy which is defined as $p^T M^{-1} p/2$ with T being the number of samples, and M being a symmetric, positive-definite covariance matrix. Typically, M is diagonal and a scalar multiple of the identity matrix.

The algorithm results in a set of samples of the posterior distribution $P(w|\mathcal{D})$, and the algorithm proceeds as follows:

1. Pick a starting point q_0.
2. Draw p_0 from a zero mean Gaussian distribution with covariance matrix M.
3. Solve the Hamiltonian system of equations for a given amount of time and record the end point as p_1:
   - $\frac{dq}{dt} = \frac{dH}{dp}$
   - $\frac{dp}{dt} = \frac{-dH}{dq}$
4. Repeat steps 1-3, $T_0 + T$ times, yielding $q_1, ..., q_{T_0}, ..., q_T$.
5. The samples from the posterior are $q_{T_0}, ..., q_T$. The initial samples up to q_{T_0} are disregarded to mitigate the effect of a poor starting point.

Differently from the two other methods discussed here, HMC does not make any assumptions on the form of the posterior distribution, and is asymptotically correct. For additional details, please see [13].

Mean Field Variational Inference Mean field variational inference (VI) proceeds by finding a Gaussian approximating distribution $q(w) \approx P(w|\mathcal{D})$ in a trade-off between approximation accuracy and scalability. The core idea is that $q(w)$ depends on some hyper-parameters that are then iteratively optimized by minimizing a divergence measure (the Kullback-Leibler divergence [72]) between $q(w)$ and $P(w|\mathcal{D})$. Samples can then be efficiently extracted from $q(w)$. The optimisation problem is given in Equation 2.26.

$$log\, P(\mathcal{D}) = KL(q(w)\, ||\, P(w|\mathcal{D})) + \mathbb{E}[log\, P(w, \mathcal{D}) - log\, q(w)] \tag{2.26}$$

Since $log\, P(\mathcal{D})$ does not depend on w, it can be considered constant, and therefore minimising KL divergence is equivalent to maximising the second term: the evidence lower bound (ELBO).

Comparison of Inference Methods HMC is an exact inference method that is asymptotically correct, compared to both VI and MCD which are approximate inference methods. The trade-off between exact and approximate inference methods is between tighter guarantees (exact) and scalability (approximate). Additionally, HMC does not make any assumptions about the shape of the posterior distribution, whereas for VI and MCD picking a bad initial shape can, in the worst case, lead to a bad approximation of the posterior. Picture trying to put a squashable square into a circular space of a similar size: it will fit but the corners of the square will be squashed into the shape of the circle.

2.3.5 Extracting Uncertainty Information

For classification tasks, there are several methods of obtaining uncertainty information. As previously mentioned, softmax probabilities are a poor indicator as they are the result of a single *deterministic* pass of a point estimate through the network, which can lead to high confidence on points far from the training data [38]. The three approaches used to summarise classification uncertainty are *variation ratios* [35], *predictive entropy* [108] and *mutual information* [108].

Variation Ratio Variation ratio is a measure of dispersion; its value is high when classes are more equally likely and low when there is a clear winner. Variation ratio, as with predictive uncertainty in regression tasks, requires T stochastic forward passes through the network for a test input $\mathbf{x}$. A set of T labels y_t is collected, where y_t is the class with the highest softmax output of that pass through, where $t \in T$. The mode of the distribution c^* and the number of times it was sampled, f_x, can then be used to obtain the variation ratio. These calculations can be seen in Equations 2.27, 2.28 and 2.29.

$$c^* = \underset{c=1,\ldots,C}{\arg\max} \sum_t \mathbb{1}[y_t = c] \tag{2.27}$$

$$f_x = \sum_t \mathbb{1}[y_t = c^*] \tag{2.28}$$

$$\text{variation-ratio}[x] := 1 - \frac{f_x}{T} \tag{2.29}$$

Predictive Entropy Predictive entropy captures the average amount of information present in the predictive distribution, $\mathbb{H}[y|\mathbf{x}, \mathcal{D}_{\text{train}}]$. In our setting, the predictive entropy can be approximated by collecting the softmax probability vectors over T stochastic forward passes, and for each class averaging the softmax probability and multiplying it by the log of that average. This can be seen in Equations 2.30 and

2.31, where f^w is the network with model parameters w.

$$\text{softmax}(f^w(x)) := \quad [P(y = 1|\mathbf{x}, \widehat{w}_t), ..., P(y = C|\mathbf{x}, \widehat{w}_t)] \tag{2.30}$$

$$\widetilde{\mathbb{H}}[y|\mathbf{x}, \mathcal{D}_{\text{train}}] := -\sum_c (\frac{1}{T}\sum_t P(y = c|\mathbf{x}, \widehat{w}_t)) \cdot \log(\frac{1}{T}\sum_t P(y = c|\mathbf{x}, \widehat{w}_t)) \tag{2.31}$$

Mutual Information The final measure is mutual information, $\mathbb{I}[y, w|\mathbf{x}, \mathcal{D}_{\text{train}}]$. Test points that maximise mutual information are points on which the model is uncertain on average, but there are model parameters that erroneously produce high confidence predictions. Mutual information is calculated similarly to predictive entropy, but with an extra term as seen in Equation 2.32.

$$\begin{aligned}\widetilde{\mathbb{I}}[y, w|\mathbf{x}, \mathcal{D}_{\text{train}}] := (&-\sum_c (\frac{1}{T}\sum_t P(y = c|\mathbf{x}, \widehat{w}_t)) \cdot log(\frac{1}{T}\sum_t P(y = c|\mathbf{x}, \widehat{w}_t))) \\ &+ \frac{1}{T}\sum_{c,t} P(y = c|\mathbf{x}, \widehat{w}_t) log P(y = c|\mathbf{x}, \widehat{w}_t)\end{aligned} \tag{2.32}$$

Variation ratios and predictive entropy are both measures of predictive uncertainty, whereas mutual information is a measure of the model's confidence in its output. Further information on this can be found in [38]. Having multiple measures of uncertainty is arguably more powerful than the sole measure available for regression tasks, as it allows for different types of uncertainty to be captured and gives us more information about the performance of the model.

2.4 Explainability

This section covers explainability and interpretability of neural networks. Explainability and interpretability are subtly different terms, where interpretability means the extent to which one can predict what will happen given an input, and explainability comprises uncovering human-interpretable reasons for why the network produces given predictions. As the definitions are similar, the terms are often used interchangeably in the literature and will be covered as part of the same section in this background chapter.

Explainability methods can be local (per input), global (per model) or a mixture of both. In this thesis, we focus on local explainability. Local explainability can be roughly categorised into one of two methods: *feature score* methods and *subset* methods. The following sections define and explain these two groups of methods, and include algorithms for the local explainability methods used in this thesis.

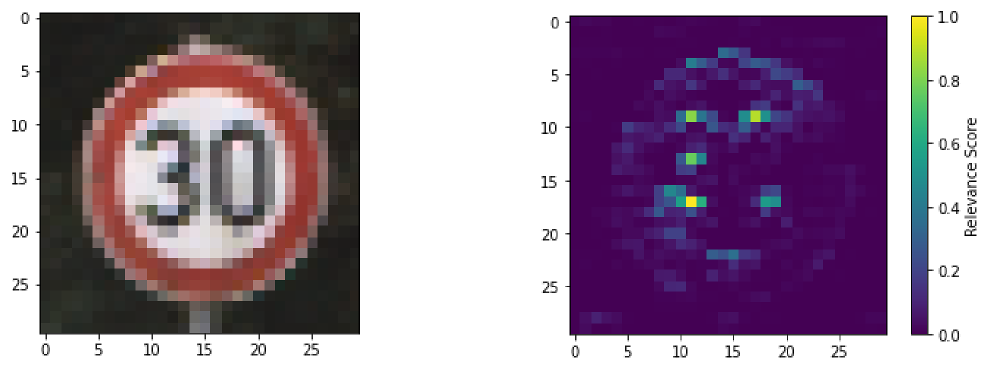
2.4.1 Local Explainability

Local explainability aims to understand the impact of input features on an individual prediction. Local explanations generally comprise a subset of the input features that contribute to the prediction (subset methods), or a score assigned to each input feature where a higher score implies a stronger contribution to the prediction (feature score methods).

Local explanations can be computed via a number of different ways. For feature score methods, this includes gradient-based methods (Layerwise Relevance Propagation [5], integrated gradients [117]), local approximating model-based methods (LIME [100]), and game theoretic methods (SHAP [81]). For subset methods, this includes local approximating model-based method Anchors [101]. Additional techniques for computing local explanations are explored in Chapter 3.

Layer-wise Relevance Propagation Layer-wise relevance propagation (LRP) [5] is a gradient-based technique for determining which features (pixels, words, etc) in a particular input vector contribute to the prediction by defining a relevance measure over the input vector such that the network output can be expressed as the sum of the values of the input vector. LRP uses the networks weights and activations in the forward pass to propagate the output back through the network.

To determine the relevant input features for a given output class c, the relevance score of class c's output neuron is set equal to its activation, and the relevance scores of all other neurons in the output are set to zero. Following this, Equation 2.33 is



(a) The GTSRB input image. (b) The resulting relevance map using LRP.

Figure 2.9: An image from the GTSRB dataset and resulting relevance map. The input is correctly classified and the relevance map is for the true class.

applied to each layer going backwards through the network.

$$R_j = \sum_k \frac{a_j w_{jk}}{\sum_{0,j} a_j w_{jk}} R_k \tag{2.33}$$

In the above, consecutive layers are denoted by j and k, a_j denotes the activation of neuron j, and w_{jk} is the weight between neurons j and k. Variations of this propagation rule exist, including LRP-γ which disproportionately favours positive evidence over negative evidence, and LRP-ϵ which helps remove noise from the relevance map by absorbing a small amount of contradicting evidence.

The output of the LRP algorithm is a mapping of input features to relevance scores for the predicted class, where a feature i with score $R_i > 0$ contributes positively to the prediction. Figure 2.9 shows the relevance map for an input from the GTSRB dataset, scaled down to 30×30 pixels. The input is correctly classified and so the relevance map is for the true class.

Integrated Gradients Integrated gradients [117] is another gradient-based local explanation method that is designed to adhere to two key axioms: sensitivity and implementation invariance. *Sensitivity* is satisfied if, for every input and baseline that differ in one feature and have different predictions (where a baseline is a neutral input, e.g the black image for object recognition), the differing feature should be given

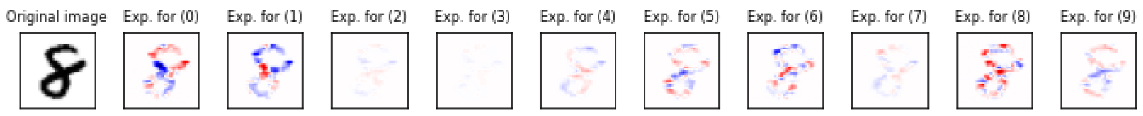


Figure 2.10: The IG explanation for each of the classes of the MNIST dataset, where red indicates a positive contribution and blue a negative.

a non-zero relevance score. *Implementation invariance* is satisfied if the relevance scores are identical for two functionally equivalent networks (networks are functionally equivalent if their outputs are equal for all inputs).

Given a network $f : \mathbb{R}^m \to [0,1]$, an input $x \in \mathbb{R}^m$ and a baseline input $x' \in \mathbb{R}^m$, the gradients of the straight line path between x and x' are calculated, and integrated gradients for each input dimension $i \in [1..m]$ are defined as the path integral of these gradients, as seen in Equation 2.34.

$$\text{IntegratedGradients}_i(x) = (x_i - x'_i) \times \int_{\alpha=0}^{1} \frac{\delta f(x' + \alpha \times (x - x'))}{\delta x_i} d\alpha \tag{2.34}$$

The output of the integrated gradients algorithm is similar to the output of LRP: a mapping of input features to relevance scores but for each class rather than only the predicted class. Positive scores contribute positively to the given class, and negative scores contribute negatively. An example of an IG explanation can be seen in Figure 2.10, which demonstrates the explanations over each class for a (correctly classified) handwritten eight from the MNIST dataset. In this example, red indicates positive contribution and blue indicates negative. Interestingly, LRP also satisfies the first axiom of *sensitivity* but fails on *implementation invariance*, as LRP replaces gradients with discrete gradients where the chain rule does not hold in general. For an explanation of why these two axioms are important, see Chapter 3.

LIME and Anchors LIME generates local explanations by assuming local linearity in a small area around an input instance, defined by some similarity measure [100]. An interpretable model is then trained on this area to produce a local approximation, called a surrogate model. The input is then explained through the weights of this surrogate model.

After assuming linearity in an area defined by a similarity metric around a given input, artificial training data is generated by perturbing the original input a number of times. The class of each perturbation is determined by using the original model, and then each data point is weighted by its distance from the original input. An interpretable model (for example, linear or logistic regression, decision trees, naive Bayes etc) is then trained on this synthetic dataset, to become the surrogate model. Equation 2.35 expresses how a surrogate model for input x is generated. Surrogate model g is chosen to minimise the loss $\mathcal{L}$, which measures the distance between g's prediction and the original model f's prediction. The model complexity Ω of g should also be minimised (e.g fewer features are preferred). d_x is the proximity measure that defines how large an area around x should be considered.

$$\text{surrogateModel}(x) = \underset{g \in G}{\arg\min}\, \mathcal{L}(f, g, d_x) + \Omega(g) \tag{2.35}$$

The surrogate model g must be a good approximation for the original model in the local area of the given input (called good local fidelity), but it does not have to be a good global approximation. The prediction of the given input is then explained by interpreting the surrogate model (such interpretable models as decision trees have well-known methods of extracting explanations, see [90] for further details on interpretable models).

The output of the LIME method is the same as the output for LRP: a mapping of each input feature to a score denoting how important that feature is to the prediction.

Anchors, the successor to LIME by the same authors [101], generates local explanations by finding decision rules that do not change the prediction if other feature values (those that are not considered by the rule) are varied. In this case, the decision rule "anchors" the prediction, and the resulting explanations are expressed using these rules, named *anchors*. Formally, the criteria for a decision rule A to be an anchor is defined in Equation 2.36.

$$\mathbb{E}_{\mathcal{D}_x(z|A)}[\mathbb{1}_{f(x)=f(z)}] \geq \tau, A(x) = 1 \tag{2.36}$$

In the above, decision rule A is a set of predicates such that $A(x) = 1$ when all feature predicates are true, $\mathcal{D}_x(.|A)$ is the distribution of neighbours of x (by some distance metric) that match A, and τ is a precision threshold (only rules with a local fidelity above τ are considered valid). This ensures that at least a fraction τ of x's neighbours are predicted the same class as x where the same rule A applies.

The output of anchors is a set of decision rules, *anchors*, for a given input.

SHAP Shapley Additive Explanations (SHAP) is a local explanation method based on Shapley Values: a game-theoretic approach to explaining model predictions [81]. Shapley values assumes that each feature value (or combinations of feature values) of an input instance x is a player in a game where the prediction is the payout. Players can form coalitions and the goal is to calculate the average contribution of a feature to the predictions of different coalitions compared to the average prediction across all instances.

As the number of features in the input increases, the number of coalitions scales exponentially, which becomes computationally intractable; this is where the SHAP method is relevant. SHAP's goal is to explain the prediction of input instance x by computing the contribution of each feature to the prediction, using Shapley value explanations represented additively as a linear model (similarly to the surrogate models seen in LIME). Formally, SHAP defines an explanation as in Equation 2.37.

$$g(z') = \phi_0 + \sum_{j=1}^{M} \phi_j z'_j \tag{2.37}$$

Here, g is the explanation model and $z' \in \{0, 1\}^M$ is the simplified feature vector, where an entry of 1 implies the corresponding feature value is present, and an entry of 0 implies that it is absent. $\phi_j \in \mathbb{R}$ is the Shapley value for feature j.

The SHAP paper proposes further improvements: KernelSHAP (for a broad class of models) and TreeSHAP (for tree-based models only, more details on TreeSHAP can be found in [81], though it will not be discussed in this thesis). KernelSHAP is an algorithm to estimate, for an input instance x, the Shapley value for each feature, that is, their contribution to the prediction.

Instead of iterating all possible simplified feature vectors, first some vectors are sampled: $z'_k \in \{0,1\}^M, k \in \{1, ..., K\}$. Then, a prediction for each z'_k is determined by converting z'_k back to the original feature space and then applying the original model f. In the conversion function, an entry of 1 in z'_k returns the value of the corresponding feature in x, and an entry of 0 returns either (1) a random value taken from another instance sampled from the data in the case of tabular data, or (2) returns a "baseline" value, e.g the average pixel value over the whole image, for image data. Similarly to LIME, the sampled instance vectors are assigned a weight, but instead of weighting by distance to the original input instance, samples are weighted according to the SHAP kernel function π, seen in Equation 2.38. M is the maximum coalition size, and $|z'|$ is the number of 1s (present features) in z'.

$$\pi_x(z') = \frac{(M-1)}{\binom{M}{|z'|}|z'|(M-|z'|)} \tag{2.38}$$

Finally, the linear model can be built as in Equation 2.37, and the coefficients ϕ_j (the Shapley values) can be learnt by optimising the loss function in Equation 2.39.

$$L(f, g, \pi_x) = \sum_{z' \in Z} \left[f(\text{conversionFunction}_x(z')) - g(z')\right]^2 \pi_x(z') \tag{2.39}$$

Figure 2.11 shows the SHAP explanation for each of the MNIST dataset classes for a given input; in this case a five that is correctly classified by the network.

2.4.2 Comparing Local Explanation Methods

Figure 2.12 shows the explanation from each of the above methods, for the true class of a handwritten digit from the MNIST dataset. The network used for this example was

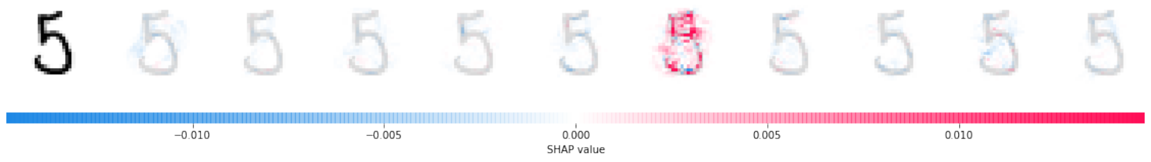
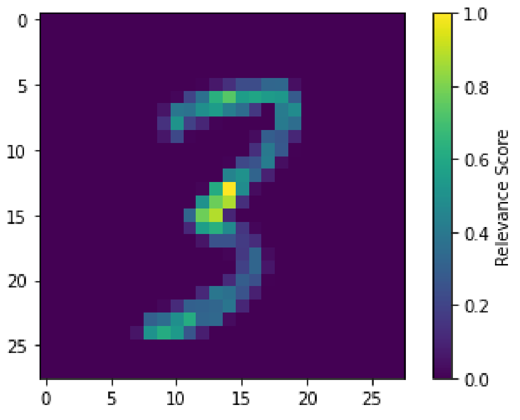


Figure 2.11: The SHAP explanation for each of the classes of the MNIST dataset, for a correctly classified "five" handwritten digit (the leftmost image).

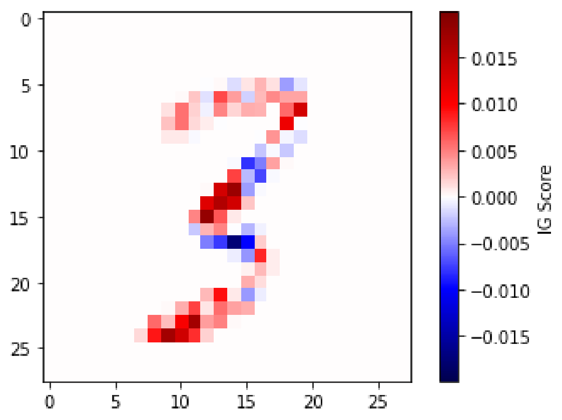
a fully-connected network with three layers of sizes 128, 64 and 10. The activations were ReLU functions except for the final layer which used the softmax function, and the loss was sparse categorical crossentropy. The model reached 97% accuracy on the test set. Note that the LIME explanation highlights whether a (super-)pixel positively contributes to the explanation or not; it is not a score. The segmentation function for LIME (usually used to determine super-pixels in large images) was forced to consider single pixels rather than groups, to align more closely with the other explanation methods.

At a glance, the explanations in the figure show us that the two gradient-based methods, LRP, which uses discrete gradients, and IG, which uses the gradient operator, only give scores to the pixels with values greater than the minimum (or different from the baseline; in the case of IG, however, the baseline in these examples is the minimum pixel value, black). That is, they do not consider negative space and therefore do not consider that the *absence* of values in given locations might contribute to the prediction. In contrast, both LIME and SHAP consider the whole input image, and a pixel with a baseline value that has been given a non-zero attribution indicates that the absence of a non-baseline value at that location *does* contribute to the prediction.

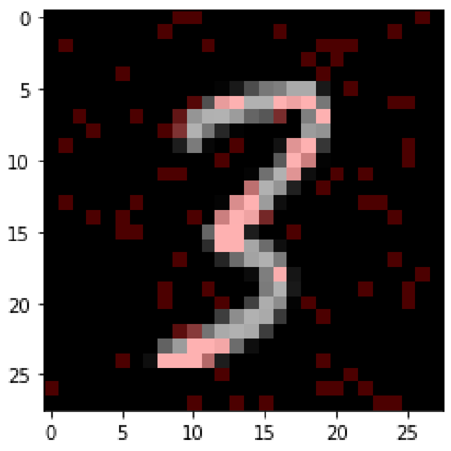
While all four methods work well for explaining models, each has its benefits and drawbacks. SHAP provides statistical guarantees for the accuracy and consistency of explanations; however, the implementation of SHAP for neural networks is very slow. In contrast, LIME may sometimes fail to approximate the behaviour of the original model correctly, so the generated explanations cannot always be trusted, but it has a fast and easily adaptable implementation. Both SHAP and LIME consider negative



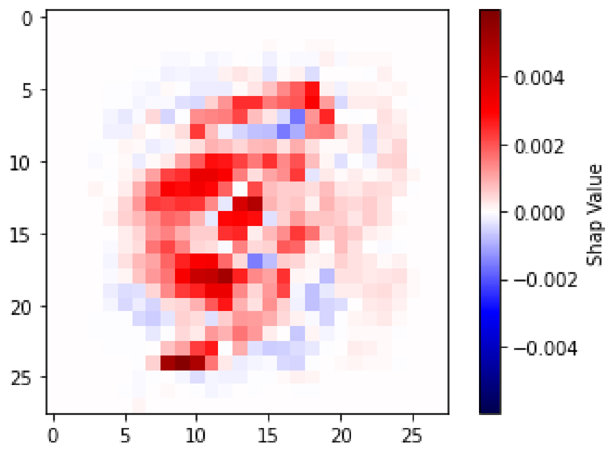
(a) LRP explanation.



(b) IG explanation where the background was black.



(c) LIME explanation.



(d) SHAP explanation.

Figure 2.12: The explanations generated by four different methods on the same input and network.

space, as demonstrated above. The two gradient-based methods, IG and SHAP, are efficient, as computing gradients of differentiable models is well supported in most machine learning frameworks, but they *require* that that model is differentiable, whereas SHAP makes no such assumptions. Additionally, IG requires a well calibrated baseline value to make sensible feature attributions. For example, if a black image is chosen as a baseline, IG will not give importance to any black pixel in the input, so a very dark grey background may be more appropriate.

2.5 Neural Network Verification

This section covers verification techniques for neural networks. The aim of verification is to provide formal guarantees that a neural network conforms to a set of desirable properties, often called correctness properties.

The most common correctness properties include robustness, safety and consistency. Robustness means that neural networks should not change their predictions when faced with adversarial examples, that is, imperceptible changes to the input that would not change the mind of a human decision maker. Safety is a wider class of correctness properties that say that the neural network should never enter a bad state. For example, in a neural network for autonomous driving, if there is a wall in front of the car, the network must always apply the brakes. Not applying the brakes in this situation would be a bad state. Finally, the correctness property of consistency states that the world view of the neural network must remain consistent. For example, in the autonomous driving scenario, if an object recognition network says there is a red Ford Fusion on the left of the autonomous car in one frame, it must not then say that the same red Ford Fusion is on the right of the vehicle in the next frame.

This thesis focuses on the *robustness* property, and the following sections cover adversarial examples, local and global robustness definitions, and different verification methods (some that verify additional correctness properties to just robustness).

2.5.1 Adversarial Examples

An adversarial example (also called an adversarial perturbation), x', is a small change to input x such that network f classifies x and x' differently, i.e., $f(x) \neq f(x')$, but a human would still give them the same classification $f(x) = f(x')$ (i.e., $||x - x'||_L$ for a given L-norm is imperceptible to the human decision maker). This "small change" refers to the semantic closeness of the inputs, and can be measured by any distance metric as in Section 2.1.3.

Generating adversarial examples is disconcertingly simple. An early, common and

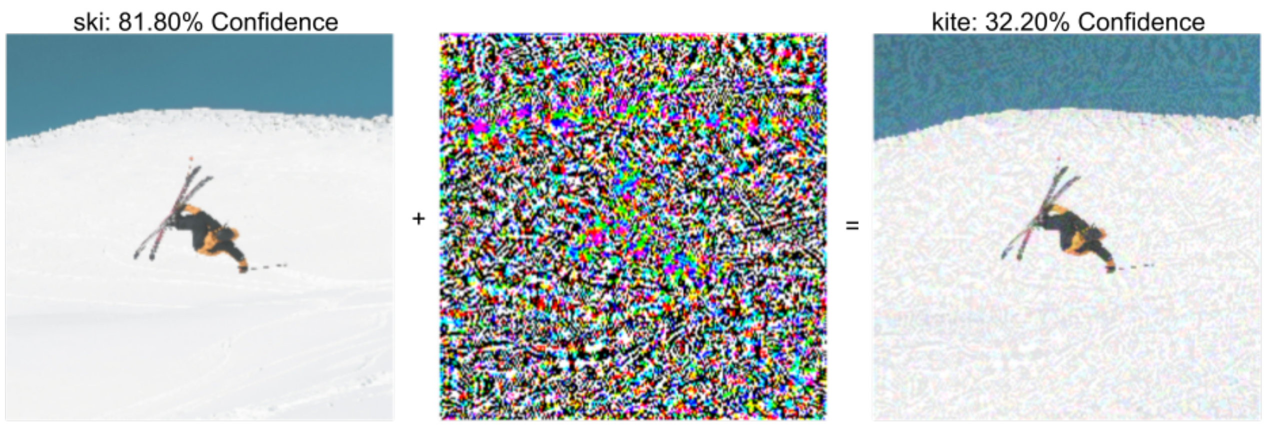


Figure 2.13: An input to the MobileNetV2 network, with the true ImageNet class 'ski', an attack generated by the FGSM with $\epsilon = 0.1$ and the resulting image and prediction.

simple method for adversarial example generation is the Fast Gradient Sign Method (FGSM) [45], as used in Chapter 4 of this thesis. FGSM is a white-box attack, meaning the attacker has full access to the underlying network (the weights, biases, etc, are visible). An adversarial example x' can be expressed as:

$$x' = x + \epsilon \times \text{sign}(\Delta_x \mathcal{L}(\theta, x, y)) \tag{2.40}$$

Here, y is the label assigned to original input x, ϵ is a multiplier to ensure the perturbations remain small, θ represents the model parameters and $\mathcal{L}$ is the loss function of the network.

Intuitively, the gradients of the loss function are taken with respect to the input image and then a new image is created by moving in the direction of these gradients (the magnitudes of the gradients are discarded). The objective here is to create an adversarial example that maximises the loss (and therefore increases model error). Since the model is fixed, the only way to maximise the loss is to change the data.

An example of the FGSM in use can be seen in Figure 2.13, which shows how the MobileNetV2 with ImageNet classes [107] changes its prediction based on an attack generated by the FGSM with $\epsilon = 0.1$.

2.5.2 Local Robustness

Local robustness (also called pointwise robustness) can be defined for both deterministic and Bayesian neural networks, as seen below.

Deterministic Local Robustness Deterministic local robustness [56] is defined with respect to an input $x \in \mathbb{R}^N$, and the neighbourhood around it in N-dimensional space. Input x is ϵ-robust if:

$$\forall x' : ||x - x'||_p \leq \epsilon \implies f(x) = f(x') \tag{2.41}$$

That is, x is locally robust if all the points within an ϵ-ball of x are given the same classification. δ takes the form of a distance function (see Section 2.1.3), for example the L_∞ distance.

Probabilistic Local Robustness Probabilistic local robustness [105] applies to Bayesian neural networks and is also defined with respect to an input x and a bounded neighbourhood of x in N-dimensional space, $T \subseteq \mathbb{R}^N$. To compute the probability for BNN $f^{\mathbf{w}}$, we assume that T is a bounded neighbourhood around x, typically an L_p ϵ-ball. The probability is then computed as:

$$\begin{aligned} &p = P(\phi(f^{\mathbf{W}})|\mathcal{D}) \text{ where} \\ &\phi(f^{\mathbf{W}}) = \exists x' \in T \text{ s.t.} ||\sigma(f^{\mathbf{W}}(x)) - \sigma(f^{\mathbf{W}}(x'))||_p > \epsilon \end{aligned} \tag{2.42}$$

Intuitively, we seek to compute the probability that there exists an input $x' \in T$ such that the output σ of the softmax layer for x' differs by more than a given threshold ϵ from the softmax output for the original input x. We say that $f^{\mathbf{W}}$ is robust with probability at least $1 - \eta$ (for $0 \leq \eta \leq 1$), with respect to set T and perturbation ϵ if and only if $p \leq \eta$.

2.5.3 Global Robustness

The expectation of local robustness over the dataset, weighted by the input distribution, gives a measure of global robustness, which is a model property. Since this notion is computable from local robustness, it will not be discussed further.

2.5.4 Verification Methods

This section covers techniques for verifying correctness properties of neural networks.

MIP Formulation In its simplest form, neural network verification can be expressed as a mixed integer programming formulation, for ReLU networks only [78]. An integer program is an optimisation problem in which some (or all) of the variables are integers, and a mixed-integer program is an integer program in which not all of the variables are discrete. The standard form of a mixed-integer program is:

$$\begin{array}{ll} \text{maximise} & \mathbf{C_1 x} + \mathbf{C_2 y} \\ \text{subject to} & \mathbf{A_1 x} + \mathbf{A_2 y} = \mathbf{B}, \\ & \mathbf{x} \geq 0, \\ & \mathbf{y} \geq 0, \\ \text{and} & \mathbf{x} \in \mathbb{R}^{n_1}; \mathbf{y} \in \mathbb{Z}^{n_2} \end{array} \tag{2.43}$$

Here, $\mathbf{A_1}$ and $\mathbf{A_2}$ are $m \times n_1$ and $m \times n_2$ integer matrices respectively. $\mathbf{C_1}$ and $\mathbf{C_2}$ are $1 \times n_1$ and $1 \times n_2$ rational matrices respectively, and $\mathbf{B}$ is an $m \times 1$ integer matrix. $\mathbf{x}$ is the real number solution vector and $\mathbf{y}$ is the integer solution vector. If integer restrictions are ignored, this problem becomes a linear programming problem.

In such a formulation, let f be the ReLU neural network with weights $\mathbf{W}$, biases $\mathbf{b}$ and layers L, and $2 \leq i \leq k$ with k the number of layers in f. $\mathbf{x^{i-1}}$ are the inputs to layer L^i and $\mathbf{x^i}$ are the outputs of layer L^i. $\beta^{\mathbf{i}}$ is a vector of binary variables and M is a "sufficiently large" constant. Using this, the set of linear constraints $\mathbf{C_i}$ encoding a layer i is defined in Equation 2.44.

$$\begin{aligned}\mathbf{C_i} = \{ & x_j^i \geq \mathbf{W_j^i x^{i-1}} + b_j^i, \\ & x_j^i \leq \mathbf{W_j^i x^{i-1}} + b_j^i + M\beta_j^i, \\ & x_j^i \geq 0, \\ & x_j^i \leq M(1-\beta_j^i) | j = 1..|L^i| \}\end{aligned} \tag{2.44}$$

Then, the set of linear constraints encoding the whole network is $\mathbf{C} = \cup_{i=2}^{k} \mathbf{C_i}$. Set $\mathbf{C}$ encapsulates all of the constraints found in the standard form of a mixed-integer program (Equation 2.43). The linear program encoding the reachability of a linearly definable input set $\mathbf{I} \subseteq \mathbb{R}^m$ to a linearly definable output set $\mathbf{O} \subseteq \mathbb{R}^n$ through the network f is given by constraints $\mathbf{C}_{\text{reach}} = \mathbf{C}_{\text{in}} \cup \mathbf{C} \cup \mathbf{C}_{\text{out}}$. $\mathbf{C}_{\text{in}}$ ($\mathbf{C}_{\text{out}}$) is a constraint set for $\mathbf{I}$ ($\mathbf{O}$), defined using the same variables in the encoding of the second (last) layer of the network. In classification problems, the final softmax activation function can be disregarded as it is a smooth approximation to the argmax function [44], and therefore does not change the output of the previous layer in a way that could change the prediction. Unfortunately, although simple, this method does not scale beyond very small networks.

ReluPlex and Marabou ReluPlex [63], and its successor Marabou [61], are SMT (satisfiability modulo theories) based solvers for the theory of linear real arithmetic $\mathcal{T}_{\mathbb{R}}$ with additional ReLU constraints. They encode the neural network as an SMT instance and solve it with a modified version of the Simplex algorithm to handle ReLU non-linearities, a decision procedure for determining the $\mathcal{T}_{\mathbb{R}}-$satisfiability of conjunctions of linear formulas, with the additional ReLU constraint.

The propositional satisfiability problem (SAT) is the problem of determining whether a set of sentences, written in propositional logic, is satisfiable. For example, let a set of sentences $\Delta = \{p \vee q, r \wedge q, \neg p\}$, then Δ is satisfiable, with only one assignment: $p = 0, q = 1, r = 1$.

SMT, satisfiable modulo theories, is the problem of determining whether a given formula ϕ, written in first-order (FO) logic, is satisfiable under a given theory $\mathcal{T}$ (or combination of theories). A theory is a pair $\mathcal{T} = (\Sigma, I)$, where Σ is a signature and I

is a set of Σ-interpretations, or models. A signature Σ consists of a set of predicate, constant and function symbols that are allowed in a given theory. Given a theory $\mathcal{T}$, a formula ϕ (where ϕ is expressed only using elements of Σ) is satisfiable modulo $\mathcal{T}$, or $\mathcal{T}$-satisfiable, if and only if there is a model in I that satisfies ϕ. In short, SMT is the problem of determining $\mathcal{T}$-satisfiability.

To exemplify SMT, consider the theory of linear real arithmetic. $\mathcal{T}_{\mathbb{R}}$ consists of the signature $\Sigma = \{\mathbb{R}, +, -, ., \leq, \geq\}$ and the standard model of real numbers. Then, consider a formula $\phi = x < y \wedge \neg(x < y - 2)$. ϕ is $\mathcal{T}_{\mathbb{R}}$-satisfiable with any assignment where $x, y \in \mathbb{R}$ and $y - 2 \leq x < y$, for example $x = 4, y = 5$.

A naive way of encoding a neural network with ReLU activations would be to use the piecewise linear nature of the ReLU function to encode these non-linearities using disjunctions: $(x \leq 0 \wedge \texttt{ReLU}(x) = 0) \vee (x > 0 \wedge \texttt{ReLU}(x) = x)$. This is not scalable, as the number of splits is 2^n, where n is the number of ReLU nodes. Instead, a binary predicate $\texttt{ReLU}(x, y)$ iff $y = \max(0, x)$ is added to extend the theory $\mathcal{T}_{\mathbb{R}}$ to a theory of reals and ReLUs $\mathcal{T}_{\mathbb{R}R}$. Again, the final softmax layer of the network is disregarded in verification, as it does not change the outcome of the network for classification (it only acts as a smoothed version of the argmax function).

Neural networks can then be directly encoded as conjunctions of $\mathcal{T}_{\mathbb{R}R}$-atoms. A ReLU node v gets encoded as a pair of variables v^b and v^f, and then $\texttt{ReLU}(v^b, v^f)$ is asserted. Here, v^b is the backward-facing variable used to express the connection of v to the preceding layer's nodes, and v^f is the forward-facing variable that expresses the connection of v to the next layer's nodes. To solve this series of conjunctions in $\mathcal{T}_{\mathbb{R}R}$, the simplex algorithm (an algorithm to determine $\mathcal{T}_{\mathbb{R}}$-satisfiability) is extended to determine $\mathcal{T}_{\mathbb{R}R}$-satisfiability.

Reluplex and Marabou can determine the satisfiability of network output properties that can also be encoded in SMT. For example, consider a network that controls the speed of a car by outputting a value a for braking/accelerating (a is negative for braking and positive for accelerating). For simplicity, it takes as input the current speed of the car v, and the current speed limit of the road l only. If the speed of the car is above the speed limit, we require that the network tells the car to brake. We can encode this as follows: $((v > l) \wedge (a < 0))$.

To specify local robustness with the l_∞-distance in a classification network, consider the following example. Let $\mathbf{x} = \{x_1, ..., x_n\}$ be the input variable to the network and $\mathbf{f} = \{f_1, ..., f_n\}$ be the concrete feature values, let $\mathbf{y} = \{y_1, ..., y_l\}$ be the output variable, let 1 be the concrete, correct classification of input $\mathbf{f}$ and let ϵ be the distance up to which we wish to verify local robustness within. Then, we can specify local robustness in SMT as follows: $(((x_1 >= f_1 - \epsilon \wedge x_1 <= f_1 + \epsilon) \wedge ... \wedge (x_n >= f_n - \epsilon \wedge x_n <= f_n + \epsilon)) \wedge (y_1 > y_2 \wedge ... \wedge y_1 > y_l))$. Both Reluplex and Marabou allow the user to specify and verify any number of these properties.

Although both Reluplex and Marabou work efficiently for smaller ReLU networks (with Marabou providing an incremental improvement over Reluplex), are both easy to use since they accepts networks in a common format (Tensorflow SavedFile format v2 [118]), and can verify any property that can be encoded in SMT (not just local robustness), neither scale beyond networks with several hundred nodes, and both can only work with ReLU networks. Finally, note that both solvers are sound (i.e. if a derivation results in SAT (UNSAT), then the original problem is satisfiable (unsatisfiable)), except in the case of floating-point numbers which are notoriously hard to represent in a digital form, and complete (i.e. given enough time, there always exists a derivation ending in SAT or UNSAT).

ReluVal and Neurify ReluVal [126], and its successor Neurify [125], rely on interval arithmetic for formal verification, rather than SMT solvers. The target of ReluVal is to calculate output intervals for some given input intervals, and then verification amounts to checking whether the output interval violates a given threshold. In the naive case, input features are set as intervals and the same arithmetic operations as those in the target neural network are performed on these input intervals to compute an over-approximation of the output intervals. However, calculating the output intervals in this way leads to very conservative bounds that are too wide to be useful for checking any safety property.

To tackle this problem, ReluVal uses a combination of symbolic and concrete intervals, as well as iterative refinement to tighten the bounds to the point they are useful for verification. Symbolic intervals preserve dependency information that concrete intervals alone cannot. To demonstrate this, let $f(x) = 2x - x$, $x \in [0, 1]$. The true output range of $f(x)$ is $[0, 1]$; however, by simply propagating the concrete interval through each operation, the output becomes $[0, 2] - [0, 1] = [-1, 2]$, which is an over-approximation. When propagating symbolic intervals instead, the output becomes $[2x, 2x] - [x, x] = [x, x]$, which is then concretised to $[0, 1]$. In ReluVal, only linear symbolic intervals are kept track of, and concretised when non-linear functions (for example, ReLU activation functions) are present.

Using symbolic interval propagation may still not lead to narrow enough bounds for verification. To improve the bounds further, ReluVal makes use of iterative refinement, which involves evenly dividing the input interval into the union of two consecutive sub-intervals. Neurify, the successor of ReluVal, further improves performance by using linear relaxation in addition to everything else. Linear relaxation of ReLU nodes strictly over-approximates the non-linear constraint of the ReLU using a linear constraint. This leads to fewer concretisation steps when propagating through the non-linearities (for ReLU networks only), and further tightens the resulting bounds.

ReluVal and Neurify can encode and verify any property that can be expressed using intervals. For example, consider a regression network and let $\mathbf{x} = \{x_0, ..., x_n\}$ be the input variable, let $\mathbf{f} = \{f_1, ..., f_n\}$ be the concrete feature values of an input instance, let y be the output variable, let $v = [v_{min}, v_{max}]$ be the interval in which we want the output to lie within, and let ϵ be the distance that we are checking local robustness within. The local robustness property can be encoded by expressing each input variable as the interval $x_i = [f_i - \epsilon, f_i + \epsilon]$, and expressing the output as $y = v$.

Both software tools require the network to be in a customised version of the already quite rare nnet file format [62]. This format is a comma delimited, text based format that has the following structure:

- Line 1: number of layers (includes output layer but excludes input layer), input dimension, output dimension, maximal number of nodes for each layer.

- Line 2: Whether each layer is convolutional (e.g., 1,1,0,0 means totally 4 layers and the first two are convolutional layers while the last two are fully connected).
- Line 3 and further: each line describes the information for the layer, such as the number of out channels, in channels, the kernel size, the stride, and the padding, for convolutional layers.

A conversion function from Keras models to the nnet format can be found in the code associated with this thesis (see Appendix B).

Both ReluVal and Neurify operate on ReLU networks only, and if an output interval is not verified as safe, it does not imply that it is unsafe (making it an incomplete but sound method, as it can prove some but not all true properties).

2.5.5 Comparing Verification Methods

When comparing performance, ReluVal outperforms Reluplex by a significant margin: a speedup of around 200x. Marabou outperforms its predecessor Reluplex; however, it only outperforms ReluVal on a minority of benchmarks, and even then only by a small amount. Neurify provides additional improvement on all other tools, with an average speedup of around 10x versus ReluVal (the fastest competitor).

The main difference between the four tools is how the underlying techniques are used for verification. Reluplex and Marabou use SMT, whereas ReluVal and Neurify use symbolic interval propagation. The SMT tools have the advantage that they are sound and complete methods and can verify more properties than output intervals alone, leading to stronger guarantees. In contrast, the symbolic interval propagation based tools are sound but incomplete methods making them faster overall, but properties must be encoded as intervals. All tools can only work with ReLU networks.

In terms of usability, we found Marabou to be the best. Marabou accepts networks in a very common file format, specification of properties is simple, and installation and setup is straightforward. Neurify and ReluVal require a custom file format that is not widely used, but specification of properties is simple once installed.

Chapter 3

Literature Review

This literature review summarises the current literature that is relevant to the methods developed in this thesis. We first cover explainability in both Bayesian and non-Bayesian settings, then we overview verification and finally Bayesian uncertainty. In every case we discuss both general methods and application-specific literature.

3.1 Explainability

According to the European Commission's technical report on 'Robustness and Explainability of Artificial Intelligence' [49], explainability corresponds to the understanding of the underlying mechanisms of the model. In addition, their definition requires the demonstration that the model follows specifications and is aligned with human values such as fairness.

There are two main approaches to explainability, depending on the nature of the model. The first is the use of interpretable models, which are designed to provide reliable and easy to understand explanations. Interpretable models include methods such as decision trees and linear regression; however, due to their complexity, neural networks are not considered in the class of interpretable models. The second approach is post-hoc explainability, which is used to extract explanations from black-box models such as neural networks with large numbers of parameters.

As discussed in Chapter 2, explainability methods can be further divided into two groups: local explainability and global explainability, where local explainability aims to understand the impact of input features on an individual prediction, and global explainability aims to gain an understanding of the model's overall behaviour. This thesis focuses on post-hoc local explainability. However, for completeness, a range of approaches are discussed here.

3.1.1 Heuristic Local Methods

Gradient-based Methods Layer-wise relevance propagation (LRP) [5], discussed in Section 2.4.1, is the first example of a gradient-based method for local explainability. To recap, LRP defines a relevance measure over the input feature vector such that the network output can be expressed as the sum of the values of the input vector. The most commonly used variant of LRP, called LRP-ϵ, can be seen as computing a backward pass through the network where the gradient of the activation functions is replaced by their average gradient.

DeepLIFT [110] is another of such methods, which generalises the use of average gradients to compute attributions. Similarly to LRP, DeepLIFT also proceeds in a backward fashion from output to input. Each neuron is assigned a score that represents the contribution of that neuron when activated for the original network input, compared to the activation for some reference input. Reference values for each neuron are calculated by running a forward pass through the network for the reference input. DeepLIFT improves on LRP-ϵ as it does not assume a zero baseline, and does not assume a particular shape for the non-linearity.

The fundamental problem with both LRP and DeepLIFT is that the chain rule (used in the gradient calculations) does not hold generally for discrete gradients. This means that the quantity computed when replacing gradients with their average gradients does not always result in the average gradient of the function overall, and that two implementations of the same function could lead to different results. This is a case of failing to satisfy implementation invariance.

A generalisation of DeepLIFT, called Integrated Gradients (IG, also described in Section 2.4.1) [117], has been designed to satisfy implementation invariance. To recap, IG can be interpreted as computing scores by multiplying the input element-wise with the average partial derivative of the straight line path between an input and a baseline. Since the original gradient is used for the activation functions, the validity of the chain rule is preserved and therefore satisfies implementation invariance.

IG has a high computational cost, compared to both LRP and DeepLIFT, and although the chain rule does not hold in the case of DeepLIFT, it has been shown empirically to be a good approximation of IG.

Surrogate Model Methods LIME [100] and its successor Anchors [101], both of which are described in Section 2.4.1, are explainability methods that utilise surrogate models to explain predictions. In the case of LIME, the surrogate models take the form of neural networks that are trained to approximate the predictions of the underlying black-box model. Instead of creating feature importance maps that LIME, LRP or SHAP do, Anchors creates a region in the feature space and then comes up with an interpretable rule (an IF-THEN rule called an anchor) which guarantees the prediction. Anchors has a better generalisability than LIME, and is more computationally efficient than game-theoretic methods such as SHAP. Additionally, Anchors is able to explain non-linear decision boundaries since it works on feature predicates and is not trying to fit a local (linear) explainable model.

Game-Theoretic Methods SHAP [81] is a game-theoretic method that tries to attribute contribution of each feature to the deviation of the prediction from the mean model prediction. SHAP has the benefit that it can provide both local and global explanations, though it requires a large amount of computational power. Additionally, SHAP explanations explain the deviation from the mean model prediction, rather than the prediction itself.

3.1.2 Logic-based Local Methods

In addition to explainability, the work in [109] provides robustness guarantees on the explanations it produces, in the sense of requiring that the computed explanation is invariant under ϵ-bounded perturbations. Ensuring robustness to adversarial perturbations is desirable for stability of an explanation method, especially in applications where one must be certain of the reason for a prediction. This work identifies a minimal set of features whose current state is sufficient for the classification. The method, which works on naive Bayes classifiers only, compiles classifiers into ordered decision diagrams and then computes such explanations from this representation.

The solution proposed in [58] extends the same guarantees to neural networks, but requires that the neural network can be expressed as a set of constraints in a reasoning system that can be answered by some oracle. This essentially limits the approach to networks with ReLU activation functions only. The algorithm uses abductive reasoning to provide abduction-based explanations (ABEs) that are sufficient to imply the prediction, and are cardinality- or subset- minimal. The experiments, however, only pertain to boosted tree models.

DeepCover [116] is a subset method that utilises statistical fault localisation techniques to produce explanations that are non-obvious, sufficient and approximately minimal. Sufficient in this case means that the subset of pixels that make up the explanation produce the same output as the original input when the left-out pixels are set to the background colour. The method works by generating a number of mutations of the original input (by setting some pixels to the background colour each time) that are then annotated with y if the mutation still produces the correct output class, and $\neg y$ if it does not. Then, statistical fault localisation measures are used to rank each pixel by how important they are to the networks output, and the explanation is constructed by iteratively adding pixels in descending order of rank until the explanation is sufficient (as defined above). This work benefits from a linear time implementation and good accuracy (76%) on a benchmark with annotated ground truth explanations. However, the resulting explanation is only approximately minimal.

3.1.3 Global Methods

While SHAP can be used as both a local and global explanation method, there are other global-specific methods available. The work in [124] is an interpretable model approach that provides a framework for learning falling rule lists from data. The approach does not rely on traditional, computationally intensive, decision tree learning methods and the subsequent falling rule lists, which are an ordered list of IF-THEN rules with probabilities, are perfectly interpretable and provide a good level of precision and recall on the test dataset presented. However, this approach is not model-agnostic, and is not guaranteed to work well on all datasets.

A global explainability method that *is* model agnostic can be seen in [47]. This work uses partial dependence plots, which are low-dimensional graphical renderings of the prediction function (neural network), to allow the human interpreter to understand more easily the relationship between input features and the prediction. The method only requires that the model can provide predictions on new data. However, it is sensitive to outliers in the feature space and works on a low-dimensional approximation so the explanations generated may not always be accurate.

Reference [4] is a meta-level paper that provides a declarative language, using a logic called FOIL, to specify different explainability queries. Given the current volume of explainability methods, and the need for flexibility, reliability and ease of application, a language that standardises such explainability queries is a welcome addition. At present, the language can only be used to define explainability queries for explainable models, though they suggest how the language could be extended to include post-hoc interpretability.

3.1.4 Bayesian Explainability

A handful of Bayesian explainability methods have been developed. BayLIME [133] is a Bayesian extension to the LIME framework which exploits prior knowledge and

Bayesian reasoning to improve the consistency in repeated explanations of a single prediction of a neural network. It also benefits from improved explanation fidelity compared to non-Bayesian methods (e.g LIME, SHAP) because of the integration of this prior knowledge. Instead of the LIME approach of training a deterministic surrogate model, BayLIME trains a Bayesian local surrogate model, which is shown analytically to be a weighted sum of the prior knowledge and the estimates based on new samples (similarly to LIME). The weights of this model can either be automatically fitted from data samples or decided based on application-specific prior knowledge. The focus of this work is on whether incorporating priors improves the stability of the resulting explanations. Similarly, [48] introduces a Bayesian non-parametric approach to fitting a *global* surrogate model. While both of these works provide well-calibrated explanations, neither focuses on modelling the uncertainty of the explanations, and they do not tackle problems such as estimating hyperparameters or improving the efficiency of computing explanations.

Two methods that *do* focus on modelling uncertainty in Bayesian explanations can be found in [111] and [16]. Both works present an explanation-method-agnostic Bayesian framework for generating local explanations with uncertainty. The key difference is that [111] induces a distribution over the input, samples from this distribution, and produces a Bayesian explanation with uncertainty based on the explanations generated from a single deterministic network on each sample. In contrast, [16] applies directly to Bayesian neural networks, and takes weight samples from the BNN. Then, an explanation is generated for each network sample on the same input, using any existing explanation method, and a novel method for combining these explanations is described. Along with providing a justification that the mean explanation is sufficient to explain the decision making process of the BNN, this work introduces the concept of a union and intersection (UAI) explanation. A UAI explanation combines the relevance maps at different percentile levels (denoted as α), where a high percentile corresponds to a union explanation (union of all Bayesian relevance map samples) and a low percentile corresponds to an intersection explanation (intersection of all Bayesian relevance map samples). Figure 3.1, taken from [16], shows an

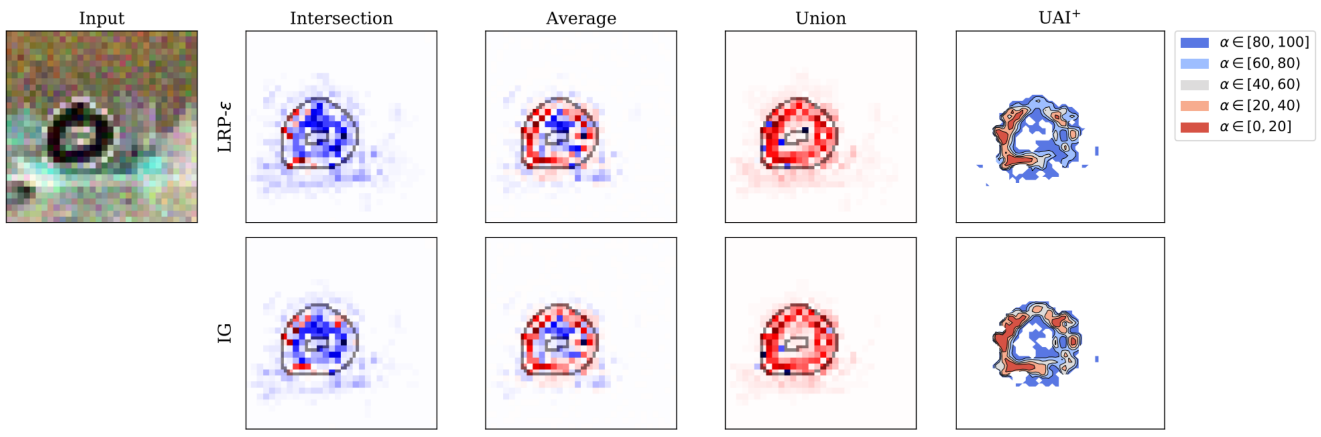


Figure 3.1: A figure showing the UAI method on the MNIST dataset with a random background. Two explanation methods were considered; LRP-ϵ and IG, and the value of α determines the percentage of explanations that agree on the importance of a given feature (in this case, pixel). Taken from [16].

example of the UAI method.

3.1.5 Specific Applications

Explainability in End-to-End Controllers Explainability of neural networks, especially end-to-end controllers, is a vital step towards finding out how they work, being able to trust in their decisions, and ultimately to verify their outputs. Nvidia's paper, titled Explaining How a Deep Neural Network Trained with End-to-End Learning Steers a Car [11], tries to do exactly this: it explains what PilotNet controller actually learns, and how it makes its decisions. The main aim of the paper was to develop a simple method for highlighting the parts of an image that were the most salient in determining steering angles, with these image sections called the salient objects. There are several existing methods for saliency detection, for example layer-wise relevance propagation; however Nvidia argue that the simplicity of their method, its fast execution time (real-time), and its almost pixel level resolution makes it especially advantageous for the task of end-to-end control in self-driving.

The procedure for this type of saliency detection is as follows. First, in each layer, the activations of the feature maps are averaged. Next, the top most averaged map is scaled up to the size of the layer below using deconvolution with the same parameters

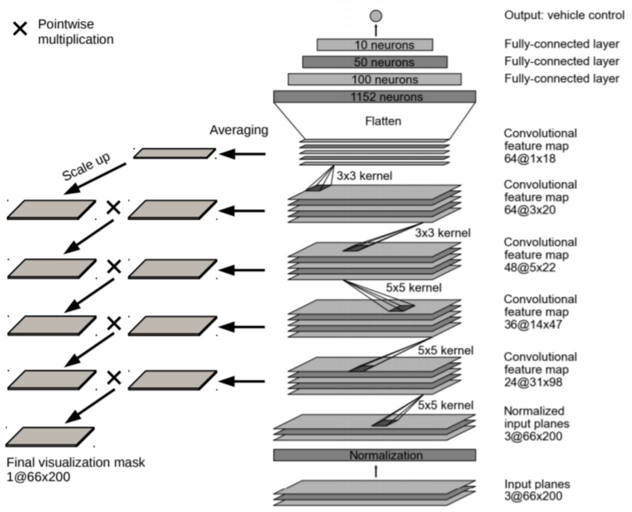


Figure 3.2: The PilotNet architecture, along with a diagram of the salient object visualisation method. Taken from [10].

used in the generation of the feature map. The up-scaled map is then combined with the averaged map from the layer below by multiplication, and this process is repeated from the lowest level until the input is reached, and a final visualisation mask is created. A diagram of this process can be seen in Figure 3.2.

It was found that the salient objects detected by this method clearly appeared to be those that should influence the steering control of the car, and they conducted some experiments to validate this. Most of the salient objects in the image corresponded to the edge of roads, or to lane markings, and overall this method substantially contributed to the understanding of what PilotNet controller learns.

A different approach to explainability in end-to-end controllers for self-driving cars can be found in the paper by Kim and Canny [69]. The paper opens by stressing the importance of easily explainable models, and the importance of easy-to-interpret rationales for their behaviour. Their approach comes in two stages. The first stage uses a visual attention model to train a CNN as an end-to-end controller, from image to steering angle. In this stage, the model highlights regions of the image that may potentially influence the network's output, as well as acting as an end-to-end controller for a self-driving car. The second stage then applies causal filtering to determine which of these potentially influencing regions actually do influence the output. Their results showed that incorporation of attention does not degrade control accuracy compared to an identical CNN without attention, as their network, the base of which was the same as PilotNet, achieved comparable accuracy to PilotNet. They also found that raw attention highlights interpretable features in the image, and causal filtering achieves a useful reduction in explanation complexity by removing features that do not significantly affect the output of the network. An interesting extension of this work that is mentioned is to compare the salient image areas with human drivers' gaze movement to determine whether the network uses the same visual cues as a driver would. Whilst this work provides a useful explainability method in the relevant domain, the method provides no formal guarantees of robustness or minimality.

Explainability in Natural Language Processing Natural language processing problems, such as machine translation, document summarising and sentiment analysis, are another application where explainability of the models used is of great importance: both to understand the rationale behind a prediction, as well as to instil confidence in the public and the human decision maker. The work presented in a paper by Wallace et al [123] introduces AllenNLP Interpret: a flexible framework for interpreting NLP models. It is a toolkit that provides interpretation primitives, for example input gradients, for any model trained within the framework, and a suite of existing interpretation methods and visualisation components. While this toolkit is immensely useful for training, using and interpreting NLP models, it does not provide explanation methods for NLP specifically and therefore is vulnerable to the out-of-distribution (OOD) data problem found in many NLP explanation methods. The OOD problem occurs in methods that answer the question "What does the model predict if each input token was not there?", which can be measured by the difference in softmax probabilities after erasing each token. If the sentence with the erased token deviates from the model's training data distribution, lower scores are assigned to OOD samples, which results in an overestimated contribution of less important tokens.

Work in [70] aims to tackle this out-of-distribution problem for NLP models through the use of input marginalisation. Instead of asking the above question, this paper instead asks “How would the model react differently if there were other tokens instead of each token?”. They marginalise each token out to mitigate the OOD problem of removing a token entirely. This method successfully shows clearer interpretations where uninformative tokens such as punctuation and “to” are given much lower attribution scores compared to other methods.

Finally, [120] presents a novel interpretation approach that relies on the human brain. Brain image recordings of people reading complex texts are compared to intermediate representations of a given layer of the neural network to interpret word and sequence embeddings from NLP models, including state-of-the-art NLP model BERT [29].

3.2 Verification

Formal verification aims to provide proof of correctness of software or hardware systems with respect to specific properties, using mathematical proofs [49]. As explained in Chapter 2, the most common correctness properties include robustness, safety and consistency. In this thesis, we use tools for the verification of neural networks, and, as such, this section of the literature review also focuses on the literature regarding verification of neural networks specifically.

While verification of linear classifiers is straightforward, neural networks prove challenging as they are highly non-linear with potentially millions of parameters. Current methods show promising results, but none scale to what are considered to be large networks.

3.2.1 Adversarial Examples

The robustness property of verification is the property that the network must not change its prediction when faced with *adversarial examples*: small, often imperceptible, changes to the input that should not affect the prediction. The seminal work on explaining and generating adversarial examples is the paper by Goodfellow, Schlens and Szegedy [45]: Explaining and Harnessing Adversarial Examples. This paper attributes the linear nature of neural networks to their vulnerability to adversarial examples, and provides a method for generating such examples called the Fast Gradient Sign Method (FGSM), which is explained in Chapter 2. While FGSM is a very efficient attack, is it both a white-box attack (meaning it needs access to the whole network, weights, etc) and it perturbs every feature.

Jacobian-based Saliency Map Attack (JSMA) [95] is an approach that minimises the number of perturbed features whilst still causing misclassification. The gradient of the loss function for each class label with respect to every component of the input (i.e. the Jacobian matrix) is used to extract the sensitivity direction. Then, a saliency map is used to select the dimension(s) that produce the maximum error and the input is perturbed only in those dimensions. This attack is more effective than FGSM, but it is less computationally efficient, and it is also still a white-box method.

Zeroth Order Optimisation (ZOO) [19] is an example of a black-box attack that estimates the gradients of the models without access to the classifier itself. The gradients are estimated by querying the target model with modified individual features and uses an optimiser such as ADAM [71] to optimise perturbations. The only downside to this method is that a lot of queries to the target classifier are required, meaning that it can be ineffective if there are mitigation methods such as rate limiting on the target.

3.2.2 Local Robustness Verification

To recap, deterministic local robustness ensures that a model classifies all inputs within an ϵ-ball consistently, and probabilistic local robustness ensures that a model classifies all inputs within an ϵ-ball consistently with some probability above a given threshold.

Constraint-based Methods Most constraint-based methods for local robustness verification apply to neural networks with ReLU activation functions only. ReluPlex

[63] and its successor Marabou [61] are SMT-based solvers for such networks, and ReluVal [126] with successor Neurify [125] are interval arithmetic-based solvers. All of these approaches are described in detail in Chapter 2 of this thesis. ReluVal outperforms ReluPlex in speed by over 200× and is able to verify instances that ReluPlex times out on. However the successor of ReluPlex, called Marabou, manages to outperform ReluVal on average, given enough cores with which to perform calculations. The creators of ReluVal then produced an updated version, Neurify, which provides an incremental improvement and manages to outperform all three of ReluVal, Marabou and ReluPlex.

Planet [33] is another constraint-based method for verifying properties, including local robustness, of ReLU neural networks. Planet combines SAT solving with linear programming and creates a linear approximation of the overall network behaviour. Using this approximation, large areas of the search space can be ruled out and do not need to be considered in the formal verification process, which leads to significantly shorter verification times.

Optimisation-based Methods DeepGO [104] uses another technique to verify local robustness, based on adaptive nested optimisation. More specifically, given a set of inputs (the local area around an input point) and a feed-forward deep neural network, the lower and upper bounds of the networks output are calculated. This work makes the assumption, and further proves that, feed-forward neural networks are Lipschitz continuous functions. DeepGO can handle all known layer types used in classification tasks (maxpool, sigmoid etc). However, due to the fact that only a finite number of iterations are performed in practice, the resulting bounds may not converge on the global minimum and maximum, meaning that inputs that could not be verified as safe are not necessarily unsafe (it is an incomplete method).

Three other optimisation-based neural network verification methods are described in sequential works by Kumar et al: [79], [14] and [15]. [79] and [15] formulate formal verification as a Branch and Bound problem and propose novel algorithms to speed up and scale verification of neural networks. [14] presents a formulation based on

Lagrangian Decomposition, along with an algorithm that produces bounds as tight as existing methods, but with better performance. As above, all of these methods are incomplete.

Probabilistic Methods Probabilistic methods aim to calculate the probability p that a Bayesian neural network output is robust for an input perturbed within an ϵ-ball around it. [17] gives a solution to estimate that probability with statistical guarantees on the accuracy of the estimate. The drawback of this method is that it requires the solution of many non-linear optimisation problems (one for each sample of the posterior), making it computationally expensive.

[129] improves on the previous work by providing guarantees on the lower bound of probabilistic safety for BNNs. The framework is based on relaxation techniques from non-convex optimisation, and derives procedures based on interval bound propagation (IBP) and linear bound propagation (LBP), for BNNs trained with variational inference, to compute the set of weights for which the corresponding neural network (sampled from the BNN posterior) is safe. This work scales to verify BNNs with millions of parameters in minutes, making it a powerful tool for probabilistic verification.

Adv-BNN [77] takes the approach of training a BNN to be robust in the first place. They note that fusing randomness can improve the robustness of neural networks, but adding noise blindly is not optimal. Instead, randomness is modelled under the framework of a BNN to learn the posterior in a scalable way. The resulting BNNs are more accurate than adversarially trained networks. Another method of training for certifiably robust BNNs is detailed in [128], where constraint relaxation techniques and a modification of the cross-entropy error function are used to enforce posterior robustness to worst-case ϵ-balls around the inputs.

Finally, the first and only approach to compute probabilistic guarantees for *iterative predictions* is described in [130]. Iterative predictions are sequences of predictions correlated across a time interval, for example in sequential planning, time-series forecasting, reinforcement learning or autonomous navigation. Over this series of predictions, a reach-avoid specification is defined that requires the goal region be reached over a given time period, whilst avoiding unsafe states. They derive a guaranteed lower bound on safety by back-propagating lower bounds of reach-avoid probabilities for discretised portions of the state space. The method is demonstrated on non-trivial, high-dimensional control tasks such as navigation and obstacle avoidance, and produces useful lower bounds that are not far away from empirically estimated bounds.

3.2.3 Global Robustness Verification

Here, global robustness verification refers to the absence or not of adversarial examples in the input space. Most of the approaches in the constraint-based methods section above also work for global robustness verification. For example, ReluPlex can verify anything that can be expressed in SMT. Although possible, this option for global robustness verification is slow and not scalable to the types of networks used commercially today. The work in [64] notes the difficulty of globally verifying a model, including the need to encode two identical copies of the network in the chosen logic. They instead suggest a hybrid approach that redefines local robustness as (δ, ϵ)-local robustness which checks whether, for all points up to δ away from an input, the confidence level of the classification is within ϵ of the confidence of the original input's classification, as opposed to the same classification. They postulate that sampling a set of points to be tested for (δ, ϵ)-local robustness can prove global robustness with some probability. However, it is an open question to determine the number of such samples and what values of δ and ϵ should be tested.

A fully global robustness verification approach is found in [84]. The method is based on abstract interpretation and importance sampling, and checks whether a network is probabilistically robust. Abstract interpretation is used to approximate the behaviour of the network and compute an overapproximation of the input regions that violate robustness. Importance sampling is then used to counteract the effect of the overapproximation and compute accurate probabilistic robustness probabilities. While this approach sounds promising, the authors have not yet implemented the algorithm in full, so no information about the efficiency and quality of the probabilities is available.

Since this thesis does not focus on verification methods specifically, a “Uses of Verification” section has been omitted.

3.3 Bayesian Uncertainty

It has long been known that neural networks are related to Gaussian processes, as putting a probability distribution over each weight will recover a Gaussian process when there are infinitely many weights. With a finite number of weights, you can still recover uncertainty estimates by placing distributions over the weights, and one of the original proposals for this was by MacKay in 1992 [83]. The method this paper proposed offers robustness to overfitting, but comes with heavy network architecture alterations, challenging inference and additional computational cost, and, as a result, was not very widely adopted.

As discussed in Section 2.3, variational inference is a technique that calculates an approximation of the posterior distribution. Variational inference has been applied to Bayesian neural networks in an attempt to make calculating this posterior feasible with respect to current scale of deep networks. One such approach is detailed in the paper by Hinton and Van Camp [50]. The authors present the ideal Bayesian approach, and then how their method can be used to get as close as possible to that. The ideal, but intractable, approach would put a prior distribution over all points in the weight space, then construct a posterior at each point by multiplying the prior by the probability of getting the outputs in the training set given those weights. Finally, normalisation is applied to get the full posterior distribution that can be used to make predictions for new inputs. The closest that can be achieved is to use Monte Carlo sampling to sample from the posterior distribution by considering random moves in weight space and accepting a move with a probability that depends on how well the resulting network fits the desired outputs. The authors then note that, if simplifying assumptions are made, the time-consuming Monte Carlo simulations can be avoided, and the method detailed involves using a simpler Gaussian approximation (with no diagonal terms in the covariance matrix) but also takes this distribution into account during the learning.

Some additional variational inference methods can be found in [46], [94] and [53]. These papers introduce new techniques such as sampling-based variational inference, and stochastic variational inference, which have been used to obtain new approximations for Bayesian neural networks. The downside to these techniques is that they are, still, prohibitively expensive computationally, as the number of parameters needed to represent uncertainty doubles for the same network size, and they require more time to converge on answers that do not improve on existing methods.

A breakthrough in the feasibility of Bayesian deep learning methods for real-world systems occurred when it was shown that a neural network with arbitrary depth and non-linearities, with dropout (or any other stochastic regularisation technique) applied before every weight layer, is equivalent to an approximation of the probabilistic Gaussian process. This work, which appears as [40], allows for easily accessible model uncertainty information. For details of this method, please see Section 2.3.4.

Concrete dropout [41] is a newly proposed variant of dropout that gives improved performance and better calibrated uncertainties. The authors point out that, to obtain well-calibrated uncertainty estimates, the technique requires a grid-search over the dropout probabilities, which is prohibitively difficult with large models. The general idea of their approach is to relax continuously dropout's discrete mask, and pair this with a principled objective function that allows for automatic tuning of the dropout probability in large models. Continuously relaxing dropout's discrete mask means replacing dropout's discrete Bernoulli distribution with its continuous relaxation, called the concrete distribution relaxation, and its purpose is to approximate discrete random variables. This allows for optimisation of the dropout probability itself. The technique was analysed on a wide variety of tasks, and it showed a significant reduction in experiment time, as well as improved model performance and uncertainty calibration.

Another variation on dropout is generalized dropout [114]. Generalized dropout is a family of regularisers which generalise dropout, and included in this family is a set of methods called Dropout++, which is a version of dropout with trainable parameters, much like Concrete dropout. Another member of the family selects the width of network layers, and classic dropout emerges as a special case. Their experiments showed that these methods improve generalisation performance over regular dropout.

An important questions to ask is, what uncertainties are even useful for Bayesian deep learning? A paper by Kendall and Gal answers exactly that, in the context of computer vision [67]. The paper makes the claim that it is most effective to model aleatoric uncertainty for large datasets as it cannot be explained away, whereas epistemic uncertainty can often be significantly reduced by the vast number of training images usually available in computer vision tasks. They also show that aleatoric uncertainty is best modelled in real-time applications, as these models can be formed without expensive Monte Carlo samples. With regards to epistemic uncertainty, the authors claim that it is better to model this uncertainty in safety-critical tasks and small datasets. In the rest of the paper, they present a Bayesian deep learning framework, which allows the learning of a mapping between input data and aleatoric uncertainty. This can optionally be composed with epistemic uncertainty approximations, and they compare the performance with each uncertainty modelled, and with both together. It was found that the highest accuracy in classification tasks, and the lowest RMS error in regression tasks, was achieved when both uncertainties were modelled, although each uncertainty on their own achieved better results than the baseline model. The idea that epistemic uncertainty should be modelled in safety-critical situations, for example self-driving, has been taken forward in the research detailed in parts of this thesis.

Work on discovering the *source*, aleatoric or epistemic, of uncertainty can be seen in [32]. Whilst most works focus on extracting and using uncertainty estimates, this paper seeks to identify examples the model is uncertain on *and* the underlying source of said uncertainty. They explore avenues such as how targeted data augmentation can improve uncertainty and whether the rate of learning in the presence of more information differs between different types of examples. To determine the source of uncertainty they leverage the fact that epistemic uncertainty is reducible in the presence of additional data, whereas reducing aleatoric uncertainty requires the use of techniques such as down-weighting [115] or elimination through data cleaning. Their results suggest that targeted data augmentation can amplify the differences in distributions between noisy (aleatoric) and atypical (epistemic) examples.

3.3.1 Uses of Uncertainty

The following section presents some practical applications of Bayesian neural networks and uncertainty information in different application domains related to this thesis.

Uncertainty in Image Recognition Uncertainty information combined with point estimates for prediction can lead to a more informed decision in the case of image segmentation for medical diagnoses. Kwon et al [73] explore the use of Bayesian neural networks in ischemic stroke legion segmentation. They decompose predictive uncertainty into aleatoric and epistemic uncertainty, which they show adds important insight to the point estimates from the ischemic stroke legion segmentation challenge.

Similar work can be found in [119], where the use of Bayesian neural networks is explored in the classification of breast histopathology images. The authors employ a dimensionality reduction technique to explain the uncertainty of the prediction, and they show that the Bayesian convolutional neural network that they craft performs much better than state-of-the-art transfer learning CNNs. Most importantly for medical applications such as this, their method significantly reduces the false negative rate of classifications.

Uncertainty in Natural Language Processing One example of work that studies the benefit of characterising model and data uncertainties for natural language processing tasks is by Xiao and Wang [131]. They show that explicitly modelling uncertainties enhances model performances in NLP tasks, including sentiment analysis, named entity recognition and language modelling. They show these results for both convolutional and recurrent neural networks, and they investigate the characteristics of inputs with exceptional uncertainty measures in two NLP datasets (Yelp 2013 and CoNLL 2003). They observe that their models exhibit higher data uncertainties for more difficult predictions.

A second application of BNNs in NLP can be seen in [21]. This work applies Bayesian recurrent neural networks to the task of language modelling: a language model determines the probability of the next word in a sequence by analysing text data. Recurrent neural network language models (RNN-LMs) are difficult to train given their high number of parameters, large dictionary size and high dimensional nature. By instead using a Bayesian RNN-LM, the authors are able to improve the robustness and performance of the system.

Uncertainty in Autonomous Driving There exist several examples of the use of uncertainty in neural networks for autonomous driving, for a range of different tasks. The paper by McAllister et al [85] explores three topics in autonomous vehicle systems; safety, interpretability and compliance, and discusses how Bayesian deep learning can be used to solve problems in these domains. The section on safety focuses on how components of a self-driving system should, at the very least, record their uncertainty and pass it to the next component so that more informed decisions can be made, and that the way to model uncertainty is to use Bayesian deep learning either to place distributions over the model weights, or to learn direct mapping to probabilistic outputs. The section on interpretability focuses on several important goals: to help passengers trust autonomous vehicle technology by informing them what the vehicle is doing and why, to help society understand the technology, to aid engineers understanding of the model to be able to validate against safety standards, and to provide accountability for insurance and legal reasons. They suggest that saliency mapping is a good start to this work, and that Bayesian deep learning can be used to give different interpretations of saliency. The section on compliance suggests two important aspects to compliance: law abidance and passenger reassurance. The

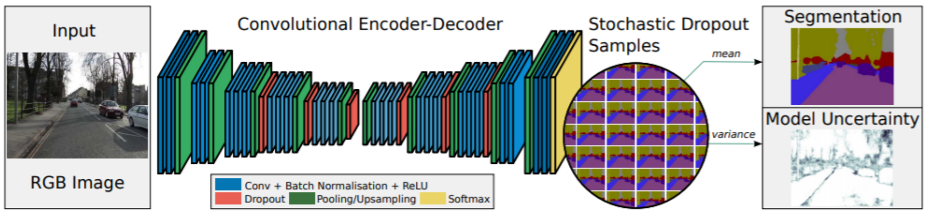

Input
RGB Image
Convolutional Encoder-Decoder
Conv + Batch Normalisation + ReLU
Dropout
Pooling/Upsampling
Softmax
Stochastic Dropout Samples
mean
variance
Segmentation
Model Uncertainty

Figure 3.3: The architecture of Bayesian SegNet, including how the segmentation and uncertainty maps are obtained. Taken from [65].

authors consider transfer learning or reinforcement learning to solve these issues. The paper ends by discussing the importance of clear evaluation metrics for each component of the self-driving system.

A clear use of the uncertainty information available from neural networks comes in the paper by Kendall et al, with a framework called Bayesian SegNet [65]. Bayesian SegNet is a framework for probabilistic pixel-wise semantic segmentation, which allows for pixel-wise prediction of class labels with a measure of model uncertainty. In order to obtain model uncertainty, they use the technique by Gal and Ghahramani [40], which uses Monte Carlo sampling with dropout at test time to generate a posterior distribution of classes. The architecture of SegNet consists of a sequence of non-linear processing layers, called encoders, and then a corresponding set of decoders, followed by a pixel-wise classifier. A diagram of this architecture can be seen in Figure 3.3. The authors found that the model is most uncertain at object boundaries, and with visually complex and ambiguous objects. Bayesian SegNet outperformed a number of state-of-the-art architectures by 2-3% and achieved a significant increase in performance for small datasets, where modelling the model uncertainty is more effective.

Another paper that uses uncertainty in semantic segmentation is from Kampffmeyer et al [60]. This paper focuses on land cover mapping in remote sensing images, with an emphasis on urban areas. The authors first present a problem that often occurs in remote sensing: class imbalance. This means that small objects often get less prioritised in an effort to achieve the best overall accuracy. The authors then propose a novel approach to maintain this high overall accuracy while still achieving high accuracy on the smaller objects, by quantifying the uncertainty on both the pixel level, and on a patch-based level. The first pixel-by-pixel approach was found to be too slow for real-time use in remote sensing. The patch-based pixel classification approach split the image into patches, and then a CNN was trained on these. The results for both methods showed promise, with the pixel-wise classification creating better segmentation maps, but in a longer time period. The authors noted that, in semi-automatic approaches for remote sensing, areas with high uncertainty could then be presented to an operator for manual clarification.

A different use of uncertainty in self-driving can be seen in [66]. This paper makes use of a Bayesian CNN to obtain the relocalization uncertainty of the model to improve the accuracy and, ultimately, to outperform state-of-the-art relocalization models. They achieve this by leveraging uncertainty to estimate metric relocalization error, which they show strongly correlate, and to detect the presence or absence of the scene in the input image. Interestingly, although a fully Bayesian model should perform dropout after every convolutional layer, the authors found that in practice this was not empirically optimal, as adding dropout after every layer acted as too strong a regulariser and degraded performance by around 10%. The paper concludes by evaluating the efficiency of the system, which processes each batch of 128 images in 6ms on an Nvidia Titan X graphics card. The authors also note that they have provided sufficient evidence to show that the model is more uncertain about images which are far from the training examples, which is the expected behaviour in this scenario.

Finally, [3] proposes a Bayesian neural network for end-to-end control that estimates uncertainty using the same technique by Gal and Ghahramani [40]. They show how more principled uncertainty estimates can be obtained by exploiting feature map correlation during training and using spatial dropout. They also show how these uncertainty estimates can improve the prediction of the network. They found that spatial dropout, as opposed to regular dropout, produced lower errors, and they used the uncertainty estimates from this to develop a shared controller, which fuses a human's actions with a network's output according to the uncertainty with some tunable threshold parameter.

3.4 Literature Summary

All of the literature reviewed in this chapter contributes to the safety of AI systems, and represents a positive step towards accurate *and* safe systems becoming more widely adopted.

Explaining the decision of a given decision function is a popular research area, and many works employ heuristic methods to produce both local and global explanations that allow the human interpreter to understand better the behaviour of the decision function [5, 117, 100, 110, 101, 81, 124, 47, 116]. Explanations that imply the prediction and have robustness guarantees are especially desirable in situations where one must be certain of the reason for a prediction, yet there are few works that consider this [109, 58]. Additionally, only [58] considers neural networks as the decision function, and the perturbations it considers are unbounded, which can lead to the entire input being included in the explanation in certain applications. For these reasons, there is a need for an explanation method with robustness guarantees that is also *informative* (i.e. is not the whole input), with minimality guarantees that enforce this.

Another important avenue in ensuring the safety of AI systems is the use of Bayesian neural networks and the uncertainty information they provide. Bayesian neural networks were thought to be too computationally complex to be viable for real-world applications but recent advances of posterior inference techniques [50, 46, 53, 94, 40, 41, 114] mean that they are becoming more popular. Work has also been undertaken on uncovering what types of uncertainty (aleotoric, epistemic) are useful to model in Bayesian learning, as well as discovering the underlying source of the different types of uncertainty [67, 32]. BNN uncertainty information has seen use in many applications, including image recognition [73, 119], natural language processing [131, 21] and autonomous driving [85, 65, 60, 66, 3]. Despite the range of applications described here, there is little work on understanding how we can quantify and use BNN uncertainty information in ensuring AI safety with a focus on obtaining statistical guarantees of the safety of the BNN, and methods within this area must be developed.

Finally, work on explanations of decision functions that incorporate uncertainty is extremely limited [16, 111], and further work in this area is a necessity to ensure AI safety benefits from the uncertainty information that Bayesian neural networks can provide.

Chapter 4

Explainability and Safety

In this chapter, we consider the role of explainability in ensuring AI safety. Amongst existing techniques, *subset methods* produce *local explanations* that explain the individual prediction in terms of a subset of the input features that justify the prediction. This chapter introduces a novel type of subset method that provides both safety guarantees and optimality with respect to a cost function, and shows how this method can be used to improve safety. Although demonstrated using NLP as a proxy application, we stress that our methods extend to any decision function with inputs represented as feature vectors. Some of the work in this chapter appeared in [74].

4.1 Motivation

In this chapter, we consider deterministic decision functions of the form $f : X \to Y$ and feature vectors X in the form of word embeddings, described as setting A in Section 1.1. Existing state-of-the-art explanation methods, such as LIME [100] and Anchors [101], lack robustness to adversarial perturbations, and methods that *do* consider adversarial robustness, for example abduction-based explanations [58], only consider unbounded perturbations that often result in the explanation being equal to the entire input. We seek to develop a method that yields high quality explanations: explanations that are stable (locally robust) and explanations that are useful (optimal with respect to a user-defined cost function).

Local robustness is an important property of explanations, as it ensures stability of the explanation and provides the guarantee that the explanation is logically sufficient to imply the prediction. For example, consider a neural network that predicts the instruction of a traffic sign, and an explanation method that highlights a number of pixels in the image that contributed towards the prediction. If the input image is perturbed such that it is imperceptible to a human, both the classification *and* the explanation should remain the same. It would be problematic if the explanation changed but the classification remained the same in this case (i.e an unstable explanation), since that explanation may be used to e.g. control an autonomous vehicle. It would also be problematic if parts of the input that were used for prediction did *not* appear in the explanation (i.e an insufficient explanation). Thus, it is important that both the prediction and the explanation are robust.

Optimality (with respect to a user-defined cost function) is another important property of explanations, as it allows the user to have more control over both which features are important in an explanation, and what properties of the explanation are important. For example, the cost function could be defined as the length of the resulting explanation, and in that case the smallest (but still sufficient) explanation is returned. We could also define the cost function to be the length of the explanation with an extra penalty given for unimportant features (in NLP, this could be padding, punctuation or similar), which would result in a minimal explanation without any unimportant information. Overall, optimality improves usefulness of explanations for the human decision maker.

As explained in Section 1.1, we work with the NLP task of sentiment analysis as a proxy application to demonstrate our methods. Due to the nature of the task, many robustness queries need to be made per input, and the black-box solvers available to us at the time of writing are not powerful enough to handle the high dimensionality and large networks seen in tasks such as image classification. Despite this, we stress that our methods are applicable to a wide range of applications, and any advancements in solver technology will directly impact the methods we develop here.

4.2 Robust Explanations

We consider a standard NLP classification task, where we classify some given input text t into a plausible class y from a finite set Y. For example, t is a movie review that we want to classify as either good or bad ($Y = \{\text{good,bad}\}$).

We assume that t is a sequence of input features in the form of words. t is of length l, and $t = (w_1, \ldots, w_l)$, where $w_i \in W$, and W is a finite vocabulary that can include the padding character '<PAD>'. Text inputs are then encoded using a continuous *word embedding* function $\mathcal{E} : W \to \mathbb{R}^d$, where d is the size of the embedding, as described in Section 2.1.3. Thus, given a text $t = (w_1, \ldots, w_l)$, we define the embedding $\mathcal{E}(t)$ of t as the sequence $x = (x_{w_1}, \ldots, x_{w_l}) \in \mathbb{R}^{l \cdot d}$, where $x_{w_i} = \mathcal{E}(w_i)$.

We denote with $W_{\mathcal{E}} \subseteq W$ the vocabulary used to train the embedding function $\mathcal{E}$. We consider embedding vectors trained from scratch on the sentiment analysis task, which involves classifying the sentiment of the input as positive or negative. This technique forces words that are positively correlated to each of the output classes to be gathered closer in the embedding space [6], which is considered a good proxy for semantic similarity with respect to the sentiment analysis task compared to count-based embeddings [2]. For classification, we consider a *neural network* $f : X \to Y$ that operates on the text embedding ($X = \mathbb{R}^{l \cdot d}$).

We seek to provide *local explanations* for the predictions of a neural network NLP model. For a text embedding $x = \mathcal{E}(t)$ and a prediction $f(x)$, a local explanation E is a subset of the features of t, i.e., $E \subseteq F$ where $F = \{w_1, \ldots, w_l\}$, that is sufficient to imply the prediction. We denote with I_E and I_F the indices of variables in sets E and F respectively, where $I_F = \{1, ..., l\}$ and $I_E \subseteq I_F$. We focus on deriving *robust explanations*: extracting a subset E of the text features F which ensure that the neural network prediction remains invariant for any bounded perturbation of the other features $t \setminus E$. Thus, the features in a robust explanation are *sufficient to imply the prediction* that we aim to explain, a clearly desirable feature for a local explanation. The reason we focus on explanations that are *robust with respect to* *bounded perturbations in the embedding space of the input text* is that unbounded perturbations often result in the whole input being an explanation.

We extract word-level explanations by means of word embeddings, but we note that our methods work, without further extensions, with diverse representations (for example, sentence-level or character-level explanations). For a word $w \in W$, with embedding $x_w = \mathcal{E}(w)$ we denote with $\mathcal{B}(w) \subseteq \mathbb{R}^d$ a generic set of word-level perturbations. We consider the following kinds of perturbation sets, depicted in Fig. 4.1.

ϵ-ball: $\mathcal{B}(w) = \{x \in \mathbb{R}^d \mid \|x - x_w\|_p \leq \epsilon\}$, for some $\epsilon > 0$ and $p \geq 1$. In words, the ϵ-ball is the d-dimensional ball of radius ϵ around a point in the d-dimensional feature space. This is a standard measure of local robustness in computer vision, where ϵ-variations are interpreted as manipulations of the pixel intensity of an image. For an example, see the red circle in Figure 4.1, centered around the input "movie". This concept has also been adopted in early NLP robustness works [89], but then replaced with better representations based on actual word replacements and their embeddings, see below, as it includes as perturbations vectors that do not have a corresponding word in the vocabulary [75].

k-NN box closure: $\mathcal{B}(w) = BB(\mathcal{E}(NN_k(w)))$. Here, the function $BB(X)$ produces the minimum bounding box for set X. For an example, see the blue box around the set of blue inputs in Figure 4.1. For a set of words $W' \subseteq W$, $\mathcal{E}(W') = \bigcup_{w' \in W'} \{\mathcal{E}(w')\}$, that is to say, $\mathcal{E}(W')$ is shorthand for the embedding function applied to a set of words. $NN_k(w)$ is the set of the k nearest neighbours to w in the embedding space: words w' with the smallest distance $d(x_w, \mathcal{E}(w'))$, where d is a valid notion of distance between embedded vectors. In our case, we use the cosine distance metric (see Section 2.1.3). Note that, in general, the box closure can be calculated for any set of embedded words. This provides an over-approximation of the k-NN convex closure, for which constraint propagation (and thus robustness checking) is more efficient [59, 55].

For some word-level perturbation $\mathcal{B}$, set of features $E \subseteq F$, and input text t with embedding $(x_1, \ldots, x_l)$, we denote with $\mathcal{B}_E(t)$ the set of *text-level* perturbations obtained from t by keeping constant the features in E and perturbing the others

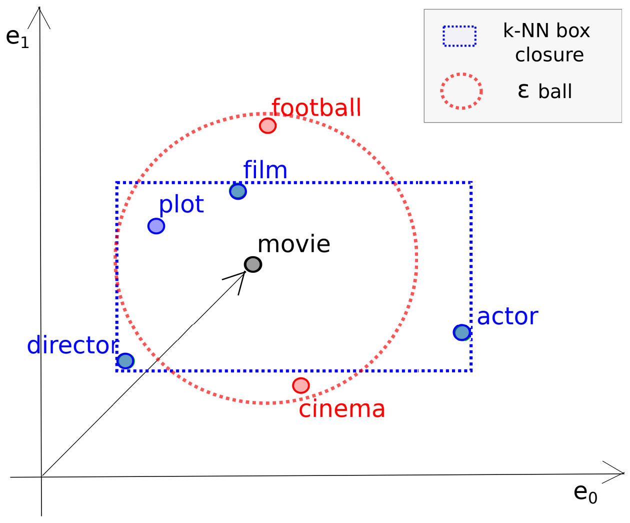


Figure 4.1: A graphical representation of the perturbation sets we define in the embedding space.

according to $\mathcal{B}$:

$$\mathcal{B}_E(t) = \{(x'_1, \dots, x'_l) \in \mathbb{R}^{l \cdot d} \mid x'_w = x_w \text{ if } w \in E;\ x'_w \in \mathcal{B}(w) \text{ otherwise}\}. \quad (4.1)$$

A robust explanation $E \subseteq F$ ensures prediction invariance for any point in $\mathcal{B}_E(t)$, i.e., any perturbation (within $\mathcal{B}$) of the features in $F \setminus E$.

Definition 4.1 (Robust Explanation) *For a text $t = (w_1, \dots, w_l)$ with embedding $x = \mathcal{E}(t)$, word-level perturbation $\mathcal{B}$, and classifier f, a subset $E \subseteq F$ of the features of t is a* robust explanation *iff*

$$\forall x' \in \mathcal{B}_E(t).\ f(x') = f(x). \quad (4.2)$$

We denote Equation 4.2 with predicate $\mathsf{Rob}_{f,x}(E)$.

As previously discussed, *robust* explanations are both stable and sufficient to imply the prediction, which is important in many applications, especially safety-critical domains such as healthcare.

4.3 Optimal Robust Explanations (OREs)

While robustness is a desirable property, it is not enough alone to produce *useful* explanations. Indeed, we can see that an explanation E including all the features, i.e., $E = F$, trivially satisfies Definition 4.1. Typically, one seeks short explanations, because these can generalise to several instances beyond the input t and are easier for human decision makers to interpret. We thus introduce *optimal robust explanations (OREs)*, that is, explanations that are both locally robust and optimal with respect to an arbitrary cost function that assigns a penalty to each word.

Definition 4.2 (Optimal Robust Explanation) *Given a cost function $\mathcal{C} : W \to \mathbb{R}^+$, and for $t = (w_1, \ldots, w_l)$, x, $\mathcal{B}$, and f as in Def. 4.1, a subset $E^* \subseteq F$ of the features of t is an* ORE *iff*

$$E^* \in \arg\min_{E \subseteq F} \sum_{w \in E} \mathcal{C}(w) \ \textit{s.t.}\ \mathsf{Rob}_{f,x}(E). \tag{4.3}$$

Note that Equation 4.3 is always feasible, because its feasible set always includes at least the trivial explanation $E = F$.

A special case of our OREs is when $\mathcal{C}$ is *uniform* (it assigns the same cost to all words in t), in which case E^* is (one of) the *robust explanations of smallest size*, i.e., with the least number of words. We note that there is not necessarily one unique ORE for each input, but rather there can be multiple OREs with the same minimal cost, although this is less likely when a more complex cost function is employed.

4.4 Proof of Sufficient Reason

We find that every ORE can be formulated as a prime implicant [58], a property that connects our OREs with the notion of sufficient reason introduced in [27].

For an input text $t = (w_1, \ldots, w_l)$, let $C = \bigwedge_{i=1}^{l} \chi_i = x_{w_i}$ be the *cube* representing the embedding of t, where χ_i is a variable denoting the i-th feature of x. Let $\mathcal{N}$ represent the logical encoding of the classifier f, and $\hat{y}$ be the formula representing the output of $\mathcal{N}$ given $\chi_1, \ldots, \chi_l$. Let $B = \bigwedge_{i=1}^{l} \chi_i \in \mathcal{B}(w_i)$ be the constraints encoding our perturbation space.

Proposition 4.1 then shows that OREs are minimal cost subsets, and are prime implicants.

Proposition 4.1 *Let E^* be an ORE and C^* its constraint encoding. Define $\phi \equiv (B \wedge \mathcal{N}) \rightarrow \hat{y}$, where $\mathcal{N}$, B, $\hat{y}$ and C are as above. Then, the following definitions apply to C^*:*

1. *C^* is a minimal cost subset of C such that $C^* \models \phi$.*

2. *C^* is a prime implicant of ϕ.*

Proof 4.1 *With abuse of notation, in the following we use C^* to denote both an ORE and its logical encoding.*

1. *If C^* is an ORE, then $\phi \equiv (B \wedge f) \rightarrow \hat{y}$ is true for any assignment x' of the features not in C^*. In particular, ϕ is trivially satisfied for any x' outside the perturbation space B, and, by Definition 4.1, is satisfied for any x' within the perturbation space.*

2. *C^* is said a prime implicant of ϕ if $C^* \models \phi$ and there are no proper subsets $C' \subset C^*$ such that $C' \models \phi$. This holds regardless of the choice of the cost $\mathcal{C}$, as long as it is additive and assigns a positive cost to each feature as per Definition 4.2. Indeed, for such a cost function, any proper subset $C' \subset C^*$ would have cost strictly below that of C^*, meaning that $C' \not\models \phi$ (i.e., is not robust) because, otherwise, C' (and not C^*) would have been (one of) the robust explanations with minimal cost.*

4.5 Logical Formulation

In order to facilitate the necessary entailment checks when searching for an ORE, i.e. checking that our candidate explanation $E \subseteq C$, perturbations of the input (excluding

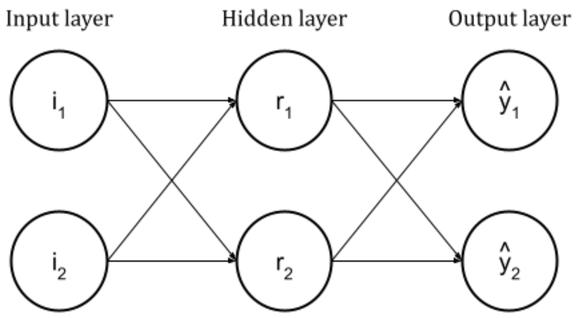


Figure 4.2: A very simple neural network architecture for classification with two inputs, one hidden layer with two neurons, and two output classes.

the candidate explanation) $\mathcal{B}_{F \setminus E}(t)$ and network $\mathcal{N}$ entail the prediction $\mathcal{B}_{F \setminus E}(t) \wedge E \wedge \mathcal{N} \models \hat{y}$, we must first formulate our network, explanation and perturbation set using SMT.

To encode the network f itself, we use the Marabou framework, which translates a model (with ReLU activations only) written in Keras or Tensorflow into constraint form $\mathcal{N}$. The exact formulation and resulting format of constraints is discussed in background Section 2.5.4. However, any logical formulation of neural networks is sufficient, for example, consider the network in Figure 4.2.

To encode this network in naive SMT, we assume that the hidden layer uses ReLU activation functions. We omit biases for simplicity and we omit the softmax function on the output layer because the non-linearity cannot be handled (we use logits instead). Then, the following set of constraints can be used (where $W_{l,j,k}$ is a weight for layer l, neuron j and connection k): $\{$ $(r_1 = 0 \wedge (W_{1,1,1}i_1 + W_{1,1,2}i_2) \leq 0) \vee (r_1 = (W_{1,1,1}i_1 + W_{1,1,2}i_2) \wedge (W_{1,1,1}i_1 + W_{1,1,2}i_2) > 0)$, $(r_2 = 0 \wedge (W_{1,2,1}i_1 + W_{1,2,2}i_2) \leq 0) \vee (r_1 = (W_{1,2,1}i_1 + W_{1,2,2}i_2) \wedge (W_{1,2,1}i_1 + W_{1,2,2}i_2) > 0)$, $\hat{y}_1 = W_{2,1,1}r_1 + W_{2,1,2}r_2$, $\hat{y}_2 = W_{2,2,1}r_1 + W_{2,2,2}r_2$ $\}$. The conjunction of this constraint set is then the encoding of the network.

From the notation defined in Section 4.4, recall that for an input text $t = (w_1, \ldots, w_l)$, $C = \bigwedge_{i=1}^{l} \chi_i = x_{w_i}$ is the cube representing the embedding of t, where χ_i is a variable denoting the i-th feature of x. We can then add the perturbation constraints and the candidate explanation, $\mathcal{B}_{F \setminus E}(t) \wedge E$, together. For any feature i

in the explanation being queried, it must be fixed to the concrete value in the input and therefore we just copy over that feature constraint from the cube. That is, we set $\bigwedge_{i \in I_E} \chi_i = x_{w_i}$. For indices not in the explanation $i \in I_F \setminus I_E$, we allow these to vary within the calculated perturbation bounds by adding the following two constraints per index:

- $\chi_i \geq \text{lower\_bound}$
- $\chi_i \leq \text{upper\_bound}$

For example, using the ϵ-ball perturbation set with the L_∞ distance, the lower bound of a feature will be equal to $x_{w_i} - \epsilon$ and the upper bound will be $x_{w_i} + \epsilon$. Every perturbation set can be distilled down into upper and lower bounds for each variable in the input in this way.

Finally, we must add the output variable $\hat{y}$ constraint. If there is only one output, then we can add the constraint $\hat{y} = y_{\text{true}}$. However, in classification problems, the number of outputs is (most of the time) equal to the number of classes. If the output layer has c neurons, then let $\hat{y}_i$ represent output neuron i, where $i \in \{1, ..., c\}$. We can then add the constraints $\hat{y}_{y_{\text{true}}} > \hat{y}_k$, for all $k \in \{1, ..., c\} \setminus y_{\text{true}}$.

4.6 Include and Exclude Constraints

Constraining our OREs allows us to improve usefulness for the human decision maker by enabling two crucial use cases: *detecting biased decisions*, and *enhancing non-formal explainability frameworks.*

We consider OREs E^* derived under constraints that enforce specific features F' to be included/excluded from the explanation:

$$E^* \in \underset{E \subseteq F}{\arg\min} \sum_{w \in E} \mathcal{C}(w) \text{ s.t. } \mathsf{Rob}_{f,x}(E) \wedge \phi(E), \tag{4.4}$$

where $\phi(E)$ is one of $F' \cap E = \emptyset$ (*exclude*) and $F' \subseteq E$ (*include*).

Note that adding *include* constraints does not affect the feasibility of our problem, because the feasible region of Equation 4.4 always contains at least the explanation $E^* \cup F'$, where E^* is a solution of Equation 4.3 and F' are the features to include. See Definition 4.1. Conversely, *exclude* constraints might make the problem infeasible when the features in F' don't admit perturbations, i.e., they are necessary for the prediction, and thus cannot be excluded.

Such constraints can be easily accommodated by any solution algorithm for non-constrained OREs: for *include* ones, it is sufficient to restrict the feasible set of explanations to the supersets of F'. For *exclude* constraints, we can manipulate the cost function so as to make any explanation with features in F' strictly sub-optimal with respect to explanations without. That is, we use cost $\mathcal{C}'$ such that $\forall_{w \in F \setminus F'} \mathcal{C}'(w) = \mathcal{C}(w)$ and $\forall_{w' \in F'} \mathcal{C}'(w') > \sum_{w \in F \setminus F'} \mathcal{C}(w)$. The ORE obtained under cost $\mathcal{C}'$ might still include features from F', which implies that Equation 4.4 is infeasible (i.e., no robust explanation without elements of F' exists).

4.6.1 Detecting Bias

Following [27], we deem a classifier decision *biased* if it depends on protected features, i.e., a set of input words that should not affect the decision. One example is a movie review that is classified based on the the director's name that happens to be mentioned in the review, rather than the sentiment behind the review. Detecting bias is clearly important in ensuring the safety of AI applications because it allows the human decision maker to determine whether the model is carefully considering all the information available to it, or whether the model is simply predicting based on one feature being present in the input, no matter the context.

In particular, a decision $f(x)$ is biased if we can find, within a given set of feature-level perturbations, an input x' that agrees with x on all but protected features and such that $f(x) \neq f(x')$.

Definition 4.1 *For classifier f, input t with features F, protected features F' and embedding $x = \mathcal{E}(t)$, decision $f(x)$ is* biased *w.r.t. some feature-level perturbation $\mathcal{B}$,* *if*

$$\exists x' \in \mathcal{B}_{F \setminus F'}(t). f(x) \neq f(x').$$

The proposition below allows us to use exclude constraints to detect bias.

Proposition 4.2 *For f, t, F, F', x and $\mathcal{B}$ as per Definition 4.1,* decision $f(x)$ is biased iff Equation 4.4 is infeasible under $F' \cap E = \emptyset$.

Proof 4.2 *Call A = "$f(x)$ is biased" and B = " Equation 4.4 is infeasible under $F' \cap E = \emptyset$". Let us prove first that $B \rightarrow A$. Note that B can be equivalently expressed as*

$$\forall E \subseteq F. (E \cap F' \neq \emptyset \vee \exists x' \in \mathcal{B}_E(t). f(x) \neq f(x')).$$

If the above holds for all E then it holds also for $E = F \setminus F'$, and so it must be that $\exists x' \in \mathcal{B}_{F \setminus F'}(t). f(x) \neq f(x')$ because the first disjunct is clearly false for $E = F \setminus F'$.

We now prove $A \rightarrow B$ by showing that $\neg B \rightarrow \neg A$. Note that $\neg B$ can be expressed as

$$\exists E \subseteq F. (E \cap F' = \emptyset \wedge \forall x' \in \mathcal{B}_E(t). f(x) = f(x')), \tag{4.5}$$

and $\neg A$ can be expressed as

$$\forall x' \in \mathcal{B}_{F \setminus F'}(t). f(x) = f(x'). \tag{4.6}$$

To see that Equation 4.5 implies Equation 4.6, note that any E that satisfies Equation 4.5 must be such that $E \cap F' = \emptyset$, which implies that $E \subseteq F \setminus F'$, which in turn implies that $\mathcal{B}_{F \setminus F'}(t) \subseteq \mathcal{B}_E(t)$. By Equation 4.5, the prediction is invariant for any x' in $\mathcal{B}_E(t)$, and so is for any x' in $\mathcal{B}_{F \setminus F'}(t)$.

4.6.2 Enhancing Non-Formal Explainers

The local explanations produced by heuristic approaches like LIME or Anchors are not automatically locally robust like OREs. We can use our approach to *minimally extend* (with respect to the chosen cost function $\mathcal{C}$) any non-robust local explanation

F' in order to make it robust, and therefore stable, by solving Equation 4.4 under the *include* constraint $F' \subseteq E$.

In particular, with a uniform $\mathcal{C}$, our approach would identify the smallest set of *extra* words that can be added to make F' robust. Such an extension retains to a large extent the original explainability properties as it is guaranteed to be a minimal extension. Enhancing these non-formal explainers is of interest when trying to ensure safety of AI, as plurality of explanations is desirable. Therefore, being able to repair non-formal explainers in this way ensures provision of explanations with robustness and stability guarantees.

4.7 Model Debugging

In addition to using OREs with include/exclude constraints to detect biased decisions and repair non-formal explainers, another important use-case of OREs is when the model commits a misclassification. By examining the ORE, we are able to uncover sensitivities of the model to polarized features, that is, features that are most commonly associated with one particular class, which can then be dealt with by providing more training data to balance the over-sensitivity, or by adjusting the architecture of the model (e.g. more regularisation). Detecting such over-sensitivities is an important diagnostic step in ensuring the safety of these AI systems.

4.8 Relation with Abductive Explanations and Anchors

Our OREs have similarities with the *abduction-based explanations (ABEs)* of [58], see Chapter 3, in that they also derive minimal-cost explanations with robustness guarantees. For an input text $t = (w_1, \ldots, w_l)$, let $C = \bigwedge_{i=1}^{l} \chi_i = x_{w_i}$ be the *cube* representing the embedding of t, where χ_i is a variable denoting the i-th feature of x. Let $\mathcal{N}$ represent the logical encoding of the classifier M, and $\hat{y}$ be the formula representing the output of $\mathcal{N}$ given $\chi_1, \ldots, \chi_l$.

Definition 4.2 (Abduction-based Explanation [58]) *An* abduction-based explanation (ABE) *is a minimal cost subset C^* of C such that $C^* \wedge \mathcal{N} \models \hat{y}$.*

As shown in Proposition 4.1, OREs are *also* minimal cost subsets of this form.

The key difference with ABEs is that our OREs are robust to *bounded* perturbations of the excluded features, while ABEs must be robust to *any* possible perturbation. This is an important difference because it is hard (often impossible) to guarantee prediction invariance with respect to the entire input space when this space is continuous and high-dimensional, like in NLP embeddings. In other words, if for our NLP tasks we allowed any word-level perturbation as in ABEs, in most cases the resulting OREs will be of the trivial kind, $E^* = F$ (or $C^* = C$), and thus of little use. For example, if we consider ϵ-ball perturbations and the review ``‘‘the gorgeously elaborate continuation of the lord of the rings’’``, the resulting smallest-size explanation is of the trivial kind (it contains the whole review) already at $\epsilon = 0.1$. For more detail, see [74].

Anchors [101] are a state-of-the-art method for ML explanations. Given a perturbation distribution $\mathcal{D}$, classifier f and input x, an anchor A is a predicate over the input features such that $A(x)$ holds and A has high *precision* and *coverage*, defined next.

$$\mathsf{prec}(A) = P_{\mathcal{D}(x'|A(x'))}(f(x) = f(x')); \ \mathsf{cov}(A) = P_{\mathcal{D}(x')}(A(x')) \tag{4.7}$$

In other words, $\mathsf{prec}(A)$ is the probability that the prediction is invariant for any perturbation x' to which explanation A applies. In this sense, precision can be intended as a robustness probability. $\mathsf{cov}(A)$ is the probability that explanation A applies to a perturbation. To discuss the relation between Anchors and OREs, for an input text t, consider an arbitrary distribution $\mathcal{D}$ with support in $\mathcal{B}_\emptyset(t)$ (the set of all possible text-level perturbations), see Equation 4.1, and consider anchors A defined as subsets E of the input features F, i.e., $A_E(x) = \bigwedge_{w \in E} x_w = \mathcal{E}(w)$. Then, our OREs benefit from the following properties.

Proposition 4.3 *If E is a robust explanation, then* $\mathsf{prec}(A_E) = 1$.

Proof 4.3 *A robust explanation $E \subseteq F$ guarantees prediction invariance for any $x' \in \mathcal{B}_E(t)$, i.e., for any x' (in the support of $\mathcal{D}$) to which anchor A_E applies.*

Note that, when $\mathcal{D}$ is continuous, $\mathsf{cov}(A_E)$ is always zero unless $E = \emptyset$, in which case $\mathsf{cov}(A_\emptyset) = 1$ (as $A_\emptyset = \mathsf{true}$). Indeed, for $E \neq \emptyset$, the set $\{x' \mid A_E(x')\}$ has $|E|$ fewer degrees of freedom than the support of $\mathcal{D}$, and thus has both measure and coverage equal to zero. We therefore illustrate the next property assuming that $\mathcal{D}$ is discrete (when $\mathcal{D}$ is continuous, the following still applies to any empirical approximation of $\mathcal{D}$).

Proposition 4.4 *If $E \subseteq E'$, then $\mathsf{cov}(A_E) \geq \mathsf{cov}(A_{E'})$.*

Proof 4.4 *For discrete $\mathcal{D}$ with probability mass function $pmf_\mathcal{D}$, we can express $\mathsf{cov}(A_E)$ as*

$$\mathsf{cov}(A_E) \quad = \sum_{x' \in supp(\mathcal{D})} pmf_\mathcal{D}(x') \, \cdot \, \mathbf{1}_{A_E(x')} \quad = \sum_{x' \in supp(\mathcal{D})} pmf_\mathcal{D}(x') \, \cdot \, \prod_{w \in E} \mathbf{1}_{x'_w = \mathcal{E}(w)}$$

To see that, for $E' \supseteq E$, $\mathsf{cov}(A_{E'}) \leq \mathsf{cov}(A_E)$, observe that $\mathsf{cov}(A_{E'})$ can be expressed as

$$\mathsf{cov}(A_{E'}) = \sum_{x' \in supp(\mathcal{D})} pmf_\mathcal{D}(x') \cdot \prod_{w \in E'} \mathbf{1}_{x'_w = \mathcal{E}(w)} =$$
$$\sum_{x' \in supp(\mathcal{D})} pmf_\mathcal{D}(x') \cdot \prod_{w \in E} \mathbf{1}_{x'_w = \mathcal{E}(w)} \cdot \prod_{w \in E \setminus E'} \mathbf{1}_{x'_w = \mathcal{E}(w)}$$

and that for any x', $\prod_{w \in E \setminus E'} \mathbf{1}_{x'_w = \mathcal{E}(w)} \leq 1$.

The above proposition suggests that using a uniform $\mathcal{C}$, i.e., minimizing the explanation's length, is a sensible strategy to obtain high-coverage OREs.

4.9 Solution Algorithms

In our published work [74], we present two solution algorithms to derive OREs, respectively based on the hitting-set (HS) paradigm of [58] and the minimum satisfying assignment (MSA) algorithm of [30]. The results in this thesis use the HS based algorithm, so only the details of this algorithm will be discussed. Please see [74] for details of the second algorithm. Both algorithms rely on repeated entailment/robustness checks $\mathcal{B}_{F\setminus E}(t) \wedge E \wedge \mathcal{N} \models \hat{y}$ for a candidate explanation $E \subset C$. For this check, we employ two state-of-the-art neural network verification tools, Marabou [61] and Neurify [125]: they both give provably correct answers and, when the entailment is not satisfied, produce a counterexample $x' \in \mathcal{B}_E(t)$, i.e., a perturbation that agrees with E and such that $B \wedge C' \wedge \mathcal{N} \not\models \hat{y}$, where C' is the cube representing x'. The implementation of this entailment check involves encoding the network and input into SMT, as described in Section 4.5.

4.9.1 Minimum Hitting Set Algorithm

For a counterexample C', let I' be the set of feature variables where C' does not agree with C (the cube representing the input). Then, every explanation E that satisfies the entailment must hit all such sets I' built for any counterexamples C' [57]. Thus, the HS paradigm iteratively checks candidates E built by selecting the subset of C whose variables form a minimum HS (with respect to cost $\mathcal{C}$) of said I's.

The hitting set paradigm [58] exploits the relationship between diagnoses and conflicts [98]: the idea is to collect perturbations and to calculate on their indices a minimum hitting set (MHS), i.e., a minimum-cost explanation. We extend this framework to find a word-level explanation for non-trivial NLP models. At each iteration of Algorithm 1, a minimum hitting set E is extracted (line 3) from the (initially empty, line 1) set Γ. If function *Entails* evaluates to *False* (i.e., the neural network f is provably safe against perturbations on the set of features identified by $F \setminus E$) the procedure terminates and E is returned as an ORE. Otherwise, (at least) one feasible attack is computed on $F \setminus E$ and added to Γ (lines 7-8): the routine then re-starts. Differently from [58], as we have experienced that many OREs with a large perturbation space – i.e. when ϵ or k are large – do not terminate in a reasonable amount of time, we have extended the *vanilla* hitting set approach by introducing the *SparseAttacks* function (line 7). At each iteration *SparseAttacks* introduces into the hitting set Γ a large number of sparse adversarial attacks on the set of features $F \setminus E$. It is known [57] that attacks that use as few features as possible help convergence on instances that are hard (intuitively, a small set is harder to "hit" and hence contributes substantially to the optimal solution compared to a longer one).

The *SparseAttacks* procedure is based on random search and is inspired by recent works in image recognition and malware detection [25]: pseudo-code is reported in Algorithm 2, while a detailed description follows in the next paragraph.

Sparse Adversarial Attacks Without sparse adversarial attacks, we found that this method often struggles to converge for our NLP models, especially with large perturbations spaces (i.e., large ϵ or k). We solved this problem by extending the HS approach with a sub-routine that generates batches of *sparse adversarial attacks* for the input C. This has a two-fold benefit: 1) we reduce the number of entailment queries required to produce counterexamples, and 2) sparsity results in small I' sets, which further improves convergence.

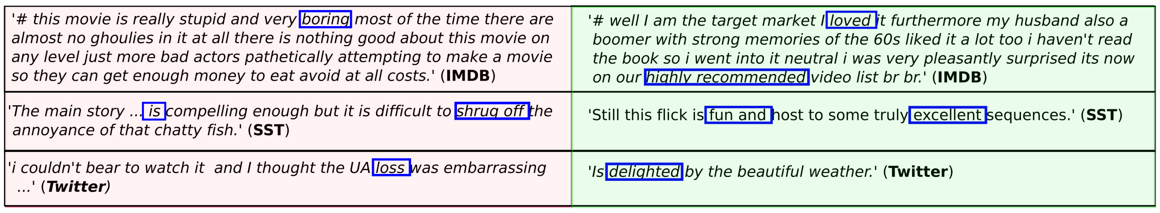
$GeneratePerturbations(k, n, \mathcal{B}, f)$ (line 2) returns a random population of n perturbations that succeed at changing f's classification: for each successful attack p, a subset of k out of d features has been perturbed through a Fast Gradient Sign attack (FGSM) [45], whilst ensuring that the point lies inside the convex region (perturbation set) $\mathcal{B}$. If no perturbation is found in this way (i.e., population size of the attacks is zero, as in line 3), the iterations budget is decreased (line 4) and another trial of $GeneratePerturbations(k, n, \mathcal{B}, f)$ is performed (e.g., with fewer features as targets and a different random seed to guide the attacks). Function $AccuracyDrop(f, P)$ returns the best perturbation a, where k is increasingly minimised (line 7). The algorithm terminates when either no attacks are possible (all the combinations of features have been explored) or after a fixed number of iterations has been performed (line 1).

Algorithm 1: ORE computation via minimum hitting sets and sparse attacks

```
Data: a neural network $f$ and its encoding $\mathcal{N}$, the input text $t$, the initial
      set of features $F$, a network prediction $\hat{y}$ , a cost function $\mathcal{C}$ against
      which the explanation is minimised
Result: an optimal ORE E
1  $\Gamma = \emptyset$
2  while $true$ do
3      $E = MinimumHS(\Gamma, \mathcal{C})$
4      if $Entails((E \wedge \mathcal{B}_{F \setminus E}(t) \wedge \mathcal{N}), \hat{y})$ then
5          return $E$
6      else
7          $A = SparseAttacks(f, F, E, \mathcal{B})$
8          $\Gamma = \Gamma \cup \{A\}$
```

Algorithm 2: Computing a successful perturbation that minimises the number of perturbed features.

```
Data: a neural network $f$, the initial set of features $F$, explanation $E$,
      number of initial perturbations $k \in \mathbb{Z}^+$ (default 1000), perturbation
      set $\mathcal{B}$, number of elements generated at each iteration $n \in \mathbb{Z}^+$ (default
      100), number of iteration before stopping $budget$ (default 1000)
1  while $k > 0 \;\wedge\; budget > 0$ do
2      $P \leftarrow GeneratePerturbations(k, n, \mathcal{B}, E, f)$
3      if $length(P) == 0$ then
4          $budget \leftarrow budget - 1$
5          continue
6      end
7      $a \leftarrow \arg\max_{p \in P} AccuracyDrop(M, P)$
8      $k \leftarrow k - 1$, $budget \leftarrow budget - 1$
9  end
10 return $a$
```



'# this movie is really stupid and very boring most of the time there are almost no ghoulies in it at all there is nothing good about this movie on any level just more bad actors pathetically attempting to make a movie so they can get enough money to eat avoid at all costs.' (**IMDB**)	'# well I am the target market I loved it furthermore my husband also a boomer with strong memories of the 60s liked it a lot too i haven't read the book so i went into it neutral i was very pleasantly surprised its now on our highly recommended video list br br.' (**IMDB**)
'The main story ... is compelling enough but it is difficult to shrug off the annoyance of that chatty fish.' (**SST**)	'Still this flick is fun and host to some truly excellent sequences.' (**SST**)
'i couldn't bear to watch it and I thought the UA loss was embarrassing ...' (***Twitter***)	'Is delighted by the beautiful weather.' (**Twitter**)

Figure 4.3: OREs for IMDB, SST and Twitter datasets (all the texts are correctly classified), that show high quality OREs. Models employed are FC with 50 input words each with accuracies respectively 0.89, 0.77 and 0.75. OREs are highlighted in blue, positive classifications are green and negative are red. Technique used is kNN boxes with k=15. The cost function used here is uniform.

4.10 Experimental Results

The following section describes the various results we have achieved using OREs, and reiterates how they can be used to ensure safety. Experiments were performed on the SST, IMDB and Twitter sentiment analysis datasets, using both fully connected and CNN models. Full details of the experimental setup used in this section can be found in Appendix A.2; however, we once again stress that OREs are not restricted to NLP applications.

4.10.1 Effect of Classifier Accuracy and Robustness

We find that our approach generally results in meaningful and compact explanations for NLP. In Figure 4.3, we show a few OREs extracted for *negative* and *positive* texts, where the returned OREs are both concise and semantically consistent with the predicted sentiment, using the uniform cost function. However, the quality of our OREs depends on that of the underlying classifier. Indeed, enhanced models with better accuracy and/or trained on longer inputs tend to produce higher quality OREs. We show this in Figure 4.4 (longer inputs), Figure 4.5 (fully connected vs convolutional networks), and Figure 4.6 (low vs high accuracy), where we observe that enhanced models tend to result in more semantically consistent explanations. For lower-quality models, some OREs include seemingly irrelevant terms (e.g., *"film"*, *"and"*, in Figure 4.5), thus exhibiting shortcomings of the classifier. When we uncover such irrelevant terms in the explanation of a given classifier, we are then able to use

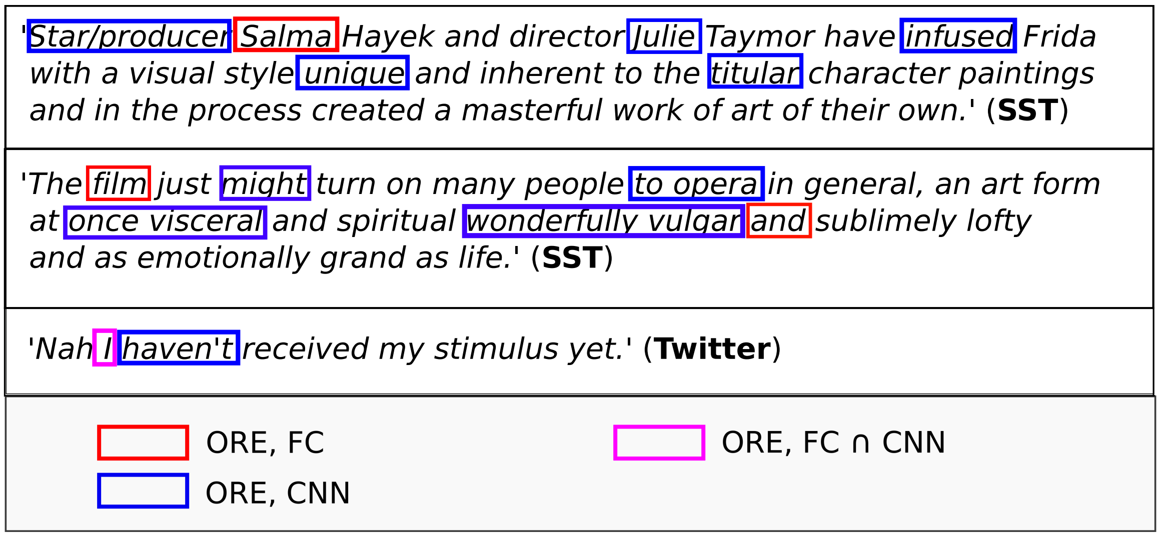
'# I've seen Foxy Brown, Coffy Friday Foster Bucktown, and Black Mama White Mama of these this is Pam Griers worst movie poor acting bad script boring action scenes theres just nothing there avoid this and rent Friday Foster Coffy or Foxy Brown instead' (**IMDB, predicted as *negative***)
*'# I gave this a 2 and it only avoided a 1 because of the occasional unintentional laugh the film is excruciatingly. Boring and incredibly cheap its even worse if you know anything at all about the Fantastic Four.', (**IMDB, predicted as negative**)*
'# a few words for the people here in cine club the worst crap ever seen on this honorable cinema a very poor script a very bad actors and a very bad movie dont waste your time looking this movie see the very good or any movie have been good commented by me say no more' (**IMDB, predicted as *negative*)**

Figure 4.4: Examples of high quality OREs generated using long inputs (highlighted in blue). OREs were extracted using kNN boxes with 25 neighbours per word: fixing words in an ORE guarantees that the model is locally robust. The cost function was uniform. The examples come from the IMDB dataset, and the model employed is an FC network with 100 input words (accuracy 0.81).

techniques such as data augmentation to retrain our classifier and therefore improve safety.

4.10.2 Detecting Bias

As per Proposition 4.2, we can apply exclude constraints to detect biased decisions. We do this by using a cost function that assigns the cost of each word to be 1, except for any protected features, where by 'protected' we mean, for example, proper names, which are set to a cost greater than the cumulative cost of all other words in the input. In Figure 4.7, we provide a few example instances exhibiting such a bias, i.e., where *any* robust explanation contains at least one protected feature. These OREs include proper names that should not constitute a sufficient reason for the model's classification. When we try to exclude proper names, no robust explanation exists, indicating that a decision bias exists. We note that the type of bias we are detecting here is direct bias, and not proxy bias [36].

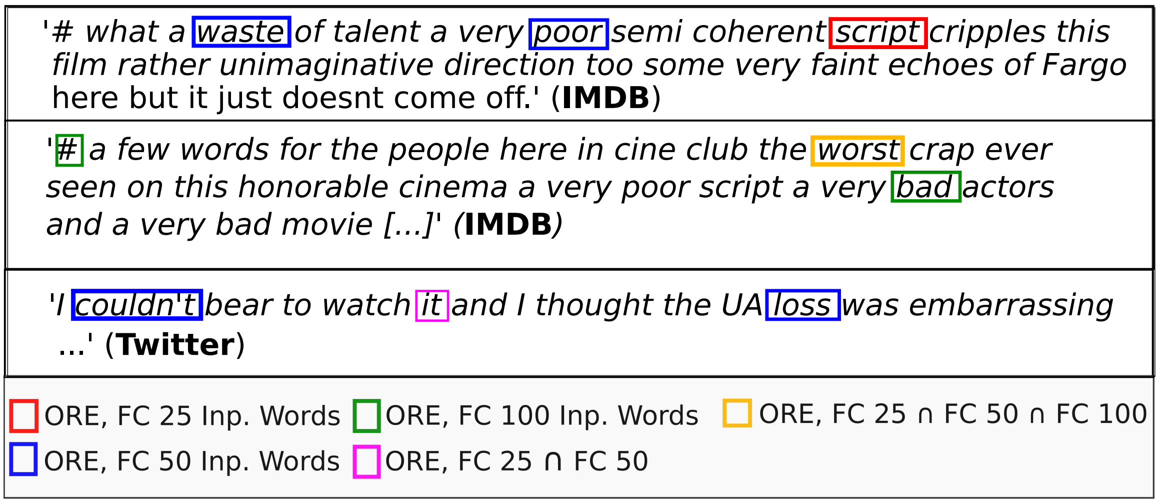

'*Star/producer Salma Hayek and director Julie Taymor have infused Frida with a visual style unique and inherent to the titular character paintings and in the process created a masterful work of art of their own.*' (**SST**)

'*The film just might turn on many people to opera in general, an art form at once visceral and spiritual wonderfully vulgar and sublimely lofty and as emotionally grand as life.*' (**SST**)

'*Nah I haven't received my stimulus yet.*' (**Twitter**)

🟥 ORE, FC
🟦 ORE, CNN
🟪 ORE, FC ∩ CNN

Figure 4.5: Comparison of OREs (with the uniform cost function) for SST and Twitter texts on FC (red) vs CNN (blue) models (common words in magenta). The first two are *positive* reviews, the third is *negative* (all correctly classified). Accuracies of FC and CNN models are, respectively, 0.88 and 0.89 on SST, 0.77 (both) on Twitter. Models have an input length of 25 words and OREs are extracted with kNN boxes ($k = 25$).

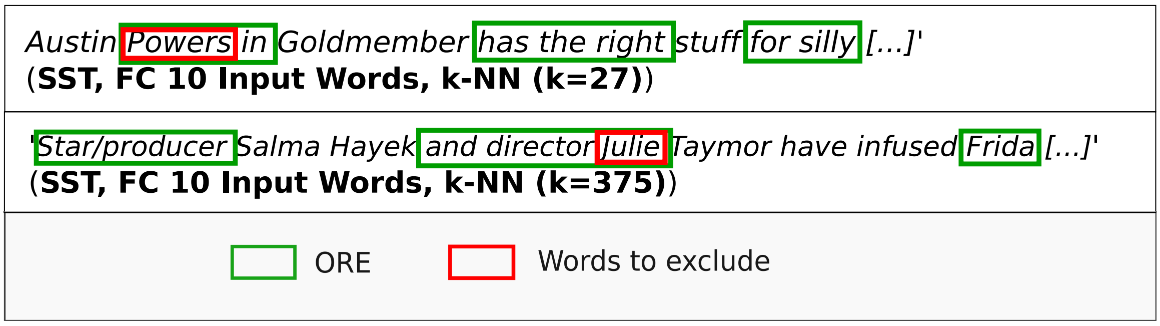

'*# what a waste of talent a very poor semi coherent script cripples this film rather unimaginative direction too some very faint echoes of Fargo* here but it just doesnt come off.' (**IMDB**)

'*# a few words for the people here in cine club the worst crap ever seen on this honorable cinema a very poor script a very bad actors and a very bad movie [...]*' (**IMDB**)

'*I couldn't bear to watch it and I thought the UA loss was embarrassing* ...' (**Twitter**)

🟥 ORE, FC 25 Inp. Words
🟦 ORE, FC 50 Inp. Words
🟩 ORE, FC 100 Inp. Words
🟪 ORE, FC 25 ∩ FC 50
🟧 ORE, FC 25 ∩ FC 50 ∩ FC 100

Figure 4.6: Comparison of OREs on *negative* IMDB and Twitter inputs for varying accuracies of FC models. The first and third examples are trained with 25 (red) vs 50 (blue) input words (words in common to both OREs are in magenta). The second example additionally uses an FC model trained with 100 input words (words in common to all three OREs are in orange). Accuracy is respectively 0.7, 0.77 and 0.81 for IMDB, and 0.77 for both Twitter models. All the examples are classified correctly. OREs are extracted with kNN boxes ($k = 25$), and the uniform cost function.

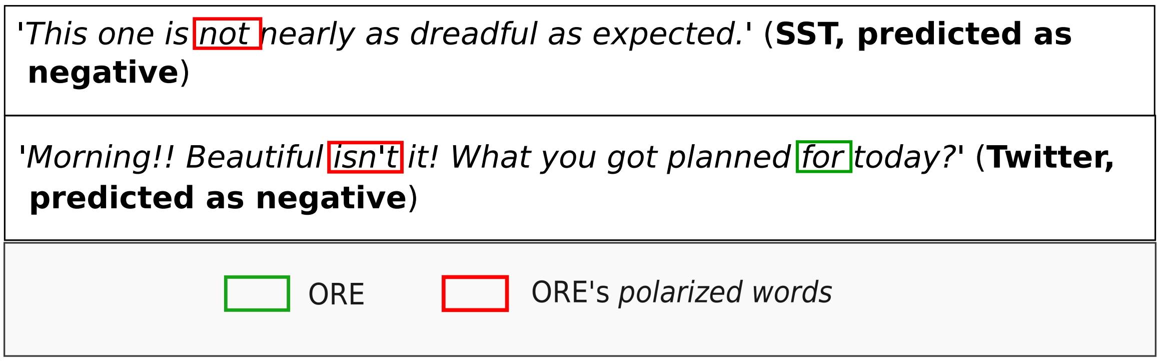


'*Austin Powers in Goldmember has the right stuff for silly [...]*'
(**SST, FC 10 Input Words, k-NN (k=27)**)

'*Star/producer Salma Hayek and director Julie Taymor have infused Frida [...]*'
(**SST, FC 10 Input Words, k-NN (k=375)**)

🟩 ORE 🟥 Words to exclude

Figure 4.7: Two examples of decision bias from an FC model with an accuracy of 0.80.



'*This one is not nearly as dreadful as expected.*' (**SST, predicted as negative**)

'*Morning!! Beautiful isn't it! What you got planned for today?*' (**Twitter, predicted as negative**)

🟩 ORE 🟥 ORE's *polarized words*

Figure 4.8: Two examples of over-sensitivity to polarized terms (in red). Other words in the OREs are highlighted in green. Models used are FC with 25 input words (accuracy 0.82 and 0.74). Method used is kNN with k respectively equal to 8 and 10.

4.10.3 Debugging Prediction Errors

An important use-case for OREs is when a model commits a *misclassification*. Misclassifications in sentiment analysis tasks usually depend on over-sensitivity of the model to polarized terms. In this sense, knowing a minimal, sufficient reason behind the model's prediction can be useful to debug it. As shown in the first example in Figure 4.8, the model cannot recognize the *double negation* constituted by the terms *not* and *dreadful* as a syntax construct, hence it exploits the negation term *not* to classify the review as *negative*.

4.10.4 Varying Cost Functions

As previously mentioned, the uniform cost function (where all words are assigned a cost of 1) produces OREs of minimal length. Whilst OREs of minimal length are still considered useful for human decision makers, it may be more desirable to customise the cost function so that explanations become more meaningful in the specific context they are applied in.

We consider three more cost functions as follows:

- $\mathcal{C}_{\text{SIMPLE}}$: A simple cost function where PAD tokens, solitary punctuation and words not in the embedding are set to $+\infty$ (in implementation, this is set to the maximum value of a integer) and all others are set to 1.

- $\mathcal{C}_{\text{PROB}}$: A probabilistic cost function, equal to $1/\text{prec}(w)$ where $\text{prec}(w)$ is the probability (with respect to the data distribution) that input feature w being included in an input corresponds to the same prediction as the current input for the network of interest f.

- $\mathcal{C}_{\text{HYBRID}}$: A hybrid of the two previous cost functions, where the cost for PAD tokens and words not in the embedding are set to $+\infty$, and the cost for everything else is probabilistic as above.

The intuition behind $\mathcal{C}_{\text{SIMPLE}}$ is that we are able to remove uninformative information from the explanation, and more meaningful explanations (still sufficient to imply the prediction) are retrieved for the human decision maker. The intuition behind $\mathcal{C}_{\text{PROB}}$ is that this cost function will prioritise words that are polarised to specific classifications. This is useful if the model designer wishes to check whether the model is over-sensitive to such polarised terms, and correct the model accordingly. The hybrid cost function combines the benefits of both the previous cost functions.

In order to demonstrate these cost functions in action, it is useful to generate *all* of the minimum cost explanations for a given input, so that we can view all possible sufficient reasons and compare between cost functions. To do this, we generate all possible candidate explanations of the same minimal cost as the first ORE uncovered, and then run the entailment check to verify that the candidate is also an ORE. Example 1 demonstrates how $\mathcal{C}_{\text{SIMPLE}}$ removes unimportant tokens from the resulting ORE(s). The input texts are movie reviews from the SST dataset and we use the

ϵ-ball perturbation set with $\epsilon = 0.05$.

Example 1 $\mathcal{C}_{\text{SIMPLE}}$ removes PAD tokens from OREs:

```
Input: ['strange', 'funny', 'twisted', 'brilliant', 'and', 'macabre', '<PAD
   >', '<PAD>', '<PAD>', '<PAD>']

Uniform cost (5 OREs, minimal cost = 6):
['strange', 'funny', 'twisted', 'brilliant', '<PAD>', '<PAD>']
['strange', 'funny', 'twisted', '<PAD>', '<PAD>', '<PAD>']
['strange', 'twisted', 'brilliant', '<PAD>', '<PAD>', '<PAD>']
['strange', 'twisted', 'and', '<PAD>', '<PAD>', '<PAD>']
['strange', 'twisted', 'macabre', '<PAD>', '<PAD>', '<PAD>']

Simple cost (1 ORE, minimal cost = 106.0):
['strange', 'funny', 'twisted', 'brilliant', 'and', 'macabre']
```

In Example 2, we demonstrate how each of the non-uniform cost functions improves the quality of the OREs generated. The experimental setup is the same as in Example 1. Note that both $\mathcal{C}_{\text{PROB}}$ and $\mathcal{C}_{\text{HYBRID}}$ produce the same result in this case, as there are no words in the input that $\mathcal{C}_{\text{SIMPLE}}$ applies to that have not already been given a cost by $\mathcal{C}_{\text{PROB}}$, resulting in the same outcome.

Example 2 Non-uniform costs gives better quality OREs:

```
Input: ['...', 'spellbinding', 'fun', 'and', 'deliciously', 'exploitative',
    '<PAD>', '<PAD>', '<PAD>', '<PAD>']

Uniform cost (13 OREs, minimal cost = 2):
['...', 'spellbinding']
['...', 'and']
['...', '<PAD>']
['...', '<PAD>']
['...', '<PAD>']
```

```
['spellbinding', '<PAD>']
['spellbinding', '<PAD>']
['fun', '<PAD>']
['and', '<PAD>']
['and', '<PAD>']
['exploitative', '<PAD>']
['<PAD>', '<PAD>']
['<PAD>', '<PAD>']

Simple cost (10 OREs, minimal cost = 3.0): ['spellbinding', 'fun', 'and']
['spellbinding', 'fun', 'deliciously']
['spellbinding', 'fun', 'exploitative']
['spellbinding', 'and', 'deliciously']
['spellbinding', 'and', 'exploitative']
['spellbinding', 'deliciously', 'exploitative']
['fun', 'and', 'deliciously']
['fun', 'and', 'exploitative']
['fun', 'deliciously', 'exploitative']
['and', 'deliciously', 'exploitative']

Probabilistic cost (6 OREs, minimal cost = 2.92):
['spellbinding', 'fun', 'and']
['spellbinding', 'and', 'deliciously']
['spellbinding', 'and', 'exploitative']
['fun', 'and', 'deliciously']
['fun', 'and', 'exploitative']
['and', 'deliciously', 'exploitative']

Hybrid cost (6 OREs, minimal cost = 2.92):
['spellbinding', 'fun', 'and']
['spellbinding', 'and', 'deliciously']
['spellbinding', 'and', 'exploitative']
```

```
['fun', 'and', 'deliciously']
['fun', 'and', 'exploitative']
['and', 'deliciously', 'exploitative']
```

In our final example, Example 3, we show how $\mathcal{C}_{\text{PROB}}$ generates shorter and more relevant explanations, by removing conjunctions and similar uninformative types of word. The same experimental setup as in Examples 1 and 2 is used.

Example 3 $\mathcal{C}_{\text{PROB}}$ gives shorter and more relevant explanations:

```
Input: ['it', 'is', 'a', 'satisfying', 'summer', 'blockbuster', 'and', '
    worth', 'a', 'look']

Uniform cost (9 OREs, minimal cost = 1):
['it']
['is']
['a']
['satisfying']
['summer']
['blockbuster']
['and']
['a']
['look']

Simple cost (9 OREs, minimal cost = 1.0):
['it']
['is']
['a']
['satisfying']
['summer']
['blockbuster']
['and']
```

	Average # OREs		
Cost	$\epsilon = 0.01$	$\epsilon = 0.05$	$\epsilon = 0.1$
Uniform	32.67	29.34	12.82
Simple	22.44	17.26	10.19
Prob	16.56	6.95	5.30
Hybrid	9.78	4.17	3.64

Table 4.1: A comparison of uniform, simple, probabilistic and hybrid cost functions, w.r.t. the average number of OREs that each input generates. The ϵ-ball perturbation set is used, with values of ϵ specified in the table.

```
['a']

['look']


Probabilistic cost (4 OREs, minimal cost = 1.0):

['satisfying']

['summer']

['blockbuster']

['look']
```

We note that the total number of OREs for a given input decreases as the complexity of the cost function increases. Having fewer OREs per input is desirable, as there is then a smaller (but more distilled, and more informative) volume of information for the human decision maker to consider. Table 4.1 shows how the choice of cost function affects the number of possible OREs, averaged over the test set.

4.10.5 Comparison to Anchors

We evaluate the robustness of Anchors for FC and CNN models on the SST and Twitter datasets. Accuracies are 0.89 for FC with SST, 0.82 for FC with Twitter, 0.89 for CNN with SST, and 0.77 for CNN with Twitter. To compute robustness, we assume a kNN-box perturbation space $\mathcal{B}$ with $k = 15$ for FC and $k = 25$ for CNN models. To extract Anchors, we set $\mathcal{D}$ to the standard perturbation distribution of [101], defined by a set of context-wise perturbations generated by a powerful language model. Thus, defined $\mathcal{B}$s are small compared to the support of $\mathcal{D}$, and so one would expect high-precision Anchors to be relatively robust with respect to said $\mathcal{B}$s.

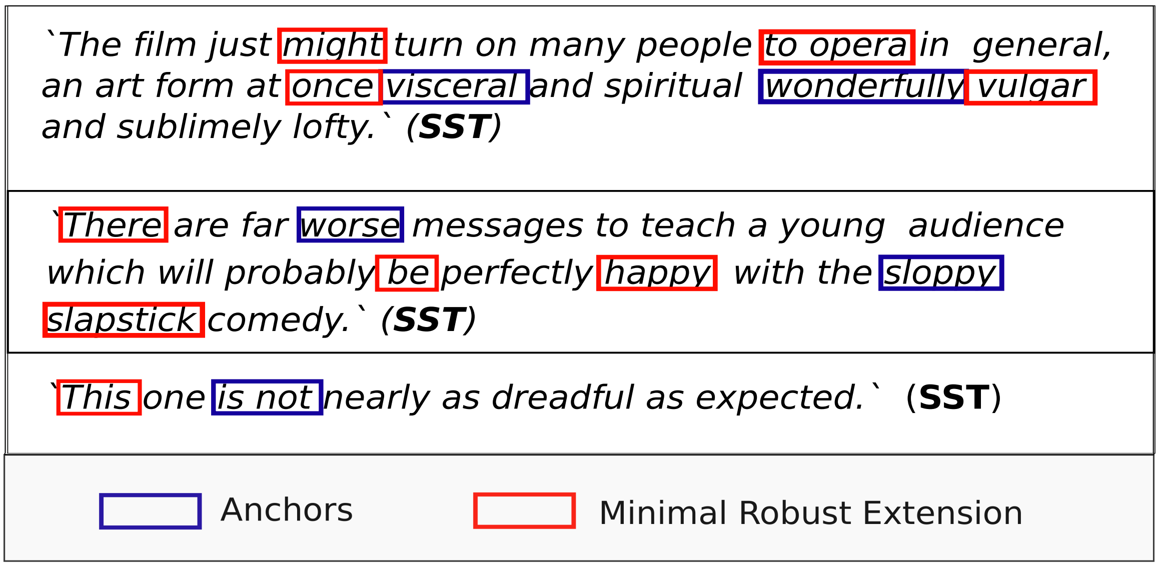
On the contrary, the Anchors extracted for the FC models attain an average precision of 0.996 on SST and 0.975 on Twitter, but only 12.5% of them are robust for the SST case and 7.5% for Twitter. With CNN models, high-quality Anchors are even more brittle: 0% of Anchors are robust on SST reviews and 5.4% on Twitter, despite an average precision of 0.995 and 0.971, respectively.

We remark, however, that Anchors are not designed to provide such robustness guarantees. Our approach becomes useful in this context, because it can *minimally extend* any local explanation to make it robust, by using *include constraints* as explained in Section 4.6. To do this, we add include constraints for all of the features of the non-formal local explanation, and then run our ORE solution algorithm as normal. In Figure 4.9 we show a few examples of how, starting from non-robust Anchors explanations, our algorithm can find the minimum number of words to make them provably robust. Obviously, such a minimal robust extension increases the length of the original explanation: OREs are designed to prioritize robustness over length (other design principles could be valid as well but are beyond the scope of this work).

4.10.6 Computational Performance and ϵ Choice

We have found a few instances where distilling an ORE from an ϵ-bounded input was computationally infeasible (i.e. our method hit the initial timeout of $7200s$ within the solver, and when extended to $86,400s$, 24 hours, still reached a timeout), thus motivating us to develop and use the kNN-boxes technique for the majority of the results in this chapter. In Figure 4.10 we compare how OREs grow for increasing values of ϵ and k (i.e., the control parameters of respectively ϵ-balls and kNN-boxes).

With a perturbation set defined as an ϵ-ball around each input vector, Table 4.2 shows ORE length and average execution time for the MHS method of generating OREs, using the cost function $\mathcal{C}_{\text{SIMPLE}}$. There is a tradeoff when selecting the value of ϵ, too small and the explanation may become too short to be useful (or even empty, indicating that the input is completely robust within that ϵ-ball) or too large and the


`The film just might turn on many people to opera in general, an art form at once visceral and spiritual wonderfully vulgar and sublimely lofty.` (**SST**)

`There are far worse messages to teach a young audience which will probably be perfectly happy with the sloppy slapstick comedy.` (**SST**)

`This one is not nearly as dreadful as expected.` (**SST**)

Figure 4.9: Examples of Anchors explanations (in blue) along with the minimal extension required to make them robust (in red). Examples are classified (without errors) with a 25-input-word CNN (accuracy 0.89). OREs are extracted for kNN boxes and k=25.

ϵ	**Explanation Length**	**MHS Execution Time**
0.01	5 ± 5	63.70 ± 63.69
0.05	5.5 ± 4.5	339.96 ± 334.66
0.1	7.5 ± 2.5	3563.4 ± 3535.84

Table 4.2: Execution time for the MHS algorithm for different values of ϵ, and the corresponding explanation length.

explanation may become equal to the entire input (as seen with the abduction-based explanation method [58]). In practice, we use a refinement search over the parameter space to decided on a value of ϵ that is applicable for each situation.

Figure 4.11 shows how using adversarial attacks speeds up convergence of the hitting set based algorithm, where the timeout was set to 7200s (two hours).

Finally, we note that we were not able to consider inputs longer than 25 words, or embeddings of dimensions larger than 100 due to the lack of scalability of the black-box solvers used (such inputs timeout in every case, even when the timeout is set to over 24 hours). As already discussed, improvements in solver technology will directly benefit our methods and allow us to scale further than currently possible.

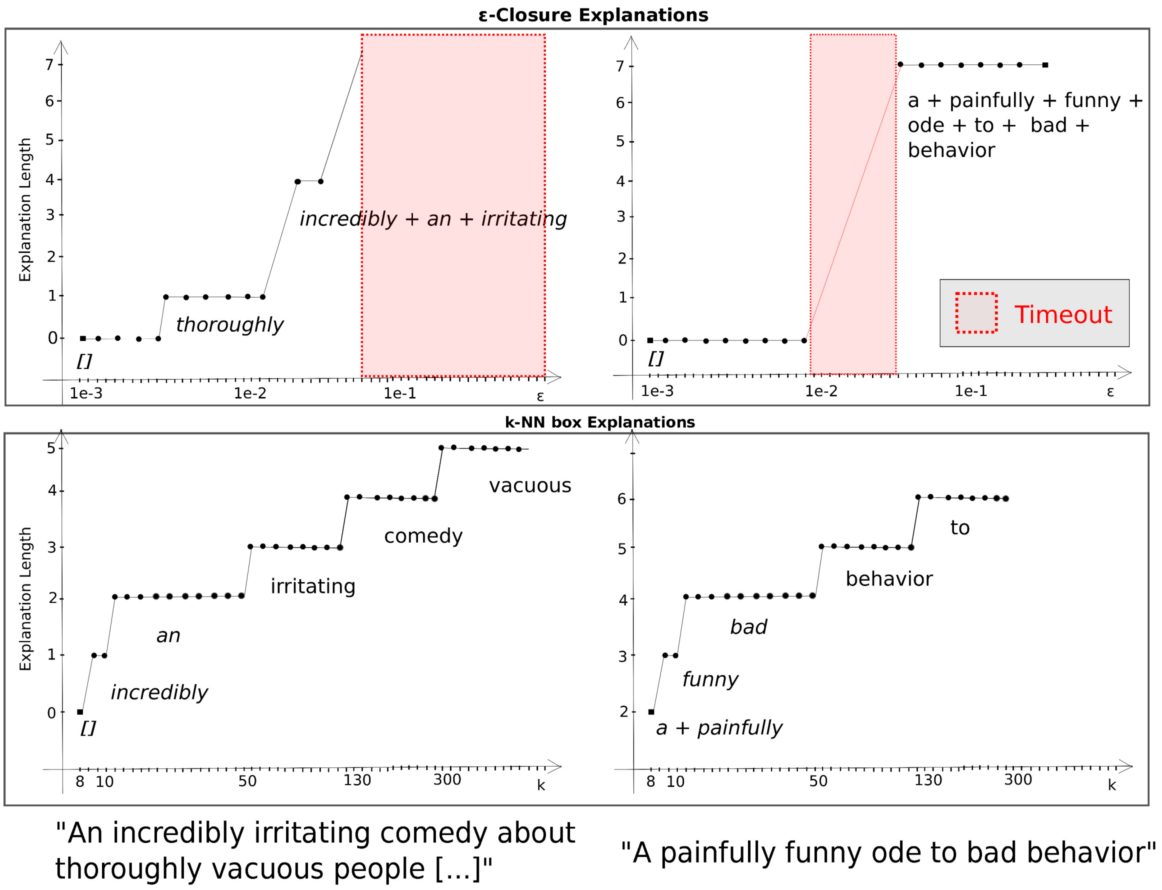


Figure 4.10: How an explanation grows when either ϵ (top) or k (bottom) is increased. Model considered is an FC model with 50 input words on SST dataset (0.89 accuracy), the cost function is $\mathcal{C}_{\text{SIMPLE}}$. On the left is a *negative* review that is correctly classified, on the right is a *positive* review that is misclassified (i.e., the model's prediction is *negative*). For specific ranges of ϵ, timeout was reached (timeout of $7200s$, highlighted in red).

	INPUT INSTANCE	**EXECUTION TIME [s]** **HS *Vanilla*, HS + Adversarial Attacks**
ε = 0.05	'insanely hilarious!'	**87.66, 8.67**
ε = 0.05	'this one is not nearly as dreadful as expected'	**114.99, 10.49**
ε = 0.05	'this one is baaaaad movie!'	**Timeout, 79.2**
ε = 0.1	'so your entire day was spent doing chores ay??!!' [...]	**Timeout, 1520.80**
ε = 0.05	'I just seen ur tweetz plz write bak' [...]	**Timeout, 159.11**

Figure 4.11: Examples of explanations that were enabled by the adversarial attacks routine for the MHS based method, along with the time taken. Timeout was set to 2 hours, $7200s$.

4.11 Summary and Discussion

We have introduced optimal robust explanations (OREs) and applied them to enhance usefulness of NLP models, for use in ensuring safety of AI systems. OREs provide concise and sufficient reasons for a particular prediction, as they are guaranteed to be both minimal with respect to a given cost function and robust, in that the prediction is invariant for any bounded replacement of the left-out features. These properties combined are key in enabling safer AI models, as detecting biased decisions, debugging misclassifications, and repairing non-robust explanations are some of key use cases that our OREs enable. We have presented a solution algorithm (along with another in [74]) that builds on the relationship between our OREs and abduction-based explanations. We have demonstrated the usefulness of our approach on widely-adopted sentiment analysis tasks, providing explanations for neural network models beyond reach for existing formal explainers. However, we stress that OREs are applicable to many domains other than NLP, and any advancements in solver technology will directly benefit our methods.

This chapter has shown how we can improve the safety of deterministic decision functions using optimal robust explanations. However, these explanations alone are not sufficient to provide the level of safety that is required in many common AI applications, for example, autonomous driving. Using OREs and the inherent property that the explanation implies the decision, we could, for example, check that the car always uses the pixels of a traffic light to decide whether to stop (something that is required by the laws of the road), or that the car never considers the colour of a car when overtaking (something you would not want to have an effect on decision making). On the other hand, OREs do not give us any information about how sure the model is of its prediction, and if the model only weakly predicts one class over another, the resulting explanation may be nonsensical to a human (and the human would not understand why this was). Bayesian neural networks, networks with a prior distribution over their weights, have the ability to capture the uncertainty within the network and enable safer decision making. The next chapter explores how we can improve the safety of AI applications using Bayesian networks and the uncertainty information that they capture.

Chapter 5

Bayesian Uncertainty and Safety

In this chapter, we consider the role of uncertainty in ensuring AI safety. Following on from Chapter 4, we note that explaining the decision of a deterministic neural network alone certainly helps with safety assurance, but is not sufficient to achieve the high level of safety required in many cases. In this chapter, we focus on a setting where uncertainty measures are returned along with the model decision and study safety in the presence of uncertainty. More specifically, we consider deep Bayesian neural networks as end-to-end controllers in safety-critical systems such as self driving, and explore the use of uncertainty in ensuring safety. This chapter introduces methods for evaluating the safety of these controllers by defining two safety measures with statistical guarantees, and then explores their use in practical applications.

5.1 Motivation

In this chapter, we consider decision functions of the form $f : X \to Y$ with added decision confidence measures (via uncertainty information), described as setting B in Section 1.1. Feature vectors X, for demonstration purposes, are in the form of images (both grayscale and colour).

Bayesian neural networks (BNNs) conform to the above definition of a decision function. A BNN, as defined in Section 2.3, is a neural network with a prior distribution on its weights. BNNs have the ability to capture the uncertainty within the learning model, while retaining the main advantages intrinsic to deep neural networks [83]. As a consequence, they are particularly appealing for safety-critical applications, such as autonomous driving, where uncertainty estimates can be propagated through the decision pipeline to enable safe decision making [85].

Recall that an end-to-end controller is a controller in which the entire process involves a single neural network without modularisation. The input to the network is sensor data, and the output is motor actuation. In the context of self-driving, the sensors may include camera input from one or more cameras (often facing front straight, front left, front right and rearwards), infrared sensors, LiDAR sensors and many more. The outputs are typically a steering angle, either relative or absolute, and braking and throttle values. In this thesis we work with Nvidia's PilotNet [11] as an example of an end-to-end controller, which we extend to a Bayesian neural network.

To illustrate the role of uncertainty in the context of end-to-end-controllers, we return to the example discussed in the Introduction, of a self-driving car that, while driving, observes an obstacle in the middle of the road. Then, the controller may be uncertain on the steering angle to apply and, in order to avoid the obstacle, may choose angles which turn the car either right or left, with equal probability at each timestep. If we consider the optimal decision according to this steering angle distribution and a squared loss, then the controller will simply select the mean value of the distribution [8] and aim straight at the obstacle. However, if the network can provide an uncertainty measure in addition to a control action then we can utilise this information in the context steering the car. We refer to this as uncertainty-aware decision making.

In this chapter we focus on the setting of Bayesian end-to-end controllers for autonomous driving and propose safety measures that can be used for safety assurance. In particular, we consider *probabilistic safety*, which is the probability that the controller will keep the system safe for a given time horizon, and *real-time decision confidence*, which is the probability that the BNN is certain of a given decision. We demonstrate how to obtain *a priori* statistical guarantees on the safety of the application of the BNN in both online and offline scenarios, and evaluate the quality of uncertainty measures provided by different BNN inference techniques. While we focus on autonomous driving, our methods can be configured with any simulator for any control task and assumes that paths can be sampled efficiently and are endowed with a probability measure.

5.2 Modelling Approach

We model the autonomous driving scenario considered in this chapter as a discrete-time stochastic control process $(\mathbf{x}_k, k \in \mathbb{N})$ [42]. A discrete-time stochastic process is a collection of random variables (see Section 2.1.1), indexed by a countable set, in this case the natural numbers $\mathbb{N}$, and a discrete-time stochastic *control* process is a mathematical framework for modelling decision making in situations in which the outcomes are partly random and partly controlled by a decision maker. For example, the commonly used Markov decision process (MDP) and its partially observable generalisation (POMDP) are discrete-time stochastic control processes.

$\mathbf{x}_k$ is a probabilistic model that describes the status of the entire system and it takes values in a state space $\mathcal{X}$, which includes information on the position, velocity and acceleration of the car, as well as that of all the other vehicles, pedestrians and obstacles in the environment. Intuitively, in this chapter, $\mathbf{x}_k$ just represents a white-box system that we assume we can simulate arbitrarily many times.

The control space of the process, which represents the set of variables a controller can modify to drive the behaviour of $\mathbf{x}_k$, is denoted by $\mathcal{U} \subseteq \mathbb{R}^m$ and is typically given by steering angle, braking and acceleration values of the autonomous vehicle; however, more complicated systems may have many more controllable parameters. We assume the controller can only observe a noisy image of the state space coming from the available sensors. These sensors might include camera input, infrared (IR) sensors, light detection and range sensors (LiDAR), GPS, or a combination of these in addition to many others depending on the simulator used. Due to this noise, and the fact that a typical sensor on a car does not have an infinitely large range and cannot observe the whole environment, $\mathbf{x}_k$ is only *partially observable*. This means that a controller will never have a full view of the current state of the system.

We denote by $\mathcal{O}$ the observation space, which is the set of all possible observations. Intuitively, given the current state of $\mathbf{x}_k$, the controller receives an observation of $\mathbf{x}_k$, $o \in \mathcal{O}$, and synthesizes an action $u \in \mathcal{U}$ based on this observation. Then, $\mathbf{x}_k$ transitions to a new state at time $k+1$. Given action u, the evolution of $\mathbf{x}_k$ is probabilistic, as traffic, weather conditions, noise and other variables are uncertain.

A (memoryless and deterministic) control strategy for $\mathbf{x}_k$, $\pi : \mathcal{O} \to \mathcal{U}$, associates an action with a given observation. In this chapter, as explained in detail in the next section, we train a BNN controller to synthesize this control strategy π.

We denote a *path* of $\mathbf{x}_k$ by $\omega : \mathbb{N} \to \mathcal{X} \times \mathcal{U}$. ω is a sequence of states and actions in an execution of the system. In the self-driving example, an execution of the system means driving (or letting the car drive) for some amount of time within the simulator and recording the state and action information. Given a strategy π we assume there exists a well-defined probability measure over the paths of $\mathbf{x}$ such that, for $X \subseteq \mathcal{X}$, $P(\omega(k) \in X|\pi)$ is the probability that $\mathbf{x}_k$ is in X at time k given π.

For instance, this probability measure is well-defined for POMDPs [18]. However, note that the uncertainty quantification techniques derived in this chapter will work also for more general, possibly non-Markov, processes.

5.3 Safety Measures For Autonomous Driving

In this section we formulate two key properties that can be evaluated to provide safety assurance of autonomous driving controllers in offline and online settings.

5.3.1 Probabilistic Safety

The first problem we consider in Definition 5.1 is that of computing the probability that a given strategy π synthesized by the BNN keeps the system safe. In the autonomous driving domain, one interpretation of "safe" could mean keeping the car within the bounds of the lane it is in, but we stress that our framework allows for alternative definitions of "safe", in any BNN control scenario.

This probability can be used to certify that a given controller is safe with high probability given the available information. Computing this value can be done in any simulator. Prior to the deployment of a safety-critical controller of some kind, it is common for large companies to evaluate the safety of specific test cases, often in proprietary simulators that the general public do not have access to [99]. As a consequence, we believe that a quantifiable notion of the safety of a given controller is pivotal in order to certify a controller, especially if this incorporates learning elements.

Definition 5.1 *(Probabilistic Safety) Let $X \subseteq \mathcal{X}$ be a safe set, ω denote a path of $\mathbf{x}_k$, $[0,T] \subseteq \mathbb{N}$ be a time horizon, and π be a control strategy synthesised by the BNN. Compute*

$$\begin{aligned}&\eta_1 \equiv P(\phi_1(\pi, \omega, [0,T])),\\ &\quad \textit{where } \phi_1(\pi, \omega, [0,T]) = \forall k \in [0,T], \omega(k) \in X \mid \pi\end{aligned}$$

Then, for $\delta > 0$, we say that π is δ-safe in $[0,T]$ iff $\eta_1 \geq \delta$.

η_1 is satisfied if the probability that a path of $\mathbf{x}_k$ is safe during the interval $[0,T]$ is greater than or equal to a threshold δ. Note that similar probabilistic measures of safety are widely used to certify cyber-physical system models [1, 12].

An illustration of how Definition 5.1 could work for checking the safety of a BNN-controlled car along a single lane road can be seen in Figure 5.1.

5.3.2 Real-time Decision Confidence

As explained in greater detail in the next section, in order to synthesize a control strategy π we train a BNN and obtain that, for an observation (in our implementation, an image) $o \in \mathcal{O}$, $\pi(o)$ is determined by the BNN predictions. However, notice that $\pi(o)$ is still deterministic. Hence, it does not take into account the uncertainty in the model predictions, which is intrinsic in the BNN and could be used to quantify the confidence of the model in its decisions. To tackle this issue, for $o \in \mathcal{O}$, in the

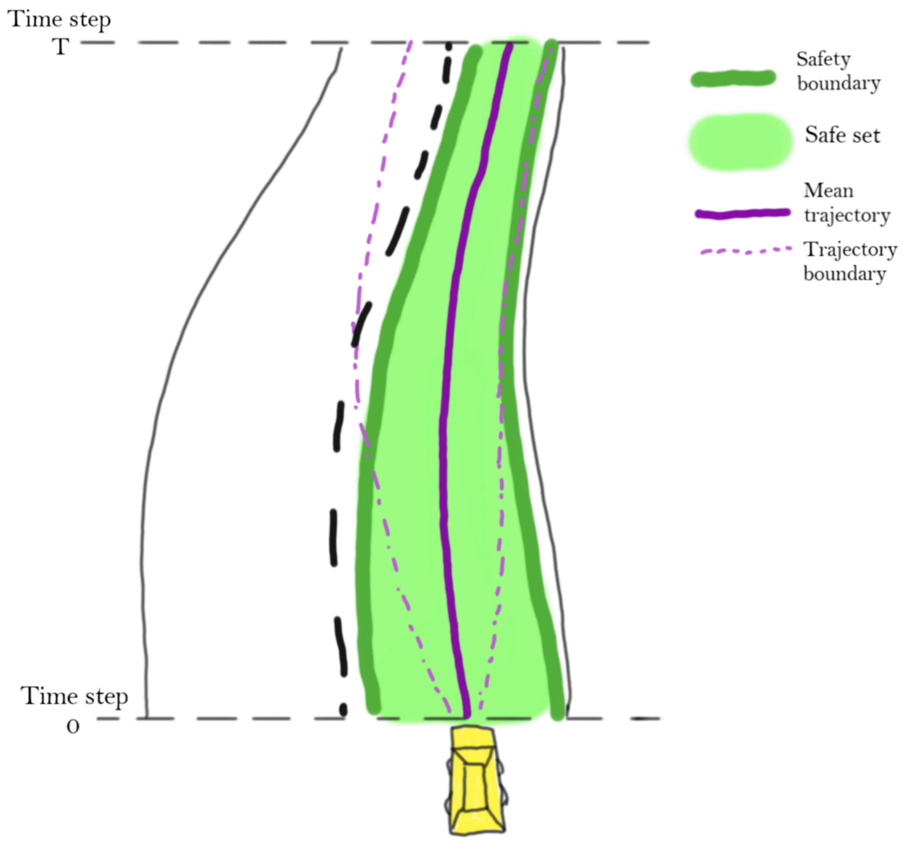


Figure 5.1: An illustration of how Definition 5.1 works on a single lane road scenario. A trajectory is δ-safe if the probability of the car remaining within the safe set (the correct lane) for a given time horizon T is greater than or equal to δ.

following definition, we consider a notion of trust of $\pi(o)$ based on the probability mass of the BNN around $\pi(o)$. The following problem is stated for regression tasks, but can be trivially extended to classification problems by changing the definition of S^{o_k} to be a user-defined set of safe classifications for observation o_k (most likely the singleton set containing only o_k, but may be a larger set if the class labels are ordinal).

Definition 5.2 *(Real-time decision confidence) Given $\epsilon > 0$, let o_k be the observation received at time k, w a weight sampled from $\mathbf{w}$, and $S^{o_k} = \{u \in \mathcal{U} \; s.t. \; |u - \pi(o_k)| \leq \epsilon\}$. Compute*

$$\eta_2 = P\big(\phi_2(S^{o_k}, f^w(o_k))\big),$$
$$\textit{where } \phi_2(S^{o_k}, f^w(o_k)) \equiv f^w(o_k) \in S^{o_k}.$$

Then, we say that the decision at time k is δ-confident iff $\eta_2 \geq \delta$.

Note that the probability measure in the above definition comes from the distribution of the weights in the BNN. In fact, by definition of probability, we can equivalently write $\eta_2 = \mathbb{E}_{w\sim\mathbf{w}}[\mathbf{1}_{f^w(o_k)\in S^{o_k}}]$, where $\mathbf{1}_E$ is the indicator function for event E. Hence, real-time decision confidence, as defined in Definition 5.2, seeks to compute the probability mass in a ϵ-ball around $\pi(o)$ and classify a decision as certain if the resulting probability is greater than or equal to a threshold. Definition 5.2 can be violated either when there is high uncertainty (i.e., variance is large) or when the control distribution is multivariate and the most likely mean of $P(f^{\mathbf{w}})(o)$ is far from $\pi(o)$, see Figure 5.2 for an illustration of this.

In Section 5.6 we show that this measure of uncertainty can be employed together with commonly used measures of uncertainty, such as *mutual information* [108], to quantify in real time the degree that the model is confident in its predictions and can thus offer a notion of trust in its predictions. In the next section we consider a statistical framework that allows us to compute the measures of Definition 5.2 and 5.1 with guarantees in terms of confidence intervals.

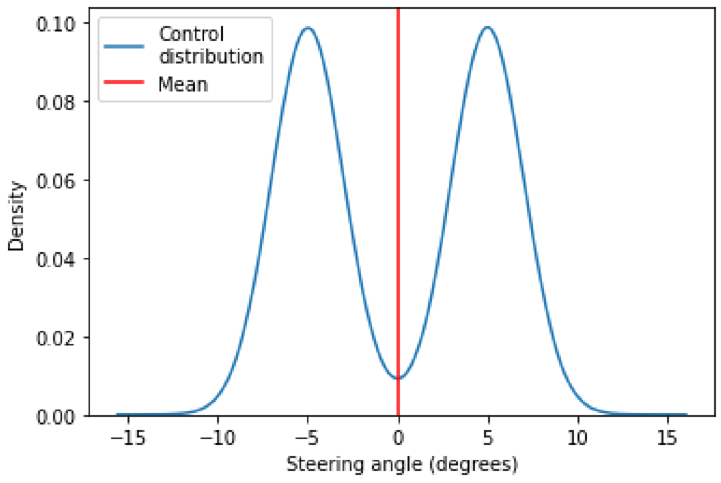


Figure 5.2: An example of a multivariate control distribution, where the mean is far from either preferred steering angle.

5.3.3 Probabilistic Guarantees

In [17], a solution is given to *estimate* the probability p defined in Equation 2.42 (local probabilistic robustness). The solution, described below, provides statistical guarantees on the accuracy of the probability estimation: it ensures arbitrarily small estimation error with arbitrarily large confidence.

This method, based on the observation that a sample of the weights of the BNN induces a deterministic neural network, uses existing formal verification techniques to decide the satisfaction of ϕ for each deterministic sample. Given a BNN $f^{\mathbf{W}}$, ϕ is verified over deterministic networks f^w where w is a weight vector sampled from the posterior distribution $P(w|\mathcal{D})$ (or its approximation $q(w)$, depending on the inference method used).

Following on from this, we can see the satisfaction of $\phi(f^{\mathbf{W}})$ as a Bernoulli random variable Z, which can be sampled by sampling the BNN weights and verifying the resulting deterministic network. Then, we compute the probability p that ϕ is true with respect to the BNN, by computing the expected value of the Bernoulli random variable $\mathrm{E}[Z]$. An estimator $\hat{p}$ for p is defined as in Equation 5.1.

$$\hat{p} = \frac{1}{n}\sum_{i=1}^{n} \phi(f^{w_i}) \approx \mathrm{E}[Z] = p \tag{5.1}$$

Since we want to provide statistical guarantees, $\hat{p}$ must satisfy the following *a priori* statistical guarantee, for an arbitrary absolute error bound $0 < \theta < 1$ and arbitrary confidence $0 < \gamma \leq 1$:

$$P(|\hat{p} - p| > \theta) \leq \gamma \tag{5.2}$$

One method to ensure that Equation 5.2 is satisfied is to use Chernoff bounds to determine the sample size n required for a given θ and γ. $\hat{p}$ satisfies Equation 5.2 after n samples if Equation 5.3 holds. In practice, a different method using Massart bounds [17] is commonly used, as Chernoff bounds are often more conservative.

$$n > \frac{1}{2\theta^2}\log(\frac{2}{\gamma}) \tag{5.3}$$

Using all of the ingredients above, the estimation method can be performed as outlined in Algorithm 3, which is a simplified version of the same algorithm found in [17].

Algorithm 3: BNN local robustness estimation

```
  Input: $x, T, f, P(w|\mathcal{D}), \phi, \theta, \gamma$
  Output: $\hat{p}$
1 $n^C \leftarrow$ number of samples by Chernoff bounds using $\theta$ and $\gamma$
2 $n, k \leftarrow 0, 0$
3 while $n < n^C$ do
4     $w \leftarrow$ sample from $P(w|\mathcal{D})$
5     SAT $\leftarrow$ verify $\phi$ over $f^w$
6     $k \leftarrow k+$ SAT; $n \leftarrow n + 1$
7     $\hat{p} \leftarrow k/n$
8 end
9 return $\hat{p}$
```

The inputs to the algorithm are the test point x, the search space for adversarial inputs T (in this case the neighbourhood around x), the BNN f, the posterior distribution (or estimation of the posterior) $P(w|\mathcal{D})$, robustness property ϕ as defined in Equation 2.42, and Chernoff bound parameters θ and γ. The maximum number of iterations of the algorithm is set to the value of the Chernoff bound for error bound

θ and confidence γ. On iteration n, weights w are sampled from the posterior distribution and then property ϕ is verified over the deterministic network f^w to obtain the n-th Bernoulli sample (the `SAT` variable). This algorithm is independent of the deterministic verification method used, but the reachability method detailed in [17] is used. The number of successes k and number of trials n are then updated, and used to update the estimator $\hat{p}$. If $n < n^C$, then the algorithm continues looping. However, if enough samples have been taken (according to the Chernoff bound), then the `while` loop terminates and the estimator $\hat{p}$ is returned.

5.4 Statistical Methods for Safety Evaluation

For the computation of properties η_1 and η_2 of Definition 5.2 and 5.1, we consider statistical methods, inspired by the techniques developed for statistical analysis of probabilistic models [17, 76]. In particular, we observe that the satisfaction of both ϕ_1 and ϕ_2 can be seen as Bernoulli random variables (see Section 2.1.1), which we can observe by sampling from $\mathbf{w}$, the weights of the BNN in the case of real-time decision confidence, and by sampling $\mathbf{x}_k$ in the case of probabilistic safety. After we collect N samples of each random variable, we can build the following empirical estimators

$$\hat{\eta}_1 = \frac{1}{N}\sum_{i=1}^{n} \phi_1(\pi, \omega_i, [0, T])$$
$$\hat{\eta}_2 = \frac{1}{N}\sum_{i=1}^{n} \phi_2(\pi(o), S^o, f^{w_i}(o)),$$

where $\{w_1, ..., w_n\}$ are weights sampled from $\mathbf{w}$ and $\{\omega_1, ..., \omega_n\}$ are paths sampled from $\mathbf{x}$. Then, for an arbitrary absolute error bound $0 < \theta < 1$ and confidence $0 < \gamma \leq 1$, we obtain that if Equation 5.3 is true, then for $i \in \{1, 2\}$, it holds that

$$P(|\hat{\eta}_i - \eta_i| > \theta) \leq \gamma. \tag{5.4}$$

The above bound is based on Chernoff bounds [20], but other sequential schemes, potentially requiring fewer samples, could be employed [17]. However, the bound in

Figure 5.3: An example of an image from the CARLA simulator.

Equation 5.3 has the advantage to allow one to determine the required sample size n for a given precision before performing the experiments. Hence, it can be trivially parallelised by assigning one sample per process, and no inter-process communication is needed.

5.5 Bayesian End-to-end Controllers For Autonomous Driving

We evaluate our methods on experiments performed on the CARLA driving simulator [31], where we consider a deep end-to-end controller given by a modified Nvidia's PilotNet (formerly known as DAVE-2) neural network architecture [10], which we train with three different BNN inference methods, Monte Carlo dropout [40], mean-field variational inference [9], and Hamiltonian Monte Carlo [13]. We consider different training scenarios, including obstacle avoidance and driving on a roundabout, demonstrating how to quantify the uncertainty of the controller's decisions and utilise uncertainty thresholds in order to guarantee the safety of the self-driving car with high probability.

5.5.1 Experimental Setup

CARLA The experiments in this paper use the CARLA simulator, a state-of-the-art, open-source simulator for autonomous driving research [31]. CARLA has been developed specifically to support the training of autonomous driving systems and is completely open-source. The environment CARLA provides ranges from simple country roads to full city simulations, including other vehicles, cyclists, pedestrians, emergency services, fully dynamic weather, working traffic lights and more. Importantly, the 3D models created for the CARLA simulator mean that images taken within the simulator are very similar to the real world. Finally, CARLA provides some autonomous driving baselines within it, and allows the user to generate training data using an "autopilot", which knows the whole state of the environment. However, we stress that any simulator can be used within this framework, assuming it can simulate car trajectories and generate images that can be used by the controller, for example the much simpler Udacity simulator [122]. All training data, which consists of (*image, steering angle*) pairs, was acquired within the CARLA simulator, either through manual driving or use of the built-in autopilot.

BNN Controller Architecture The architecture of the neural network models used in this chapter are based on PilotNet [11], Nvidia's first end-to-end controller for self-driving. The architecture, as seen in Figure 5.5, consists of nine layers. The first layer is simply a normalisation layer that is hard coded and not adjusted during learning. This layer takes as input an image split into its YUV planes. The next five layers are convolutional layers that were designed to perform feature extraction. The first three convolutional layers had a 2 x 2 stride with a 5 x 5 kernel, and the last two convolutional layers had a 3 x 3 kernel and no stride. The final layers are all fully connected until reaching the output layer, which consists of a single output indicating the inverse turning radius of the vehicle.

The PilotNet paper [11] does not specify what type of non-linearities were used, nor does it mention the loss function. For the purposes of this thesis, ReLU non-

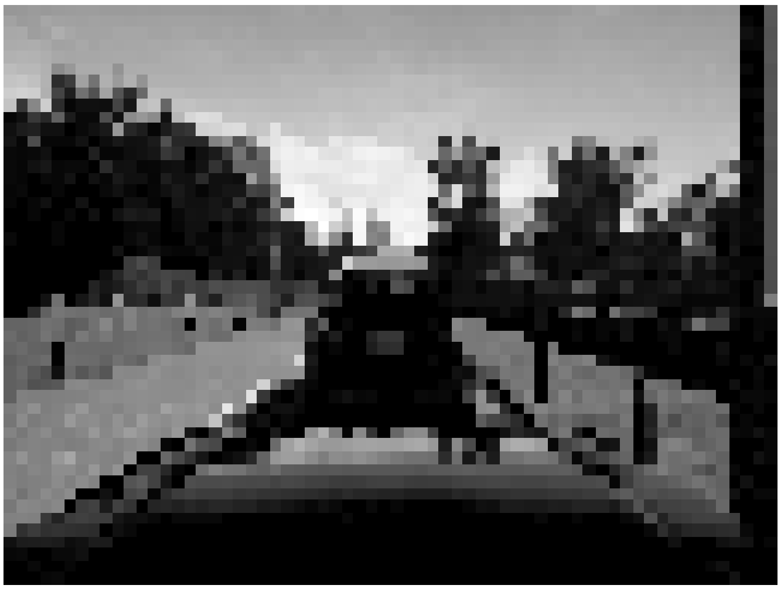


Figure 5.4: An example of an image from the CARLA simulator after preprocessing.

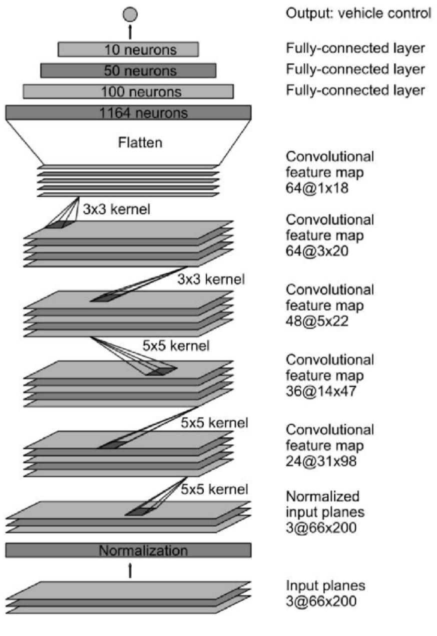


Figure 5.5: The architecture of Nvidia's PilotNet.

linearities were assumed for all but the final layer where arctan was used (as its output is in the range $(-\frac{\pi}{2}, \frac{\pi}{2})$ which matches the steering angle range). The loss function was assumed to be the mean-squared loss.

Full details of the network architectures and training regimes for each inference method can be found in Appendix A.3.

Modelled Scenario and BNN Decisions In the experiments performed in this thesis, we consider a setting where the observation space $\mathcal{O}$ is given by images from a single camera input, placed on the front centre of the car facing forwards. An example image can be seen in Figure 5.3. The control space $\mathcal{U}$ is the steering angle, given between -1 and 1 (corresponding to 90 degrees left and right). Again, we stress that the techniques developed in this chapter are general and not limited to this scenario.

In our experiments, for an observation o we have that $\pi(o)$, the BNN decision, is given by the most likely class. However, we stress that other choices for $\pi(o)$ are possible according to the particular loss function (see e.g., [8]) and the methods presented in this paper are independent of the criteria for assigning $\pi(o)$. For example, $\pi(o)$ could be the mean of n samples of the BNN in the case of regression with the mean squared loss function (called model averaging [39]).

5.6 Experimental Results

In this section, we describe a generic experimental set up for computing the measures in Definition 5.1 and Definition 5.2. We first consider probabilistic safety as defined in Definition 5.1 and we show that this measure can be effectively used in order to identify problematic scenarios in which further data acquisition should occur. We then show that use of the measure in Definition 5.2 in conjunction with classical measures of uncertainty can greatly increase the safety of an autonomous vehicle when it is in unfamiliar scenarios.

5.6.1 Probabilistic Safety Estimates

In order to measure the safety of a BNN controller in a particular setting, one must simulate scenarios (e.g. turns, collision avoidance, intersections) in various conditions in order to satisfy the bound in Equation 5.3. Though we perform simulations in order to test the safety of a turn scenario, running the correct number of simulations with diverse environmental conditions works on any scenario one would like to test. For example, the notion of probabilistic safety is also used to calculate the safety in Figure 5.8. We also study the effect of the BNN inference method used (HMC, VI and MCD) on the quality of the uncertainty measure produced.

Figure 5.6 shows the test setup for probabilistic safety estimates. We place a vehicle approximately 10 meters from the entrance of a roundabout in fixed weather conditions. We then collect training data using the built in autopilot, with the weather set to sunny noon. The autopilot is set to drive the car through the roundabout, taking the first exit. We then use our safety boundaries (marked in the figure) to determine the probability that a specific controller will drive safely, that is, stay within our safety boundaries. We segment the road into small hexagonal sections so that we are left with safety probabilities of visiting that section, for each section of road tested, for each controller.

While we expect the controller to be able to navigate safely from its trained starting point to the end point in the weather it has seen, we seek to test the robustness of posterior distributions to changes in scenery and weather conditions in order to also include simulations of potential worst-case deployment performance. In row (a) of Figure 5.6 we see that, while the variance can be useful in collision avoidance (see the section below), the wide variance of HMC causes a larger proportion of trajectories to fall outside of the safety boundary. The estimated probability of safety for HMC, across all weathers, was 0.766 ($\pm$0.05). Row (b) of Figure 5.6 reports the consistency of VI across different weather conditions with a cumulative safety probability estimate of 0.91 ($\pm$0.05) in this particular test case. The main reason for lack of safety in VI was veering into the center lane of the roundabout, which is arguably safer than the

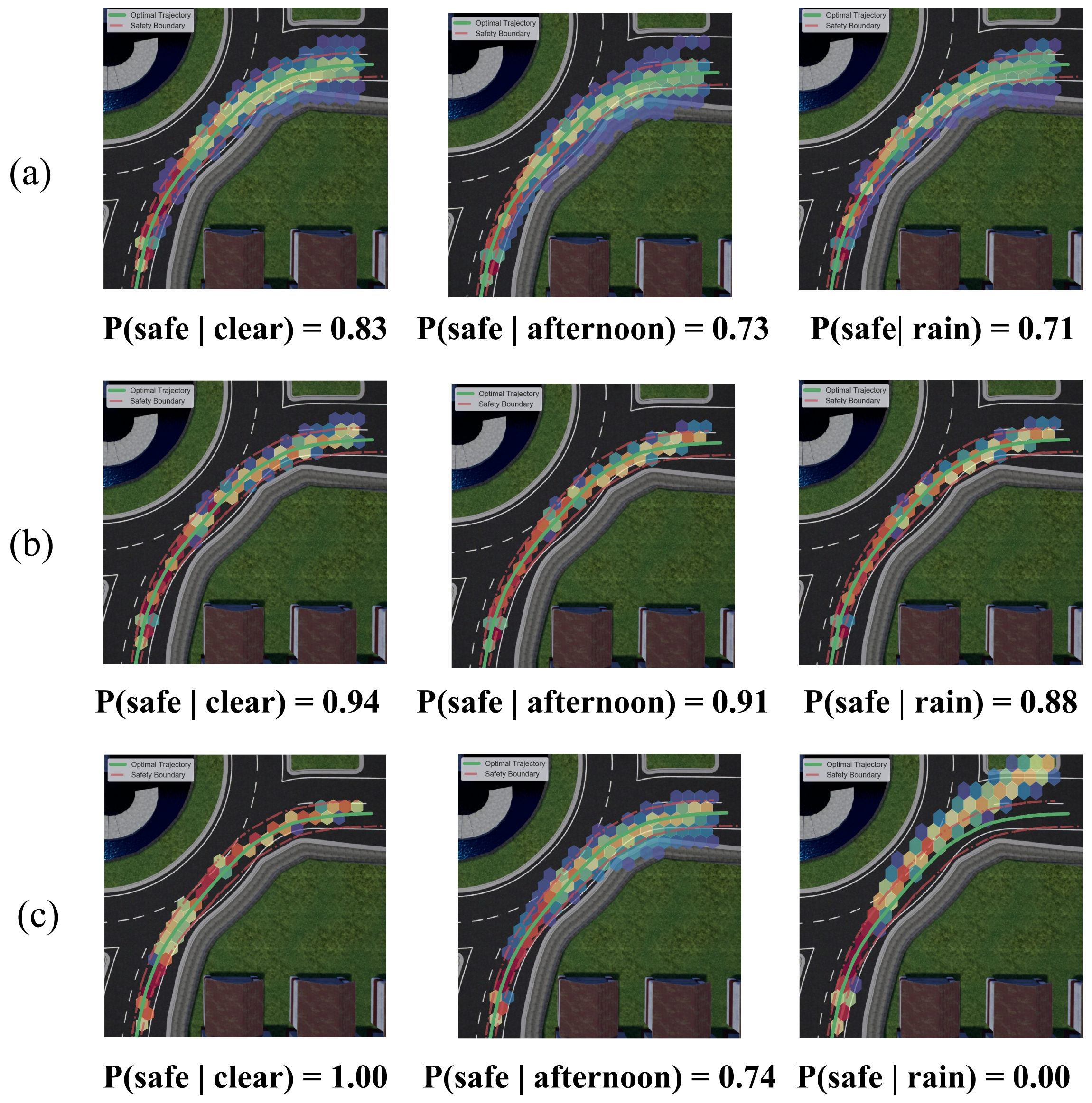


Figure 5.6: Offline safety probabilities of (a) HMC, (b) VI and (c) MCD respectively. Each hexagon is shaded with the probability of the car visiting that area, and the optimal trajectory is plotted with a green line. The red lines show safety boundaries, where outside of this is considered unsafe, and inside is safe.

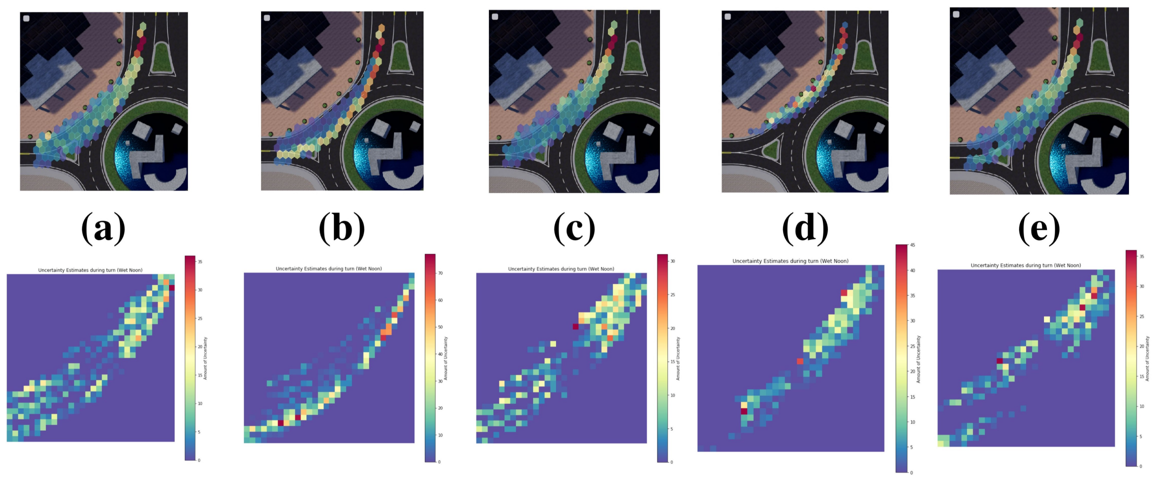


Figure 5.7: Top row: Performance of the self-driving car (HMC controller) on turn C; Bottom row: spatial uncertainty distribution. Columns represent different weather conditions (a) clear noon, (b) cloudy noon, (c) wet noon, (d) heavy rain, (e) wet sunset

HMC controller that veered into the curb and side wall. Finally, in row (c) we see the performance of MCD. In the training environment, it was the only method to achieve a perfect safety score; however, we see the network fails to generalize well to other weather conditions. While MCD performs slightly better than HMC in the more dim light of the afternoon, it fails catastrophically in the rain. MCD's overall probabilistic safety, prior to the consideration of rain, was 0.87. When we factor rainy environments, the overall probabilistic safety of MCD falls to 0.58 ($\pm$0.05). It is likely that if we were to retrain MCD in all weather conditions and re-run the safety analysis we would see a perfect safety score, as we do currently with clear weather. In this way, we can use our offline safety probability as a guide for active learning in order to increase data coverage and scenario representation in training data.

5.6.2 Uncertainty-aware Real-time Collision Avoidance

Figure 5.7 demonstrates that, by monitoring the uncertainty (as defined in Definition 5.2), we can see that in instances prior to important decision making, in this case how to navigate the roundabout, there is a spike in uncertainty. Each column of the figure represents the turn in different weather conditions: clear noon, cloudy

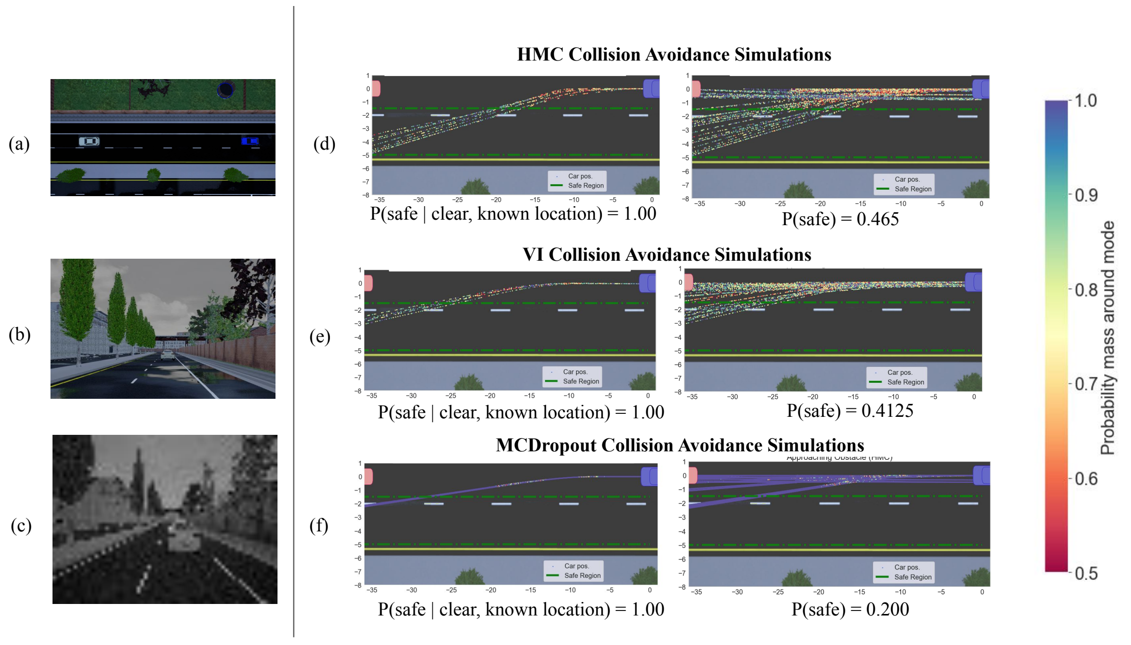


Figure 5.8: To the left of the black line, we visualize the experimental set up. (a) Spatial distribution of cars; (b) original camera signal; (c) input to BNN. In the centre of the figure we plot the course of the vehicle controlled by a BNN, where each row represents a different posterior approximation (d) HMC, (e) VI, (f) MCD. Each dot, representing the position of the car during its trajectory is colored based on the uncertainty of the controller.

noon, wet noon, heavy rain and wet sunset. The top row of the figure shows (for the controller trained with HMC as an example) the trajectory of the car over 273 simulations (for an error margin of 7.5%), where the colour of a hexagon gets further towards red as the probability of visiting it increases. The bottom row shows the uncertainty spatially, as defined in Definition 5.2. It is clear that the top right of each of these spatial uncertainty plots has a higher number of yellow and red cells, indicating that the uncertainty is higher when about to make a decision as opposed to in the middle of one.

Since Definition 5.2 has shown spikes prior to decision making, we explore a new scenario in which the car is presented with an obstacle. In Figure 5.8, we can see an example of a collision avoidance test set up. We place a stationary vehicle 40 meters away from the self-driving car in fixed weather conditions along a single roadway. We then train a BNN controller, for each inference method, on data collected from safe human driving in this scenario. Below, we describe a general algorithm for performing collision avoidance which generalizes to any scenario one would like to test. Further, the system that we use can be implemented for any BNN that is trained to drive autonomously, and can detect situations in which the car is uncertain in order to improve safety.

The uncertainty-aware decision system is designed in two stages. In the first stage, we simulate more runs of the vehicle driving without any collision avoidance system present. We rely only on the learned behaviour of the vehicle (plots of these runs can be seen in Figure 5.8). At this stage, we are able to qualitatively understand the behaviour of each network posterior in terms of the uncertainty it produces as it approaches the obstacle. We note that it is possible, though less desirable, to perform this qualitative evaluation using a held-out, test data set. Because the input we observe at time t depends on all of the decisions made up to that time, generating safety or uncertainty estimates based on another controller's decisions may be inaccurate due to the potentially low probability of ever observing those states with the current controller under consideration. In the second stage, we use the captured information about uncertainty in order to generate actionable warning thresholds. For example, if we see that there is typically a large spike in uncertainty as the car approaches the obstacle, we can use a threshold in order to stop the car when we experience a similar peak in the future. We also use mutual information, a well recognised measure of uncertainty, alongside our real-time decision confidence measure. The behaviour of mutual information can roughly be seen in the bottom left-hand corner of Figure 5.9.

We use a three-tiered warning system based on real-time decision confidence, as defined in Definition 5.2. That is, given an image at time k we bin network decisions into four categories based on the value of η_2. Algorithm 4 describes this uncertainty-aware controller, where the algorithm is run for the observation at each timestep k. The inputs to the algorithm are: a BNN controller $\mathbf{f}^{\mathbf{w}}$ with weights variable $\mathbf{w}$, the radius of the safe space around the prediction ϵ, the observation (input image) o, the warning thresholds $\delta_0, \delta_1, \delta_2$ for warnings 0, 1 and 2, respectively and the number of samples n that should be taken from the BNN. If any of the relevant measures rise above a threshold, the output of the algorithm is a warning, and if no measures are high enough, the algorithm returns ‘safe’.

Algorithm 4: Uncertainty aware BNN control algorithm

```
Data: a BNN $\mathbf{f^w}$ with weight variable $\mathbf{w}$, epsilon $\epsilon$, observation $o$, thresholds
      $\delta_0, \delta_1, \delta_2$ for warnings 0, 1 and 2, respectively, number of samples $n$
Result: a warning (or lack thereof)
1  $t = 0$
2  $c = 0$
3  $\pi(o) = \mathbf{f^w}(o)$
4  $preds = \emptyset$
5  for $i = 1..n$ do
6      $f^w = sample(\mathbf{f^w})$
7      if $f^w(o) \in \{u \ s.t. \ |u - \pi(o)| \leq \epsilon\}$ then
8          $t \mathrel{+}= 1$
9      $c \mathrel{+}= 1$
10     $preds = preds \cup \{f^w(o)\}$
11 $\hat{\eta_2} = \frac{t}{c}$
12 $m = mutual\_inf(preds)$
13 if $\hat{\eta_2} \leq \delta_2$ then
14     throw severe
15 else if $\hat{\eta_2} \leq \delta_1$ then
16     throw standard
17 else if $m \geq \delta_0$ then
18     throw mutual information
19 else
20     return safe
```

Frequently, no warning will be thrown, i.e., $\eta_2 \geq \delta_1$ for a given $\delta_1 \in [0, 1]$. However, in the case that we are less than δ_1-certain ($\eta_2 < \delta_1$), a standard warning (warning 1) is thrown. A severe warning (warning 2) is thrown when the network is less that δ_2-certain (this assumes $\delta_2 < \delta_1$). Finally, we consider a warning (warning 0) which is thrown when neither a severe nor standard warning are thrown ($\eta_2 \geq \delta_1$), but the predictive distribution exhibits high mutual information (calculated as in Section 2.3.5 using the `mutual_inf` function), above yet another threshold, in our case 0.45. For our experiments, the constants δ_1 and δ_2 are set to a threshold of 0.7 and 0.6, respectively. The actions that occur at each of these warnings are also configurable. However, we have set up our system such that mutual information warnings slow down the vehicle,

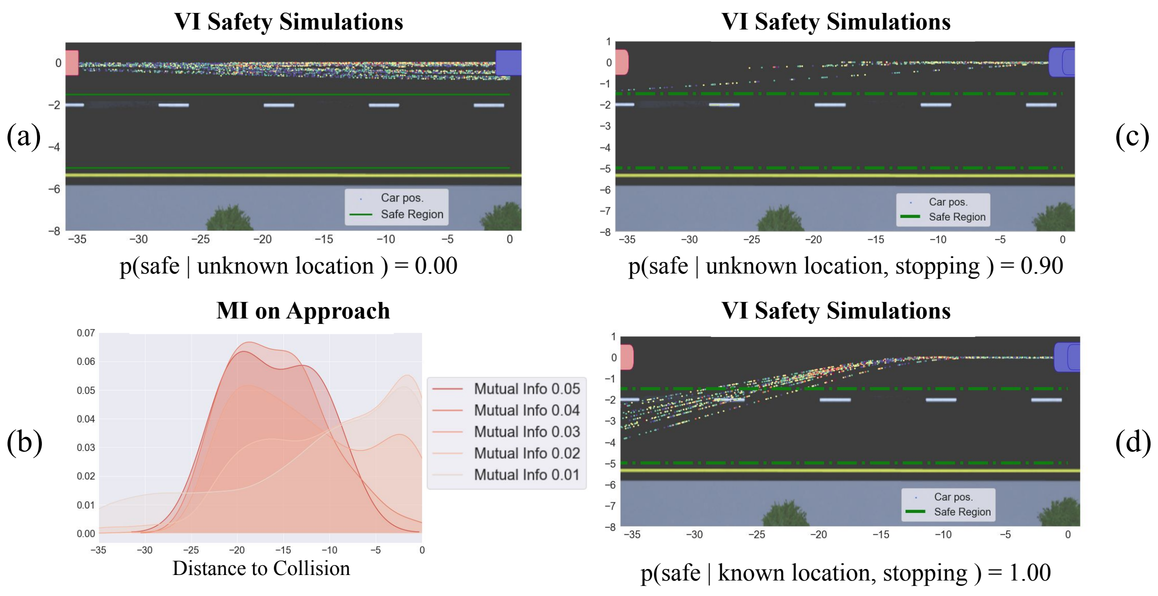


Figure 5.9: Demonstration of how the uncertainty-aware stopping procedure performs. (a) Original safety of VI without stopping algorithm. (b) Mutual information signal spikes as we approach the obstacle. (c) VI safety with stopping. (d) VI performance in a known environment with stopping.

standard warnings slow down the vehicle and alert the operator of potential hazard, and severe warnings cause the car to safely brake and alert the operator that they need to assume control of the vehicle.

Setting these thresholds requires a delicate trade-off between autonomy and safety. If the thresholds are set too low, then the system will operate more autonomously (that is, without asking for user intervention), but may be less safe. Setting the thresholds too high may be safer, but causes the car to operate less autonomously as the user is constantly prompted for input. In Figure 5.9, we show that these sorts of collision avoidance systems can perform well in practice (here demonstrated with the VI controller). We show that we can detect and reduce the rate of collisions (the inverse of probabilistic safety), improving the safety in unknown conditions from 0.00 (± 0.05) to 0.90 (± 0.05), see Figure 5.8. Moreover, we test that implementation of this strategy does not affect the autonomy of the car in known situations. For this we simulate the situation in which the car was trained and we find that the car still operates with safety probability 1.00, with error margin of 0.05 according to

Equation 5.3, and full autonomy.

Finally, Figure 5.10 further demonstrates the warning system for the HMC controller. All plots demonstrate how far away the car is from the obstacle when each type of warning is thrown, and how many warnings are given. The most warnings were thrown around 17m away in clear noon weather, and rainy sunset weather.

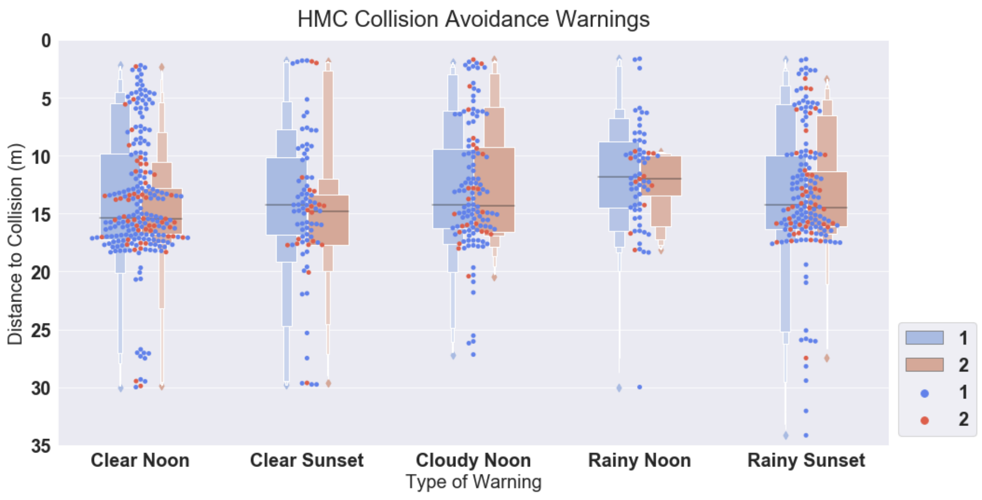
We note that the calculation of real-time uncertainty adds only a minimal amount of overhead (microseconds) into the end-to-end controller, which is not enough to affect its operation.

5.7 Summary and Discussion

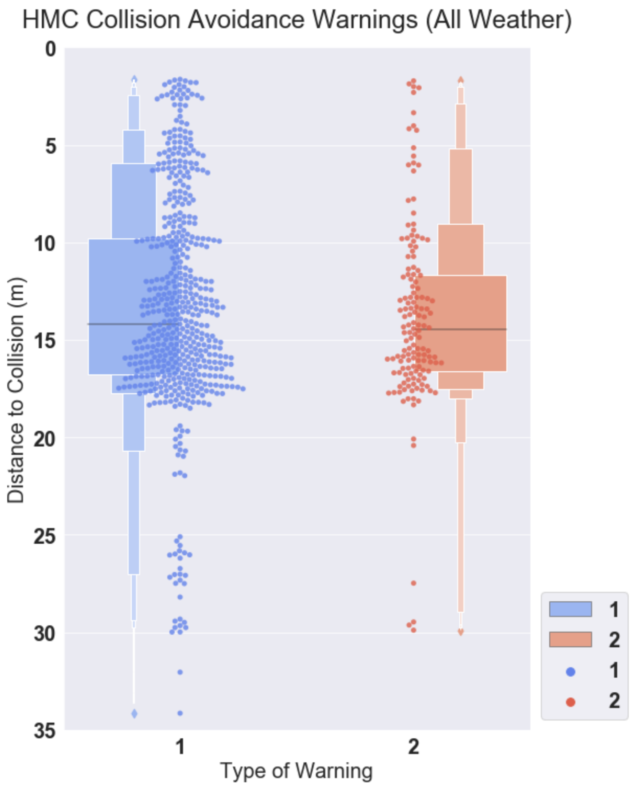
We presented methods for evaluating the safety of end-to-end BNN controllers, exemplified using a BNN variant of Nvidia's end-to-end autonomous driving controller. The proposed methods allow one to obtain uncertainty estimates for the controller's decisions with a priori statistical guarantees. On experiments performed on the CARLA driving simulator we showed that our statistical framework can be used to evaluate model robustness to changes in weather, location, and observation noise. Further, we propose an uncertainty-aware decision system for autononous driving and illustrate how our techniques can be employed to detect and avoid a high percentage of collisions.

We also evaluate three inference methods for BNNs to analyse the quality of uncertainty estimates that they produce. Our experiments suggest that MCD may underestimate the uncertainty of the BNN posterior and therefore is not able to capture a sufficiently accurate notion of uncertainty for use during real-time collision avoidance. This is in line with what was also observed in [17, 91]. On the other hand, HMC is too inefficient to scale beyond small datasets. Therefore, our results suggest that Gaussian VI approximations, such as *Bayes by Backprop* [9], may be particularly suitable for applications such as real-time collision avoidance.

While open-source simulators use maps of realistic, yet fictional cities, industrial simulators may be able to replicate the conditions and environments that exist on real



(a) Warnings broken down by weather.



(b) Warnings in all weather.

Figure 5.10: Using uncertainty (as in Definition 5.2) as a method for real-time detection of unsafe conditions. Dots represent raw warnings and boxes represent concentration of warnings within intervals from the mean. Network trained with HMC begins throwing warnings around 20 meters before a collision.

world street corners. In this case, our method could be used to perform statistical verification of safe coverage of places for which the original network may not have enough data. We note that generating a safety test of the scale shown in Figure 5.6 takes approximately 1.5 hours on a single desktop GPU, thus parallelisation using industrial grade equipment would allow one to test much larger geographical settings.

This chapter demonstrates the value of quantifying the uncertainty of Bayesian neural networks, within safety-critical applications, to evaluate network safety. However, the methods introduced in this chapter do not use all of the available information that a network can provide, only the inputs and outputs in a black-box manner. By using the whole network, it is possible to understand *why* a network makes a given prediction and therefore open up another avenue to improve safety. For example, we can make sure that the network does not use spurious characteristics for predictions, we can check for biased decisions, and we can determine where the model needs tuning or more training. We already have a method for extracting such information from deterministic networks, OREs, as defined in Chapter 4. The following chapter extends OREs to the Bayesian setting, and explores what different measures of uncertainty we can extract and their possible uses.

Chapter 6

Uncertainty, Explainability and Safety

The previous two chapters demonstrated the usefulness, individually, of local explainability and Bayesian uncertainty information for safety assurance of AI-based systems. This chapter explores the *combination* of the two, to give us a deeper insight into model behaviour. Our exploration includes formulating the notion of a Bayesian explanation for different types of explanation methods, and how we can use the distributions over individual features to extract uncertainty information and augment existing explanation methods. We evaluate the proposed techniques on Bayesian neural networks trained on image classification and sentiment analysis tasks.

6.1 Motivation

In this chapter, we, again, consider decision functions of the form $f : X \to Y$ that provide confidence estimates via uncertainty information. It is straightforward to see that a deterministic neural network encodes a single decision rule over its input space and that the goal of local feature attribution is to provide a human-interpretable explanation for this decision. On the other hand, Bayesian neural networks represent a distribution over such decision rules and thus one must consider the corresponding distribution over explanations for the decision. In this work we focus on two types of explanation methods separately, for ease of use: feature score methods and subset methods.

Feature score methods are methods, such as LRP, IG or SHAP, that return a relevance score for each feature given an input and a set of (fixed) weights. In this case, we write $r^w(x)$ to denote the generic score method relevance function. Since this relevance function is dependent on the fixed weights of the network, we can see that, when a Bayesian network is considered, the posterior weight distribution $P(w|\mathcal{D})$ invokes a distribution over the relevance scores (Equation 6.1).

$$P(r|x, \mathcal{D}) = \int r^w(x) P(w|\mathcal{D})\, dw \tag{6.1}$$

Then, relevance samples $\mathbf{r} \sim P(r|x, \mathcal{D})$ can be obtained by first sampling from the posterior weight distribution $\mathbf{w} \sim P(w|\mathcal{D})$ and then applying the relevance function. Note that feature score methods return only one score map per network sample.

On the other hand, subset methods are methods, such as Anchors or OREs, that return a subset of the input features as the most important features for prediction. This leads to a feature either being in or out of the explanation. In the case of a deterministic network, these methods can (and often do) generate multiple explanations for one input as they rely on minimising (or maximising) a score function where multiple subsets of input features may produce the same minimal (or maximal) result.

Since both types of methods, and therefore the resulting explanations, are dependent on the fixed weights of the network, in the Bayesian setting the posterior distribution $P(w|\mathcal{D})$ induces a probability over the explanations. We now formalise the concept of explanations for BNNs and propose a range of methods for their extraction.

6.2 Conceptual Description

Formally, we write $f^{\mathbf{w}}(x) = [f_1^{\mathbf{w}}(x), \dots, f_{n_c}^{\mathbf{w}}(x)]$ to denote a BNN with n_c output units, c classes and an unspecified number of hidden layers, where $\mathbf{w}$ is the weight vector random variable. Given a distribution over $\mathbf{w}$ and $w \in \mathbb{R}^{n_w}$, a weight vector sampled from the distribution of $\mathbf{w}$, we denote with $f^w(x)$ the corresponding deterministic neural network with weights fixed to w.

We consider a classification task where an input $\mathbf{x} \in \mathbb{R}^d$ is classified into a plausible class $y \in \mathcal{Y}$ by a neural network with parameters w, namely $f^w : \mathbb{R}^d \to \mathcal{Y}$. We denote with F the set of features in $\mathbf{x}$, namely $F = \{x_1, .., x_d\}$. E is a subset of F, i.e., $E \subseteq F$. Indices of variables in sets F and E are denoted respectively as $I_F = \{1, .., d\}$ and $I_E \subseteq I_F$.

We define a generic explanation for a deterministic neural network as in Definition 6.1.

Definition 6.1 (Explanation) *Let $g : \mathcal{P}(F) \times \mathbb{R}^{n_w} \to \mathbb{R}^+$ be a function mapping explanations and weights to non-negative rewards (including 0). Then we say that a subset $E^* \subseteq F$ is an* explanation *for features $\mathbf{x}$ and network f^w w.r.t. g iff*

$$E^* \in \arg\max_{E \subseteq F} g(E, w). \tag{6.2}$$

Any subset E^* of input features is called an explanation if it maximises the function g. Function g takes the form of a user-defined cost function, which could be any new or existing explanation method that considers both the subset and weights of a network as input, for example layer-wise relevance propagation.

To exemplify this definition, and others throughout this section, we present a running example. Let input $x = \{0.0, 0.1, 0.3, 0.9, 0.0\}$, let deterministic network f^w have weights $w = \{0.1, 0.1, 0.1, 0.1, 0.1\}$, and function $g(E, w) = \sum_{i=0}^{|E|} E_i w_{I_{E,i}}$. Note that this is not a realistic cost function or network architecture and is for demonstration purposes only. We see that subset $E^* = \{x_2, x_3, x_4\}$ maximises function g with score $(0.1 \times 0.1) + (0.3 \times 0.1) + (0.9 \times 0.1) = 0.13$ and is therefore an explanation.

Note that, in many cases, there is not a unique subset E^* that maximises the cost function g, which leads us to Definition 6.2.

Definition 6.2 (Explanation Collection) *For a deterministic neural network f^w, input $\mathbf{x}$, a set $\mathbf{E}^* = \{E_1^*, ..., E_n^*\}$ of subsets $E_i^* \subseteq F$ of the features of $\mathbf{x}$ is an* explanation collection *w.r.t. a function* $g : \mathcal{P}(F) \times \mathbb{R}^{n_w} \to \mathbb{R}^+$*, iff*

$$E^* \in \mathbf{E}^* \iff E^* \in \underset{E \subseteq F}{\arg\max}\ g(E, w). \tag{6.3}$$

An explanation collection is the set of *all* of the subsets of input features $E_i^* \subseteq F$ that maximise the cost function g. Considering *all* of the possible explanations for one input and network pair is integral in the Bayesian setting as we retain more information about the distributions of the deciding features. From our example, we see that there are three additional subsets that maximise g using the zero valued x_1 and x_5 variables, which, when multiplied by their corresponding weight, do not add or remove from the maximum score. The full explanation collection is then: $\mathbf{E}^* = \{\{x_2, x_3, x_4\}, \{x_1, x_2, x_3, x_4\}, \{x_2, x_3, x_4, x_5\}, \{x_1, x_2, x_3, x_4, x_5\}\}$.

Suppose now that we want to explain the decision of a BNN whose output for input x is class $s \in \{1, ..., c\}$ (hence preferring it to all other classes). Given a sample of $N > 0$ deterministic neural networks from the BNN's weight distribution $\mathbf{w}$, namely $\hat{f^w} = (f_0^w, ..., f_{N-1}^w)$, we split $\hat{f^{\mathbf{w}}}$ into c disjoint sets: $\hat{f^{w0}}, ..., \hat{f^{wc}}$, depending on the output of each deterministic neural network. More formally, $\hat{f^{\mathbf{w}s}} = \{f_j^w \ s.t.\ f_j^w \in \hat{f^{\mathbf{w}}} \wedge f_j^w(x) = s\}$.

We now define the concept of an explanation for the ensemble of networks in $\hat{f^{\mathbf{w}s}}$. The explanations of all classes will account for the BNN epistemic uncertainty on x.

Definition 6.3 (Ensemble Explanation) *Given a sample* $\hat{f^{\mathbf{w}s}} = \{f_0^{ws}, .., f_{N-1}^{ws}\}$ *of* $N > 0$ *deterministic neural networks, an ensemble explanation is the multiset of explanation collections* $\{\mathbf{E}^*{}_0,\ ..., \mathbf{E}^*{}_{N-1}\}$ *(as in Definition 6.2), where* $\mathbf{E}^*{}_i$ *is the explanation collection for network* f_i^{ws}*.*

Intuitively, an ensemble explanation is the combination of all explanation collections, one from each BNN sample. In our running example, the weight vector w will change with each BNN sample. Let us generate the ensemble explanation for 2 BNN samples: let $w_1 = \{0.1, 0.1, 0.1, 0.1, 0.1\}$ be the first sampled weights, $w_2 = \{0.1, 0.2, 0.1, -0.1, 0.0\}$ be the second, and our input $x = \{0.0, 0.1, 0.3, 0.9, 0.0\}$ as before. From above, the first explanation collection is $\mathbf{E_1^*} = \{\{x_2, x_3, x_4\}, \{x_1, x_2, x_3, x_4\},$

$\{x_2, x_3, x_4, x_5\}$, $\{x_1, x_2, x_3, x_4, x_5\}\}$, and we calculate the second explanation collection as $\mathbf{E_2^*} = \{\{x_2, x_3\},\ \{x_1, x_2, x_3\},\ \{x_2, x_3, x_5\},\ \{x_1, x_2, x_3, x_5\}\}$. The ensemble explanation is then $\{\mathbf{E_1^*}, \mathbf{E_2^*}\}$.

6.3 Explaining the Decisions of Bayesian Neural Networks

This section describes how ensemble explanations can be distilled into something of use to a human decision maker, and how function g can be defined for different types of explanation method.

6.3.1 General Case

While ensemble explanations (Definition 6.3) contain a lot of information about how a BNN makes a decision, that information is not easily interpretable by a human. As shown in the running example above, the resulting ensemble explanation is long, even for only two BNN samples, and interpreting it is difficult. We take inspiration from the concept of code coverage [88] to define a generic *covering explanation*, which takes our ensemble explanation and distills the content down into useful information for the human decision maker.

Definition 6.4 (Covering Explanation) *Given an ensemble explanation* $\mathbf{E^{**}}_{\mathbf{w}s} = \{\mathbf{E^*}_0, ..., \mathbf{E^*}_{N-1}\}$ *(as in Definition 6.3), a subset of features* $E_{COVER} \subseteq F$ *is a covering explanation if:*

$$E_{COVER} \in \underset{E \in F}{\arg\max} \sum_{\mathbf{E^*} \in \mathbf{E^{**}}} \mathbb{1}_{E \in \mathbf{E^*}} \tag{6.4}$$

We further define the covering explanation probability to be:

$$Prob(E_{COVER}) = \frac{1}{N} \sum_{\mathbf{E^*} \in \mathbf{E^{**}}} \mathbb{1}_{E_{COVER} \in \mathbf{E^*}} \tag{6.5}$$

In short, the covering explanation is the explanation that occurs in ("covers") the most explanation collections of the BNN samples, i.e. the dominant explanation[1]. For example, if we take three network samples, and the resulting ensemble explanation $\mathbf{E}^{**}{}_{\mathbf{w}0} = \{\{\{x_1, x_3\}, \{x_2, x_3\}\}, \{\{x_2, x_3\}, \{x_3, x_4\}, \{x_3, x_4, x_5\}\}, \{\{x_5\}, \{x_2, x_3\}\}\}$, then the covering explanation for class 0 is $\{x_2, x_3\}$ as it occurs in the highest number of individual samples‘ explanation collections.

The covering explanation probability is the probability that the covering explanation will occur in any sampled networks‘ explanation collection. Using the small example above, the covering explanation $\{x_2, x_3\}$ occurs in all three network samples‘ explanation collections, therefore $P(E_{\text{COVER}}) = \frac{1+1+1}{3} = 1$.

Similarly to Section 5.3.3, we can obtain an *a priori* statistical guarantee on the covering explanation probability using the Chernoff bounds to determine the necessary sample size for a given absolute error bound and confidence level.

By characterising function g, we can define the notion of a covering explanation and covering explanation probability for different explanation methods. Function g can be defined using any neural network explanation method. For example, we could set function g to be equal to the cumulative layer-wise relevance score of M input features, then maximising g would amount to choosing the M input features with the highest cumulative score (as mentioned above, there may be multiple such sets that result in this maximum score).

In the next section, we redefine the notion of a covering explanation and covering explanation probability for both feature score methods and subset methods. For feature score methods, we show how any method that returns a relevance map can be used, and in the case of subset methods we define the concept of a Bayes-optimal robust explanation (B-ORE), which extends our previous work on OREs in Chapter 4.

[1]Since we wish to retain the properties associated with the base explanation method (e.g. robustness and optimality from OREs), we choose the dominant explanation instead of what one might think of as a true "covering" explanation that is made up of parts of all the explanations in the explanation collections.

6.3.2 Bayesian Feature Score Methods

Feature score methods can be defined as a generic relevance function r over input x and BNN weight sample w, to produce a relevance score map $r^w(x)$. In the simplest case, we could design function g to return the sum of all relevance scores of the input features given to it. More specifically, for input x, subset of input features E, indices I_E of variables in E, BNN network sample f^w with weights w, relevance method r and relevance score map $r^w(x)$, we have:

$$g(E, w) = \sum_{j \in I_E} r^w(x)_j \tag{6.6}$$

However, the explanations that maximise g in this case will all be the same: they will include every input feature with a non-zero relevance score and then a varying number of zero-scoring features. Furthermore, it is apparent that this naive method results in the loss of information as we end up disregarding the magnitude of the score.

To prevent this, we can introduce a threshold value, where only scores over this threshold are considered in the sum, and zero features are removed by penalising the length of E as discussed above.

$$g(E, w) = \sum_{j \in I_E : r^w(x)_j > \delta} r^w(x)_j \tag{6.7}$$

Here, δ is a threshold parameter. Note that this definition of g means that only one explanation exists that maximises g: the subset of non-zero features that are above δ.

Whilst a covering explanation, as in Definition 6.4, can be computed from the ensemble explanation of a BNN using this definition of function g, the covering explanation probability (Definition 6.4) will usually be low, as we find that, empirically, generating the same (single) explanation from function g from multiple network samples is unlikely. Additionally, by taking the features over a given score threshold, we are losing information about the magnitudes of the scores of these remaining features which is disregarded once chosen.

To retain more information about individual feature relevance scores, whilst still being able to extract covering explanations, we propose the following adjustment to the definition of an explanation.

Definition 6.5 (Feature Score Explanation) *Let* $g : \mathcal{P}(F) \times \mathbb{R}^{n_w} \to \mathbb{R}^+$. *Then we say that a subset* $E^* \subseteq F$ *is a* feature score explanation *for features* $\mathbf{x}$ *and network* f^w *w.r.t.* g *iff*

$$E^* \in \{E | E \subseteq F \wedge g(E, w) > \gamma\}. \tag{6.8}$$

Instead of taking only the maximising subsets as explanations, a feature score explanation is any subset that pushes the cost function over a user-defined threshold. This definition loosens the strict requirement that an explanation must *maximise* g and instead allows for a range of explanations that are close to the optimal explanation (depending on how close the threshold is to the maximum score) whilst retaining important relevance information for the human decision maker.

We present the following pair of algorithms for calculating the set of feature score explanations for an input and network pair, for any feature score method.

Algorithm 5: Feature score explanation set computation

```
Data: the input $x$, the relevancy score function $r$, the network $f$, threshold
      $\delta$, threshold $\gamma$
Result: a set of explanations $S$
1 $E \leftarrow \mathbf{r}(x, f)$
2 for $i \leftarrow 0$ to $|E|$ do
3     if $E_i < \delta$ then
4         $E_i \leftarrow 0$
5 $S \leftarrow \mathbf{genExps}(E, \gamma)$
```

Algorithm 5 is a wrapper function that first filters out any scores below a given threshold δ, then passes the filtered explanation to the recursive function defined in Algorithm 6.

This recursive function generates a set of explanations that satisfy Definition 6.5. For each index in the input E (which, to begin with, is the whole feature score map

Algorithm 6: Feature score explanation helper function **genExps**(E,γ)

```
  Data: the explanation $E$, threshold $\gamma$
  Result: a set of explanations $S$
1 $S \leftarrow \emptyset$
2 for $i \leftarrow 0$ to $|E|$ do
3     $sliced \leftarrow E \setminus E_i$
4     if $\textbf{\textit{sum}}(sliced) < \gamma$ then
5         $S \leftarrow S \cup E$
6     else
7         $T \leftarrow \textbf{genExps}(sliced, \gamma)$
8         $S \leftarrow S \cup T$
9 return $S$
```

generated from a given explanation method), that index is removed from the feature score map and then the sum of the remaining entries is taken. If the sum is less than the user-defined threshold γ, then all of the entries in E are required to retain a score of above γ, and so E is added to the explanation set S. If the sum is not less than γ then we recurse with $E \setminus E_i$ and join the result with set S.

Similarity With UAI As discussed in the literature review in Section 3.1.4, [16] attempts to explain Bayesian neural networks by using a method called UAI (union and intersection). This method is defined in Equation 6.9, where $\mathcal{P}_\alpha$ is an operator that computes the feature-wise α percentiles.

$$\mathrm{UAI}_\alpha(x) = \mathcal{P}_\alpha[P(R|\mathbf{x}, \mathcal{D})] \tag{6.9}$$

When α is high (> 0.5), a union explanation is recovered, as the resulting relevance map accumulates features across many sampled explanations. When α is low, an intersection explanation is generated, as only features that $\alpha\%$ of the explanations agree on are included.

UAI considers the contribution of each individual feature separately to generate a Bayesian explanation. Our notion of a covering explanation considers *entire ex-*

planations instead. This difference allows us to retain information between features, instead of looking at them individually. In addition to this, UAI does not provide any statistical guarantees, and does not use the uncertainty information available from BNNs.

6.3.3 Bayes-Optimal Robust Explanations

Moving into the category of subset methods, this section describes how to extend OREs to the Bayesian setting for enhanced explainability. First, we recall robust explanations of Definition 4.1. A robust explanation is any subset of input features E that must remain unperturbed whilst a bounded perturbation is applied to the set $F \setminus E$, else a misclassification will occur.

We then introduced the notion of an optimal robust explanation (ORE) in Definition 4.2, where the difference is that an explanation must now also satisfy optimality with respect to a user-defined cost function (for example, the length of the explanation). OREs are generated using a hitting-set based algorithm that generates candidate explanations and checks their validity using entailment checks performed by a black-box solver.

Following the conceptual design outlined in Section 6.2, we now define an ORE Collection. While we give the definition for word features, we remark that it can be generalised to other types of inputs.

Definition 6.6 (ORE Collection) *For a deterministic neural network f^w, cost function $\mathcal{C} : W \to \mathbb{R}_{\geq 0}$, input text $t = (w_1, ..., w_l)$ with embedding $x = \mathcal{E}(t)$, word level perturbation $\mathcal{B}$, a set $\mathbf{E}^* = \{E_1^*, ..., E_n^*\}$ of subsets $E_i^* \subseteq F$ of the features of $\mathbf{x}$ is an* ORE collection*, iff*

$$E^* \in \mathbf{E}^* \iff E^* \in \arg\min_{E \subseteq F} \sum_{w \in E} \mathcal{C}(w) \ s.t.\ \mathsf{Rob}_{f^w,x}(E). \tag{6.10}$$

An ORE collection, similarly to an explanation collection, is the set of all subsets that satisfy the properties of an ORE. Moving into the Bayesian setting, we can then also redefine an ensemble explanation (Definition 6.3) in terms of OREs.

Definition 6.7 (Bayes-Optimal Robust Explanation) *Given a sample of $N > 0$ deterministic neural networks from the BNN's weight distribution that predict class s, $f^{\hat{\mathbf{w}}s} = \{f_0^{ws}, ..., f_{N-1}^{ws}\}$, cost function $\mathcal{C} : W \to \mathbb{R}_{\geq 0}$, input text $t = (w_1, ..., w_l)$ with embedding $x = \mathcal{E}(t)$, word level perturbation $\mathcal{B}$, a* B-ORE (Bayesian optimal robust explanation) *is the multiset of ORE collections $\mathbf{E}_{B-ORE,\mathbf{w}s} = \{\mathbf{E_0^*}, ..., \mathbf{E_{N-1}^*}\}$ (as in Definition 6.6), where $\mathbf{E_i^*}$ is the ORE collection for network f_i^{ws}.*

A Bayes-optimal robust explanation is the set of ORE collections, where each ORE collection is from one BNN sample.

Finally, we define the concepts of a covering explanation and covering explanation probability in terms of OREs.

Definition 6.8 (Bayes-Optimal Robust Covering Explanation) *Given the posterior distribution of the BNN $P(w|\mathcal{D})$, a sample of $N > 0$ deterministic neural networks from the BNN's weight distribution that predict class s, $f^{\hat{\mathbf{w}}s} = \{f_0^{ws}, .., f_{N-1}^{ws}\}$, a* B-ORE $\mathbf{E}_{B-ORE,\mathbf{w}s} = \{\mathbf{E}^*{}_0, ..., \mathbf{E}^*{}_{N-1}\}$ *(as in Definition 6.7), where $\mathbf{E}^*{}_i$ is the ORE collection (as in Definition 6.6) for network f_i^{ws}, then a* B-ORCE (Bayes-optimal robust covering explanation) *is defined as:*

$$E_{\textit{B-ORCE}} \in \arg\max_{E \in F} \sum_{\mathbf{E}^* \in \mathbf{E}_{B-ORE,\mathbf{w}_s}} \mathbb{1}_{E \in \mathbf{E}^*} \tag{6.11}$$

The B-ORCE is the explanation that occurs in the most ORE collections of the BNN samples.

Definition 6.9 (B-ORCE Probability) *Given a sample of $N > 0$ deterministic neural networks from the BNN's weight distribution that predict class s, $f^{\hat{\mathbf{w}}s} = \{f_0^{ws}, .., f_{N-1}^{ws}\}$, a B-ORE $\mathbf{E}_{B-ORE,\mathbf{w}s} = \{\mathbf{E}^*{}_0, ..., \mathbf{E}^*{}_{N-1}\}$ (as in Definition 6.7), where $\mathbf{E}^*{}_i$ is the ORE collection (as in Definition 6.6) for network f_i^{ws}, and a B-ORCE $E_{\textit{B-ORCE}}$, then the* probability *of the B-ORCE appearing in $\mathbf{E}^*$ is:*

$$\mathrm{P}_{\mathbf{E}^*}(E_{\textit{B-ORCE}}) := \mathrm{P}_{ws_i \sim P(w|\mathcal{D})} E_{\textit{B-ORCE}} \in \mathbf{E}^*{}_i.$$

B-ORCE probability is the probability that the B-ORCE appears in the ORE collection for a given BNN sample. As in Section 5.3.3, this probability comes with statistical guarantees based on the number of samples.

6.4 Covering Results

This section demonstrates the notion of covering explanations for both feature score methods, including LRP, IG and SHAP, and for the subset method B-OREs for image and NLP tasks.

6.4.1 Bayesian Feature Score Methods

For the MNIST dataset experiments, we have trained a Bayesian fully connected (FC) model, and a Bayesian convolutional (CNN) model for the GTSRB dataset. The full experimental setup including architecture and hyper-parameters for the following experiments can be found in Appendix A.4.

All experiments have been implemented in Python3, and libraries for each of the three explanation methods are as follows: [34] for LRP, [52] for IG and [80] for SHAP. The number of samples taken for each input network pair in all of the experiments is 100.

Results For Equation 6.7 Figures 6.1 and 6.2 display the covering explanations generated using the simple version of function g, defined in Equation 6.7. Figure 6.1 shows three different input images chosen from the MNIST dataset, and the resulting covering explanations and their probabilities from LRP, IG and SHAP. As discussed above, the covering explanation probability is very low, and information about the magnitudes of the remaining scores is discarded leading to information loss. Similar findings can be observed in Figure 6.2, which displays an input chosen from the GT-SRB dataset and the resulting covering explanations and probabilities from the three explanation methods. Again, we see that the covering explanation probability is very low in every case.

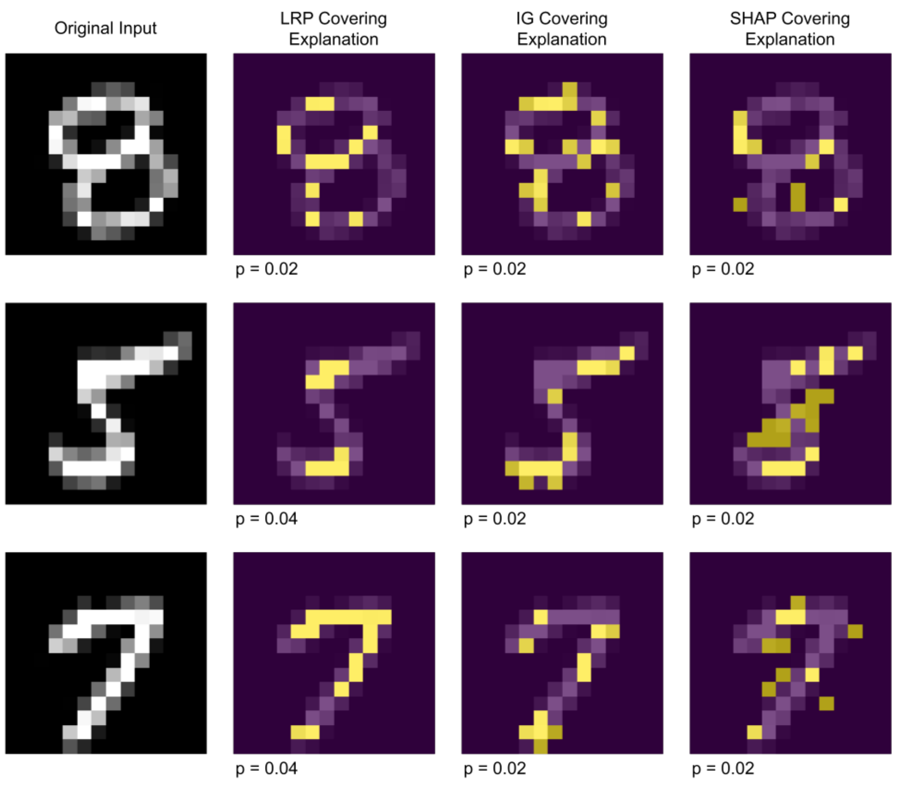


Figure 6.1: Covering explanations and covering explanation probabilities (p) generated from the MNIST dataset using LRP, IG and SHAP explanation methods, and using Equation 6.7 to define function g. Yellow pixels imply inclusion in the explanation. Threshold value δ was set to 60% of the maximum relevance score.

Results For Feature Score Explanations Figures 6.3 and 6.4 display the covering explanations generated using feature score explanations of Definition 6.5. Figure 6.3 shows the same three input images as the above experiment, chosen from the MNIST dataset, and the resulting covering explanations and their probabilities from LRP, IG and SHAP. The covering explanation probability is considerably higher in this case, which implies that many Bayesian network samples agree on which features contribute highly to the resulting prediction. Interestingly, there is quite a difference in the magnitude of the covering explanation probability between different explanation methods. The covering explanations generated using LRP with feature score explanations have a higher average probability than those generated using IG or SHAP.

Similar findings can be observed in Figure 6.2, which displays the same input as

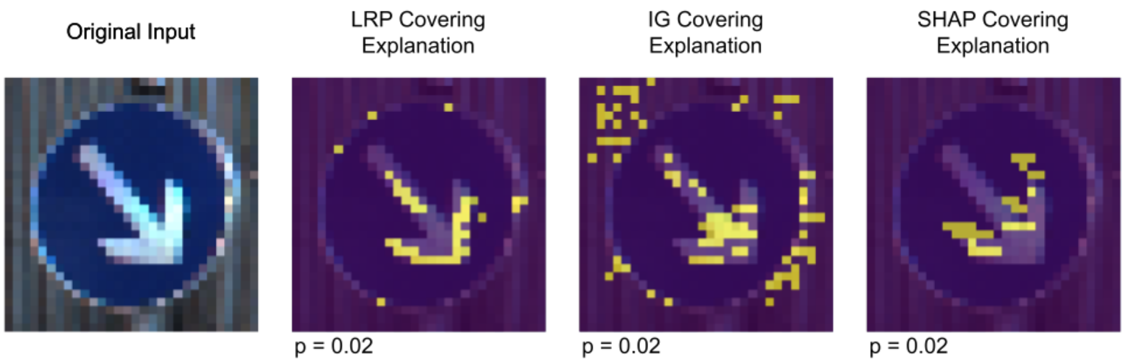


Figure 6.2: Covering explanations and covering explanation probabilities (p) generated from the GTSRB dataset using LRP, IG and SHAP explanation methods, and using Equation 6.7 to define function g. Yellow pixels imply inclusion in the explanation. Threshold value δ was set to 15% of the maximum relevance score.

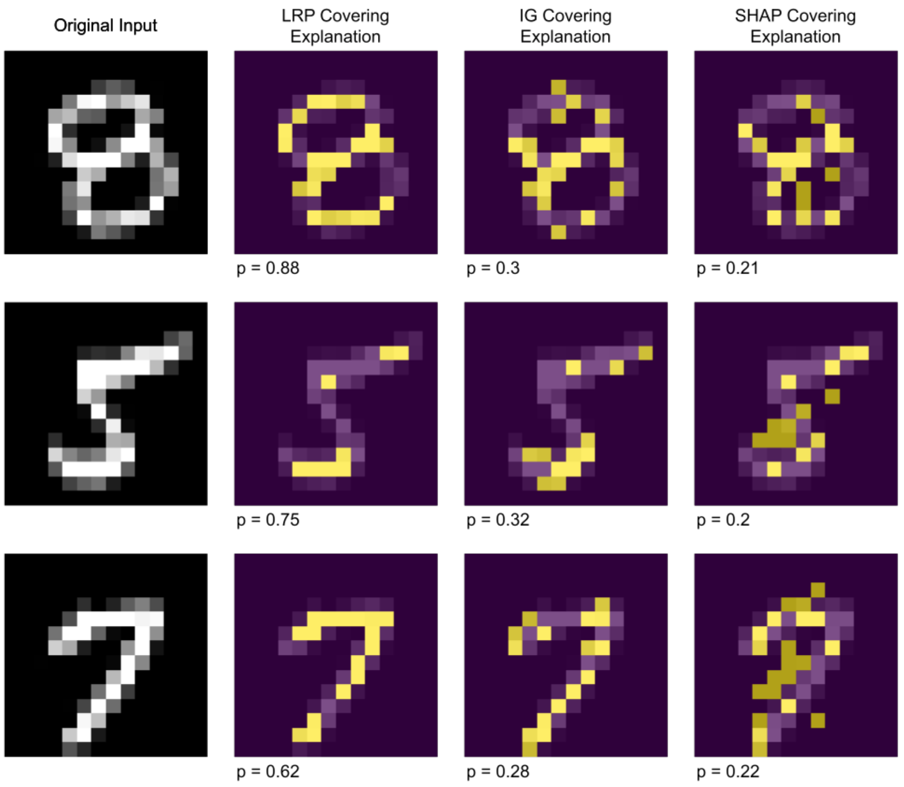


Figure 6.3: Covering explanations and covering explanation probabilities (p) generated from the MNIST dataset using LRP, IG and SHAP explanation methods, and using feature score explanations (Definition 6.5). Yellow pixels imply inclusion in the explanation.

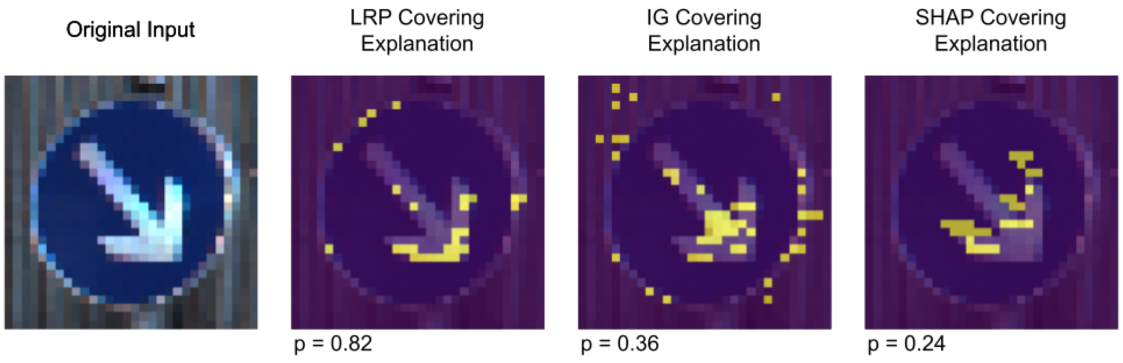


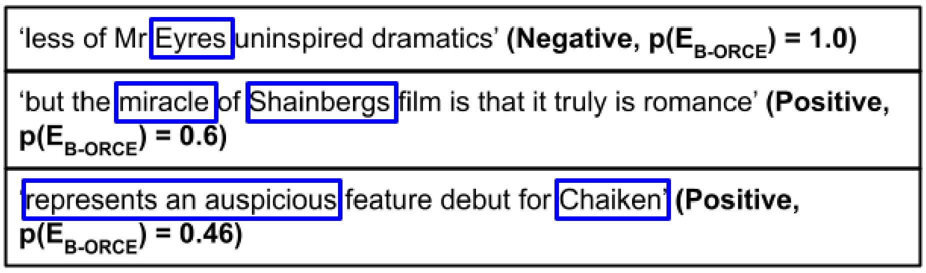
Figure 6.4: Covering explanations and covering explanation probabilities (p) generated from the GTSRB dataset using LRP, IG and SHAP explanation methods, and using feature score explanations (Definition 6.5). Yellow pixels imply inclusion in the explanation.

above, chosen from the GTSRB dataset, and the resulting covering explanations and probabilities from the three explanation methods. Again, we see that the covering explanation probability is much higher than before, and LRP is higher than both IG and SHAP.

6.4.2 Bayes-Optimal Robust Explanations

Similarly to the experimental setup for our ORE analysis, we have trained both fully connected (FC) and convolutional (CNN) models on the Stanford Sentiment Treebank (SST) sentiment analysis dataset, except that these networks are Bayesian in nature. Full details of the experimental setup can be found in Appendix A.4.

All experiments have been implemented in Python3 and use Neurify [125] to answer robustness queries for determining possible OREs, as described in Section 4.9. In the experiments below, similarly to the ORE analysis, we opted for the kNN-box perturbation space as the k parameter is easier to interpret and tune than the ϵ parameter for the ϵ-ball space. The B-ORCE cost function was defined as in Equation 6.12, where the set of unwanted words includes padding and lone punctuation. Intuitively, this cost function encourages short explanations without uninterpretable tokens. In all of these experiments, we take $N = 100$ as the number of Bayesian


'less of Mr Eyres uninspired dramatics' **(Negative, p(E$_{\text{B-ORCE}}$) = 1.0)**

'but the miracle of Shainbergs film is that it truly is romance' **(Positive, p(E$_{\text{B-ORCE}}$) = 0.6)**

'represents an auspicious feature debut for Chaiken' **(Positive, p(E$_{\text{B-ORCE}}$) = 0.46)**

Figure 6.5: A selection of reviews from the SST dataset that include a name in their Bayes-optimal robust covering explanation. The words included in the B-ORCE are highlighted in blue, and the Bayesian network prediction and B-ORCE probability is given alongside each.

network samples.

$$\mathcal{C}(\text{word}) = \begin{cases} \text{length(input\_text)} + 1, & \text{if word} \in \text{unwanted\_words} \\ 1, & \text{otherwise} \end{cases} \tag{6.12}$$

B-ORCE Results Table 6.1 shows a range of reviews from the SST dataset, along with the Bayes-optimal robust covering explanation, the B-ORCE probability, the Bayesian network prediction (all examples were classified correctly), and the predictive variance (the variance of the predictions of the N sampled networks).

Over all 50 input texts, the average B-ORCE probability is 0.63. Interestingly, we note that the B-ORCEs with higher probabilities tend to contain a greater proportion of words with positive or negative connotations than those with lower probabilities, at least in this selection of input texts. For example, the highest probability in the table is $p = 1.0$ for the first entry, and the B-ORCE contains two words with highly positive connotations, {*'fluid','mesmerizing'*}, in the context of movie reviews. In contrast, the last entry in the table has the lowest probability at $p = 0.28$, and the B-ORCE contains words including *'those'* and *'viewers'* which are not necessarily associated either positively or negatively.

Figure 6.5 displays cases where a name appears in the B-ORCE. In the context of movie reviews, the name of, for example, an actor or director, can be viewed as a protected feature and should not affect the decision of the network. Especially in the first example, we see that the name *'Eyres'* constitutes the entire B-ORCE with

Review	$E_{\textbf{B-ORCE}}$	$P(E_{\textbf{B-ORCE}})$	**Pred**	**Var**
‘a fluid and mesmerizing sequence of images’	{‘fluid’, ‘mesmerizing’}	1.0	Pos	0.0
‘gives ample opportunity for largescale action and suspense’	{‘largescale’}	0.74	Pos	0.0
‘prosaic and dull’	{‘dull’}	0.66	Neg	0.0
‘you’re likely to have one helluva time at the movies’	{‘helluva’, ‘movies’}	0.52	Pos	0.002
‘there’s a lot of good material here’	{‘a’, ‘lot’ ‘good’}	0.5	Pos	0.01
‘impress even those viewers who have little patience for eurofilm pretension’	{‘impress’, ‘those’, ‘viewers’, ‘eurofilm’}	0.28	Pos	0.03

Table 6.1: A selection of reviews from the SST dataset with their Bayes-optimal robust covering explanation, B-ORCE probability, Bayesian network prediction (all reviews were correctly classified) and predictive variance (calculated as the variance of the N network samples‘ predictions). Since predictions are either 0 (negative) or 1 (positive), the maximum possible predictive variance is 0.25.

$p = 1$, indicating that the name was an explanation for every sample of the BNN and implies the decision. Since *‘Eyre’* is the name of filmmaker, this is unintended and unwanted behaviour that has been exposed. In such cases, the model creator can make use of data augmentation or similar techniques to avoid this type of model bias.

In addition to these experiments, we explored the relationship between predictive variance, B-ORCE probability and B-ORCE length (as this is what the cost function tries to minimise). Note that there are only a few input texts with prediction variance higher than 0, as the models were accurate enough so that finding such examples was difficult. Figures 6.6, 6.7 and 6.8 are scatter plots comparing these values. Figure 6.6, which plots the prediction variance versus B-ORCE probability, shows that when there is no prediction variance, the B-ORCE probability is still completely variable. However, when the prediction variance takes values above 0, we see a moderate to strong negative correlation (Pearson correlation coefficient = -0.35 (moderate), Spearman correlation coefficient = -0.60 (strong)) between prediction variance and B-ORCE probability. This is somewhat expected, as a higher prediction variance implies a more uncertain Bayesian network, which in turn leads to wider

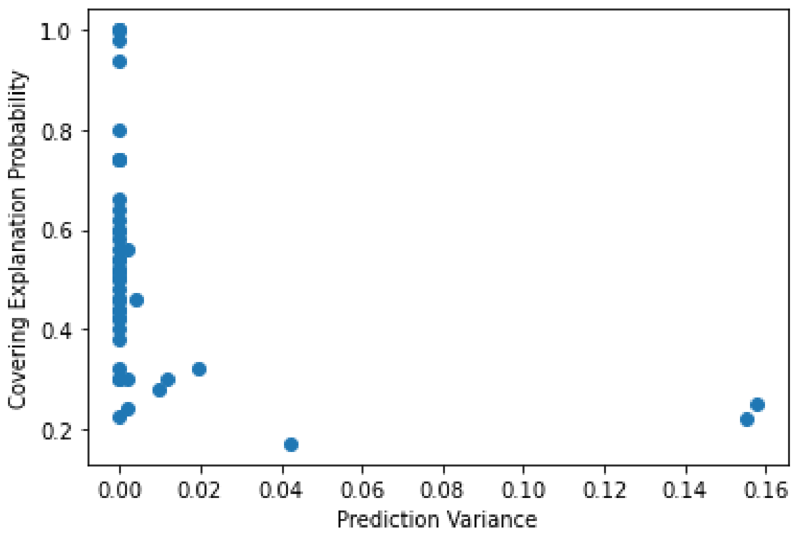


Figure 6.6: A scatter plot of prediction variance (the variance of the N sampled networks predictions, in this case $N = 100$) versus the B-ORCE probability, for 50 different input texts.

weight distribution. BNN samples are expected to be further apart and that could cause a greater difference in the ORE collections of each network, and therefore lower the B-ORCE probability.

Figure 6.7, which plots the prediction variance versus the length of the B-ORCE, shows that there were very few single word explanations (only 1) for prediction variances greater than 0. In fact, there is a weak to moderate positive correlation between the two (Pearson correlation coefficient $= 0.42$ (moderate), Spearman correlation coefficient $= 0.23$ (weak)).

Finally, Figure 6.8 plots the B-ORCE probability versus the B-ORCE length. There is a moderate to strong negative correlation between the two (Pearson correlation coefficient $= -0.55$ (strong), Spearman correlation coefficient $= -0.57$ (moderate)), which complements the correlations in the plots above, as a shorter B-ORCE correlates with a lower prediction variance (from Figure 6.7), and lower prediction variance allows for the maximum range of B-ORCE probabilities (from Figure 6.6).

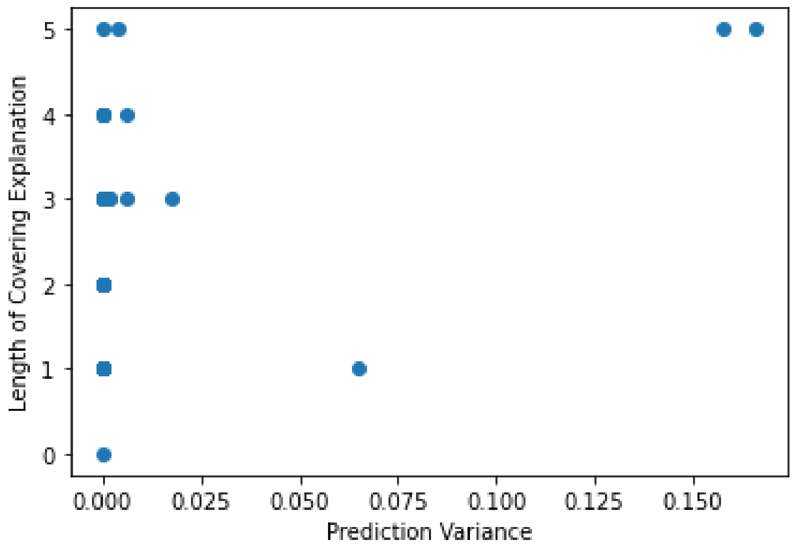


Figure 6.7: A scatter plot of prediction variance (the variance of the N sampled networks predictions, in this case $N = 100$) versus the length of the resulting B-ORCE, for 50 different input texts.

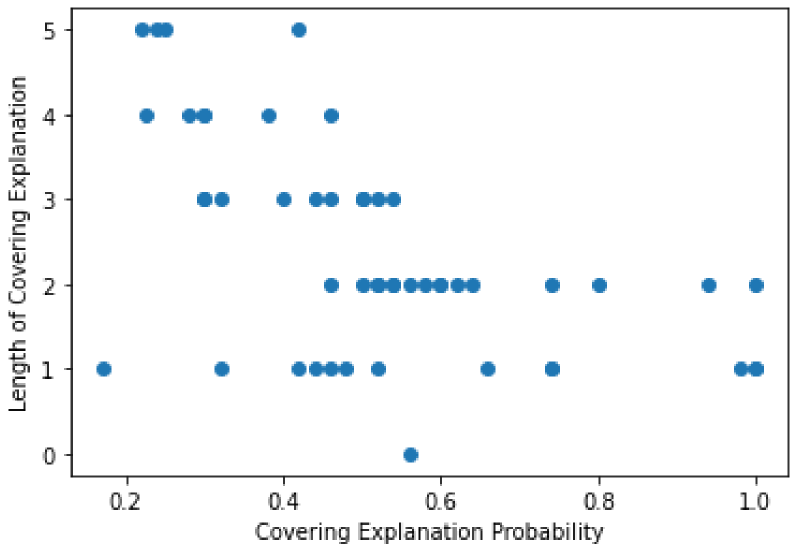


Figure 6.8: A scatter plot of the B-ORCE probability versus the length of the resulting B-ORCE, for 50 different input texts.

6.5 Analysing Bayes-Optimal Robust Explanations

In addition to generating a B-ORCE from a B-ORE, there is extra information contained within the B-ORE that we can use to help us learn more about our models. This section explores *feature probability* and what it can tell us about our model and embedding space, whereas the next section goes further and derives a measure of uncertainty from the feature probability, as well as showing its uses.

From the B-ORE, we can find the probability of a feature (in our experiments, a word) being in each network samples‘ ORE collection, for each class. Since the posterior distribution $P(w|\mathcal{D})$ of the BNN induces a probability distribution over the ORE collections, we can express the probability that a particular feature t_i is included in an ORE collection as below. Note that, although this definition is based on NLP (words as features), features of any form can be used and the embedding function can be set to the identity function, or omitted entirely (x can be set equal to the input features directly) to produce the same effect.

Definition 6.10 (Linear Feature Attribution) *Given the posterior distribution of the BNN $P(w|\mathcal{D})$, a sample of $N > 0$ deterministic neural networks from the BNN's weight distribution that predict class s, $f^{\hat{\mathbf{w}}s} = \{f_0^{ws}, .., f_{N-1}^{ws}\}$, cost function $\mathcal{C} : W \to \mathbb{R}_{\geq 0}$, input $t = (w_1, ..., w_l)$ with embedding $x = \mathcal{E}(t)$, feature level perturbation $\mathcal{B}$, B-ORE $\mathbf{E}_{B\text{-}ORE,\mathbf{w}s} = \{\mathbf{E^*}_0, ..., \mathbf{E^*}_{N-1}\}$ (as in Definition 6.7), where $\mathbf{E^*}_i$ is the ORE collection (as in Definition 6.6) for network f_i^{ws}, and a feature $t_i \in t$, then the probability of feature t_i appearing in an ORE collection $\mathbf{E}^*$ is defined as:*

$$\mathrm{P}_{\mathbf{E}^*,t}(t_i) := \mathrm{P}_{ws_j \sim P(w|\mathcal{D})} t_i \in \mathbf{E^*}_j$$

Now that we have access to this probability, we are able to explore how the value changes over different inputs. Most notably, since a B-ORE can be generated for every possible class, we can study the value over different classes for the same input. Tables 6.2, 6.3 and 6.4 contain example input texts from the SST dataset, and the resulting word probabilities for each class, according to Definition 6.10. To generate

P(w)	‘feminist’	‘action’	‘fantasy’
c = Pos	0.84	0.90	0.90
c = Neg	0.92	0.96	0.94

Table 6.2: A movie review from the SST dataset and the resulting word probability. The review was classified as negative and the ground truth was positive. The prediction variance was 0.24.

P(w)	‘the’	‘innate’	‘theatrics’	‘that’	‘provide’	‘its’	‘thrills’	‘and’	‘extreme’	‘emotions’
s = Pos	0.41	0.91	0.48	0.88	0.62	0.60	0.83	0.83	0.83	0.64
s = Neg	1.0	1.0	0.88	1.0	0.25	0.625	1.0	0.875	1.0	0.75

Table 6.3: A movie review from the SST dataset and the resulting word probability. The review was classified as positive and the ground truth was negative. The prediction variance was 0.14.

a sufficient number of BNN samples that predict each possible class, input texts with a high prediction variance have been chosen. The same experimental setup as in Section 6.4.2 is used as these results come from the same set of experiments.

Table 6.2 shows an example where all words have a high probability in both classes. Since *all* probabilities are high, the most likely implication is that this input as a whole is very sensitive to perturbations, and so all words must almost always appear in the explanations, rather than implying anything else about the model behaviour.

Tables 6.3 and 6.4 show examples where only some words have a high probability in both classes. The words that have a high probability in one class and a low probability in the other show that the model knows with more certainty what class these words contribute to. Words with similar probabilities in both classes (whether low, medium or high) show that the network is unsure about the classification of these words, and that they may not contribute either way (i.e., they are neutral words). The words with high probability in both classes are a more difficult case to interpret. Due to the nature of OREs, a word with a high probability in both classes indicates

P(w)	‘Todd’	‘Solondz’	‘takes’	‘aim’	‘on’	‘political’	‘correctness’	‘and’	‘suburban’	‘families’
c = Pos	0.36	0.90	0.31	0.56	0.07	0.64	0.17	0.10	0.50	0.36
c = Neg	0.30	0.70	0.35	1.00	0.30	0.95	0.30	0.85	0.60	0.35

Table 6.4: A movie review from the SST dataset and the resulting word probability. The review was classified as positive and the ground truth was negative. The prediction variance was 0.13.

that this word, in the embedding space, is very close to a decision boundary. By examining the word probabilities over each class, it may give us more insight into the quality of our embedding and indicate when more training might be necessary.

6.6 Uncertainty Quantification In B-OREs

This section explores how uncertainty information can be used alongside Bayesian explanations to improve AI safety, illustrated on NLP tasks.

6.6.1 Feature Uncertainty

Using Definition 6.10, we have a probability that a given word $t_i \in t$ is in the ORE collection $\mathbf{E}^*$ for a BNN sample. We can obtain an *a priori* statistical guarantee on the linear feature attribution using the Chernoff bounds to determine the necessary sample size for a given absolute error bound and confidence level, similarly to Section 5.3.3. The probability for a word t_i, in essence, captures our uncertainty about the word t_i being part of an ORE implying the classification. It is clear that we are most uncertain when $\mathrm{P}_{\mathbf{E}^*,t}(t_i) = 0.5$, thus indicating that, for half of our networks in the Bayesian posterior, this word implies the classification and for the other half it does not. Thus, intuitively, we can measure our uncertainty, which we label *feature uncertainty*, in this quantity as the binary entropy of the probability.

Definition 6.11 (Feature Uncertainty) *Given $P(w|\mathcal{D})$, N, t, $f^{\hat{\mathbf{w}}s}$, $\mathcal{C}$, x, $\mathcal{E}$, $\mathcal{B}$ and $\mathbf{E}_{B\text{-}ORE,\mathbf{w}s} = \{\mathbf{E}^*{}_0, ..., \mathbf{E}^*{}_{N-1}\}$ as in Definition 6.10,* feature uncertainty *of input word $t_i \in t$ is given as:*

$$H(q) = -q\log_2 q - (1-q)\log_2(1-q)$$

where $q = \mathrm{P}_{\mathbf{E}^*,t}(t_i)$.

Now that we have a measure of uncertainty for each input word, we can measure the uncertainty of the entire input text using one of several possible aggregation strategies. First, we can use the Average Explanation Uncertainty (AEU):

$$\mathrm{AEU}(t) := \frac{1}{l}\sum_{i=0}^{l} H(\mathrm{P}_{\mathbf{E}^*,t}(t_i)) \tag{6.13}$$

We can also use the Maximum Explanation Uncertainty (MEU):

$$\mathrm{MEU}(t) := \max_{i\in\{0,\ldots,l\}} H(\mathrm{P}_{\mathbf{E}^*,t}(t_i)) \tag{6.14}$$

These measures stand in contrast to the typical notion of predictive uncertainty that are used in Bayesian deep learning, which rely only on the variance of the output variable. However, we can validate the usefulness of these two measures by examining their correlation with the predictive uncertainty. Intuitively, we expect that both the MEU and AEU will be high when predictive uncertainty is high and allows us to have a robust attribution of our predictive uncertainties to particular input tokens.

Discussion of Usage To the best of our knowledge, this is a novel notion of uncertainty attribution which can be used for debugging, active learning or ranking features by importance to the prediction. Feature uncertainty can be used in *a posteriori* debugging of Bayesian neural networks, for example, features with high uncertainty can be examined (either input by input, or globally over the whole training set) and one can determine whether more training data is needed, or whether a feature is (correctly) not associated with any class. In active learning frameworks, feature uncertainty could be used as a portion of an acquisition function to allow the model to *learn* new words. In the next section, we explore the use of feature uncertainty in explanation feature ranking, in order to provide the human decision maker with information on the importance of features to the decision. All of these uses contribute to safety assurance for AI systems in some way, and there are likely many more possible uses of feature uncertainty that we have yet to discover.

6.6.2 Feature Ranking

We now focus on the use of individual feature uncertainties (Definition 6.11) in conjunction with Bayes-optimal robust covering explanations (Definition 6.8). A B-ORCE gives us the explanation that most BNN samples agree on. By examining the feature uncertainty of each feature (in our case, word) in the B-ORCE, we can learn about the ordering of importance of the features that imply the prediction, among other uses.

Features in the B-ORCE with lower feature uncertainty tell us that the BNN is more certain that those features imply the prediction. For this reason, we can give the features a ranking of importance from most important (lowest feature uncertainty) to least important (highest feature uncertainty). Note that *all* features in the B-ORCE combined are what implies the prediction, but within that set it may be useful to know the ranking. For example, if a protected feature (e.g. the name of a director) appears in the B-ORCE with a low feature uncertainty (and therefore high rank), we can postulate that the BNN relies on this feature for prediction. However, if the same feature appears in the B-ORCE with a high feature uncertainty, we are less sure about the role of this feature in the BNN prediction, and, even though it is included in the B-ORCE, we may deem that to be acceptable over a given uncertainty threshold.

6.6.3 Global Feature Uncertainty Analysis

We now turn to explore what information feature uncertainty can provide about the BNN model as a whole. Consider looking at the feature uncertainty of all of the words in explanations that imply one class (positive or negative, in the case of the SST dataset). We hypothesise that the words in this set with the lowest feature uncertainty are most strongly associated with the predictive class. By examining these sets for each class, we get a global view of the words that contribute to each prediction along with their ranking, allowing the human interpreter to understand what is important to the model, and to spot potential problems.

In contrast to this, looking at the set of words with the highest feature uncertainty over all classes gives us an insight into which words the model does not associate with either class. We expect that, for a well-trained BNN on the sentiment analysis task, the majority of this set of words will be words not associated with a strong negative or positive sentiment (for example, conjunctions and adverbs). Empirically, we found this to be true, as demonstrated in the results section below.

6.7 Uncertainty Results

For the following experiments, we use the same experimental setup as in Section 6.4.2: a Bayesian FC model trained on the SST dataset. See Appendix A.4 for more detail.

6.7.1 Feature Uncertainty Results

Figure 6.9 displays several input texts and the resulting feature uncertainty of each word. We observe that, in almost all of our 50 input examples, the words with the most semantic meaning have the lowest feature uncertainty. As it is a known fact that OREs are sufficient for the prediction, this shows that, for B-OREs, there exists a robust explanatory landscape composed of relevant words (i.e. polarised terms).

Figures 6.10 and 6.11 show scatter plots of the average and maximum values of feature uncertainty versus predicitve uncertainty. In both cases, there is no significant correlation (the Pearson and Spearman coefficients are both close to zero).

6.7.2 Word Ranking Results

Figure 6.12 shows the same selection of input texts as in Figure 6.9, but with the B-ORCE highlighted in red. Our results show that the words in the B-ORCE consistently have a lower feature uncertainty than the majority of other words in the input. This solidifies the use of a B-ORCE to explain the prediction of a BNN, as we can see that the BNN is, in general, most certain about the words in the B-ORCE. Table 6.5 shows the same input texts with the actual ranking of the words in the B-ORCE. We observe that the highest ranking words have a significant semantic meaning, ex-

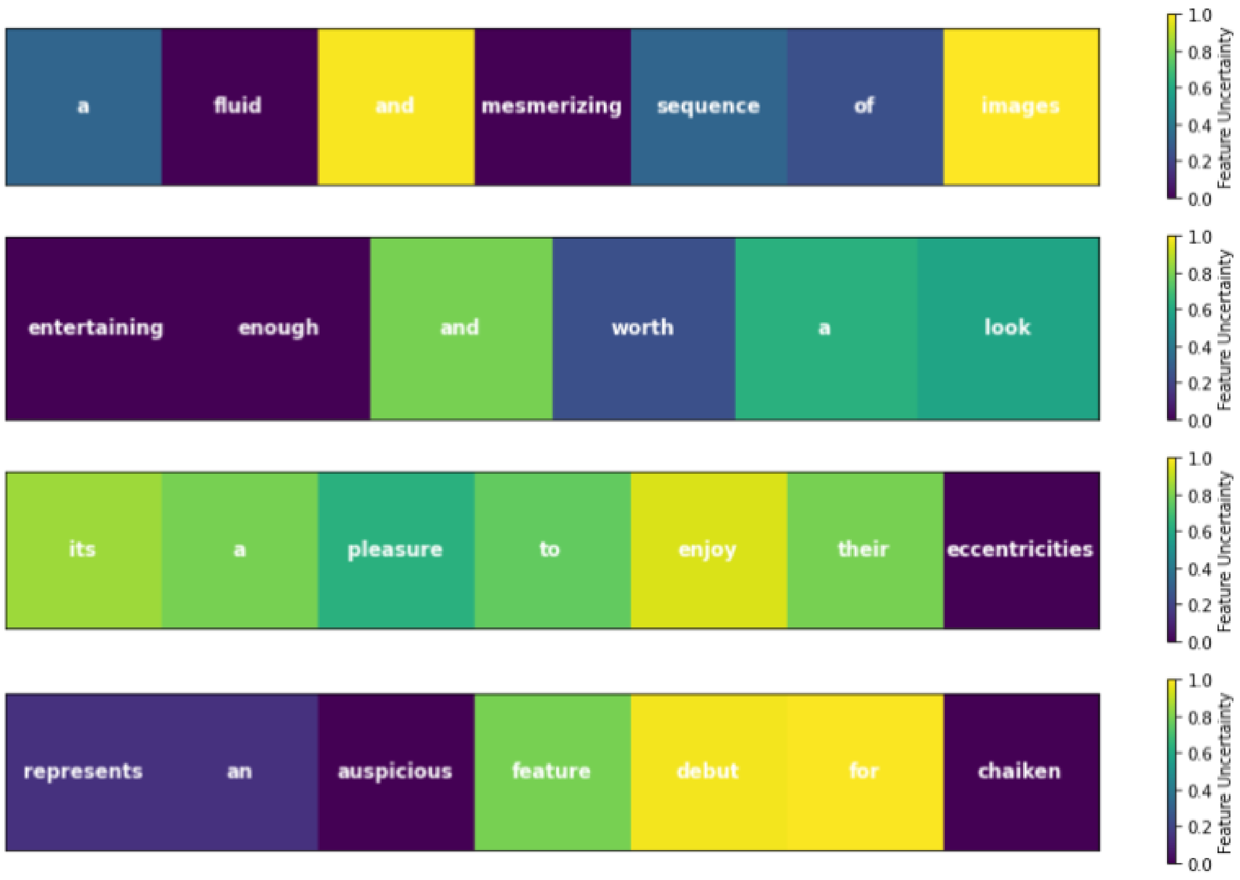


Figure 6.9: A selection of reviews from the SST dataset and the resulting feature uncertainty from the B-ORE.

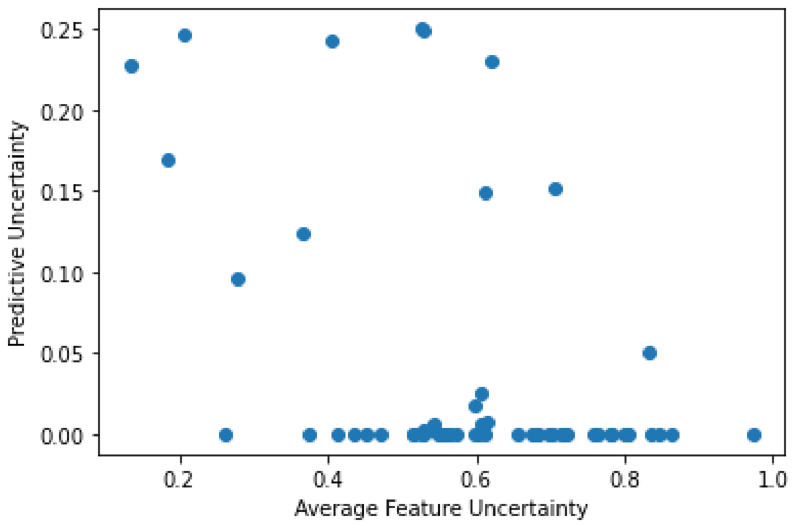


Figure 6.10: A scatter plot of the average feature uncertainty against predictive uncertainty over a sample of 50 reviews from the SST dataset.

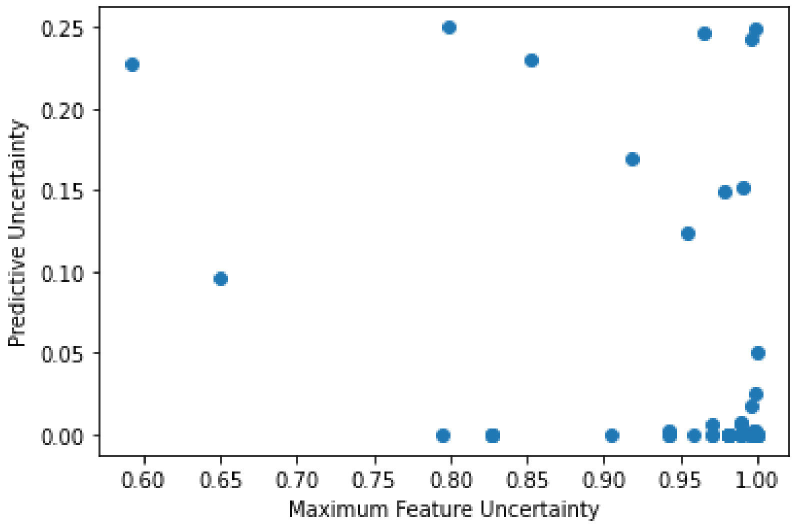


Figure 6.11: A scatter plot of the maximum feature uncertainty against predictive uncertainty over a sample of 50 reviews from the SST dataset.

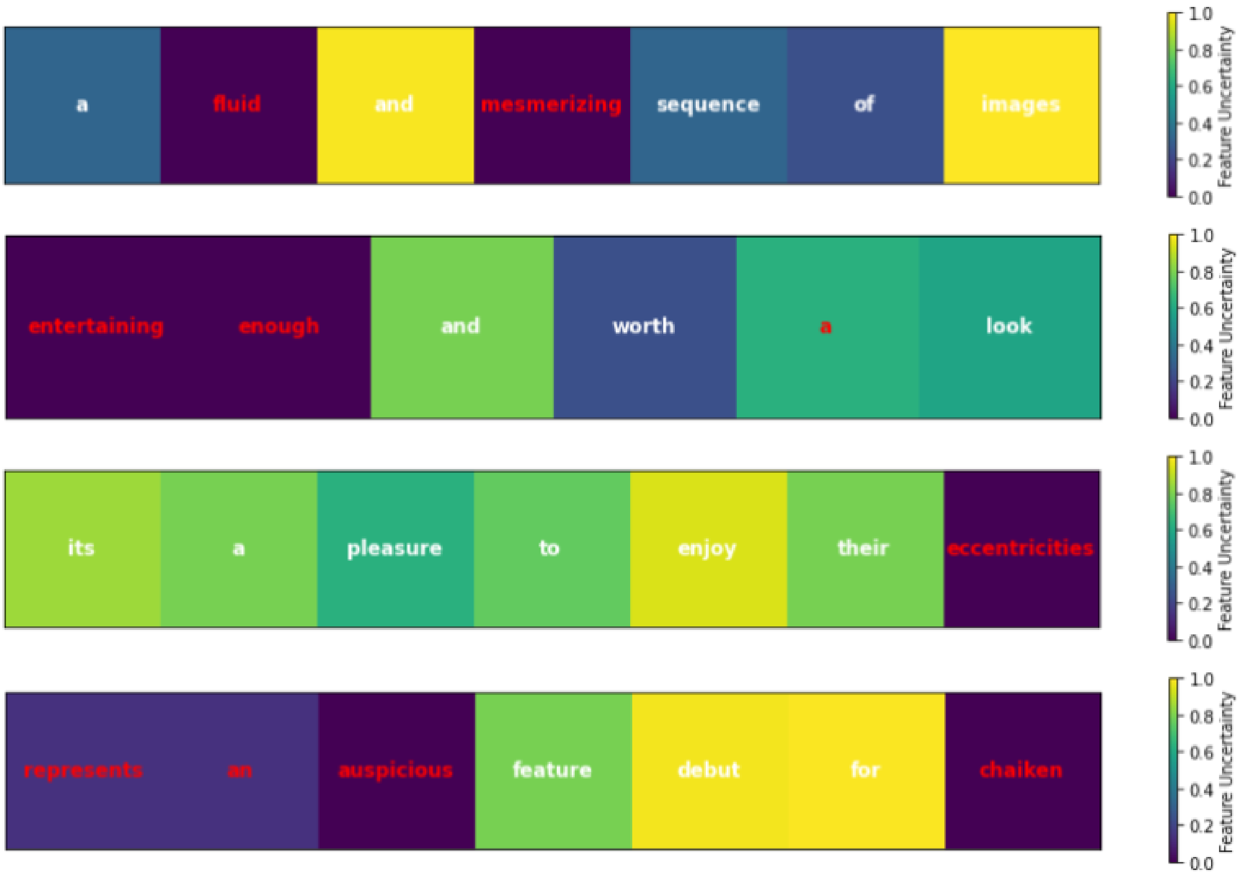


Figure 6.12: A selection of reviews from the SST dataset and the resulting feature uncertainty from the B-ORE. The B-ORCE is highlighted in red.

Review	**Ranked $E_{\textbf{B-ORCE}}$**
'a fluid and mesmerizing sequence of images'	{'fluid':1, 'mesmerizing':2}
'entertaining enough and worth a look'	{'entertaining':1, 'enough':2, 'a':3}
'it's a pleasure to enjoy their eccentricities'	{'eccentricities':1}
'represents an auspicious feature debut for Chaiken'	{'represents':3, 'an':4, 'auspicious':2, 'Chaiken':1}

Table 6.5: A selection of reviews from the SST dataset and their B-ORCE, with a ranking given to each word in the B-ORCE. 1 means most important (lowest feature uncertainty).

cept in the last case. The last review is showing us that the name 'Chaiken' is most important to the prediction, indicating a weakness in the BNN model and that data augmentation or more training is needed.

6.7.3 Mutual Information Analysis

The analysis of the mutual information (MI) of a set of B-OREs can shed light on the role played by word pairs in a model decision. Consider the first entry in Figure 6.9 and 6.12, where the B-ORCE is {fluid, mesmerizing}. When we analyse the mutual information (MI) between each pair of words, as depicted in Figure 6.13 (left), we observe that 'fluid' and 'mesmerizing' are both highly predictable when 'of' is in an ORE: please note that both 'fluid' and 'mesmerizing' have high probability of being in an ORE, opposed to 'of' whose odds are very low (right plot). On the other hand, the MI between 'fluid' and 'mesmerizing' is low, thus informing us that both these words appear with words different than 'of' in multiple OREs.

6.7.4 Global Feature Uncertainty Results

Figures 6.14 and 6.15 depict word clouds of all of the words appearing in positive and negative reviews respectively. The size of the word is determined by the feature uncertainty of that word (the average if the word appears in more than one review), where a larger word indicates a lower feature uncertainty. In both the positive and

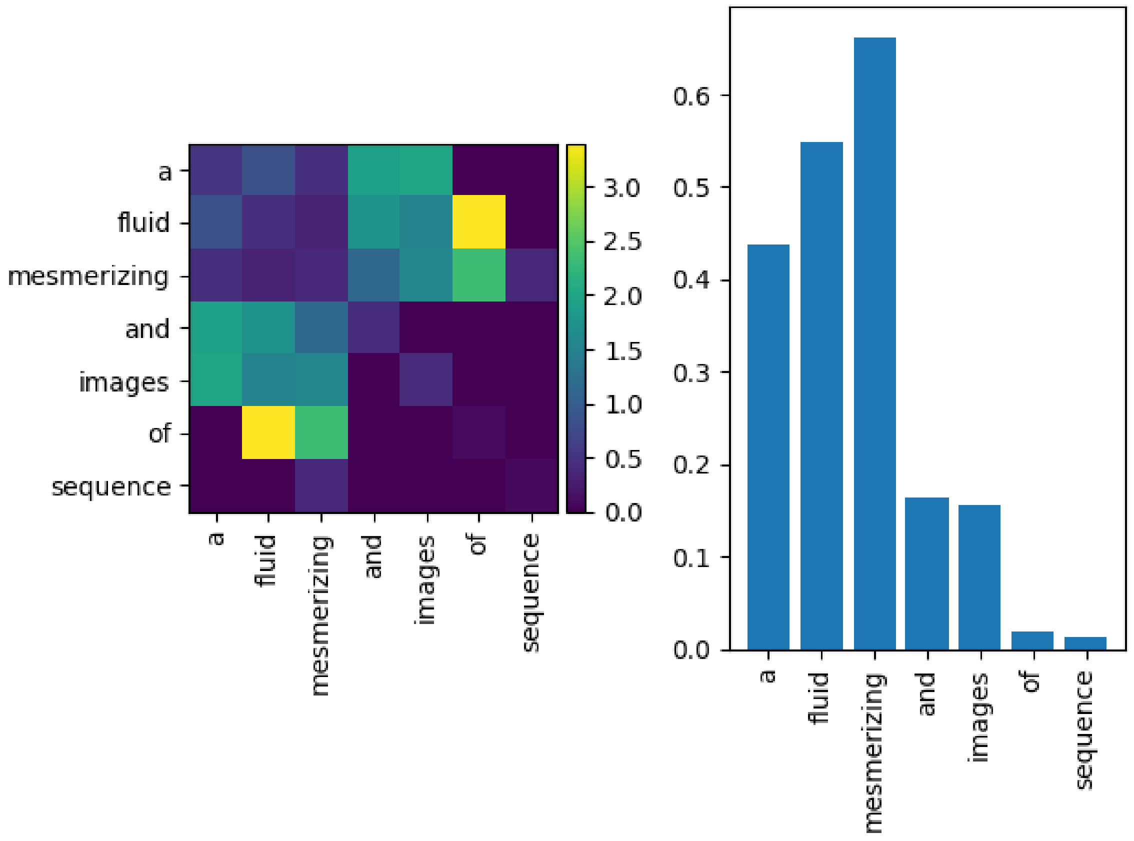


Figure 6.13: Mutual information of each word pair as they appear in a B-OREs (left), alongside the probability of each word to being part of an ORE when a deterministic neural network is sampled from the posterior distribution (right).

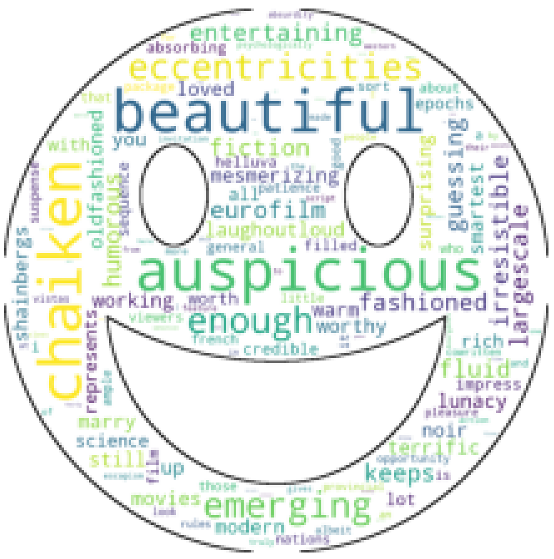


Figure 6.14: A word cloud depicting the words in reviews predicted as positive, where the size of the word indicates the average feature uncertainty of that word. A larger word indicates a lower feature uncertainty.

negative class, we observe that the largest words are, in general, words that are strongly associated with a positive (or negative) sentiment. This allows the human decision maker to view, at a glance, what words the BNN relies on to predict one class or the other. We observe that, in both word clouds, infrequent words ('chaiken', 'chafing' to name one for each figure) are ORE building blocks (i.e., their uncertainty is very low). As those words probably never appear in a training set, both the word clouds tell us that our BNN is overly robust to rare/out-of-vocabulary words and might thus require a re-calibration in that sense. We observe, based on this empirical study only, that in Figure 6.14 the terms are well-known and commonly used as opposed to those in Figure 6.15 that are infrequent. From these word clouds, we determine that this specific model behavior can be approximated by a rule such as 'if there is a positive term it is likely to be a positive review; if there is an infrequent term it is likely to be negative.'

On the other hand, Figure 6.16 depicts a word cloud of all the words appearing in any class, where the size of the word is again determined by their feature uncertainty. In this case, a larger word indicates higher feature uncertainty. This word cloud gives the human decision maker a view of what words the BNN is most unsure about, that

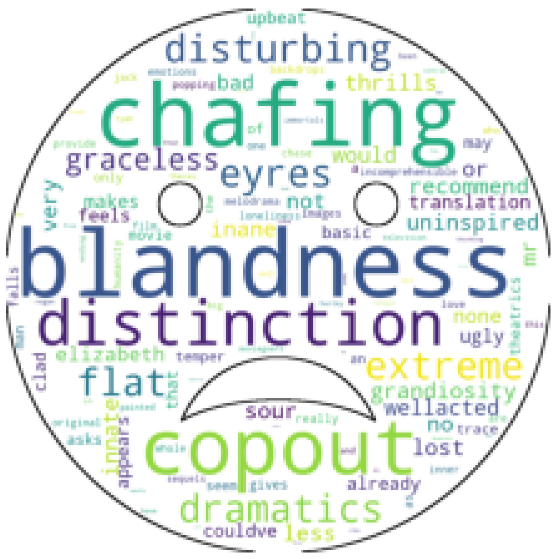


Figure 6.15: A word cloud depicting the words in reviews predicted as negative, where the size of the word indicates the average feature uncertainty of that word. A larger word indicates a lower feature uncertainty.

is, those words that the BNN does not strongly associate to either class.

Finally, Figure 6.17 groups words into their different categories and then plots the average feature uncertainty of each word type. Words are given types using the Natual Language Toolkit Python library [7]. Whilst most labels are straightforward, we note that in WordNet (where the label data comes from), adjectives are grouped into adjectives and adjective satellites where an adjective is the “center” of its group, and adjective satellites are synonyms of that adjective. For example, an adjective may be ‘dry’ and its satellite set might contain the words ‘arid’, ‘rainless’ etc. The ‘Other’ category covers all words not included in any other category.

In a well-trained BNN, we would expect the lowest feature uncertainty to occur when the word is an adjective, as descriptive words are most likely to be associated with a positive or negative sentiment. This is indeed the case, as demonstrated in the figure.

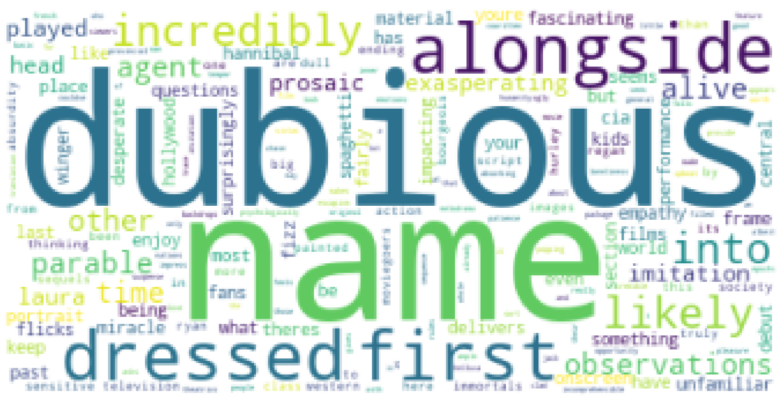


Figure 6.16: A word cloud depicting the words in all reviews, where the size of the word indicates the average feature uncertainty of that word. A larger word indicates a higher feature uncertainty.

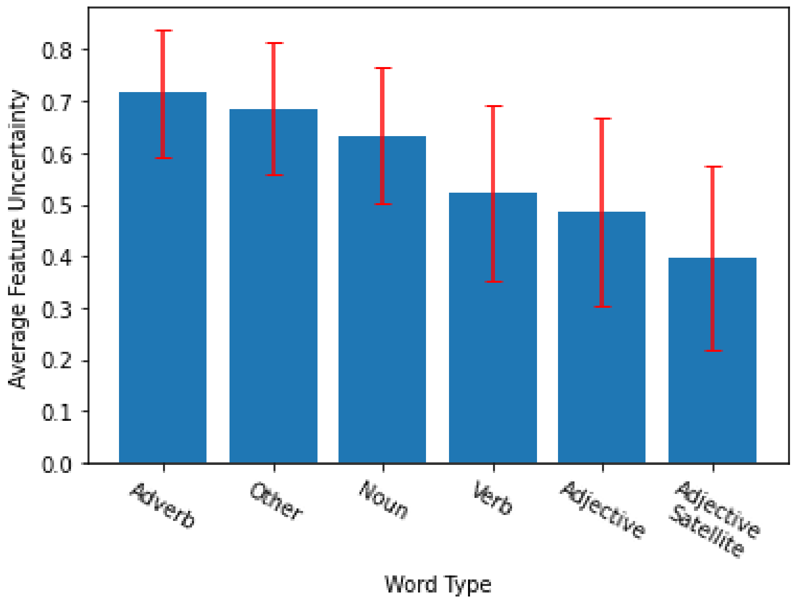


Figure 6.17: The average feature uncertainty, over all classes and reviews, of a group of words of a given type. Type ‘Other’ contains all words not grouped into another type.

6.8 Summary

In summary, we present a series of experiments and definitions that combine the uncertainty information available from Bayesian neural networks with local explanation methods. We show how to formulate Bayesian versions of feature score-based explanation methods such as LRP, as well as a Bayesian version of our OREs, called Bayes-optimal robust explanations (B-OREs). We define the notion of a covering explanation, which condenses the information produced from a number of BNN posterior samples into a single explanation, with a probability of the likelihood that that explanation is an explanation of a random sample. In the case of Bayes-optimal robust covering explanations, we obtain a probability for how likely the explanation is to imply the prediction. Further to this, we combine Bayesian covering explanations with feature uncertainty, to give an ordering of importance to each feature that appears in the covering explanation. Finally, we show that feature uncertainty can be used to give a global overview of the input features that the model most associates with each class.

Chapter 7

Conclusion and Future Work

Summary The questions posed and answered within this thesis can be summarised by the more abstract question: how can we improve the safety of AI systems in real-word safety-critical applications? In short, we answer this question by developing robust explanation methods with provable guarantees, and we advocate the use of Bayesian neural networks for the power of the uncertainty information that can be extracted from them.

Explainability can play a key role in ensuring safety, and explainability for neural networks is a fairly well studied field, yet very few existing techniques provide any guarantees on the resulting explanations. The techniques that *do* provide guarantees, e.g. that the explanation is logically sufficient to imply the prediction (robustness), do not scale well to even small CNNs and the unbounded nature of the perturbations used in verifying that the explanation is robust can often result in trivial explanations equal to the entire input. Formal methods that produce explanations with unbounded perturbations are very useful for certain applications, but not for many others. For example, if we consider an image of a traffic sign that we want a *robust* explanation for, then an all green image is a valid perturbation and would lead to every single pixel of the image being labeled as necessary to justify the prediction, and therefore constitute part of the explanation. Chapter 4 introduces the notion of an *optimal* robust explanation (ORE) that is both sufficient to imply the prediction *and* is optimal with respect to a cost function. We define multiple such cost functions, but the simplest is the length of the explanation. This leads to much more principled explanations that can be used by the human model creator or decision maker to monitor and combat model bias, to debug models and repair other non-formal explainers.

Traditional, deterministic neural networks suffer from a lack of information on confidence in the model prediction. For example, if we have a neural network that is trained to classify cats and dogs and we give it an image of an emu, it will tell us it is either a cat or dog since it has no way of telling us that it is neither. In contrast, Bayesian neural networks allow a measure of uncertainty to be paired with the output, so in our example, a BNN could tell us that the emu was a dog, but only with 5% certainty (where certainty is taken as the variance of N samples of the BNNs posterior weight distribution). Evidently, uncertainty information is very useful in ensuring safety, and, using that uncertainty information, we can decide that the decision of the network is nowhere near certain enough for us to believe it. Chapter 5 presents a statistical framework for evaluating the safety of BNN end-to-end controllers, with demonstration in a self-driving context. Using a BNN within this framework, we are able to provide bounds on the safety of the system with *a priori* statistical guarantees, among other things.

Finally, we combine the above work in Chapter 6. This is an exploratory chapter, with little prior literature to reference, that investigates how we can combine the powerful uncertainty information available from BNNs with neural network explanation methods and proposes a variety of metrics to measure robustness, to allow us to glimpse even more insight into model behaviour and stronger robustness guarantees.

Contributions The main contributions of this thesis are summarised here again.

- We present a method to derive local explanations, primarily for NLP models, with provable robustness and optimality guarantees. We call such explanation Optimal Robust Explanations (OREs). This method shares similarities with abduction-based explanations (ABEs) [58], but is better suited to any model (ín particular NLP models) where the unbounded nature of ABEs may result in trivial explanations, of no use to the human interpreter. We demonstrate that our approach can provide useful explanations for non-trivial fully-connected and convolutional neural networks on three widely used sentiment analysis benchmarks, and we provide a framework for computing our OREs with a choice of solution algorithms based on either hitting sets or minimum satisfying assignments with the Marabou and Neurify solvers. Finally, we compare OREs with a state-of-the-art explanation method, called Anchors, and we show that Anchors often lack prediction robustness in our benchmarks. We demonstrate the usefulness of our framework on tasks such as model debugging, bias evaluation and repair of non-formal explainers like Anchors.

- We present a statistical framework for evaluating the safety of end-to-end BNN controllers for applications in self-driving. This framework allows one to obtain and quantify the quality of uncertainty estimates for the controller's decisions, and provides bounds on the safety of the entire system with respect to a given criteria, with high probability and *a priori* statistical guarantees. We show that this framework can be used to evaluate model robustness to changes in weather, location and observation noise, and we empirically demonstrate that our real-time statistical estimates can be used to avoid a high percentage of collisions.

- Finally, we present a framework that combines the uncertainty information available from Bayesian neural networks with local explanation methods. We show how to formulate Bayesian versions of feature score-based explanation methods such as LRP, as well as a Bayesian version of our OREs, called Bayes-optimal robust explanations (B-OREs). We define the notion of a covering explanation, which condenses the information produced from a number of BNN posterior samples into a single explanation, with a probability of the likelihood that the explanation is an explanation of a random sample. In the case of Bayes-optimal robust covering explanations, we obtain a probability for how likely the explanation is to imply the prediction. Further to this, we combine Bayesian covering explanations with feature uncertainty, to give an ordering of importance to each feature that appears in the covering explanation. Finally, we show that feature uncertainty can be used to give a global overview of the input features that the model most associates with each class.

Appraisal Whilst we have made the strengths of our contributions to AI safety apparent, it is worth reiterating them along with taking time to reflect on the limitations of our methods.

In Chapter 4, our notion of OREs provide the human interpreter with useful, optimal and robust explanations for neural network models. This novel method provides guarantees on the explanations it produces, which is key in improving safety and advancing the field of AI explanations. A, perhaps temporary, drawback of our method is its ability to scale. Whilst OREs can be computed easily for applications such as NLP, where the input is a low-dimensional feature vector, our use of black-box solving oracles means that higher-dimensional feature vectors and larger networks are too computationally complex to be feasible. Any improvement in these third-party black-box oracles will lead to improvements in the performance of our methods, however.

Chapter 5 provides new methods to quantify and harness uncertainty information from Bayesian neural networks in safety-critical and real-time applications, with bounds on safety of the entire system. Our notion of probabilistic safety allows for greater safety in event-planning over a given time horizon, and our notion of real-time decision confidence allows a system to monitor and react to possible danger (areas of low confidence) that was previously unseen. A limitation of our work is the number of samples needed to achieve a small error bound and high confidence. For example, to achieve an error bound of 0.01 and confidence of 0.99, we must take 3516 samples. Whilst this is feasible on most computers using batch computations, this may pose a problem when considering smaller embedded systems found in some autonomous vehicles or similar, as it may be too slow to operate in real time.

In Chapter 6, we explore how useful explanations can be derived from decision functions with uncertainty information. Since there is so little literature in this area, any insights here are important and our work shows promise in helping to ensure the safety of AI applications. However, our work is by no means fully comprehensive and there are a lot of avenues left to explore. For example, one limitation of our work is that we do not systematically evaluate a range of posterior inference methods to determine which produces the highest quality uncertainty information for use in Bayesian explanations. Our work also suffers from the same lack of scalability as OREs due to the black-box oracle used.

Insights and Observations We now take some time to make note of a range of interesting insights and observations that we have noticed over the course of this thesis.

First, we discuss some general insights gained over the course of this project. Laws and regulations from governments around the world have caught up with the problems AI systems pose, and governance surrounding such systems is being put in place. Despite this, the *safety* of AI systems certainly does not seem to be a priority of the AI creators. The volume of research on these topics has significantly increased over the past decade. However, so has the volume of research on improving the accuracy, performance and architecture of neural networks without regard to safety (including explainability and local robustness, among other things), and with a much steeper gradient.

Bayesian uncertainty is an incredibly powerful asset to AI creators, but Bayesian models are underused. The literature introducing Bayesian models is definitely increasing, but more research must be done to convince the general AI community that Bayesian neural networks hold promise for a multitude of AI applications.

Following on from this, we note some specific observations that may be of use to researchers following in our footsteps. The highest quality of uncertainty information that we have seen is from Bayesian neural networks trained with the Hamitonian Monte Carlo (HMC) method of Bayesian inference. Unfortunately, this also seems to be the most difficult inference method to train a model with, and the author has spent countless hours optimising hyperparameters. Compared to networks trained with variational inference or Monte Carlo dropout, it is very difficult to achieve the same level of accuracy without spending a lot of time adjusting parts.

We conclude that explanation methods that consider negative space are more valuable than methods that do not. By negative space, we refer to parts of an input that are not above the average background level, for example, the black space surrounding the digits in the MNIST dataset. This negative space is often very important in classification: for example, we could consider a ‘5’ digit on a digital display as an ‘8’ without two sticks. By disregarding this information, we may overlook important features for the classification.

Producing black-box safety guarantees appears to be an unsolved and incredibly difficult problem. For this reason, we stress the need for an easy-to-use suite of software tools that allow AI developers to incorporate safety and robustness into their everyday work. Along similar lines, producing black-box explanation methods with robustness guarantees seems equally difficult. We expect that focus on scaling white-box explanation methods, and incorporation into common software tools, will increase uptake and improve safety and explainability over many fields of AI.

Further Work There are many directions one could take based on the work in this thesis. With regards to optimal robust explanations, a first step would be to explore a more general class of perturbations beyond the embedding space. For example, we could examine how OREs are formed with perturbations such as *adding* positively or negatively charged words (that should not affect the outcome), or we could consider perturbations in the input space directly. While the current work focuses on fully-connected and convolutional neural network architectures only, we note that our method would work with recurrent neural networks also. The difficulty here is that recurrent neural networks add a layer of complexity which is computationally challenging for our current framework, even though there are preliminary verification tools for recurrent networks. However, since our method is independent of the particular robustness oracle used, it can directly benefit from any advancement in these kinds of techniques, alongside a more compact representation of neural network models.

Examining how our Bayesian safety framework, detailed in Chapter 5, performs with real driving data, as opposed to simulation, could reveal some interesting insight. The well known “reality gap”, which describes the difficulty of transferring simulated experience into the real word, could pose unexpected problems, or even reveal previously unknown facts, and research into this would help to advance the safety of such real-world systems. In addition to this, we would like to evaluate safety in more environmental conditions and over longer journeys (even entire cities), which would allow us to identify problematic areas and which conditions they occur in. Furthermore, we could explore the use of our offline safety probabilities as a guide for active learning to increase data coverage. If the safety probability in a given city area is particularly low, more training data of that area could be used to improve the accuracy of the controller.

Similarly to the exploration of different inference methods in Chapter 5, we would like to explore the effect of different inference methods whilst training our Bayesian neural networks on the feature uncertainties we extract from our Bayes-optimal robust covering explanations. We suspect that, since different inference methods appear to provide a range of calibration levels of uncertainty, we will see a variation in uncertainty estimates between methods. Determining the best inference method for each specific application (primarily sentiment analysis since that is what we focus on) will provide the human decision maker with higher quality information about the models they are using. Scaling our B-ORCEs to networks with far more inputs and layers, such as for image classification, is another important step for this work. Similarly to the work in Chapter 4, our method will directly benefit from any advancements in robustness oracles of network representations. Finally, we would like to explore the use of our B-ORCEs and uncertainty estimates in the area of fairness. Since we are able to detect bias accurately in the model, this is an obvious extension of our work.

In the future, it would be desirable to combine all of the frameworks and tools developed within this thesis into a software package for Bayesian neural network safety and interpretation, so that researchers around the world may benefit from our work.

Appendix A

Experimental Setup

This Appendix details the datasets and experimental setup used throughout the thesis.

A.1 Datasets

This section introduces both industry standard and author-constructed datasets that are used in this thesis.

MNIST The Modified National Institute of Standards and Technology (MNIST) dataset is a large collection of images of handwritten digits that is commonly used for training image processing systems. It can be found here [28], and some example images can be seen in Figure A.1.

SST The Stanford Sentiment Treebank is a corpus with fully labeled parse trees that allows for a complete analysis of the compositional effects of sentiment in language. It can be found here [113], and an example input can be seen in Figure A.2.

Twitter The Twitter dataset contains 1.6 million Tweets, each labelled with their sentiment (positive or negative). It can be found here [43], and an example input can

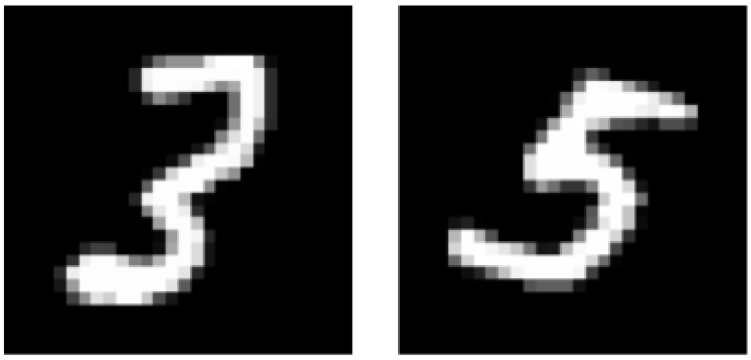
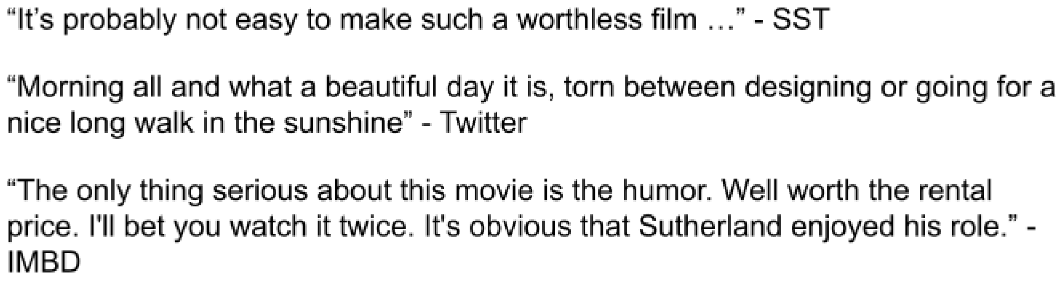


Figure A.1: Two example images from the MNIST dataset.


“It’s probably not easy to make such a worthless film …” - SST

“Morning all and what a beautiful day it is, torn between designing or going for a nice long walk in the sunshine” - Twitter

“The only thing serious about this movie is the humor. Well worth the rental price. I'll bet you watch it twice. It's obvious that Sutherland enjoyed his role.” - IMBD

Figure A.2: Three examples from the SST, Twitter and IMDB datasets.

be seen in Figure A.2.

IMDB The IMDB dataset contains 25, 000 movie reviews, each labelled with their score between 1 and 10, except that we map scores of 4 and below to 0 (negative sentiment) and 5 and above to 1 (positive sentiment), just as the original authors have done. It can be found here [82], and an example input can be seen in Figure A.2.

GTSRB The German Traffic Sign Recognition dataset is an image classification dataset with photos of traffic signs distributed over 43 classes. It can be found here [54], and some example images can be seen in Figure A.3.

Autonomous Driving Dataset The dataset used in Chapter 5 of this thesis was handcrafted by the author. Using the Carla simulator [31], the autonomous vehicle

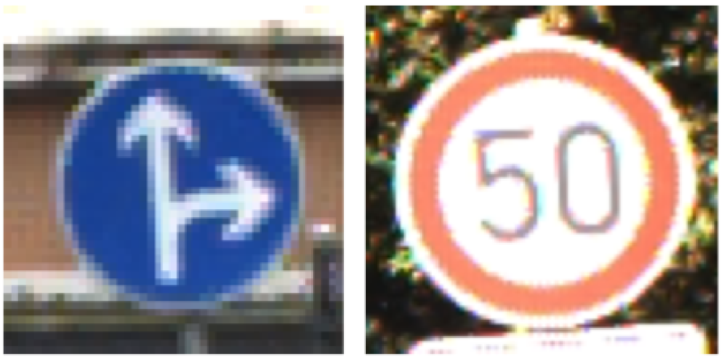


Figure A.3: Two example images from the GTSRB dataset.

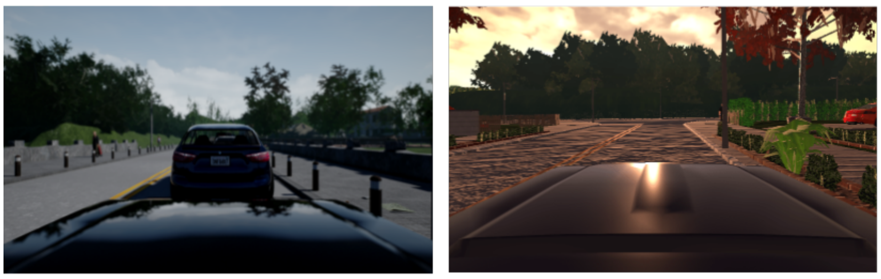


Figure A.4: Two example images from the autonomous driving dataset.

was driven around the map using a games controller, whilst images along with the steering angle (the angle of the controllers analogue stick), location and speed were recorded. This was done over a range of different environments (city, countryside) and for a range of different light levels and weather events. Some example images can be seen in Figure A.4.

A.2 Experimental Setup for Chapter 4

We have trained fully connected (FC) and convolutional neural network (CNN) models on sentiment analysis datasets that differ in the input length and difficulty of the learning task. Experiments were parallelized on a server with two 24-core Intel Xenon 6252 processors and 256GB of RAM, but each instance is single-threaded and can be executed on a low-end laptop. We considered 3 well-established benchmarks for sentiment analysis: SST [112], Twitter [43] and IMDB [82] datasets. From these, we have chosen 40 representative input texts, balancing *positive* and *negative* examples.

Table A.1.1: Training Details

	TWITTER	**SST**	**IMDB**
Inputs (Train, Test)	$1.55M, 50K$	$117.22K, 1.82K$	$25K, 25K$
# Classes	2	2	2
Input Length (max, max. used)	88, 50	52, 50	2315, 100
Model Types	FC, CNN	FC, CNN	FC, CNN
# Layers (min,max)	3,6	3,6	3,6
Test Accuracy (min, max)	0.77, 0.81	0.82, 0.89	0.69, 0.81
# Parameters (min,max)	$3K, 18K$	$1.3K, 10K$	$5K, 17K$

Table A.1.2: Experimental Details

	TWITTER	**SST**	**IMDB**
Sample Size	40	40	40
Review Length (min-max)	10, 50	10, 50	25, 100

Table A.1: Datasets used for training/testing and extracting explanations. We report various metrics concerning the networks and the training phase (including accuracy on the test set), while in Table A.1.2 we report the number of texts for which we have extracted explanations along with the number of words considered when calculating OREs: samples were chosen to reflect the variety of the original datasets, i.e., a mix of long/short inputs equally divided into positive and negative instances.

Embeddings are pre-trained on the same datasets used for classification [22].

We performed our experiments on networks with up to 6 layers and $20K$ parameters. FC networks are made up of a stack of fully connected layers, while CNNs additionally employ *convolutional* and *max pool* layers: for both CNN and FC models, the decision is taken through a *softmax* layer. *Dropout* is added after each layer to improve generalization during the training phase. With regard to the embeddings that the models equip, we found that the best trade-off between the accuracy of the network and the formal guarantees that we need to provide is reached with low-dimensional embeddings, thus we employed optimized vectors of dimension 5 for each word in the embedding space. This is in line with the experimental evaluations conducted in [96], where for low-order tasks such as sentiment analysis, compact embedding vectors allow one to obtain good performance, as shown in Table A.1. Table A.1 gives a full view of the models used.

Both the HS and MSA algorithms have been implemented in Python and use Marabou [61] and Neurify [125] to answer robustness queries. Marabou is fast at verifying ReLU FC networks, but it becomes memory intensive with CNNs. On the other hand, the symbolic interval analysis of Neurify is more efficient for CNNs. A downside of Neurify is that it is less flexible in the constraint definition (inputs have to be represented as squared bi-dimensional grids, thus posing problems for NLP inputs which are usually specified as 3D tensors).

In the majority of the experiments, we opted for the kNN-box perturbation space, as we found that the k parameter was easier to interpret and tune than the ϵ parameter for the ϵ-ball space, and improved verification time.

A.3 Experimental Setup for Chapter 5

A.3.1 Data Acquisition and Processing

CARLA The experiments in this chapter use the CARLA simulator, a state-of-the-art, open-source simulator for autonomous driving research [31]. CARLA has been developed specifically to support the training of autonomous driving systems and is completely open-source. The environment CARLA provides ranges from simple country roads to full city simulations, including other vehicles, cyclists, pedestrians, emergency services, fully dynamic weather, working traffic lights and more. Importantly, the 3D models created for the CARLA simulator mean that images taken within the simulator are very similar to the real world. Finally, CARLA provides some autonomous driving baselines within it, and allows the user to generate training data using an "autopilot", which knows the whole state of the environment. However, we stress that any simulator can be used within this framework, assuming it can simulate car trajectories, and generate images that can be used by the controller, for example the much more simple Udacity simulator [122].

Processing All training data, which consists of (*image, steering angle*) pairs, was acquired within the CARLA simulator, either through manual driving or use of the built-in autopilot. During experiments, we also make use of the car's trajectory data, which is provided in the form of a list of GPS coordinates from the simulator. Images are converted to grayscale and scaled to a size of 64×48 pixels, and steering angles (recorded between -1 and 1) are binned into intervals of tenths. The data recorded consists of three scenarios: a right turn on a roundabout and a straight segment of a road with and without an obstacle (stationary vehicle). It is possible to vary the weather within the simulator, however the weather condition in all of the training data is “clear noon”.

A.3.2 Network Architecture

The architecture of the neural network models used in this chapter are based on Pilot-Net [11], NVidia's first end-to-end controller for self-driving. Traditionally, steering angle prediction has been treated as a regression problem. However, it has been shown that posing regression tasks as classification tasks often shows improvement over direct regression training [103]. In addition to this, although theoretically continuous, steering angle in the real-world is commonly a discrete variable due to mechanical limitations. Therefore, we have modified the final layers of our neural network architectures to have neurons equal to the number of classes (variable per experiment), and a softmax activation function.

We fix the convolutional layers and first fully connected layer, and use the final layers for uncertainty extraction (similarly to [93]). For MCD, we use concrete dropout [41] on the final three layers (and leave the fourth fully connected). For VI and HMC, we use an additional four fully connected layers, where the input to the first layer are the features extracted from the final fixed network layer. All other activation functions are ReLU. A diagram demonstrating these designs can be seen in Figure A.5.

In our experiments, for an observation o we have that $\pi(o)$, the BNN decision, is given by the most likely class. However, we stress that other choices for $\pi(o)$ are possible according to the particular loss function (see e.g., [8]) and the methods presented in this paper are independent of the criteria for assigning $\pi(o)$. For example, $\pi(o)$ could be the mean of n samples of the BNN in the case of regression with the mean squared loss function (called model averaging [39]).

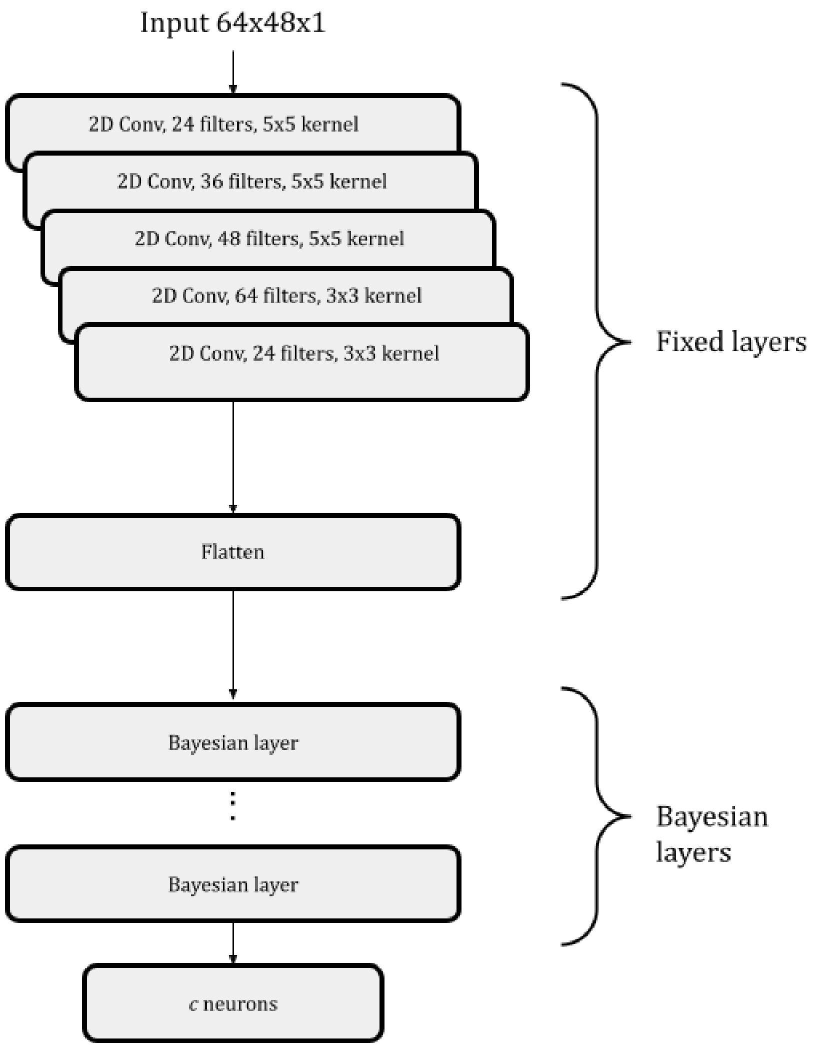

Input 64x48x1
2D Conv, 24 filters, 5x5 kernel
2D Conv, 36 filters, 5x5 kernel
2D Conv, 48 filters, 5x5 kernel
2D Conv, 64 filters, 3x3 kernel
2D Conv, 24 filters, 3x3 kernel
Fixed layers
Flatten
Bayesian layer
Bayesian layer
Bayesian layers
c neurons

Figure A.5: The architecture of the neural networks used in this chapter. Activation functions are all ReLU except the last layer which uses softmax.

A.3.3 Network Training

This section summarises how the networks for each inference technique were trained.

MCD The cross-entropy loss function is used, along with the ADAM optimizer with a learning rate of 0.0001 and the dropout probabilities tuned with concrete dropout, which converged to (0.1, 0.08, 0.08). The batch size is 16 and it was trained for 25 epochs.

VI Features are first extracted from the final fixed layer of the network using the weights from the MCD network for these initial layers. Then, we impose prior distributions on the weights of the final four, fully-connected layers. These are normal distributions with mean 0 and variable variance. Inference was then performed with a step size of 0.0001, a minibatch size of 512 and 30000 iterations, using the Edward python library [121], and the posterior is also in the form of a normal distribution.

HMC The prior distributions for the HMC networks are as above; however, the posterior here is an empirical distribution based on sampling with the HMC algorithm. We use 10 steps of numerical integration prior to judging the acceptance criteria of each sample.

A.4 Experimental Setup for Chapter 6

A.4.1 Bayesian Feature Score Methods

For these experiments, we have trained a Bayesian FC model on the MNIST dataset and a Bayesian CNN model on the GTSRB dataset. Experiments were performed on a server with two 24-core Intel Xenon 6252 processors and 256GB of RAM, but as each experiment only utilised one core and roughly 8GB of RAM, they could be run on a desktop PC. The input images have been scaled to a size of 14×14 pixels in the case of MNIST, and 30×30 pixels for GTSRB, though colour information has been retained where available.

The architecture of the Bayesian FC model is as follows: a dense layer of size 256, a dense layer of size 128 and finally a dense layer with size equal to the number of classes (10). The total number of parameters is $183,268$. All activation functions are *ReLU* except the last which is softmax. The network, constructed and trained using the DeepBayes python library [127], uses variational online Gauss Newton (VOGN) [68] during inference, along with the sparse categorical crossentropy loss. The model was trained for 25 epochs with a learning rate of 0.25, and achieved an overall accuracy of 97.1% on the test dataset.

The architecture of the Bayesian CNN model is as follows: a convolutional layer with 4 filters and kernel size 3×3, a max pooling layer, a convolutional layer with 8 filters and kernel size 3×3, a max pooling layer, a flattening layer, a dense layer of size 128 and finally a dense layer with size equal to the number of classes (43). The total number of parameters is $56,259$. All activation functions are *ReLU* except the last which is softmax. The network, constructed and trained using the DeepBayes python library, uses VOGN during inference, along with the sparse categorical crossentropy loss. The model was trained for 25 epochs with a learning rate of 0.25, and achieved an overall accuracy of 95.2% on the test dataset.

All experiments have been implemented in Python3, and libraries for each of the three explanation methods are as follows: [34] for LRP, [52] for IG and [80] for SHAP. The number of samples taken for each input network pair in all of the experiments is 100.

A.4.2 Bayes-Optimal Robust Explanations

Similarly to the experimental setup for our ORE analysis, we have trained both fully connected (FC) and convolutional (CNN) models on the Stanford Sentiment Treebank (SST) sentiment analysis dataset, except that these networks are Bayesian in nature. Experiments were performed on a server with two 24-core Intel Xenon 6252 processors and 256GB of RAM, but as each experiment only utilised one core and roughly 8GB of RAM, they could be run on a desktop PC. From the SST dataset, we have chosen 50 representative input texts, balancing positive and negative examples. Embeddings are pre-trained on the same datasets used for classification. As the majority of the movie reviews within the SST dataset are very short, we limit our input to 10 words.

In practice, we have found that increasing the number of words above 10 does not significantly increase the accuracy of our networks.

We performed our experiments on both FC and CNN networks with up to 6 layers and $20K$ parameters, where FC networks are made up of a stack of fully connected layers, while CNNs additionally employ convolutional and max pooling layers. The final decision for both networks types is through a softmax layer, and all other activations are $ReLU$. The specific experimental results below come from a Bayesian FC network with 3 layers of sizes 16, 8 and 2. The network, constructed and trained using the DeepBayes python library, uses VOGN during inference, along with the sparse categorical crossentropy loss. The model was trained for 25 epochs with a learning rate of 0.25, and achieved an overall accuracy of 81.5% on the test dataset. We chose this model to demonstrate our method as both larger and CNN models did not increase the accuracy by more than 1%; however, they significantly increased experimental runtime, as the blackbox oracles that answer robustness queries are very sensitive to network size.

A.5 Uncertainty Results

For these experiments, we use the same experimental setup as in Section A.4.2: a Bayesian FC model trained on the SST dataset.

Appendix B

Code Guide

This Appendix details how to access, install and use the code provided with this thesis, including third party requirements. All code is written in Python3. The code can be found here:

`https://github.com/Rhiba/thesis-code`

B.1 Installation

This section describes the installation process for parts of the code, including installation of third party programs and libraries.

B.1.1 Prerequisite Python3 Packages

The code in this thesis uses several Python packages that can be installed by invoking the command: `pip3 install <package name>`. These packages are:

- `tensorflow`
- `keras`
- `pickle`
- `numpy`
- `matplotlib`

- `scipy`
- `socketio`
- `eventlet`
- `flask`
- `pillow`
- `python-sat`
- `tqdm`
- `pandas`

B.1.2 Carla Simulator

To run the Carla simulator, it must first be downloaded from here `https://github.com/carla-simulator/carla/releases`. Since Carla is built using the Unreal Engine, this must also be installed following the instruction on this page `https://docs.unrealengine.com/4.27/en-US/Basics/InstallingUnrealEngine/`. Once the Unreal Engine has been installed and the Carla simulator has been downloaded, one can follow the instructions on the following page `https://carla.readthedocs.io/en/stable/getting_started/` to install Carla.

B.1.3 Marabou

In order to install Marabou, the repository must be cloned using the command: `git clone https://github.com/NeuralNetworkVerification/Marabou`. One must then follow the installation instructions on the same Github page to fully install the package.

B.1.4 Neurify

First, Neurify requires that the package `lpsolve` is installed. This can be installed using your systems package manager, for example, the Ubuntu command is `apt install lp-solve`. Neurify should then be downloaded using the command `git clone https://github.com/Rhiba/Neurify`, and the installation instructions on the same Github page should then be followed. Note that any input into Neurify should be normalised in the range $0 - 1$. This version of Neurify is a fork of the original library that has been slightly altered for our work.

B.1.5 Custom Neurify Shared Library

Since Neurify is written in C and the code in this thesis is all Python, we have created our own shared library so that we are able to call Neurify commands from within Python. In the Neurify directory, go into the general folder then follow these instructions:

1. Edit the `makefile` to change paths to the relevant paths on your computer.
2. Run `make`.
3. Run `cp libentails.so /home/username/your/lib/dir`
4. Make sure your shared library directory is in your LD library path by running `export LD_LIBRARY_PATH=/home/username/your/lib/dir:$PATH`
5. Run `cp entails.h /home/username/your/include/dir`

NB. You may need to compile lp_solve on your own machine if your version from a package manager doesn't work.

B.1.6 Explainer Libraries

The library containing the Layerwise Relevance Propagation implementation can be installed using this command: `git clone https://github.com/KaiFabi/LayerwiseRelevancePropagation.git`.

The library containing the Integrated Gradients implementation can be installed using this command: `git clone https://github.com/hiranumn/IntegratedGradients`.

The library containing the SHAP implementation can be installed using this command: `git clone https://github.com/slundberg/shap`.

B.1.7 DeepBayes

DeepBayes is the library used to train Bayesian neural networks with various different inference methods. It can be installed by running the command: `git clone git@github.com:matthewwicker/deepbayes.git`.

B.2 Usage

This section gives examples of code usage, separated by chapter.

B.2.1 Chapter 4 Code Usage

To launch an Optimal Robust Explanation (ORE) experiment, go to the `chap-4\OREs\abduction_algorithms\experiments\{DATASET}\` folder, where `{DATASET}` is either 'SST' 'Twitter' or 'IMDB', and run one of the python files (`.py` extension). We report a few usage examples.

ORE for a Fully Connected Model on a Sample Review

Let's suppose that you want to extract an ORE from a sample review and you are using an FC model, trained on SST dataset, with 25 input words and using the k Nearest Neighbours bounding box technique with k=10. You have to navigate to `Explanations\abduction_algorithms\experiments\SST\` folder and run the following:

```
python3 smallest_explanation_SST_fc_knn_linf.py -k 10 -w 5 -n 25
-i 'This is a very bad movie'
```

The algorithm identifies the words *is* and *bad* as an ORE. Additionally, results are logged and stored in a folder inside `results\HS-clear` (the complete path that is reported in the logs of the execution).

If you want to obtain the same result but using Minimum Satisfying Assignment algorithm the command is:

```
python3 smallest_cost_explanation_SST_fc_knn_linf.py -k 10 -w 5
-i 'This is a very bad movie'
```

Solving an hard instance with Adversarial Attacks

Let's suppose that you want to extract an ORE from a Twitter text and you are using a CNN model, trained on Twitter dataset, with 25 input words and using the k Nearest Neighbours bounding box technique with k=8. You can improve the convergence on hard instances with Adversarial Attacks by simply specifying the number of attacks that you want to launch (parameter -a/–adv). You have to navigate to the `Explanations\abduction_algorithms\experiments\Twitter\` folder and run the following:

```
python3 smallest_explanation_Twitter_cnn2d_knn_linf.py -k 8 -w 5
-n 25 -a 500 -i 'Spencer is not a good guy'
```

The algorithm identifies the words *Spencer* and *not* as an ORE (alongside a *PAD* token that highlights a problem with the model robustness). Again, results are stored in a folder inside `results\HS-clear` (the complete path that is reported in the logs of the execution).

Detecting Decision Bias Using Cost Functions

Let's suppose you want to check whether an explanation will always contain the name of a character, actor or director: something which ideally should not be used for classifying whether a review is positive or negative. To do this, you can run the following command:

```
python3 smallest_HScost_explanation_SST_fc_knn_linf.py -k 27 -w 5
-a 0 -i "Austin Powers in Goldmember has the right stuff for
summer entertainment and has enough laughs to sustain interest to
the end" -u False -x 'austin,powers,goldmember'
```

If the excluded word still appears in the ORE (which it does in this example, "powers" is present), you know that no explanation exists without it and therefore that there is a decision bias here.

If you want to do the same but using MSA algorithm the command is:

```
python3 smallest_cost_explanation_SST_fc_knn_linf_alternate_cost.py
-k 27 -w 5 -i "Austin Powers in Goldmember has the right stuff for
summer entertainment and has enough laughs to sustain interest to
the end"
```

If you want to permanently exclude the word from the explanation use:

```
python3
smallest_cost_explanation_SST_fc_knn_linf_alternate_cost_exclude.py
-k 27 -w 5 -i "Austin Powers in Goldmember has the right stuff for
summer entertainment and has enough laughs to sustain interest to
the end"
```

Run multiple instances in parallel

You can run several ORE instances in parallel (e.g., on a server) by launching the `run-exp_MODEL_DATASET_knn_linf.sh` script in each `Experiments\DATASET` folder, where DATASET is either 'IMDB', 'SST' or 'Twitter' and MODEL is either 'fc' or 'cnn'. This requires the `screen` bash command (available on any Linux distribution) to manage the various instances.

B.2.2 Chapter 5 Code Usage

To launch an experiment from Chapter 5, go to the `chap-5\` folder. We report a few usage examples.

Gathering Training Data from CARLA To gather training data from the CARLA simulator, using its built in autopilot, you must first run the simulator using the command: `./CarlaUE4.sh &`. Then, run `python manual_control.py`. Now that both the simulator and control file are running, press the `P` key to turn on the autopilot, and then press the `R` key to start recording training data. To stop recording, press the `R` key again. All of your training data (images, steering control, location coordinates) can be found in the `_out` folder that is created by this script. It is also possible to record training data this way whilst manually controlling the car with the arrow keys: just omit the autopilot instruction.

Driving the Car Using Your Own Model To drive the car in CARLA simulator using your own model, first open the simulator with the command `./CarlaUE4.sh &`. Then, run the following command:

```
python drive.py --json path_to_model_json.json \
--model path_to_model_weights.h5
```

This command requires a keras model architecture saved in `json` format, and the keras model weights saved in `h5` format.

Real time Decision Confidence with VI in Sunny Weather (With Obstacle) To run this experiment, first open the simulator with the command `./CarlaUE4.sh &`. Then, run the following command:

```
python3 SafetySimulation/VI_Mean_obs_weather.py \
-f path_to_model --W 0 -l A
```

The results of this run (including warnings and their level, and uncertainty readings) can be found in the `_out` folder generated by the code.

Real time Decision Confidence with HMC in Rainy Weather (No Obstacle) To run this experiment, first open the simulator with the command `./CarlaUE4.sh &`. Then, run the following command:

```
python3 SafetySimulation/HMC_Mean_obs_weather.py \
-f path_to_model --W 10
```

The results of this run (including warnings and their level, and uncertainty readings) can be found in the `_out` folder generated by the code.

Probabilistic Safety with MCD in Sunny Weather at Dusk To run this experiment, first open the simulator with the command `./CarlaUE4.sh &`. Then, run the following command:

```
python3 SafetySimulation/Drop.py \
-f path_to_model -s B -w 2
```

The results of this run (including warnings and their level, and uncertainty readings) can be found in the `_out` folder generated by the code.

B.2.3 Chapter 6 Code Usage

To launch an experiment from Chapter 6, go to the folder `chap-6\experiments\{DATASET}\` where `{DATASET}` is either 'SST' 'GTSRB' or 'mnist', and run one of the Jupyter notebook files (`.ipynb` extension). We report a few usage examples.

Generating a Bayesian Covering Explanation for MNIST using LRP Suppose you wish to generate a Bayesian covering explanation for an input image from the MNIST dataset, using the Layerwise Relevance Propagation explanation method. To do this, open the notebook file `...mnist\mnist_LRP_bayesian_coverage.ipynb` and first change any relevant import paths to match those on your machine. Then, run the first five cells in the file, and in the sixth, change the value of `n` to correspond to the input you wish to generate a Bayesian covering explanation for. For the naive version of the explanation, as defined in Equation 6.7, run cells seven and eight, and for the improved version, run the rest of the cells in the notebook.

Generating a Bayesian Covering Explanation for GTSRB using IG Suppose you wish to generate a Bayesian covering explanation for an input image from the GTSRB dataset, using the Integrated Gradients explanation method. To do this, open the notebook file `...GTSRB\GTSRB_IG_bayesian_coverage.ipynb` and first change any relevant import paths to match those on your machine. Then, run the first seven cells in the file, and in the eighth, change the value of `n` to correspond to the input you wish to generate a Bayesian covering explanation for. For the naive version of the explanation, as defined in Equation 6.7, run cells nine, ten and eleven, and for the improved version, run the rest of the cells in the notebook.

Generating a B-ORCE for the SST Dataset Suppose you wish to generate a B-ORCE for an input from the SST dataset. To do this, open the notebook file `...SST\SST_BORCE_generation.ipynb` and change any relevant import paths to match those on your machine. Then, run all cells up to the one titled "Random positive example runs", and then edit the value of `n` in that cell to be the index of the SST review you wish to generate a B-ORCE for. Next, run the calculation cell (this is quite verbose so expect a lot of output), and finally the results analysis cell. The output gives you all possible explanations and their covering probability, with the top one (highest probability) being the covering explanation.

Ranking the Features of a B-ORCE Suppose you wish to generate a B-ORCE for an input from the SST dataset, and you also wish to rank the features in the B-ORCE by order of importance based on their feature uncertainty (as well as visualising all of this). To do this, you must first generate the B-ORCE for your desired input using the `...SST\SST_BORCE_generation.ipynb` notebook file (as detailed in the above example). This will save the B-ORCE to disk ready for use in the next notebook. Then, open the notebook file
`...SST\individual_and_global_feature_uncertainty_exploration.ipynb`
and change any relevant import paths to match those on your machine. In cell seven, change the array `ns` to contain only the index of the input you have generated a

B-ORCE for (this can also handle more than one input so long as a B-ORCE has been generated for all of them, if you wish). Now run cells $1 - 7$. Now skip down to cell 13 (beginning with the comment "`# visualise covering explanation with feature uncertainty`") and run it: this will produce your desired result.

Examining the Highly Uncertain Features of a Model Suppose you wish to examine the features of a model, over a large number of B-ORCEs, that are highly uncertain (perhaps to aid in model debugging). To do this, you must first have generated B-ORCEs for a large number of inputs using the same model (to do this, see the example above). Then, open the notebook file
`...SST\individual_and_global_feature_uncertainty_exploration.ipynb`
and change any relevant import paths to match those on your machine. In cell seven, change the array `ns` to contain all of the indexes of the inputs that you have generated B-ORCEs for. Now run cells $1 - 7$ and skip down to cell 15 (beginning with the comment "`# generate word cloud from high uncertainty words`") and run it: this will produce your desired result.

Bibliography

[1] Alessandro Abate, Maria Prandini, John Lygeros, and Shankar Sastry. Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems. *Automatica*, 44(11):2724–2734, 2008.

[2] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2890–2896, 2018.

[3] Alexander Amini, Ava Soleimany, Sertac Karaman, and Daniela Rus. Spatial uncertainty sampling for end-to-end control. In *Workshop on Bayesian deep learning, NIPS*, 2017.

[4] Marcelo Arenas, Daniel Baez, Pablo Barceló, Jorge Pérez, and Bernardo Subercaseaux. Foundations of symbolic languages for model interpretability. *Advances in neural information processing systems*, 34, 2021.

[5] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

[6] M. Baroni et al. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *ACL*, pages 238–247, 2014.

[7] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[8] Christopher Bishop. *Pattern recognition and machine learning.* Springer, 2006.

[9] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *Proceedings of the 32nd international conference on international conference on machine learning-volume 37*, pages 1613–1622, 2015.

[10] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *Nvidia developer blog*, 2016.

[11] Mariusz Bojarski, Philip Yeres, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Lawrence Jackel, and Urs Muller. Explaining how a deep neural network trained with end-to-end learning steers a car. 2017.

[12] Luca Bortolussi, Luca Cardelli, Marta Kwiatkowska, and Luca Laurenti. Central limit model checking. *ACM trans. comput. logic*, 20(4):19:1–19:35, July 2019.

[13] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of Markov chain Monte Carlo.* CRC Press, 2011.

[14] Rudy Bunel, Alessandro De Palma, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip Torr, and M Pawan Kumar. Lagrangian decomposition for neural network verification. In *Conference on Uncertainty in Artificial Intelligence*, pages 370–379. PMLR, 2020.

[15] 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.

[16] Kirill Bykov, Marina M-C Höhne, Adelaida Creosteanu, Klaus-Robert Müller, Frederick Klauschen, Shinichi Nakajima, and Marius Kloft. Explaining bayesian neural networks. 2021.

[17] Luca Cardelli, Marta Kwiatkowska, Luca Laurenti, Nicola Paoletti, Andrea Patane, and Matthew Wicker. Statistical guarantees for the robustness of bayesian neural networks. In *IJCAI*, 2019.

[18] Krishnendu Chatterjee, Martin Chmelík, and Mathieu Tracol. What is decidable about partially observable Markov decision processes with ω-regular objectives. *Journal of computer and system sciences*, 82(5):878–911, 2016.

[19] 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.

[20] Herman Chernoff et al. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The annals of mathematical statistics*, 23(4):493–507, 1952.

[21] Jen-Tzung Chien and Yuan-Chu Ku. Bayesian recurrent neural network for language modeling. *IEEE transactions on neural networks and learning systems*, 27(2):361–374, 2015.

[22] F. Chollet et al. Keras, 2015.

[23] Michael Copeland. What's the difference between artificial intelligence, machine learning, and deep learning? `https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/`, 2016. Accessed: 2018-10-22.

[24] Council of European Union. Council regulation (EU) no 2016/679, article 22, 2016. `https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN#d1e2803-1-1`.

[25] F. Croce et al. Sparse-rs: a versatile framework for query-efficient sparse black-box adversarial attacks. In *ECCV workshop on adversarial robustness in the real world*, 2020.

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

[27] Adnan Darwiche and Auguste Hirth. On the reasons behind decisions. In *ECAI 2020*, pages 712–720. IOS Press, 2020.

[28] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012.

[29] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[30] I. Dillig et al. Minimum satisfying assignments for SMT. In *CAV*, volume 7358 of *LNCS*, pages 394–409. Springer, 2012.

[31] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st annual conference on robot learning*, pages 1–16, 2017.

[32] Daniel D'souza, Zach Nussbaum, Chirag Agarwal, and Sara Hooker. A tale of two long tails. In *ICML workshop on uncertainty and robust- ness in deep learning*, 2021.

[33] 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.

[34] Kai Fabi. Layerwiserelevancepropagation. `https://github.com/KaiFabi/LayerwiseRelevancePropagation`, 2020.

[35] Linton C Freeman. *Elementary applied statistics: for students in behavioral science*. John Wiley & Sons, 1965.

[36] Peter A Frost. Proxy variables and specification bias. *The review of economics and Statistics*, pages 323–325, 1979.

[37] Yarin Gal. What my deep model doesn't know... `http://mlg.eng.cam.ac.uk/yarin/blog_3d801aa532c1ce.html`, 2015. Accessed: 2018-08-29.

[38] Yarin Gal. *Uncertainty in deep learning*. PhD thesis, University of Cambridge, 2016.

[39] Yarin Gal and Zoubin Ghahramani. Bayesian convolutional neural networks with Bernoulli approximate variational inference. In *4th International conference on learning representations (ICLR) workshop track*, 2016.

[40] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: representing model uncertainty in deep learning. In *International conference on machine learning*, pages 1050–1059, 2016.

[41] Yarin Gal, Jiri Hron, and Alex Kendall. Concrete dropout. In *Advances in neural information processing systems*, pages 3581–3590, 2017.

[42] Iosif Il'ich Gihman and Anatolij Vladimirovič Skorohod. *Controlled stochastic processes*. Springer Science & Business Media, 2012.

[43] A. Go et al. Twitter sentiment classification using distant supervision. *CS224N project report, Stanford*, 2009.

[44] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press Cambridge, 2016.

[45] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

[46] Alex Graves. Practical variational inference for neural networks. In *Advances in neural information processing systems*, pages 2348–2356, 2011.

[47] Brandon M Greenwell, Bradley C Boehmke, and Andrew J McCarthy. A simple and effective model-based variable importance measure. 2018.

[48] Wenbo Guo, Sui Huang, Yunzhe Tao, Xinyu Xing, and Lin Lin. Explaining deep learning models - a bayesian non-parametric approach. *Advances in neural information processing systems*, 2018:4514–4524, 2018.

[49] Ronan Hamon, Henrik Junklewitz, and Ignacio Sanchez. Robustness and explainability of artificial intelligence. *Publications office of the European Union*, 2020.

[50] GE Hinton and Drew van Camp. Keeping neural networks simple by minimising the description length of weights. 1993. In *Proceedings of COLT-93*, pages 5–13.

[51] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. 2012.

[52] Naozumi Hiranuma. Integratedgradients. `https://github.com/hiranumn/IntegratedGradients`, 2020.

[53] Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *The journal of machine learning research*, 14(1):1303–1347, 2013.

[54] 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*, number 1288, 2013.

[55] Po-Sen Huang, Robert Stanforth, Johannes Welbl, Chris Dyer, Dani Yogatama, Sven Gowal, Krishnamurthy Dvijotham, and Pushmeet Kohli. Achieving verified robustness to symbol substitutions via interval bound propagation. In

Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP), pages 4083–4093, 2019.

[56] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *International conference on computer aided verification*, pages 3–29. Springer, 2017.

[57] A. Ignatiev et al. On finding minimum satisfying assignments. In *CP*, volume 9892 of *LNCS*, pages 287–297. Springer, 2016.

[58] A. Ignatiev et al. Abduction-based explanations for machine learning models. In *AAAI*, pages 1511–1519. AAAI Press, 2019.

[59] Robin Jia, Aditi Raghunathan, Kerem Göksel, and Percy Liang. Certified robustness to adversarial word substitutions. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pages 4129–4142, 2019.

[60] Michael Kampffmeyer, Arnt-Borre Salberg, and Robert Jenssen. Semantic segmentation of small objects and modeling of uncertainty in urban remote sensing images using deep convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 1–9, 2016.

[61] G. Katz et al. The Marabou framework for verification and analysis of deep neural networks. In *CAV*, volume 11561 of *LNCS*, pages 443–452. Springer, 2019.

[62] Guy Katz. Nnet. `https://github.com/sisl/NNet`, 2015.

[63] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In

International conference on computer aided verification, pages 97–117. Springer, 2017.

[64] Guy Katz, Clark Barrett, David L. Dill, Kyle D. Julian, and Mykel J. Kochenderfer. Towards proving the adversarial robustness of deep neural networks. In *Workshop on formal verification of autonomous vehicles, international conference on integrated formal methods (iFM)*, 2017.

[65] A Kendall, V Badrinarayanan, and R Cipolla. Bayesian segnet: model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. In *British machine vision conference 2017, BMVC 2017*, 2017.

[66] Alex Kendall and Roberto Cipolla. Modelling uncertainty in deep learning for camera relocalization. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 4762–4769. IEEE, 2016.

[67] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in neural information processing systems*, pages 5574–5584, 2017.

[68] 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.

[69] Jinkyu Kim and John Canny. Interpretable learning for self-driving cars by visualizing causal attention. In *Proceedings of the IEEE international conference on computer vision*, pages 2942–2950, 2017.

[70] Siwon Kim, Jihun Yi, Eunji Kim, and Sungroh Yoon. Interpretation of nlp models through input marginalization. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pages 3154–3167, 2020.

[71] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[72] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[73] Yongchan Kwon, Joong-Ho Won, Beom Joon Kim, and Myunghee Cho Paik. Uncertainty quantification using bayesian neural networks in classification: application to ischemic stroke lesion segmentation. 2018.

[74] E La Malfa, A Zbrzezny, R Michelmore, N Paoletti, and M Kwiatkowska. On guaranteed optimal robust explanations for nlp models. International joint conferences on artificial intelligence, 2021.

[75] Emanuele La Malfa and Marta Kwiatkowska. The king is naked: on the notion of robustness for natural language processing. *AAAI*, 2022.

[76] Axel Legay, Benoît Delahaye, and Saddek Bensalem. Statistical model checking: an overview. In *International conference on runtime verification*, pages 122–135. Springer, 2010.

[77] Xuanqing Liu, Yao Li, Chongruo Wu, and Cho-Jui Hsieh. Adv-bnn: improved adversarial defense through robust bayesian neural network. 2018.

[78] Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward relu neural networks. 2017.

[79] Jingyue Lu and M Pawan Kumar. Neural network branching for neural network verification. *International Conference on Learning Representations*, 2019.

[80] Scott Lundberg. Shap. `https://github.com/slundberg/shap`, 2020.

[81] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st international conference on neural information processing systems*, pages 4768–4777, 2017.

[82] A. Maas et al. Learning word vectors for sentiment analysis. In *ACL-HLT*, pages 142–150, 2011.

[83] David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

[84] Ravi Mangal, Aditya V Nori, and Alessandro Orso. Robustness of neural networks: a probabilistic and practical approach. In *2019 IEEE/ACM 41st international conference on software engineering: new ideas and emerging results (ICSE-NIER)*, pages 93–96. IEEE, 2019.

[85] Rowan McAllister, Yarin Gal, Alex Kendall, Mark Van Der Wilk, Amar Shah, Roberto Cipolla, and Adrian Vivian Weller. Concrete problems for autonomous vehicle safety: advantages of bayesian deep learning. In *IJCAI*. International joint conferences on artificial intelligence, inc., 2017.

[86] Rhiannon Michelmore, Matthew Wicker, Luca Laurenti, Luca Cardelli, Yarin Gal, and Marta Kwiatkowska. Uncertainty quantification with statistical guarantees in end-to-end autonomous driving control. In *2020 IEEE international conference on robotics and automation (ICRA)*, pages 7344–7350. IEEE, 2020.

[87] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *1st international conference on learning representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, workshop track proceedings*, 2013.

[88] Joan C Miller and Clifford J Maloney. Systematic mistake analysis of digital computer programs. *Communications of the ACM*, 6(2):58–63, 1963.

[89] T. Miyato et al. Adversarial training methods for semi-supervised text classification. In *ICLR*, 2017.

[90] Christoph Molnar. *Interpretable machine learning*. Lulu.com, 2020.

[91] Pavel Myshkov and Simon Julier. Posterior distribution analysis for bayesian inference in neural networks. In *Workshop on Bayesian deep learning, NIPS*, 2016.

[92] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[93] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. *Advances in neural information processing systems*, 32:13991–14002, 2019.

[94] John Paisley, David M Blei, and Michael I Jordan. Variational bayesian inference with stochastic search. In *Proceedings of the 29th international conference on international conference on machine learning*, pages 1363–1370, 2012.

[95] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*, pages 372–387. IEEE, 2016.

[96] K. Patel and P. Bhattacharyya. Towards lower bounds on number of dimensions for word embeddings. In *IJCNLP*, pages 31–36, 2017.

[97] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[98] R. Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32(1):57–95, 1987.

[99] Hugh Reynolds. Simulation: the invisible gatekeeper. *Medium*, 2019.

[100] M. Ribeiro et al. "why should i trust you?" explaining the predictions of any classifier. In *SIGKDD*, pages 1135–1144, 2016.

[101] M. Ribeiro et al. Anchors: High-precision model-agnostic explanations. In *AAAI*, volume 18, pages 1527–1535, 2018.

[102] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[103] Rasmus Rothe, Radu Timofte, and Luc Van Gool. DEX: Deep EXpectation of apparent age from a single image. In *Proceedings of the IEEE international conference on computer vision Workshops*, pages 10–15, 2015.

[104] Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. In *IJCAI*, 2018.

[105] Wenjie Ruan, Min Wu, Youcheng Sun, Xiaowei Huang, Daniel Kroening, and Marta Kwiatkowska. Global robustness evaluation of deep neural networks with provable guarantees for the hamming distance. IJCAI, 2019.

[106] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[107] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[108] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review*, 5(1):3–55, 2001.

[109] Andy Shih, Arthur Choi, and Adnan Darwiche. A symbolic approach to explaining bayesian network classifiers. In *Proceedings of the 27th international joint conference on artificial intelligence*, pages 5103–5111, 2018.

[110] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *International Conference on machine learning*, pages 3145–3153. PMLR, 2017.

[111] Dylan Slack, Anna Hilgard, Sameer Singh, and Himabindu Lakkaraju. Reliable post hoc explanations: modeling uncertainty in explainability. *Advances in neural information processing systems*, 34, 2021.

[112] R. Socher et al. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, pages 1631–1642, 2013.

[113] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[114] Suraj Srinivas and R Venkatesh Babu. Generalized dropout. 2016.

[115] Ian J Stewart and Owen S Hamel. Bootstrapping of sample sizes for length-or age-composition data used in stock assessments. *Canadian journal of fisheries and aquatic sciences*, 71(4):581–588, 2014.

[116] Youcheng Sun, Hana Chockler, Xiaowei Huang, and Daniel Kroening. Explaining image classifiers using statistical fault localization. In *European conference on computer vision*, pages 391–406. Springer, 2020.

[117] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR, 2017.

[118] Tensorflow Team. Using the savedmodel format: Tensorflow core, Nov 2021.

[119] Ponkrshnan Thiagarajan, Pushkar Khairnar, and Susanta Ghosh. Explanation and use of uncertainty obtained by bayesian neural network classifiers for breast histopathology images. *IEEE transactions on medical imaging*, 2021.

[120] Mariya Toneva and Leila Wehbe. Interpreting and improving natural-language processing (in machines) with natural language-processing (in the brain). *Advances in neural information processing systems*, 32:14954–14964, 2019.

[121] Dustin Tran, Matthew D. Hoffman, Rif A. Saurous, Eugene Brevdo, Kevin Murphy, and David M. Blei. Deep probabilistic programming. In *International conference on learning representations*, 2017.

[122] Udacity. Udacity self-driving car simulator. `https://github.com/udacity/self-driving-car-sim`, 2018. Accessed: 2018-08-29.

[123] Eric Wallace, Jens Tuyls, Junlin Wang, Sanjay Subramanian, Matt Gardner, and Sameer Singh. Allennlp interpret: a framework for explaining predictions of nlp models. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th International joint conference on natural language processing (EMNLP-IJCNLP): system demonstrations*, pages 7–12, 2019.

[124] Fulton Wang and Cynthia Rudin. Falling rule lists. In *Artificial intelligence and statistics*, pages 1013–1022. PMLR, 2015.

[125] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. *Advances in neural information processing systems*, 31, 2018.

[126] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *27th {USENIX} security symposium ({USENIX} security 18)*, pages 1599–1614, 2018.

[127] Matthew Wicker. Deepbayes. `https://github.com/matthewwicker/deepbayes/tree/main/deepbayes`, 2021.

[128] Matthew Wicker, Luca Laurenti, Andrea Patane, Zhuotong Chen, Zheng Zhang, and Marta Kwiatkowska. Bayesian inference with certifiable adversarial robust-

ness. In *International Conference on Artificial Intelligence and Statistics*, pages 2431–2439. PMLR, 2021.

[129] 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.

[130] Matthew Wicker, Luca Laurenti, Andrea Patane, Nicola Paoletti, Alessandro Abate, and Marta Kwiatkowska. Certification of iterative predictions in bayesian neural networks. In *Conference on uncertainty in artificial intelligence*, pages 1713–1723, 2021.

[131] Yijun Xiao and William Yang Wang. Quantifying uncertainties in natural language processing tasks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 7322–7329, 2019.

[132] Danny Yadron and Dan Tynan. Tesla driver dies in first fatal crash while using autopilot mode. `https://www.theguardian.com/technology/2016/jun/30/tesla-autopilot-death-self-driving-car-elon-musk`, 2016.

[133] Xingyu Zhao, Wei Huang, Xiaowei Huang, Valentin Robu, and David Flynn. Baylime: Bayesian local interpretable model-agnostic explanations. In *Uncertainty in artificial intelligence*, pages 887–896. PMLR, 2021.