

LANCASTER UNIVERSITY

MASTERS THESIS

Adversarial Detection Through Bayesian Approximations In Deep Learning

Author:

Thomas PINDER

Supervisor:

Dr. Leandro MARCOLINO

*A thesis submitted in fulfilment of the requirements
for the degree of Data Science MSc*

in the

School Of Computing And Communications

September 13, 2018

Declaration of Authorship

Student Name: Thomas PINDER

Title: Adversarial Detection Through Bayesian Approximations In Deep Learning

Dissertation Supervisor: Dr. Leandro MARCOLINO

Programme: Data Science MSc

I certify that the material contained in this dissertation is my own work and does not contain unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation. Regarding the electronically submitted version of this submitted work, I consent to this being stored electronically and copied for assessment purposes, including the Department's use of plagiarism detection systems in order to check the integrity of assessed work. I agree to my dissertation being placed in the public domain, with my name explicitly included as the author of the work.

Date:

Signed:

LANCASTER UNIVERSITY

Abstract

School Of Computing And Communications

Data Science MSc

Adversarial Detection Through Bayesian Approximations In Deep Learning

by Thomas PINDER

Write abstract here...

Contents

Declaration of Authorship	i
Abstract	ii
List of Figures	vi
List of Abbreviations	viii
1 Introduction	1
1.1 Motivations	1
1.2 Aims	2
1.3 Report Overview	3
1.4 Contributions	4
2 Related Work	5
2.1 Deep Learning	5
2.2 Adversarial Attacks	5
2.3 Adversarial Defences	6
2.4 Bayesian Deep Learning	7
2.5 Reinforcement Learning	8
3 Background	10
3.1 Neural Networks and Deep Learning	10
3.1.1 An Overview of Neural Networks	10
3.1.2 Extending Neural Networks into Computer Vision	11
3.1.3 Dropout in Convolutional Neural Networks	13
3.2 Bayesian Machine Learning	14
3.2.1 Bayesian Inference	14
3.2.2 Markov Chain Monte-Carlo Through Metropolis Hastings	14
3.2.3 Gaussian Processes As A Bayesian Classifier	16
3.2.4 Formulating Optimisation Objectives For Variational Inference	19
3.2.5 Methods For Efficiently Carrying Out Variational Inference	21
3.3 Reinforcement Learning	23

3.3.1	The Markov Decision Process Framework	23
3.3.2	Using Q-Learning To Seek Out The Optimal Policy	23
3.3.3	Handling The Explore-Exploit Dilemma	24
4	Methodologies	25
4.1	Bayesian Neural Networks	25
4.1.1	Approximating Gaussian Processes Through Dropout	25
4.1.2	Sampling From the Network's Predictive Posterior	27
4.2	Deep Q-Networks	28
4.2.1	Markov Decision Process Modelling	28
4.2.2	Functional Approximation	28
4.2.3	Explore-Exploit Through Epsilon Annealing	30
4.3	Adversarial Examples	30
4.3.1	Fast Gradient Sign Method	30
5	Results	32
5.1	Experimental Setup	32
5.2	Computer Vision	32
5.2.1	Baselines	32
5.2.2	Off Manifold Experimentation	33
5.2.3	The Efficacy of Adversarial Attacks	35
5.2.4	Measuring Uncertainty in the Presence of Adversaries	36
5.2.5	Real Word Dataset	40
5.2.6	Adversaries In X-Rays	42
5.3	Training A Deep Q-Network With Bayesian Neural Network Functional Ap- proximation	45
6	Discussion	47
6.1	Discussion	47
6.2	Further Work	47
6.2.1	Conditionally Classifying Nearest Neighbouring Observations	47
6.2.2	Re-Calibrating A Model To Refine Uncertainty Estimates	48
6.2.3	Experimenting With Dropout Extensions	49
6.3	Limitations And Reflections	49
7	Conclusion	50
7.1	Contributions	50

	v
7.2 Closing Thoughts	50
A Appendices	i
A.1 Image Pre-processing	i
Bibliography	ii

List of Figures

1.1	An example adversarial attack to make a Macaw be classified as a Rhodesian Ridgeback with 99.7% confidence. Image generated using a trained Inception Model.	1
3.1	1-layer Neural Network	11
3.2	Depiction of a 2×2 kernel with stride length 1 moving across an image. . . .	12
3.3	Pooling of a 2×2 kernel across a 16-pixel image. The numbers here arbitrarily represent the underlying pixel values used by the network pre and post-pooling. . . .	12
3.4	Dropout ($p = 0.4$) within a single epoch of a network's training phase.	13
3.5	Prior-Posterior Example for Parameter ω	15
3.6	Identical chains from a Metropolis-Hastings algorithm, with varying variance. Burn-in set to 1000. True parameter value is 5.	16
3.7	Multivariate Gaussian Distribution	17
3.8	The relationship between an agent and its environment.	24
4.1	Example Bayesian neural network with weights modelled using a Gaussian distribution, $\sigma = 0.1$	27
4.2	Depiction of the CNN used in [32].	29
5.1	Example of eight digits from the MNIST dataset.	33
5.2	Confusion matrix heatmap of the performance of a CNN and BCNN on the MNIST dataset.	34
5.3	A comparison of Monte-Carlo uncertainty and softmax confidence as an image is rotated.	34
5.4	Example of FGSM being applied with $\epsilon = 0.5$	35
5.5	The effect of varying ϵ on a CNN and BCNN's accuracy trained and tested upon the MNIST dataset. 95% Confidence interval shown calculated for 200 experimental runs.	36

5.6	Metropolis-Hastings diagnostic plots for uncertainty inference from MNIST images. Adversarial μ and σ refers to the parameterisation of the posterior distribution representing uncertainty in the presence of adversaries. The same is true of original μ and σ for uncertainties from unperturbed images.	38
5.7	Posterior distributions for the uncertainty values originating from classification carried out on adversarially perturbed images and their unperturbed counterpart.	39
5.8	Empirical CDF of 100000 differences in posterior uncertainties.	40
5.9	Examples of a patient's X-Ray with, and without, the presence of pneumonia.	41
5.10	Confusion matrices showing the performance of a standard CNN and Bayesian CNN in classifying pneumonia from an X-Ray.	42
5.11	The effect of epsilon on a CNN and BCNN's accuracy in classifying pneumonia.	43
5.12	Metropolis-Hastings diagnostic plots for uncertainty inference from chest X-Rays searching for pneumonia. Adversarial μ and σ refers to the parameterisation of the posterior distribution representing uncertainty in the presence of adversaries. The same is true of original μ and σ for uncertainties from unperturbed images.	44
5.13	Posterior distributions for uncertainty estimates originating from BCNN predictions made on adversarially perturbed X-Ray images and their unperturbed counterpart.	45
5.14	Empirical CDF, F , for the probability a posterior sample of adversarial uncertainty is greater than an uncertainty estimate original from an unperturbed image.	46

List of Abbreviations

AI	Artificial Intelligence
BNN	Bayesian Neural Network
CAVI	Coordinate Ascent Variational Inference
CNN	Convolutional Neural Network
DQN	Deep Q-Network
ELBO	Evidence Lower Bound
GP	Gaussian Process
KL	Kullback Leibler
MCMC	Monte-Carlo Markov Chain
MDP	Markov Decision Process
M-H	Metropolis Hastings
NN	Neural Network
RL	Reinforcement Learning
SVI	Stochastic Variational Inference
VI	Variational Inference

Chapter 1

Introduction

1.1 Motivations

In recent years, deep learning has achieved state-of-the-art results in a range of domains, including, but not limited to, computer vision [24], speech recognition [11] and board games [49], [48]. However, there is a worrying fragility to these systems and the inability to understand the classifier's decision is deeply concerning.

These concerns are accentuated through adversarial examples; observations with perturbations unobservable to the human eye that can then *fool* the classifier (Figure 1.1). Adversarial examples were first established in computer vision, with the initial aim being to fool the classifier into misclassifying an image [53]. Further work leads to perturbations being created to not only fool a classifier but also coerce the classifier into misclassifying an observation as a specific class. The effects of attacks such as this are more worrying as the adversary is now able to influence the classifier, controlling the decisions it makes. Adversary attacks can be highly simplistic, with just a single pixel perturbation being enough to reduce a classifier's accuracy by up to 67% [51].

Perhaps this paragraph should be in related works

With deep learning becoming more widely used in domains such as autonomous vehicles and medical imaging, there becomes an increasing need to devise methods to evade adversarial examples. One example of a particularly sinister use of adversaries lies within

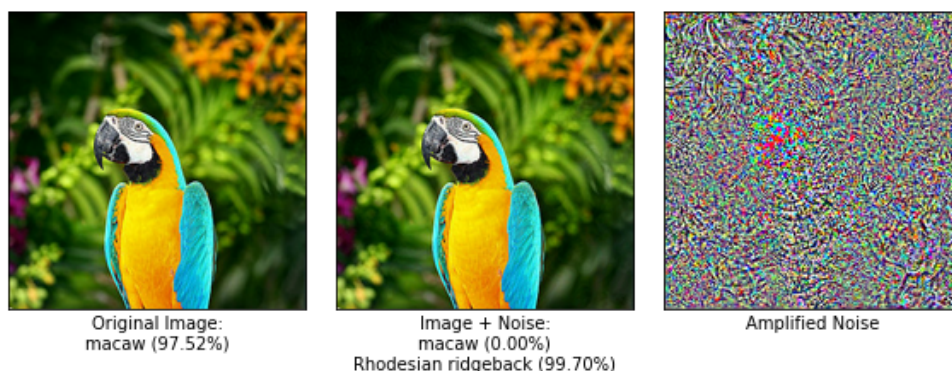


FIGURE 1.1: An example adversarial attack to make a Macaw be classified as a Rhodesian Ridgeback with 99.7% confidence. Image generated using a trained Inception Model.

the images received by the imaging sensors of an autonomous vehicle. Should the adversary be able to perturb the sensors, then the vehicle could be coerced into misclassifying pedestrian crossings as regular roads or red lights as green; the effects of which would be harrowing.

It is this notion that motivates this project, which focuses itself on devising new methods of defence against adversarial examples. We will apply our work to the domain of computer vision whereby we will study the effects of adversaries against the baseline MNIST dataset [28] and a real-world x-ray imaging dataset. Additionally, we will study the influence adversaries can have within deep reinforcement learning (RL), specifically the images used to functionally approximate future actions within Deep Q-Networks (DQNs).

In both the computer vision and RL domains, we will replace traditional deep learning model architectures with Bayesian neural networks as due to their converge to a Gaussian process (GP) we can infer measures of uncertainty from our model's predictions. Using these uncertainty values, we will study the viability of detecting adversaries based upon the respective magnitude of uncertainty. **Discuss popularity of adversaries in vision.**

1.2 Aims

Research into the effects of adversarial examples in RL is a new area, with the first paper being published in 2017 [21]. We try to further this work by investigating the following aims through the respective objectives:

1. **How effective are Bayesian neural network architectures over standard neural network architectures?**

The primary aim of this thesis will be to test the suitability of supplanting standard neural networks with Bayesian neural networks (BNNs). As BNNs allow for the derivation of accurate uncertainties from classifications, the hypothesis presented is that these uncertainties will enable us to identify the presence of an adversary.

Objectives:

- Establish a theoretical grounding for the creation of BNNs in a tractable way.
 - Implement, test and compare BNNs against NNs in a range of tasks (detailed further in the following objectives).
- ### 2. **Are BNNs able to identify adversaries in computer vision tasks while retaining state-of-the-art performance?**

While identifying an adversary is vital, it is equally imperative that the classifier is still able to perform optimally, or at least close to optimally in a given task.

Objectives:

- Evaluate the detection rate of a BNN against adversarial examples
- Test BNNs against NNs on the MNIST dataset [28] and pneumonia chest x-ray real-world dataset.

3. What effect do adversarial examples have on a DQN agent's performance?

Along with computer vision tasks, the performance of BNNs will be tested within a reinforcement learning setting. To do this, a DQN will be implemented, replicating the work of Mnih et al. Once implemented, the images produced within the environment will be perturbed to a range of degrees, with the agent's performance change being studied.

Objectives:

- Implement the work of Mnih et al. in [32].
- Invoke adversarial perturbations to the game frames and measure performance change.

4. Make work reproducible.

In order for this field to advance, reproducible work is crucial. To this extent, we will try to have modular, well-documented code that anyone can easily run.

Objectives:

- Host code with running instructions on GitHub ¹. https://github.com/thomaspinder/bayesian_uncertainty_adversaries Check Link Working
- Series of shell scripts containing exact configurations needed to reproduce experiments in Section 5. [https://github.com/thomaspinder/bayesian_uncertainty_adversaries/blob/master/scripts/README.md](#) Perhaps belongs in conclusion as a demonstration of making work reproducible.
- Document functions and classes using Sphinx ². https://thomaspinder.github.io/bayesian_uncertainty_adversaries/index.html Check Link Working

1.3 Report Overview

The work presented in this report will adopt the following structure. Chapter 2 will survey existing research in the fields of adversarial detection, Bayesian deep learning and deep reinforcement learning. In Chapter 3 we will establish fundamental ideas such as Bayesian

Check this description aligns with section content once the thesis is complete.

¹Code at https://github.com/thomaspinder/bayesian_uncertainty_adversaries

²Documentation at https://thomaspinder.github.io/bayesian_uncertainty_adversaries/index.html

inference, variational inference and neural networks. These foundations will underpin the methods described in Chapter 4 which will contain theoretical description of the methods employed. Chapter 5 presents the results of our research and highlights high-level findings. Chapter 6 will present a deeper examination of the results from Chapter 5 along with discussing the impact of this work in a generalised manner. In addition to this, Chapter 6 will also provide a discussion of this work's limitations and the opportunity for further work. Finally, Chapter 7 will be a generalisation of this work, summarising key details and tying our findings back to our initial aims and objectives.

1.4 Contributions

Within this work, several contributions are put forward, the first being the first (to the author's knowledge) empirical evaluation of Bayesian neural networks against standard neural networks in the presence of adversaries. Secondly, open source implementations of Bayesian neural networks using both the Keras [2] and PyTorch [42] libraries in Python is presented. Supplementary to this, an open source implementation of a Double Deep Q Network (DDQN) in Python is put forward. Within this DDQN, this work is the first to show the convergence of an agent's policy when the traditional neural network's are replaced by Bayesian networks.

Chapter 2

Related Work

2.1 Deep Learning

Placeholder

Perhaps a section covering current research in deep learning. May be covered by Bayesian DL with works of Gal, Graves.etc

2.2 Adversarial Attacks

Szegedy et al. first presented the notion of an adversarial attack through a method termed Box-Constrained L-BFGS. By optimising the prediction error of a classifier, adversarial examples are created. The hypothesis put forward is that neural networks have *blind spots* and adversarial examples represent “low-probability (high-dimensional) pockets in the manifold” that can exploit the network’s blind spots [53].

Remark 2.1: L-BFGS

L-BFGS is short for Limited Memory Broyden–Fletcher–Goldfarb–Shanno and is a popular optimisation strategy. The algorithm is termed a quasi-Newton strategy as, unlike Newtonian optimisation, an approximation replaces the variable’s Jacobian matrix [12].

After the vulnerability of neural networks was exposed by Szegedy et al.[53], a huge influx of research was conducted to create more effective attacks [9][51][34]. Goodfellow, Shlens, and Szegedy derived Fast Gradient Sign Method (FGSM), an algorithm that seeks to dynamically maximise a classifier’s error penalty [9].

Su, Vargas, and Kouichi were able to use differential evolutionary algorithms as an optimiser to perturb just a single pixel in an image. These single pixel attacks were tested across a range of network architectures and datasets, achieving accuracy reductions of up to 73% on the CIFAR-10 [23] image set. Perhaps more concerning was the algorithm’s ability to perform targeted attacks which not just fool the classifier, but explicitly, coerce the classifier incorrectly classifying an image as a targeted class [51].

Moosavi-Dezfooli, Fawzi, and Frossard extensively studied the minimum perturbation needed to fool a classifier. By linearising a classifier's loss function, DeepFool then searches for the smallest perturbation value η that will change the classifiers value [34].

In 2017, Huang et al. studied the effect of adversaries within a deep reinforcement learning scenario. Using the agent's trained policies, the game's images in the testing phase were perturbed using FGSM. Two types of an attack were studied, *white-box*, whereby the adversary knows the agent's classifier architecture, and *black-box*, where the adversary was blindly crafting attacks. White-box attacks were devastating, with an agent's performance decreasing by over 50% when just one-thousandth of its original value perturbed an image's value. More concerning was the effectiveness of black-box attacks which attained comparable efficacies, thus highlighting the transferability of adversarial attacks [21].

Lin et al. were able to extend the research of Huang et al. through *timed attacks* and *enchanted attacks* within Deep Q-Networks [32] setting. Timed attacks can inject an adversarial attack at a critical point in time, such as the frame that precedes striking the ball in Pong. Enchanting attacks cannot only fool the classifier but, in a similar fashion to Single Pixel attacks, tactfully craft an adversarial example to coerce the agent into making a specific action [30].

Concernedly, the efficacy of adversarial attacks extends beyond handwritten digit classification and Atari game domination. Rozsa et al. was able to adversarially perturb images of celebrities, altering aesthetic features such as the presence of lipstick [46]. Meanwhile, Eykholt et al. adversarially perturbed images of road signs to deliberately fool autonomous vehicles perception with some perturbations resulting in 100% success rates [4].

2.3 Adversarial Defences

The consequences of an adversarial attack are devastating, and unsurprisingly there have been many attempts to build classifiers that are robust to adversaries [16, 40, 45]. These efforts include applying the K-Nearest Neighbours (KNN) algorithm at each layer of a deep neural network. By identifying homogeneously labelled groups of observation points at each layer of the neural network during training, a testing observation can be classified based upon its nearest neighbour. Introducing KNN into a neural network's architecture did improve accuracies in the presence of adversaries on datasets such as MNIST by up to 27%; however, the accuracy decreased to 54.9%, a poor result for such a basic image set [40].

Ross and Doshi-Velez were able to achieve slightly higher accuracies on an MNIST image set perturbed by comparably strong FGSM attacks, retaining around 75% accuracy through a technique termed input gradient regularisation. Unfortunately, this method was only able

to achieve such accuracies when trained on a combination of perturbed and unperturbed training data which more than doubled the training time cost, meaning it will become impractical with larger datasets [45].

Xu, Evans, and Qi focussed on altering the modelled image of the dataset, attempting to hinder the efficacy of adversarial examples that fool classifiers. *Feature squeezing* induced a degree of robustness upon an image, a method that reduces the number of bits in a single pixel value, thus reducing the adversary's opportunity to create an attack. Additionally, the idea of smoothing the image by creating small *windows* over pixels in the image was presented. The pixel's values were then averaged across the window to create a blurred image. The result of both methods decreased the effectiveness of adversarial attacks by 20-30%; however, this came at the cost of a significantly reduced accuracy on unperturbed images [59].

2.4 Bayesian Deep Learning

Perhaps most concerning in all of the adversarial examples in Section 2.2 is the confidence of the classifiers in their incorrectly labelled observation. In many cases, the confidence calculated via a softmax layer in a neural network is between 86-99.9%, giving rise to a greater need for interpretability and uncertainty evaluation of black-box classifiers such as neural networks [51, 21].

The notion of using probabilities instead of point estimates within neural networks was first conceptualised by **DenkerTransformingDistributions** back in 1991. Within this work, the idea of using a Gaussian distribution to represent the network's class predictions was theorised [DenkerTransformingDistributions]. From here a flurry of research was conducted, with Bayesian methods being demonstrated in backpropagation calculations within feed-forward neural networks [31].

Computational complexity has always been a bottleneck within deep learning. Neal attempted to remedy this bottleneck by using Hamiltonian Monte-Carlo Markov Chains (MCMC) to sample from the network's posterior distribution [37]. However, Hamilton MCMC is a costly process. To alleviate this cost, work was later done by Hinton and Camp to use variational inference (VI) to derive an approximate, known distribution in place of the tractable posterior [17].

The idea of making Bayesian inferences in deep learning went somewhat dormant within the early 21st century due to its computational complexity. Welling, Bren, and Teh were able

to significantly improve the scalability and efficiency of MCMC sampling by applying random Gaussian noise to the results of Stochastic Gradient Descent [56]. At the same time, Graves introduced a method for making VI upon the network's posterior distribution that was suitable for a range of complex networks; something that had been an intractable problem previously [10].

Implementing Bayesian deep learning methods was made significantly more accessible due to work by Gal and Ghahramani who were able to show that by adding dropout [50] to every layer of a neural network, the network will converge to a Gaussian process (GP) approximation [5]. This approximation was significant as, after defining two hyperparameters, τ and l , a GP is highly robust to overfitting and allows for model uncertainty to accurately be quantified [43].

The idea of intertwining dropout layers into a neural network's architecture to make neural networks a Bayesian approximation was extended by Gal, Hron, and Kendall with the conception of *concrete dropout* [6]. Through concrete dropout, the dropout probability p is found using efficient gradient-based optimisation, thus removing the need for expensive grid-search to be done over p .

2.5 Reinforcement Learning

Reinforcement learning (RL) is a highly active area of research due to its potential to create autonomous agents; a key area of modern artificial intelligence (AI). Unfortunately, RL is not a complete area yet, and there are still a large number of existing challenges that are being tackled by researchers; one such area is that of function approximation.

A functional approximation allows the agent's act of choosing the next action to be reduced from a series of large Q-tables (which become intractable in complex problems), down to a model that learns a general solution over the entire environment. As the complexity of a reinforcement learning problem increases there becomes an increased need for accurate functional approximations to be made when considering what actions to make in a given state. Almost any classifier *can* do these functional approximations; however, much success has come from using neural networks as the classifier which gives rise to a field named Deep Reinforcement Learning (DRL) [32].

Tsitsiklis and Van Roy highlighted the hazards of using functional approximators when the classifier is non-linear, such as a neural network, by analytically showing that the desired solution will often not converge [54]. Mnih et al. was successfully able to mitigate these concerns through two psychologically inspired techniques: *experience replay* and two

independent, non-linear functional approximators, one for the current state and one for the next. These functional approximators took the form of 4-layer Convolutional Neural Networks (CNNs) that periodically bootstrapped previously experienced states and respective rewards from the experience replay queue and trained the neural network upon these samples to ensure previous experiences were not forgotten.

Along with functional approximation, another active area of reinforcement learning is the *explore-exploit dilemma*, that is, should an agent explore sub-optimal actions, or exploit the best-known action. This balancing act is critical for an agent to find the optimal policy as failure to explore enough will result in the environment not being thoroughly analysed. One of the most straightforward heuristics available to handle this problem is the ϵ -greedy strategy, whereby some value of $\epsilon \in [0, 1]$ is set, and when choosing an action, there is a *epsilon* chance that the agent will explore. Despite its simplicity, such strategies were employed in ground-breaking work [32]; however, Osband et al. argue that ϵ -greedy strategies lack a temporal consideration. By extending the work of [32], Osband et al. showed that a Bayesian-based exploration strategy termed Thompson sampling [58] allowed for the agent to reason about the long-term consequence of a given action. The results of replacing ϵ -greedy with Thompson sampling not only resulted in faster convergence rates but also higher total expected rewards [38].

Chapter 3

Background

3.1 Neural Networks and Deep Learning

3.1.1 An Overview of Neural Networks

Machine learning aims to learn and abstract functional mappings between inputs and outputs. Neural networks can do this through a stack of interconnected layers, with a series of calculations being applied to the previous layer's output values in order to learn feature representations of the underlying data.

Remark 3.1: Deep Learning

Despite being first used by Hinton, Osindero, and Teh[18], there is no universal definition for the definition for the term *deep learning*. Common consensus and the definition adopted in this work is that deep learning is a term used to describe neural networks comprised of many layers used to model large amounts of data.

Mathematically, an n -layered neural network applied to inputs $\mathbf{X} \in \mathbb{R}^d$ with labels $\mathbf{y} \in \mathbb{R}^d$ can be written a chain of functions $h = h^n \cdot h^{n-1} \cdot \dots \cdot h^1$. Within each layer h a set of weights \mathbf{W}^i being multiplied by the previous layer's output h^{i-1} . A non-linear function $\sigma(\mathbf{W}^i \cdot \sigma^{i-1})$, known as an activation function, is then applied to the resultant product. This provides a framework for learning a functional representation between \mathbf{X} and \mathbf{y} through

$$\hat{\mathbf{y}} = \sigma^n(\mathbf{W}^n \cdot \sigma^{n-1}(\mathbf{W}^{n-1} \sigma^{n-2}(\dots \sigma^1(\mathbf{W}^1 \mathbf{X}))))). \quad (3.1)$$

Upon inception, the neural network's weights are randomly initialised using draws from a specified distribution such as the Normal $\mathcal{N}(\mu, \sigma)$ or the uniform $\mathcal{U}(a, b)$. Through multiple passes through the network, a mapping between \mathbf{X} and \mathbf{y} is learned, by minimising the error at each pass. This error in prediction is measured using a *loss function* $\mathcal{J}(\omega, \mathbf{X}, \mathbf{y})$, with

the loss minimisation being done through a cost function, $\mathcal{C}(\cdot)$. An example of a 1-layer fully connected neural network is shown in Figure 3.1.

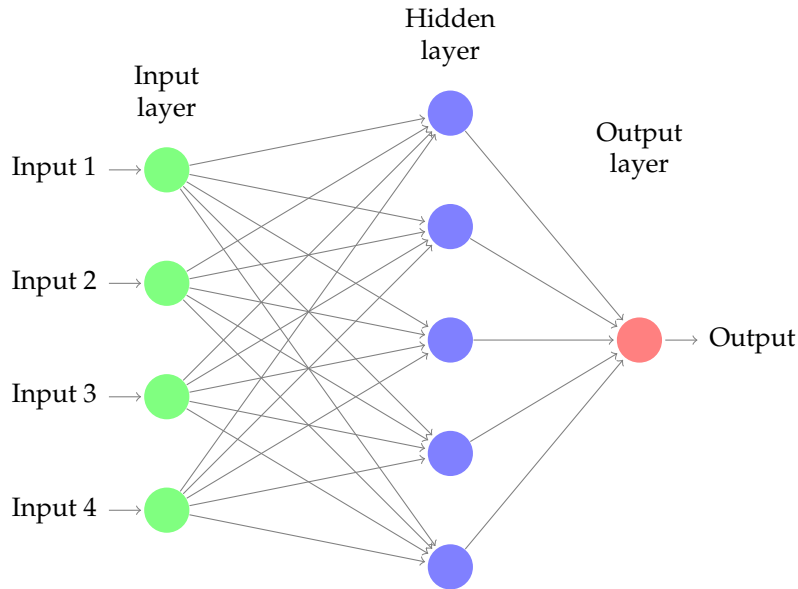


FIGURE 3.1: 1-layer Neural Network

3.1.2 Extending Neural Networks into Computer Vision

Classical NNs, as described in Section 3.1.1 work well on tasks where the inputs are numeric values organised into tabular form. However, significant improvements in the field of computer vision came as a result of the inception of CNNs [28].

A CNN differs from a classical neural network by the addition of a convolutional layer; a type of layer that attains a small sliding *window*, or kernel, that traverses across the input image (Figure 3.2). Kernels are typically square-shaped, with a width/height of three to eight pixels. The increment for which the kernel travels across the image is termed the *stride* and is usually one or two pixels. With each stride across the image, matrix multiplication is applied to the window's captured pixel values, and the resultant product flows into the network's next layer.

The addition of a sliding kernel in CNNs makes the architecture particularly pertinent to computer vision tasks as the kernel is often able to detect distinguishing image features such as edges and colour changes [60]. However, this inferential improvement comes at the cost of significantly more parameters within the model. While the computational complexity of training any neural network is heavily dependent upon the dataset and the number of epochs required until convergence, we can measure the additional complexity of convolutional layers as

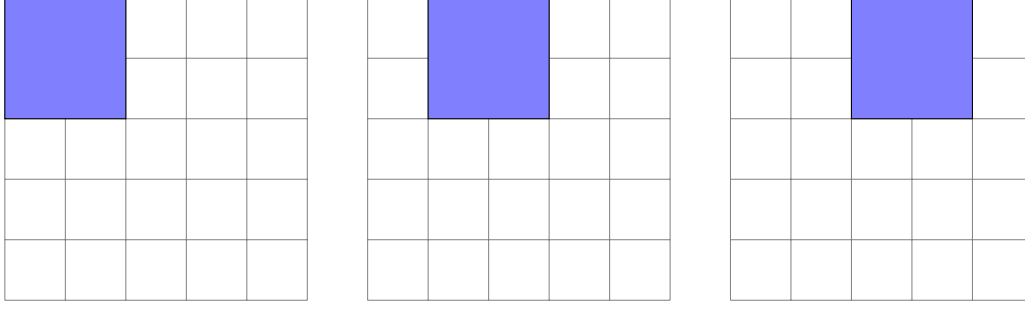


FIGURE 3.2: Depiction of a 2×2 kernel with stride length 1 moving across an image.

$$\mathcal{O}\left(\sum_{l=1}^d n_{l-1} \cdot s_l^2 \cdot n_l \cdot m_l\right), \quad (3.2)$$

where l is the convolutional layer's index and d is the number of convolutional layers. Additionally, n_l is the width of the l^{th} layer, s_l is the width/height of the l^{th} layer's kernel and the dimension of the l^{th} layer's outputted feature map [15].

By Equation 3.2, even the smallest convolutional layer is going to increase the computational complexity of training such a network significantly. Fortunately, through a technique termed *pooling* (Figure 3.3) the model's complexity is reduced by down-sampling the image's spatial representation. Consequently, this leads to fewer parameters in the model, thus a lighter computational burden. There are many different pooling algorithms, with max-pooling being the most popular. A max pooling layer contains a kernel similar to that of a convolution that strides across the image in a non-overlapping manner. With each stride, the maximum value from the kernel is selected and then used to represent the entire kernel in proceeding windows [3].

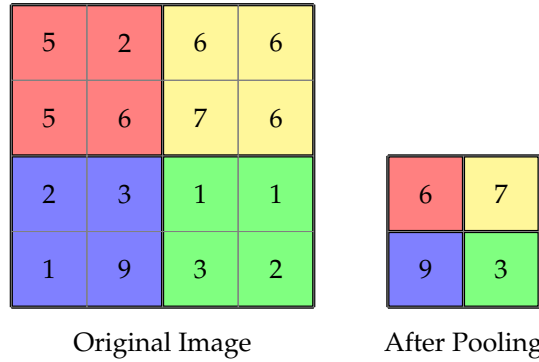


FIGURE 3.3: Pooling of a 2×2 kernel across a 16-pixel image. The numbers here arbitrarily represent the underlying pixel values used by the network pre and post-pooling.

3.1.3 Dropout in Convolutional Neural Networks

A fundamental concept in the BCNN that will be developed in proceeding sections is the notion of dropout, a technique that was initially envisioned to allow networks to generalise better and prevent overfitting [50]. Dropout is applied to the set of nodes within a single layer of a neural network during training. By specifying a parameter $p \in (0, 1)$, through dropout, each node has a $1 - p$ probability of being set to zero for that specific epoch (Figure 3.4). More formally, we can think of each node being multiplied by a random draw from the Bernoulli distribution, thus making the output of the i^{th} layer

$$\begin{aligned} \mathbf{b} &\sim \text{Bernoulli}(p), \text{ such that } \mathbf{b} = \{b_1, b_2, \dots, b_n\} \\ \mathbf{l}^i &= \mathbf{b}^i \cdot (\sigma^i(\mathbf{W}^i \cdot \mathbf{X}^{i-1})). \end{aligned} \quad (3.3)$$

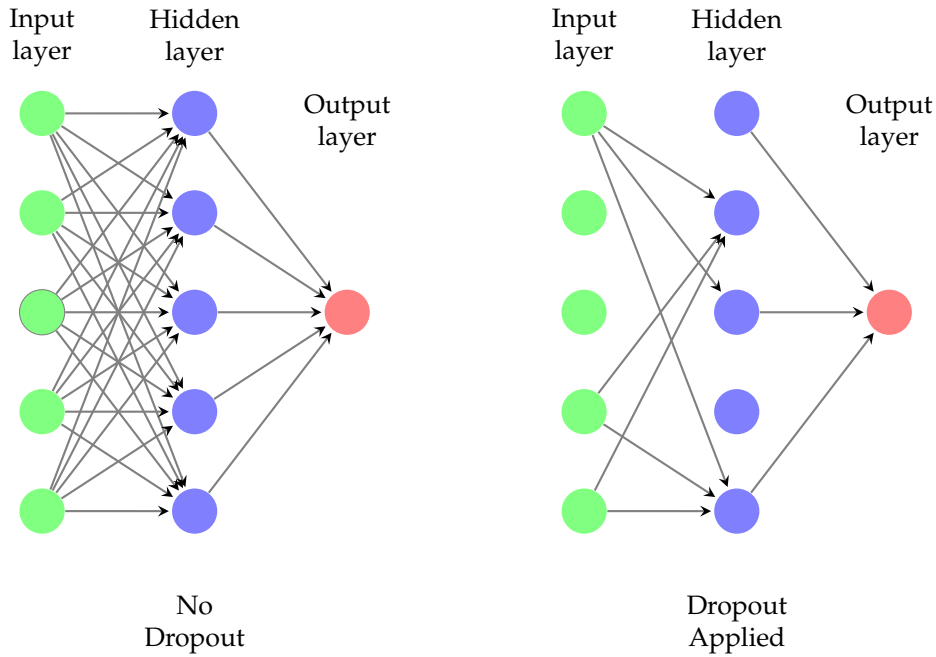


FIGURE 3.4: Dropout ($p = 0.4$) within a single epoch of a network's training phase.

This random "turning on and off" of nodes breaks dependencies between nodes as they learn their value throughout the training process, consequently leading to significantly less overfitting in the final model. An intuitive way to think of dropout is as a rough *bagging* procedure, similar to that adopted by random forests. At each new epoch, a different set of nodes will be set to zero, leading to a different *sub-network* to be trained. Upon training completion, each node is scaled by the dropout probability factor p and kept on during testing. The significant divergence between bagging in random forests and that of dropout in

neural networks is that each new tree is independent of the previous within a random forest. In neural networks with dropout, this is not the case, as each new sub-network inherits the set of weights learned by the previous sub-network [8].

3.2 Bayesian Machine Learning

3.2.1 Bayesian Inference

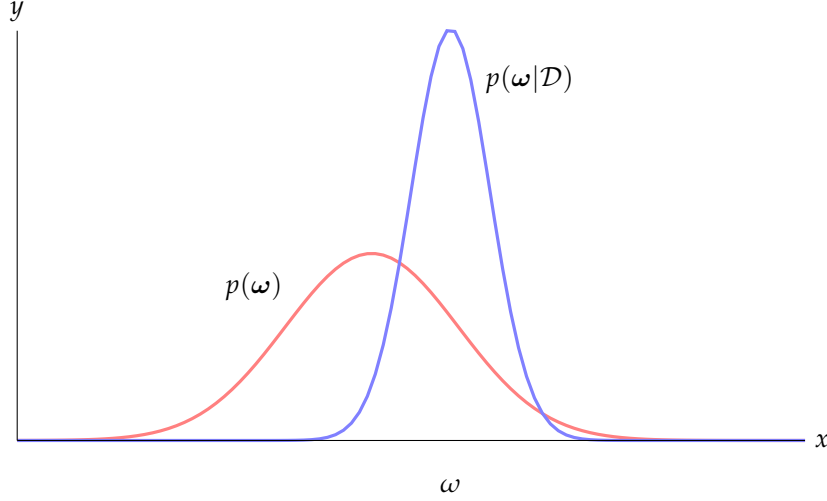
By modelling classification problems using Bayesian inference, we can leverage probabilities to quantify uncertainties in an estimate or prediction. Bayesian inference aims to learn a *posterior distribution* which probabilistically describes how confident we are about a set of parameters ω , given a set of data \mathcal{D} (Figure 3.5). *Prior beliefs* allow us to capture any initial beliefs about the values of ω before any data has been observed. This has the advantage of allowing us, as the modeller, to infuse any domain knowledge into the model. Another essential component in Bayesian inference is the *likelihood function* $p(\mathcal{D}|\omega)$ which is used to describe how probable values of the data are, dependent upon certain values of ω . With this knowledge and Baye's Theorem, we can then represent our posterior distribution as

$$p(\omega|\mathcal{D}) = \frac{p(\mathcal{D}|\omega)p(\omega)}{p(\mathcal{D})} = \frac{p(\mathcal{D}|\omega)p(\omega)}{\int p(\mathcal{D}|\omega)p(\omega)d\omega}. \quad (3.4)$$

The denominator in Equation 3.4 is known as the *marginal likelihood* and it is this function that normalises the posterior distribution by ensuring that it sums to 1. Calculating $p(\mathcal{D})$ is often an intractable problem as it is often not possible to express the integral in closed form. Being unable to compute the posterior distribution analytically does not prohibit Bayesian inference though as Monte-Carlo Markov Chain (MCMC) algorithms such as Metropolis-Hastings or Hamiltonian MCMC can be used to approximate the full posterior distribution.

3.2.2 Markov Chain Monte-Carlo Through Metropolis Hastings

As touched upon above, one solution to sample from a distribution whereby direct sampling is not possible is the Metropolis-Hastings (M-H) algorithm, part of a larger family of algorithms known as MCMC algorithms [14]. Provided we know some $f(x)$ that is proportional to the true density $P(x)$ then we are free to take these samples using M-H. The aim of any MCMC algorithms is to explore the entire parameter space, such that the time in any location of the parameter space is proportional to the height of the distribution for the given

FIGURE 3.5: Prior-Posterior Example for Parameter ω .

parameters. In the setting of Bayesian inference, we typically use the M-H algorithm to sample from the posterior when we do not know it in closed form, however, for simplicity, we will explain the M-H algorithm generally for any unknown distribution.

Given a distribution $f(x)$ for which we would like to sample from, we first define a starting point for our Markov chain x^0 . Secondly, we define a *proposal distribution* $Q(x^*|x^0)$ from which we sample a *candidate value* value x^* . Sampling from the proposal distribution should be easy and we typically use a Gaussian, $\mathcal{N}(x^*|x^0, \sigma^2)$. With a starting value and a candidate value for our Markov chain, we must now decide whether we should accept the candidate value into the chain or keep the current value. We make this decision by calculating an acceptance probability

$$\alpha(x^0, x^*) = \min \left(\frac{f(x^*) \cdot Q(x^0|x^*)}{f(x^0) \cdot Q(x^*|x^0)}, 1 \right). \quad (3.5)$$

Should the candidate value move the chain to an area of higher density, then the ratio in Equation 3.5 will be greater than one, so we will accept the candidate value into the chain. This process repeats itself for finite N terms, with a new candidate value being calculated each time, conditional upon the previous value in the Markov chain. We have no guaranteed convergence with MCMC algorithms, however we have an asymptotic result that $x(j) \rightarrow X$ such that $X \sim f(x)$. That is, as the length of our chain grows, our samples converge to a distribution X , such that X is distributed by our target density $f(x)$ [44].

Some practical issues to consider when running a M-H are that the initial samples are highly sensitive to the proposed starting point. As such, it is usually advisable to apply a *burn-in*, whereby an initial number of samples are discarded. Typically between 100-1000

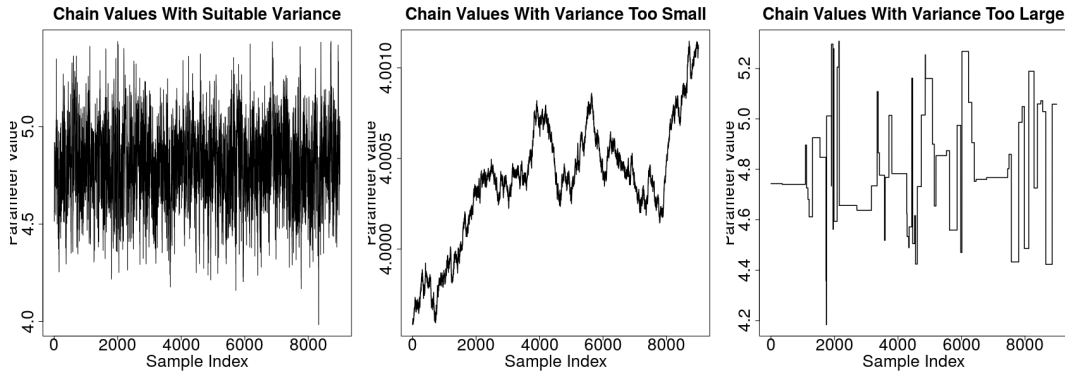


FIGURE 3.6: Identical chains from a Metropolis-Hastings algorithm, with varying variance. Burn-in set to 1000. True parameter value is 5.

samples is appropriate for a burn-in, dependent on the chain's trace plot.

Finally, care should be taken when setting the variance parameter of our proposal distribution. Setting the variance too high will make the chain very *sticky*, meaning the acceptance rate will be very low, consequently meaning that the chain will not explore the parameter space efficiently. Conversely, setting the variance too low will rectify the low acceptance rate, however, the change in candidate value will be so small that each movement of the chain will be incredibly small. Examples of such behaviour, along with a chain sampled with appropriate variance can be seen in Figure 3.6, note range of values on the y-axis. In the chain with small variance, the chain is exploring a minute portion of the parameter space, while the chain of large variance is incredibly sticky and not exploring the space sufficiently¹ [7].

3.2.3 Gaussian Processes As A Bayesian Classifier

Gaussian processes (GPs) start using standard Bayesian inference by defining a prior, however we are now defining a prior over functions, $f(\cdot)$, which we can think of as infinitely long vectors. Fortunately, we need not specify this prior over an infinite vector. Moreover, we are able to just define our prior over a finite set of values $\{x_1, x_2, \dots, x_n\}$. Through this prior and observed data, we are then able to derive a posterior distribution that allows for inference on unobserved data to be carried out.

In a regression setting, GPs differ from ordinary least squares and other frequentists approaches as they are non-parametric, or to be more specific, they are parameterised by infinitely many parameters. To lay the foundation of GPs, we will first define their setup in a two-dimensional \mathbb{R}^2 setting. The extension to \mathbb{R}^n is a case of increasing the matrices' dimensionality.

¹Code used to generate these chains was adapted from <https://theoreticalecology.wordpress.com/2010/09/17/metropolis-hastings-mcmc-in-r/>

For the bivariate case of two variables, we assume (as with any GP) that our variables are multi-variate normal distributed

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \sigma_{1,1} & \sigma_{1,2} \\ \sigma_{2,1} & \sigma_{2,2} \end{bmatrix} \right) \\ = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}),$$

where $\boldsymbol{\mu}$ is the vector of means and $\boldsymbol{\Sigma}$ the variance-covariance matrix. For the example in Figure 3.7,

$$\boldsymbol{\mu} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

as shown by the modality at $(0,0)$ and the perfect circle around the mode showing the two variables are independently normal with one another.

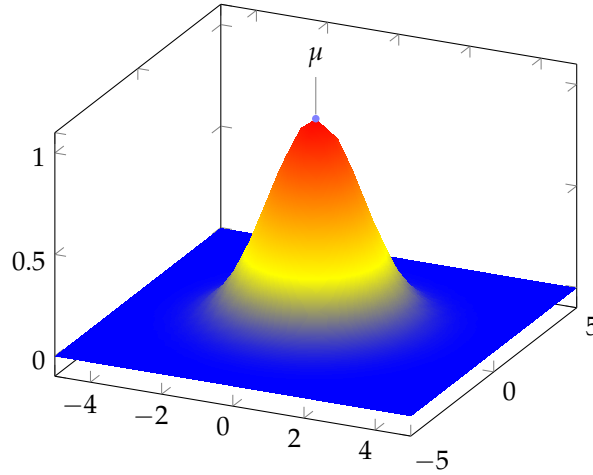


FIGURE 3.7: Multivariate Gaussian Distribution

The next step in understanding GPs is the notion of a kernel function. A kernel function allows the variance-covariance matrix to be derived from the observed data. Many different kernel functions exist, with certain kernels specialising in specific tasks. Some examples include the histogram kernel for image recognition, Jaccard kernel for binary classification and tree kernels for natural language processing [43]. While many different kernels exist, they all must satisfy the following two properties:

- Symmetric such that $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \mathcal{K}(\mathbf{x}', \mathbf{x})$.
- Positive semi-definite such that $\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \mathcal{K}(\mathbf{x}, \mathbf{x}') \cdot f(\mathbf{x})f(\mathbf{x}')d\mathbf{x}d\mathbf{x}' \geq 0$.

To demonstrate a kernel, we will demonstrate the commonly used squared exponential kernel

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp -\frac{1}{2l^2}(\mathbf{x} - \mathbf{x}')^T(\mathbf{x} - \mathbf{x}').$$

In the squared exponential kernel, l controls the width of the kernel, while σ controls the kernel's spread. Typically these parameters are found through grid search or another optimisation tool. With the notion of kernels and multivariate Gaussians defined, we can now show how posterior inference can be made into a GP.

A GP's posterior can be thought of as the joint probability of our observations. Using classical machine learning terminology, we have a set of observations for which we observed, f ; we can think of these as our training data. Similarly, we have a set of unobserved observations f^* , constituting our test data for which we would like to use our GP to infer about. Our posterior can then be defined as

$$\begin{bmatrix} f \\ f^* \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \mu \\ \mu^* \end{bmatrix}, \begin{bmatrix} \mathcal{K} & \mathcal{K}_* \\ \mathcal{K}_*^T & \mathcal{K}_{*,*} \end{bmatrix} \right) \quad (3.6)$$

In Equation 3.6, \mathcal{K} is the kernel function of our observed data, while \mathcal{K}_* calculates the similarity between f and f^* . Finally, $\mathcal{K}_{*,*}$ calculated the kernel function of f^* . As with all Bayesian inference, we would now like to sample from our posterior distribution. Murphy is able to show that we can get to $p(f^*|f)$ and sample from f^* using

$$f_i^* \sim \mathcal{N}(\mu_*, \Sigma_*). \quad (3.7)$$

In a multivariate setting this is not as simple as sampling from the standard Gaussian where we can simply compute

$$\begin{aligned} x_i &\sim \mathcal{N}(\mu, \sigma^2) \\ &= \mu + \sigma \mathcal{N}(0, 1), \end{aligned}$$

as shown by [35].

While tractable in two-dimensions, square-rooting Σ_* in the standard fashion becomes intractable in larger dimensions so we must use Cholesky decomposition that states $\Sigma = \mathbf{L}\mathbf{L}^T$ where \mathbf{L} is the lower triangular matrix of Σ . Using this, we are now able to sample from an multivariate normal distribution through

$$f_* \sim \mu + \mathbf{L}\mathcal{N}(0, \mathbf{I}), \quad (3.8)$$

where \mathbf{I} is the identity matrix.

Fitting a GP for classification problems is a harder task as in the above setting we are modelling regression which allows us to assume a Gaussian prior and likelihood. In a classification task we must assume a softmax likelihood, thus making the posterior no longer conjugate to the prior.

Should GPs for classification be expanded.

3.2.4 Formulating Optimisation Objectives For Variational Inference

MCMC algorithms work very well and are often incredibly powerful at approximating the posterior; however, they can often be very slow to converge. Within this work, we will instead use a family of algorithms known as *Variational Inference* (VI) to approximate the posterior distribution. VI replaces the posterior distribution from Equation 3.4 with an easily inferable, parametric distribution $q(\omega)$. Finding $q(\omega)$ turns the job of finding the posterior distribution into an optimisation problem, rather than sampling. While less theoretically established than MCMC, VI is significantly more efficient, particularly when the geometry of the posterior is complex or the dataset and model structure are large, as is the case in all deep learning models.

To be slightly more verbose, VI and MCMC both solve the problem of finding a posterior distribution; however, the approach of both algorithms in solving this problem is different. VI seeks out *variational parameters* θ that optimise a distribution to best approximate

a hard-to-find conditional distribution, such as the posterior. An example of such an approximating distribution is a mixture of K Gaussians, with the variational parameters: mean $\mu = \{\mu_1, \mu_2, \dots, \mu_K\}$ and variance $\sigma = \{\sigma_1, \sigma_2, \dots, \sigma_K\}$.

To carry out this optimisation, we define our family of distributions \mathcal{L} over our sought after parameters ω . We then try different *candidates* $q(\omega) \in \mathcal{L}$ with the objective of finding $q^*(\omega)$, the candidate that minimises the distance between our approximating distribution and the true posterior.

We calculate the distance between the approximating distribution and our posterior using the Kullback-Leibler (KL) divergence $\text{KL}(q_\theta(\omega) || p(\omega|\mathcal{D}))$ ² [27] such that³

$$\begin{aligned} \text{KL}(q(\omega) || p(\omega|\mathbf{X}, \mathbf{Y})) &= \int \log q(\omega) [\log q(\omega) - \log p(\omega|\mathbf{X}, \mathbf{Y})] d\omega \\ &= \int q(\omega) \log \frac{q(\omega)}{p(\omega|\mathbf{X}, \mathbf{Y})} d\omega. \end{aligned} \quad (3.9)$$

Unfortunately, this leads the optimisation in full circle as the reason for optimising q_θ was due to the intractability of $\log p(\omega|\mathcal{D})$ which is now present in our optimisation function. We can circumvent this dependency on $\log p(\omega|\mathcal{D})$ through the *Evidence Lower Bound* (ELBO), an objective that suitably approximates the KL-Divergence, plus a constant term. We can define the ELBO objective as

$$\text{ELBO}(q) = \int q(\omega) \log p(\mathbf{Y}|\mathbf{X}, \omega) d\omega - \text{KL}(q(\omega) || \log p(\omega)). \quad (3.10)$$

To make the intuition behind the ELBO objective somewhat simpler, we can re-write Equation 3.10 as follows

$$\text{ELBO}(q) = \int \log p(\mathcal{D}|\omega) - \text{KL}(q_\theta(\omega) || p(\omega)). \quad (3.11)$$

We can now see in Equation 3.11 how the optimisation's two terms will work together throughout the optimisation procedure. The first term is the data's log-likelihood function, meaning that there will be an emphasis on ensuring the modalities are well fitted. The second term is the KL-divergence between our new variational distribution and the prior on ω . Unlike maximising the first term, this will encourage probability mass to spread,

²|| is notation here to denote the distance between two densities.

³For discrete distributions we define KL-Divergences as $\mathbb{E}[\log q_\theta(\omega)] - \mathbb{E}[\log p(\omega, \mathcal{D})] + \log p(\mathcal{D})$

juxtaposing the modality concentration of the log-likelihood [Bishop2006PatternLearning, Blei2016VariationalStatisticians, 7].

3.2.5 Methods For Efficiently Carrying Out Variational Inference

With an optimisation objective defined (Equation 3.10), the question now posed is how can we best optimise this function. Coordinate Ascent Variational Inference (CAVI) is one approach whereby we incrementally climb the natural gradient (further details in Remark 3.2) of the ELBO function. In small datasets, this is a tractable process, as computing the natural gradient only requires the Fisher information matrix which is straight-forward, given that the variational distribution was purposefully selected based upon the fact it is parameterisable [Bishop2006PatternLearning]. CAVI is similar in application to Gibbs sampling (a special case of the MH-algorithm), as the algorithm cycles through the parameter set, updating each one based upon the most recent values of the remaining parameters.

Remark 3.2: Natural Gradients

We can conceptualise Natural Gradients in an optimisation setting by first thinking about standard gradient ascent which takes the form

$$\omega^{i+1} = \omega^i + \alpha \cdot \nabla \mathcal{L}(\omega). \quad (3.12)$$

Whereby ω^i is our parameter set at the optimisation's i^{th} iteration and α is our learning rate that determines how much we should adjust our parameter values at each iteration. Finally, $\nabla \mathcal{L}(\theta)$ is the derivative of our loss function applied to our current parameter set.

Natural gradients extend this idea as the gradients are computed with respect to the manifold of our KL-Divergence loss function. Computing gradients across the KL manifold is powerful, as in the sense of VI, a natural gradient quantifies the geometry of our parameter space in such a way that any move of a fixed distance will result in the same magnitudinal change of KL-Divergence, independent of the distance we move in. Natural gradients not only ensure a faster optimisation but often a more accurate one as well. To calculate the gradients of the KL manifold, we compute the Fisher Information matrix (matrix of second derivatives) of our KL-Divergence function. We use the Fisher information matrix to update our original parameter. We can mathematically express the optimisation of natural gradients as

$$\omega^{i+1} = \omega^i + \alpha \cdot \nabla_N(\omega^i), \quad (3.13)$$

such that $\nabla_N(\omega^i)$ is our natural gradient [Blei2016VariationalStatisticians, 20, 41].

As the complexity of the dataset and underlying model grow, CAVI based approaches fail to scale efficiently, and we must, therefore, seek new methods [Blei2016VariationalStatisticians]. Stochastic Variational Inference (SVI) enables us to run VI on massive datasets as, unlike CAVI approaches, they do not require computations to be made across the entire data set. Moreover, SVI optimises the variational parameters using numerous noisy samples, acting as if the underlying was made up entirely of this noisy sample. These are then used to compute directly and update global parameter estimates, thus bypassing the inefficient local optimisations that are made within CAVI approaches. The general outline of SVI computation is as follows:

1. Draw a random sample from the parameter space.
2. Calculate the local variational parameters.
3. Use the newly calculate local variational parameter to compute the global parameters.
4. Update the global parameters.

This process is repeated indefinitely [20].

3.3 Reinforcement Learning

3.3.1 The Markov Decision Process Framework

A Markov Decision Process (MDP) is a way of describing a stochastic environment whereby the environment is fully observable. The MDP framework hinges itself upon the Markov assumption, that is, *"The future is independent of the past, given the present"*. A MDP can be written as a 5-tuple containing the following components:

- Set of states \mathcal{S} of the environment.
- Set of legal actions \mathcal{A}
- Probability transition matrix $P(s'|s, a)$
- Reward function \mathcal{R}
- Discount factor γ

Given a set of states and actions, we can further define a policy π to be a mapping between state and actions that an agent follows. When solving an MDP, the primary aim is often to find a policy that maximises the agent's total expected reward $\sum_{t=0}^{\infty} \gamma^t \mathcal{R}_{a_t}(s, s')$; this is termed an optimal policy $V(\pi)$ [BertsekasDynamicControl].

3.3.2 Using Q-Learning To Seek Out The Optimal Policy

A practical and widely used approach to solving an MDP is through Q-learning; a model-free algorithm that means $P(s'|s, a)$ does not need to be known. Within Q-learning, an agent holds *Q-tables* for each state, with a single Q-table being a mapping of actions to Q-values for that given state. Through repeated simulation, the agent updates the Q-tables through

$$Q(s, a) = (1 - \alpha)Q(s, a) + \alpha(R(s) + \gamma Q(s', a')), \quad (3.14)$$

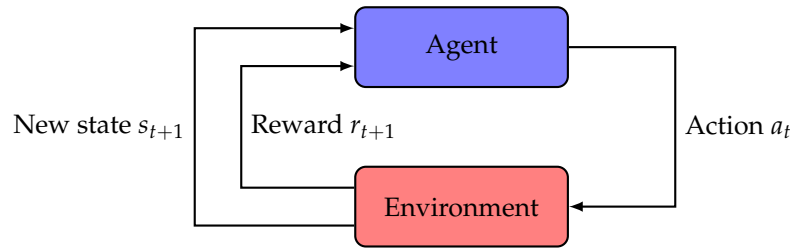


FIGURE 3.8: The relationship between an agent and its environment.

where alpha is a custom parameter specifying the learning rate [47]. In environments whereby the state space is large, holding a Q-table for every state becomes computationally intractable. A functional approximator, such as least squares regression, neural networks are used to approximate the Q-values, thus replacing the tables and consequently solving the issue of an intractably large state-space.

3.3.3 Handling The Explore-Exploit Dilemma

When carrying out Q-learning, a higher Q-value represents a larger total expected reward, based upon the agent's experience so far. However, we must ensure that the agent explores the environment adequately; otherwise, the agent may diverge and repeatedly play a sub-optimal policy just because it was fruitful early on. We handle such scenarios by posing the problem as a multi-armed bandit problem.

Discuss UCB and Thompson?

The simplest heuristic that converges to the optimal policy, and the heuristic used in this project as per Mnih et al., is the epsilon-greedy strategy. Selecting $\epsilon \in [0, 1]$, a random value u^* is sampled from $\mathcal{U}(0, 1)$ and if $u^* < \epsilon$ the agent explores and plays a sub-optimal action. Larger values of ϵ result in an increased amount of exploration by the agent. Efficient exploration is controlled by annealing ϵ through time, meaning that the proportion of exploration and exploitation is shifted to favour exploitation as the agent becomes more knowledgeable of the environment [52].

Chapter 4

Methodologies

4.1 Bayesian Neural Networks

4.1.1 Approximating Gaussian Processes Through Dropout

Neural networks are incredibly complex models, often containing hundreds of thousands of parameters. Such a vast quantity of parameters makes the training of the most simple neural networks (non-Bayesian, for now) a highly challenging job and can take several days on state-of-the-art hardware. When we extend this training process to a Bayesian framework, we are no longer learning fixed point values over each node within the network, but instead trying to learn a posterior distribution over the network's weights.

Once a posterior has been established, we are now able to make inferences about new samples x^* . To make these inferences, we must use the *predictive distribution* which we can derive from Equation 3.4, noting the approximate distribution $q_\theta(\omega)$ replaces the posterior distribution

$$\begin{aligned} p(y^*|x^*, x, \omega) &= \int p(y^*|x^*, \omega) p(\omega|X, Y) d\omega \\ &\approx \int p(y^*|x^*, \omega) q(\omega) d\omega \end{aligned} \quad (4.1)$$

However, it can be seen that the final term in this function $q_\theta(\omega)$ is a variational distribution, replacing the posterior distribution from Equation 4.1. Therefore we must turn to the VI techniques presented in Section 3.2.4 to approximate the posterior distributions. Recall the optimisation objective from Equation 3.11 which can be broken out into two parts, the first being the expected log-likelihood of the data and the second the KL-Divergence between our approximating distribution and the parameter's prior distribution.

Gal and Ghahramani were able to show that by adding dropout layers into any neural network architecture, then each layer of the network converged to a GP [5]. The proof behind this derivation is very involved, spanning several pages, so we direct the reader to [Gal2016DropoutAppendix] for a full derivation. At a high level, this is not entirely novel

as prior work by Williams and Neal showed that an infinitely wide, single layer neural network converged to a GP when distributions were placed over the network's weights matrix [36, 57]. Finitely wide neural networks rely on VI to approximate the weights posteriors, a task that becomes intractable as the size of the network grows [36, 20, 39].

With an aim of finding the predictive posterior in Equation 4.1, we must first find $q_\theta(\omega)$. To do that, we must find the set of variational parameters θ that best approximate $q_\theta(\omega)$. To do this, we use the VI objective from Equation 3.10 that defines the ELBO function. In large networks, the first term of Equation 3.10 is intractable, so we use Monte-Carlo integration to solve it.

$$\int q_\theta(\omega) \log p(\mathbf{Y}|\mathbf{X}, \omega) d\omega \approx \sum_{n=1}^n \int q_\theta(\omega) \log(y_n|x_n, \omega) d\omega. \quad (4.2)$$

By sampling $\hat{\omega} \sim q(\omega)$, we can derive an unbiased estimator for the Monte-Carlo integration result, making our objective now

$$\hat{\mathcal{L}}(\theta) = \log p(\mathbf{Y}|\mathbf{X}, \hat{\omega}) - \text{KL}(q_\theta(\omega)||p(\omega)). \quad (4.3)$$

As $\log p(\mathbf{Y}|\mathbf{X}, \hat{\omega})$ is an unbiased estimator, we can guarantee that by optimising $\hat{\mathcal{L}}$ for θ , we will arrive at the same result as if we had optimised \mathcal{L} . We carry out the SVI using the idea of optimising based upon the gradients of numerous noisy samples (detailed in Section 3.2.5). To do this, we first sample $\hat{\omega} \sim q_\theta(\omega)$ and then minimise $\hat{\mathcal{L}}(\theta)$.

Delving deeper into this, our variational parameter for θ is the M_i , such that $M_i = \frac{1}{K} \sum_{k=1}^K \mathbf{W}_k$; the mean of the i^{th} layer's weights. Therefore our variational distribution can be calculated as

$$q_{M_i} = M_i \cdot B_{i,j},$$

where $B_{i,j}$ is a matrix of Bernoulli random variables for the i^{th} layer. Finally, to draw a random sample for SVI we just need to sample $\mathbf{W}_i \sim q_{M_i}(\mathbf{W}_i)$. To conduct SVI, we draw multiple samples from $q_{M_i}(\mathbf{W}_i)$, optimising the global parameters each time, thus performing SVI over the neural networks layers.

In its current state, this result is seemingly abstract; however, it can be seen that the steps taken to compute SVI over the network are identical to the procedure performed when each

Discuss mean field approximation of the layer's weights?

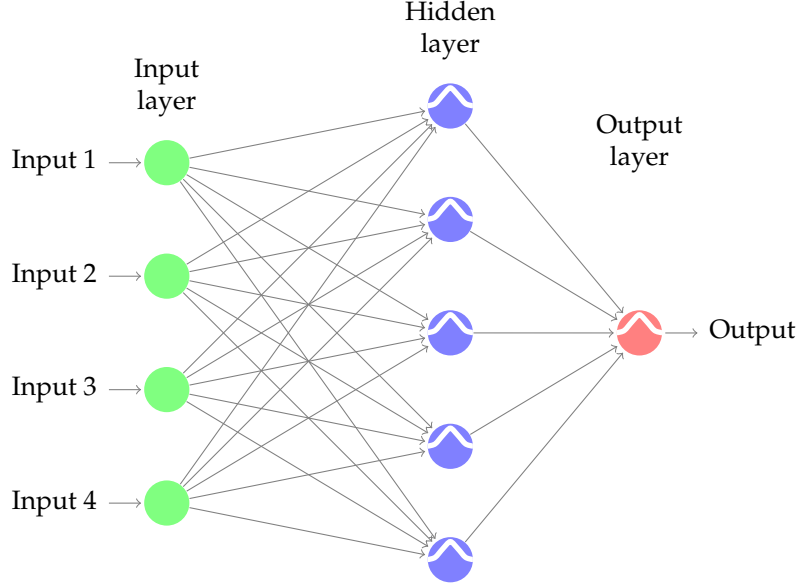


FIGURE 4.1: Example Bayesian neural network with weights modelled using a Gaussian distribution, $\sigma = 0.1$.

layer in the network has dropout applied, and we pass batches of observations through and update our network's weight after each batch. Further, the process of taking draws from the network's predictive distribution can be done by leaving dropout on a test time, a step that deviates from dropout's normal usage. By passing a test observation through the network numerous times, we are making many stochastic draws from the predictive posterior. As a result of this, we can harness the natural benefits of Bayesian inference such as uncertainty measures.

4.1.2 Sampling From the Network's Predictive Posterior

From here, the first and second order moments of expectation and variance are derived through multiple stochastic passes through Equation 4.1. The first moment is the mean value of these stochastic passes.

$$\mathbb{E}(y^*) = \frac{1}{T} \sum_{t=1}^T \hat{y}_t^* \quad (4.4)$$

Where T is the number of stochastic passes and \hat{y}^* is our prediction. Similarly, the predictive variance is the sample variance of stochastic passes.

Remark 4.1: Computational Runtime

Computationally, a Bayesian neural network will take the same time as a standard neural network to train as the only difference is the addition of dropout layers. At test time, T stochastic passes must be made through the network, meaning that test runtime will be T -times slower, however, this procedure could easily be distributed across multiple cores to reduce the test time of Bayesian neural networks.

4.2 Deep Q-Networks

A large component of this project was the replication of the Deep Q-Network (DQN) created by Mnih et al. 2015 [32]; an enhanced implementation of the original DQN paper [33]. The major divergence in the implementation here is that the CNN used by Mnih et al. is replaced by a BCNN, with the intention of utilising uncertainties within real-time game play. Within this section, we will only present the enhanced paper (Mnih et al. 2015) and to emphasise clarity, the algorithm's core components will be explained individually.

4.2.1 Markov Decision Process Modelling

Atari games are dynamic, meaning that un-controlled components of the game (example: the ball in Breakout) have attributes such as velocity and trajectory that cannot be represented by a single frame of the game. A single state is comprised of four frames, in order to capture this information. Therefore, a single frame is captured every fourth frame of the game; this reduces the number of frames held by a quarter, meaning a single state encapsulates information from the previous 16 frames. The number of actions available to the agent was game dependent. The game primarily used in our experiments was Pong which has six legal actions: up, down, static, up and fire the ball, down and fire, and static and fire.

Similar to actions, the value of rewards ranges from game to game within the Atari 2600 suite [1]. Rewards are clipped, meaning that any positive reward is re-valued to 1, similarly any negative reward to -1. Clipping rewards enables the same agent to train across multiple games where the rewards range may vary [13].

4.2.2 Functional Approximation

DQN's fundamentally originate around the Q-Learning algorithm presented in Section 3.3, however, a Q-table is supplanted by a BCNN based functional approximator. The agent

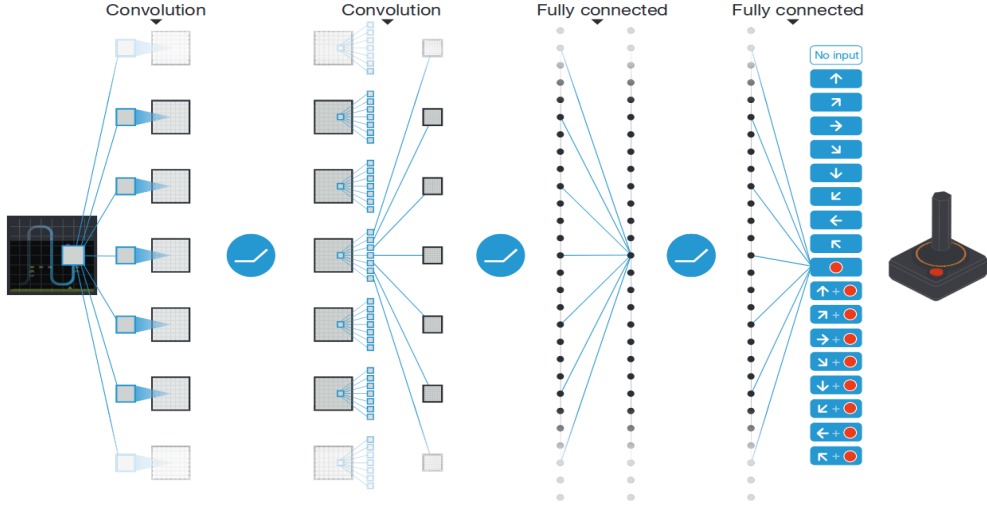


FIGURE 4.2: Depiction of the CNN used in [32].

starts with no knowledge of the game, other than the set of legal actions available. Further to this, the BCNN's weights are randomly initialised resulting in an architecture similar to that shown in Figure 4.2. The BCNN is composed of four hidden layers with the first three being convolutional layers and the final hidden layer being a fully connected layer. In line with the Bayesian approximations derived by Gal and Ghahramani, a dropout layer with $p = 0.1$ is included after each convolution and before the final softmax layer. The four hidden layers are of widths 32, 64, 64 and 512, each using rectified linear units as an activation function. The model's final visible layer is another fully connected layer, with the number of weights being equal to the game's number of legal actions.

The agent learns from accumulating experiences into a replay experience [29] which stores 5-tuples of the form $\langle s, a, r, t, s' \rangle$ where t is an indicator of the state's terminality status. Tuples are stored until the replay experience holds one million tuples, at which point a first-in-first-out attitude is adopted. A sample of 32 experiences are periodically bootstrapped from the replay experience and passed through the BCNN with the network's weights updated. Sampling experiences in this way means that the estimated Q-values are less likely to diverge which would result in the agent becoming stuck in a negative feedback loop. However, this approach is slightly limited, as every state receives an equal probability of being selected, meaning that critical states are not prioritised.

One major divergence from the 2013 DQN is that the 2015 algorithm uses slightly larger CNNs and, more significantly, two CNNs. Originally the values of $Q(s, a)$ and $Q(s', a')$ in Equation 3.14 were approximated by the same CNN. This led to a dependency being present between $Q(s, a)$ and $Q(s', a')$ and consequently resulted in policies sometimes diverging. Training two CNNs would be timely, however, so to avoid this additional expense,

a single CNN is trained for $Q(s, a)$ and then copied and made the CNN for $Q(s', a')$ every 10000 frames.

4.2.3 Explore-Exploit Through Epsilon Annealing

In the agent's infancy, it has a minimal idea over which actions are best. Tied to this acute knowledge, Atari games are very sparse, with a large number of possible state-action pairs. To ensure the agent explores enough of the game, an ϵ -greedy strategy is employed, with ϵ 's value initialised to 1. Over the first one million games, this value linearly anneals to 0.1. Annealing encourages the agent to exploit actions increasingly, as the agent's Q-values begin to converge.

4.3 Adversarial Examples

We define our image space as $\mathcal{X} = [0, 1]^{H \times W \times C}$ to contain n images $x \in \mathcal{X}$. An adversarial example x^* is defined to be an image x that is perturbed by values η in a way such that our classifier $h(\cdot)$ makes the classification $h(x) \neq h(x + \eta)$.

4.3.1 Fast Gradient Sign Method

Fast Gradient Sign Method (FGSM) will be used to compute adversarial examples due to its efficacy and efficiency. Adversarial examples are derived by first calculating perturbation values, η , which are then summed with the original image's pixel values. Values of η are calculated by

$$\begin{aligned} \eta &= \epsilon \text{sign}(\Delta_x J(\theta, x, y)) \\ \implies x^* &= \eta + x \end{aligned} \tag{4.5}$$

The value of ϵ is the l_∞ bound of the perturbed image, or more explicitly, the maximum pixel difference value between the original and perturbed images. The larger the l_∞ norm value is, the more obvious the adversarial example will be to the human eye.

Perhaps add an example here.

Decomposing Equation 4.5, we are first calculating the cost function, $J(\cdot)$, of our classifier with parameters, θ . The cost function is evaluated while the classifier is predicting a given image, x , with corresponding label y . Once calculated, the sign function of the cost function's derivative is then used to determine whether $\eta_{h,w,c}$ will be added or subtracted to the original images pixel values [9].

The mechanics behind FGSM lie within the optimisation strategies employed by classifiers that attempt to reduce the cost function to a global minimum. By computing the cost function's derivative during the construction of an adversarial example, FGSM can perturb pixel values in a way such that the optimiser moves away from the minima. A gradient-based approach such as FGSM is significantly more effective than a perturbation method such as applying a Gaussian noise as an image is being targeted at an individual pixel level.

Chapter 5

Results

5.1 Experimental Setup

Specific network architectures and hyperparameter settings vary from experiment and consequently will be acknowledged before presenting the results. However, for all experiments the variable values in Table 5.1 were constant across all experiments.

Parameter	Value
Stochastic Passes (\mathcal{T})	100

TABLE 5.1: Constant experimental variable values

5.2 Computer Vision

5.2.1 Baselines

The MNIST dataset (Figure 5.1) is comprised of images of handwritten digits between 0 and 9 is a benchmark test within computer vision and accuracies of 97% upwards is not uncommon. The entire dataset contains 60000 images of which the same 50000 images were used for training both a CNN and BCNN. For the CNN, a standard LeNet architecture [28] was used which first consists of two convolutional layers which are followed by two linear layers. Each layer uses the Rectified Linear Units (ReLU) activation function and a 2D max-pooling layer. Between the final and penultimate layer there exists a single dropout layer with probability 0.5 with this layer being disabled at test time. The only divergence from this architecture in the BCNN is that every layer contains a dropout layer of probability 0.5 and this probability is kept on at test time.

Each network was trained for a total of twenty epochs using stochastic gradient descent with a learning rate initialised to 0.01 and momentum of 0.5. Finally, to compute the loss between the model's predictions and the ground truth during training, the negative log-likelihood loss across log-softmax probabilities as it allows for the cross-entropy function to scaled out to multi-class problems [Bishop2006PatternLearning].

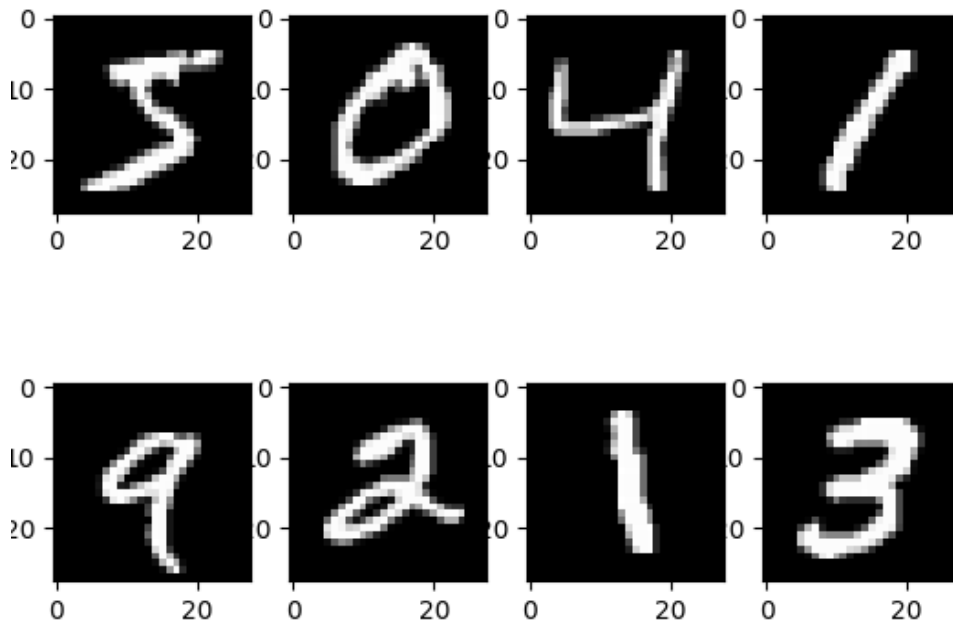


FIGURE 5.1: Example of eight digits from the MNIST dataset.

Once both networks had finished training, the baseline accuracy of each was found using the entire 10000 test set. Across the entire test set, the CNN and BCNN reported accuracies of 98.34% and 98.53% respectively. We can visualise these results at an individual class level through the confusion matrices presented in Figure 5.2

Generally the network's perform very well in classifying the MNIST digits. A high accuracy is to be expected however, given the simplicity of the task. At an individual class level, there is no one class that the network's struggle with, however, digits whereby there's a strong visual similarity to another digit (nine and eight, for example, when the 9's tail curls around too much it can look very much like an eight) do seem to have slightly lower accuracies than highly distinct integers such as zero.

5.2.2 Off Manifold Experimentation

Softmax is often the metric employed to represent confidence in a neural network's prediction. For people accustomed to this style of inference, it may seem unnecessary to now make multiple stochastic samples from a network's posterior in order to extract confidence. The problem with softmax certainties are that they will very confidently extrapolated predictions originating from observations that lie outside the datasets range. Extrapolation in isolation is not too problematic, however, when these extrapolations are accompanied by high confidence then problems begin to arise as we should significantly less confidence about predictions made on observations that lie outside the underlying datasets range.

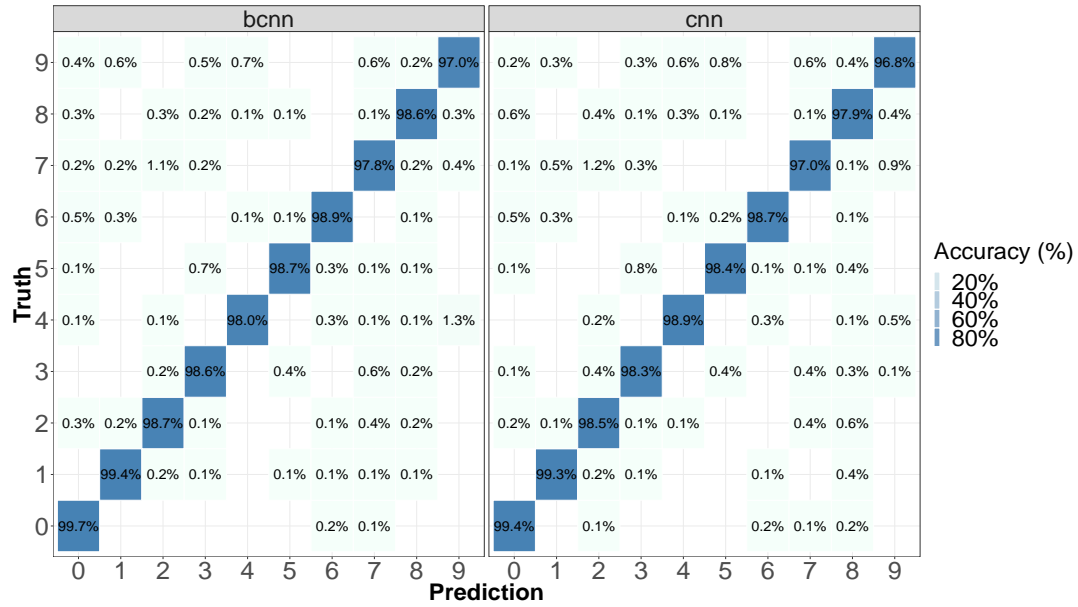


FIGURE 5.2: Confusion matrix heatmap of the performance of a CNN and BCNN on the MNIST dataset.

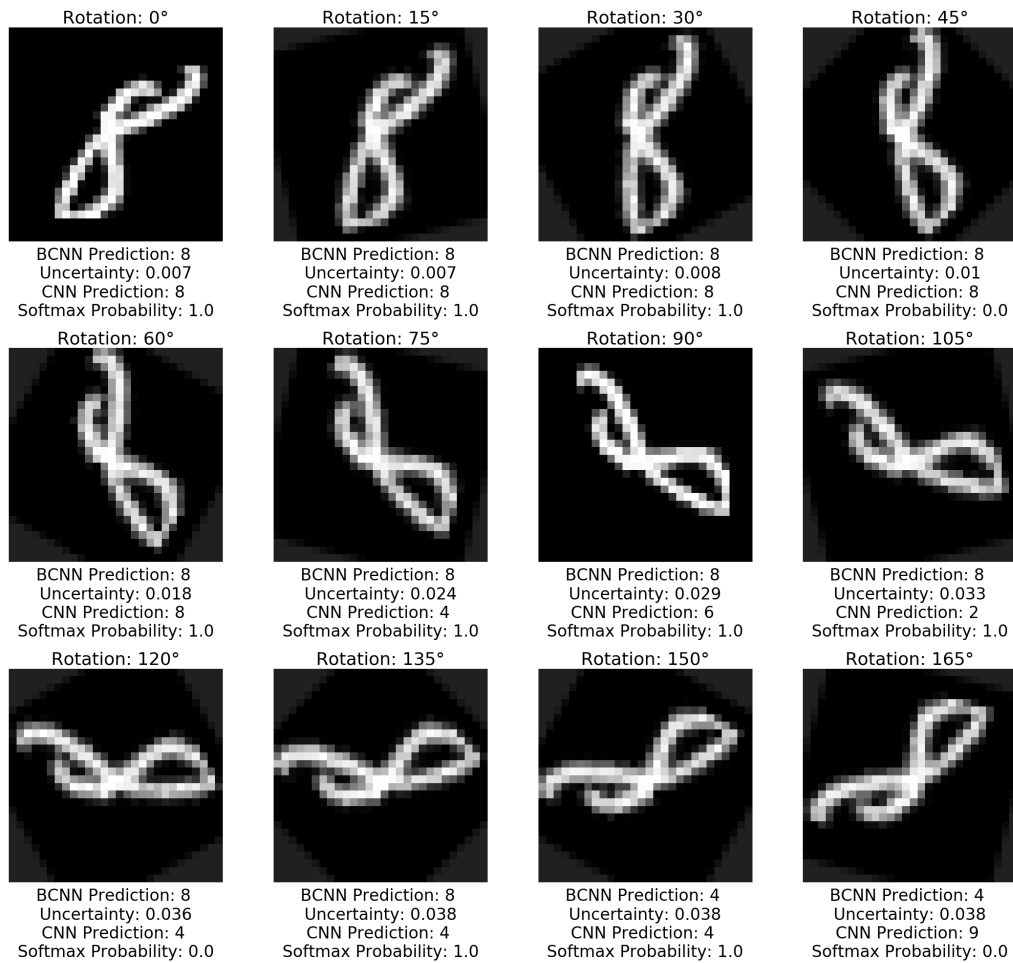


FIGURE 5.3: A comparison of Monte-Carlo uncertainty and softmax confidence as an image is rotated.

Using our trained CNN and BCNN we can demonstrate the improvement in uncertainty/confidence estimates that arise from using BCNNs. Using a single image, we rotate the image through 15 degree increments and predict the images class at each rotational increment. These predictions are made by both the CNN and BCNN, with confidence/uncertainty estimates being extracted through softmax and Monte-Carlo sampling respectively. The network should be less certain about a classification made upon an image rotated 90 degrees, as there will be little to no examples of this type of image in the underlying training data.

As can be seen in Figure 5.3, the Monte-Carlo based uncertainty estimate is slowly increasing as the image approaches 90 degrees. This is due to the fact that the figure 8 will look the same at 0 and 180 degrees. Conversely, the softmax based confidence is consistently high, even when the image's predicted class changes from 8 to 4, as seen in the image rotated 75 degrees.

5.2.3 The Efficacy of Adversarial Attacks

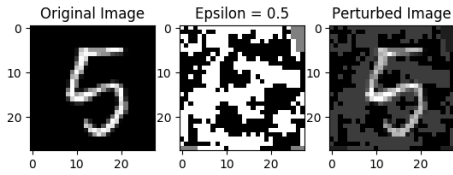


FIGURE 5.4: Example of FGSM being applied with $\epsilon = 0.5$.

With common baselines for both a CNN and BCNN established, both network's resilience to adversarial attacks is now measured. Adversarial examples are crafted using the FGSM method from Section 4.3 with values of ϵ being tested between 0.01 and 1. An example can be seen where FGSM is applied to an MNIST image of the number

5, with $\epsilon = 0.5$ in Figure 5.4. For each value of epsilon, the network being attacked has been pre-trained in the absence of any adversarial examples and all of the testing data was perturbed.

It is evident from Figure 5.5 that both networks are immune to epsilon at lower values before the effects of an increasing epsilon quickly become devastating for the classifier's accuracy. We can get some initial intuition into the effects of epsilon by fitting a simple linear regression (SLR) model to fit accuracy against epsilon, using a known intercept of 0.94 (the average of the two classifier's accuracies). This makes the model ¹

¹Note a slight deviation from common notation as typically ϵ is used to model the model's error. As ϵ here denotes the l_∞ bound of FGSM, τ is used for the model's error.

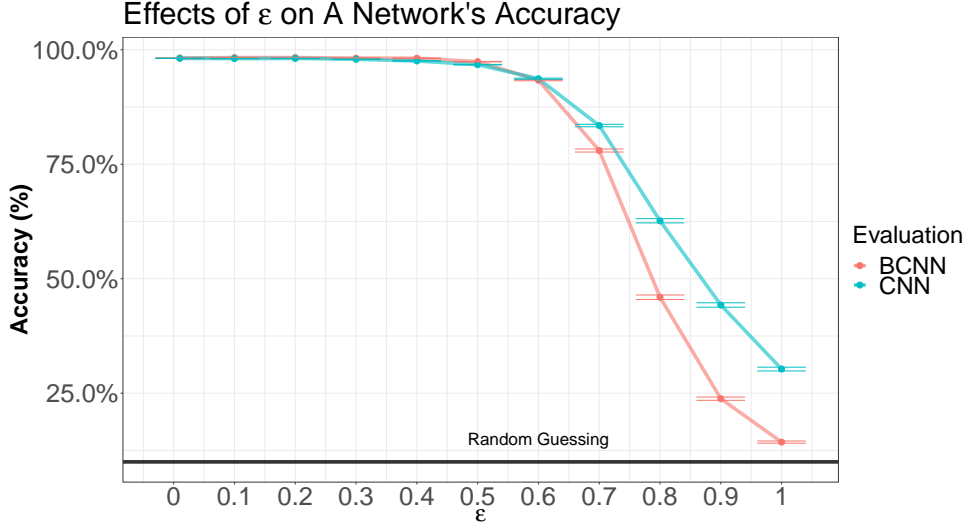


FIGURE 5.5: The effect of varying ϵ on a CNN and BCNN's accuracy trained and tested upon the MNIST dataset. 95% Confidence interval shown calculated for 200 experimental runs.

$$\hat{y} = 0.94 - 0.26\epsilon + \tau. \quad (5.1)$$

The model's -0.26 on ϵ , indicates that a unit ($+0.1$) increase in ϵ will cause a 2.5 drop in the either network's predictive power. It should be noted that this linear regressor is not a perfect fit, with a heavy skew being present in residuals, however, it does allow us to gauge at a high level the effect of epsilon on the predictive accuracy of a classifier.

5.2.4 Measuring Uncertainty in the Presence of Adversaries

To investigate the plausibility of using the uncertainty estimates of a BCNN to determine the presence of an adversary where it should be larger. To test this hypothesis, a BCNN first classifies unperturbed images, FGSM is then applied to the image and the BCNN then makes a new calculation on the adversarial example. For each of the two predictions the uncertainty estimate of the BCNN is recorded. Using the uncertainty estimates corresponding to each of these two stratifications, we can now conduct a hypothesis test on the mean uncertainty within each group. We can formally structure our hypothesis as

$$H_0: \mu_{original} = \mu_{adversary}$$

$$H_1: \mu_{original} < \mu_{adversary}.$$

We can conduct this t-test of means using

$$t = \frac{\mu_{adv} - \mu_{ori}}{\frac{\sigma_{adv}}{n} \frac{\sigma_{ori}}{n}}. \quad (5.2)$$

Within the current example, the p-value resulting from our t-test is to the magnitude of 10^{-16} , indicating that we have extremely high evidence to reject H_0 in favour of H_1 . This gives a high level result that we can use the uncertainties resulting from a BCNN prediction to detect the presence of an adversary.

In keeping with the Bayesian approach of this work so far, we can also make inferences into this proposed increased uncertainty when adversaries are present through Bayesian inference. To do this, we base our methods upon the work of Kruschke, whereby we can test for a difference in means by deriving the posterior distribution for the adversary based uncertainties and the original uncertainties [25]. Using the Bayesian inference presented in Section 3.2.1, we can derive the posterior distributions for both stratifications using Baye's theorem

$$p(y|x) \propto p(x|y) * p(y).$$

Through this posterior distribution we will be able to learn the distribution of uncertainty means, given an image. The advantage we get by conducting this analysis in terms of posterior distributions instead of fixed-point p-values is that we can quantitatively measure the probability that the adversarial based uncertainty is n units larger than the original image based uncertainty. To do this though we must first specify our likelihood, which in this case is a normal distribution.

We now assign prior distributions to the parameters of our likelihood; μ and σ . Kruschke recommend using very vague priors so as not to overly influence the final posterior. As per these recommendations, a normal distribution is selected as the prior over μ , with mean equal to the combined sample's mean and standard deviation equal to the sample's combined standard deviation, multiplied by 1000 to ensure a the probability density isn't too concentrated in a single area. For the likelihood's σ parameter, a exponential prior distribution is used with $\lambda = 0.1$ to ensure a non-negligible amount of probability density occurs in the distribution's tail.

With a likelihood defined and a prior distribution specified over each of the likelihood's parameters we can proceed to fit our posterior distribution. Multiplying our likelihood and priors together will yield a posterior, however, to know the density in its full form we must

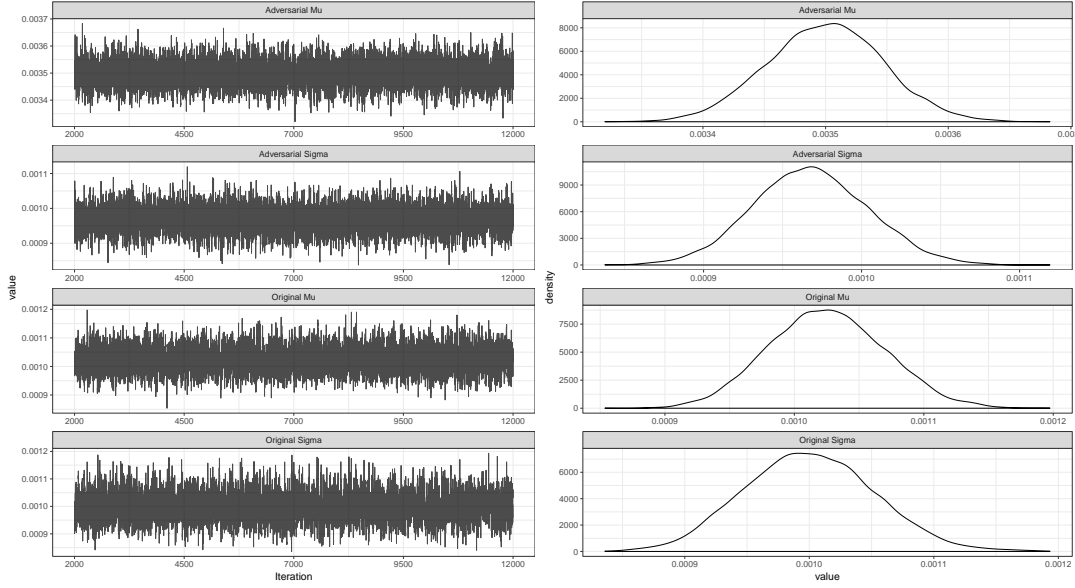


FIGURE 5.6: Metropolis-Hastings diagnostic plots for uncertainty inference from MNIST images. Adversarial mu and sigma refers to the parameterisation of the posterior distribution representing uncertainty in the presence of adversaries. The same is true of original mu and sigma for uncertainties from unperturbed images.

adopt an MCMC approach. Using the Metropolis-Hastings algorithm from Section 3.2.2, we can attempt to sample from this posterior. All we need to do now is supply to the MH-Sampler sampler with reasonable starting values for our four parameters

$$\begin{aligned}\mu_{adv} &= 0.01 & \sigma_{adv} &= 5 \\ \mu_{ori} &= 0.01 & \sigma_{ori} &= 5.\end{aligned}$$

We run the MH-sampler for 12000 iterations, taking a final burn-in of 2000 samples. From Figure 5.6, it can be seen that our Markov chain has performed extensive exploration of the parameter space with no evidence of sticking being present. This convergence allows us to now infer the difference between the uncertainty estimates arising from adversaries compared to unperturbed images.

To help visualise the difference in parameterisation of the two posterior distributions, 100000 samples from each posterior distribution have been taken and plotted in Figure 5.7. For both posterior distributions, the standard deviation parameter value is estimated to be 0.001, however, the posterior's mean values are 0.0035 and 0.0010 for the uncertainty estimates where the image is perturbed and unperturbed respectively.

Clearly the two posterior distributions are distinct, and this is backed up through testing the number of cases where a MCMC sample for μ is greater in the adversarially perturbed

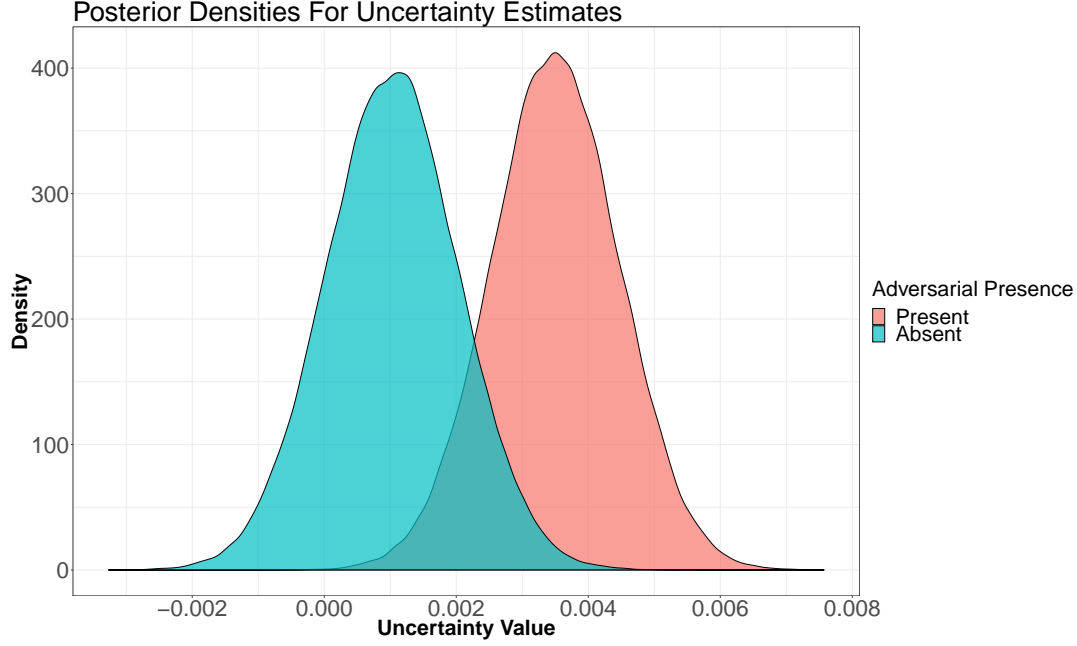


FIGURE 5.7: Posterior distributions for the uncertainty values originating from classification carried out on adversarially perturbed images and their unperturbed counterpart.

chain compared against the unperturbed chain. By calculating the mean amount of times this condition is met, we can state the probability that a mean uncertainty sample originating from an adversarially perturbed image will be greater than in value than an uncertainty sample from an unperturbed image. We can do this for 100000 samples to arrive at the following

$$\begin{aligned}
 & \frac{1}{N} \sum_{i=1}^N \text{sign}(\mathcal{N}(\mu_{adv}, \sigma_{adv}) - \mathcal{N}(\mu_{ori}, \sigma_{ori}) > 0) \\
 &= \frac{1}{100000} * 96155 \\
 &= 0.96155
 \end{aligned}$$

This calculation allows us to infer that there is 0.962 probability that an uncertainty estimate in an adversarially perturbed image will be larger than that of its unperturbed counterpart. One advantage of conducting inference in this way is that we can calculate the probability of two samples being different by a certain amount, γ , through $\frac{1}{N} \sum_{i=1}^N \text{sign}(\mathcal{N}(\mu_{adv}, \sigma_{adv}) - \mathcal{N}(\mu_{ori}, \sigma_{ori}) > \gamma)$. By taking the difference in sample values for each of the 100000 samples, we can then compute the empirical CDF (Figure 5.8) and infer the probability of an adversarial uncertainty estimate being equal to or less than its unperturbed counterpart by any set amount.

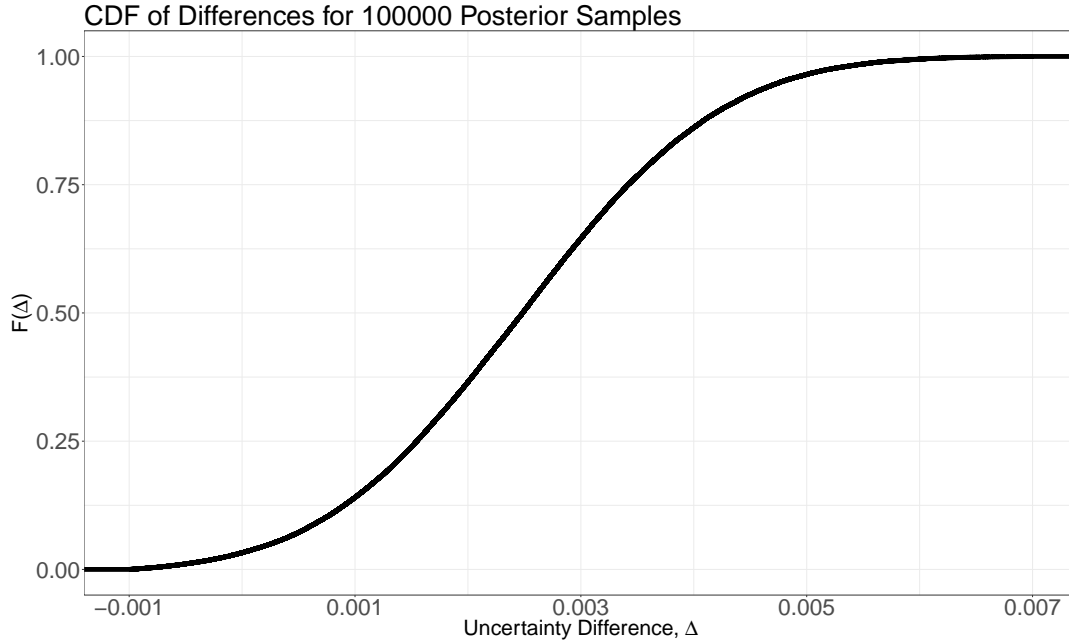


FIGURE 5.8: Empirical CDF of 100000 differences in posterior uncertainties.

5.2.5 Real Word Dataset

While establishing baselines within a community through datasets such as the MNIST dataset is important, these datasets are not particularly representative of real-world datasets. Further to this, there is the possibility that over time we will, as a community, only develop models good at predicting handwritten digits, not *true* computer vision. To extend this work to a real world domain, we test both networks on an image dataset representing 5,863 chest x-rays². Unlike the MNIST task, this is a binary classification as to whether the observation has pneumonia or not.

The nature of this task is fundamentally the same as classifying MNIST digits, however, we are now trying to extract image based details at a much finer granularity. To successfully accomplish this task we must therefore adapt our model slightly in order for the task's increased complexity to be managed. We can do this by adding an additional convolutional layer into our network to enhance the model's power for learning features of the image. Remaining parameters are kept constant and loss and cost functions are unchanged.

We train both a CNN and BCNN on 4684 images, with the CNN and BCNN achieving accuracies of 92.24% and 93.17%, respectively, in circa 9 minutes for both network types. As with the MNIST dataset, both network's achieve comparably high accuracies, the BCNN slightly outdoing the CNN's performance. Due to the task now being a binary classification task, we can easily probe deeper into each classifiers performance by examining precision and recall metrics. We can formally define these metrics as

²Dataset: <https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia/home>

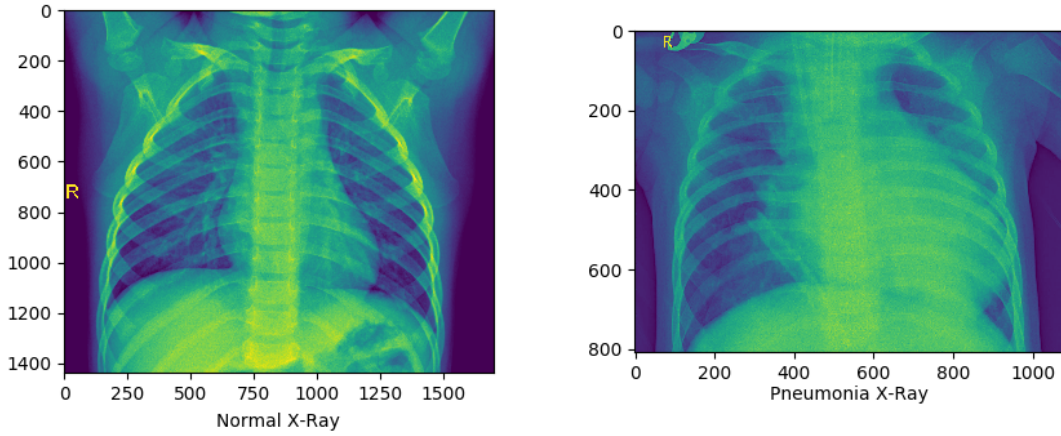


FIGURE 5.9: Examples of a patient's X-Ray with, and without, the presence of pneumonia.

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \quad \text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}.$$

Intuitively we can think of precision as identifying the number of positive classifications that had a positive ground truth value. In the current pneumonia example this corresponds with diagnosing someone with pneumonia, based upon an X-Ray image, and the person did in fact have pneumonia. Recall measures the amount of actual positives that were correctly identified as positive. Again, tying this back to the contextual case of pneumonia, recall would measure the number of patients suffering from pneumonia who were found to have the disease based upon the X-Ray image. Both of these metrics are useful in practice and often an improvement in one corresponds to a decrease in the other. A good model should not only have high accuracy, but also reasonable precision and recall values.

We can visualise the precision and recall of our pneumonia CNN and BCNN through the confusion matrix in Figure 5.10. From this, we can calculate that the CNN classifier has a recall of 77.9 and precision 92.2. Similarly, the BCNN attains a recall of 79.5 and 94.3 for precision. Both models favour precision over recall which in the medical domain is usually preferable as it means we are catching a majority of the cases where an individual is suffering from a given ailment. As with accuracy, the BCNN outperforms the CNN in both recall and precision, reinforcing the notion that on unperturbed images, a BCNN offers greater predictive power.

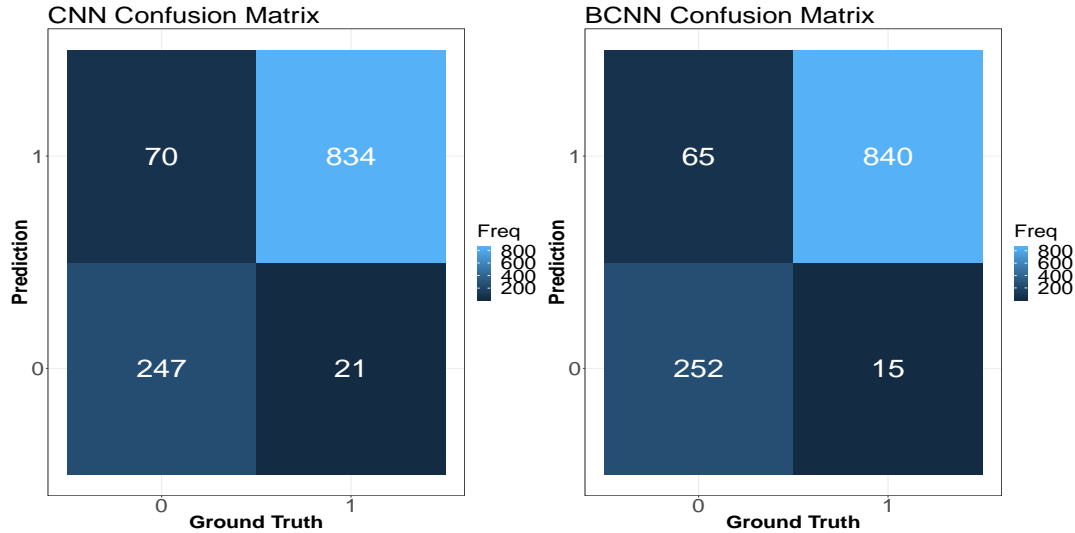


FIGURE 5.10: Confusion matrices showing the performance of a standard CNN and Bayesian CNN in classifying pneumonia from an X-Ray.

5.2.6 Adversaries In X-Rays

Just as with the MNIST digits in Sections 5.2.3 and 5.2.4, we'll now investigate the effect of adversaries on the X-Ray images to see if adversaries are more potent in more complicated image sets. Just as before, we'll test a pre-trained CNN and BCNN on adversarial examples crafted through FGSM at varying levels of ϵ . Due to computational run times, the original testing set was reduced from 1174 down to 200 images, with the original proportion of pneumonia to non-pneumonia image ratio kept equal at 73:27 respectively.

As can be seen in Figure 5.11, the results are slightly peculiar, with the accuracy being heavily impinged beyond random guessing. This is peculiar as the attacks being carried out are not targeted (as with the MNIST experiments), meaning that the adversary is not trying to make a classifier classify 1s as 0s and vice-versa. Moreover, the adversary is simply trying to apply gradient-based perturbations to the image with the intention of making classification *hard*. It is plausible that because the task here is binary, tiny perturbations act as targeted attacks and confuse the classifier into classifying an image as the opposite class as its ground truth. However, as the value of ϵ grows, the images become increasingly distorted and move so far away from the training data manifold that the classifier does not even recognise the image and resorts to random guessing then.

The effect of FGSM is clearly devastating though, with the classifier's performance dropping to random guessing at an ϵ of just 0.06, significantly smaller than the 0.9 required for the MNIST dataset. This result is consistent with results found on comparable datasets, such as CIFAR-10, in the literature. However, through BCNNs we are able to again extract and utilise uncertainty estimates on each adversarial example.

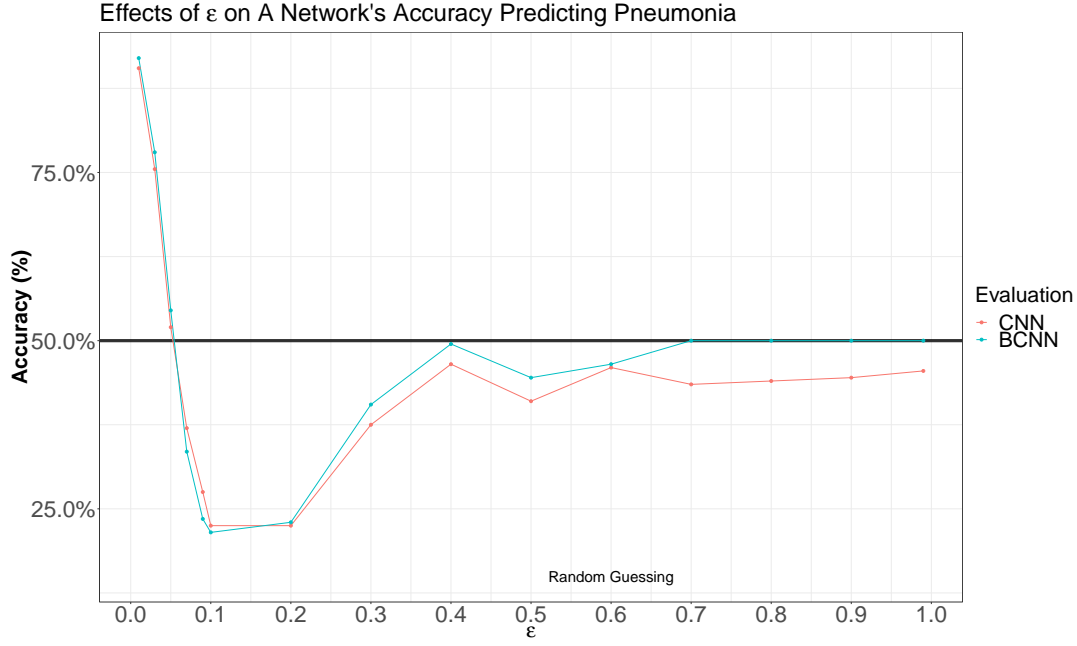


FIGURE 5.11: The effect of epsilon on a CNN and BCNN's accuracy in classifying pneumonia.

To extend the analysis conducted over the MNIST dataset, we will again carry out a single-tailed t-test to test for a difference in mean uncertainty values between uncertainty estimates originating from an adversarial prediction and the unperturbed counterpart image. Using an identical set of hypotheses to those in Section 5.2.4, a p-value of the order 10^{-16} is calculated. This reinforces the earlier result that there exists extremely strong evidence to reject H_0 in favour of H_1 , thus indicating that the mean value of the adversarial prediction's uncertainty is greater than that of the unperturbed images.

In order to derive a new CDF for the difference in uncertainty estimates, a new MH-sampler will be run using an identical setup to the one posed in section 5.2.4. Through Figure 5.12, nice, explorative Markov chains can be observed, with parameter values for μ and σ being found to be

$$\mu_{adv} = 0.0035 \quad \mu_{ori} = 0.0020 \quad (5.3)$$

$$\sigma_{adv} = 0.0011 \quad \sigma_{ori} = 0.0018. \quad (5.4)$$

Again, there is a distinct difference in the two dataset's parameter value for μ , however, unlike in the MNIST case, the spread of the data, σ is much larger in the original image's uncertainty estimates. This could be because the performance was so poor for the BCNN over perturbed X-Ray images that the uncertainty was always high, however, there could

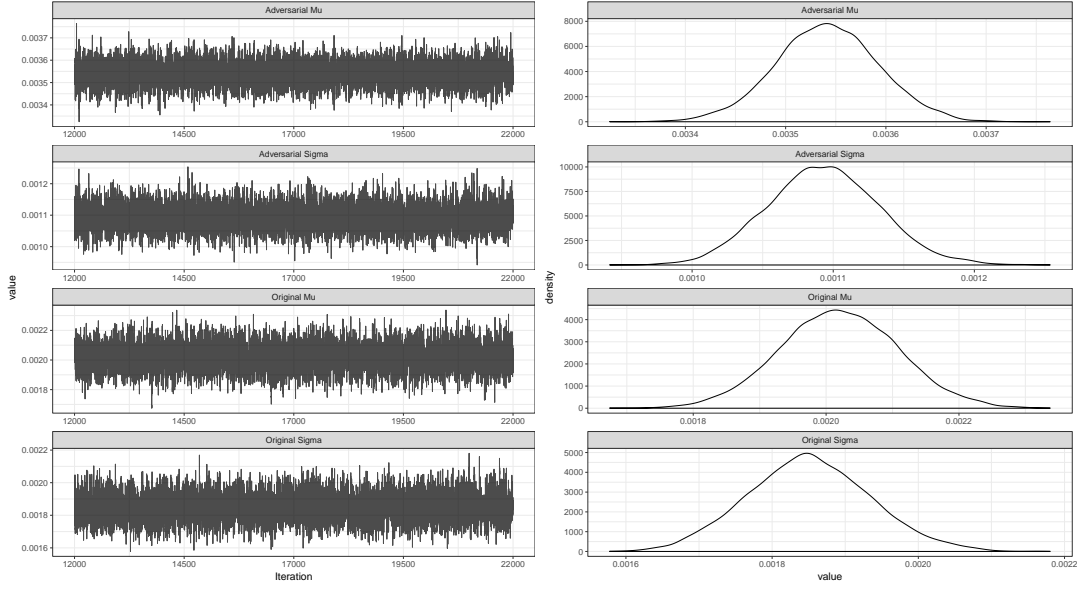


FIGURE 5.12: Metropolis-Hastings diagnostic plots for uncertainty inference from chest X-Rays searching for pneumonia. Adversarial mu and sigma refers to the parameterisation of the posterior distribution representing uncertainty in the presence of adversaries. The same is true of original mu and sigma for uncertainties from unperturbed images.

have been numerous unperturbed images that were just hard to classify, thus leading to a high uncertainty value and consequently more variance within the estimates. By making 100000 draws from a normal distribution parameterised by our newly found parameters, we can now visualise the posterior distributions for both sets of uncertainty estimates in Figure 5.13.

There is considerably more overlap present in these two posterior distributions when compared to those seen for the MNIST dataset in Figure 5.13. This indicates that the difference between uncertainty values, while present, is not quite as significant as in the MNIST dataset. We can test the probability of the adversarially perturbed image's uncertainty value being larger through the mean number of posterior samples that are greater than the unperturbed counterpart's sample. Running this comparison for 100000 samples, it can be found that there is 0.76 probability an adversarially uncertainty estimate will be larger. A probability so high is powerful as it enforces the earlier analysis, thus strengthen the notion that BCNN uncertainties can be used to identify the presence of an adversary.

For completeness, we again derive the CDF for the two posterior distribution's difference, such that the difference is calculated by $\Delta_i = \tau_{adv,i} - \tau_{ori,i} \forall i \in [1, 100000]$, where $\tau_{adv,i}$ is the i^{th} sample from the adversarial uncertainties' posterior. We can clearly see from Figure 5.14 that the probability of a sample having a difference less than or equal to the x-axis' value. For example, there is a 75% chance that an adversarially perturbed image's uncertainty estimate will be 0.0027 units greater than an unperturbed image's uncertainty

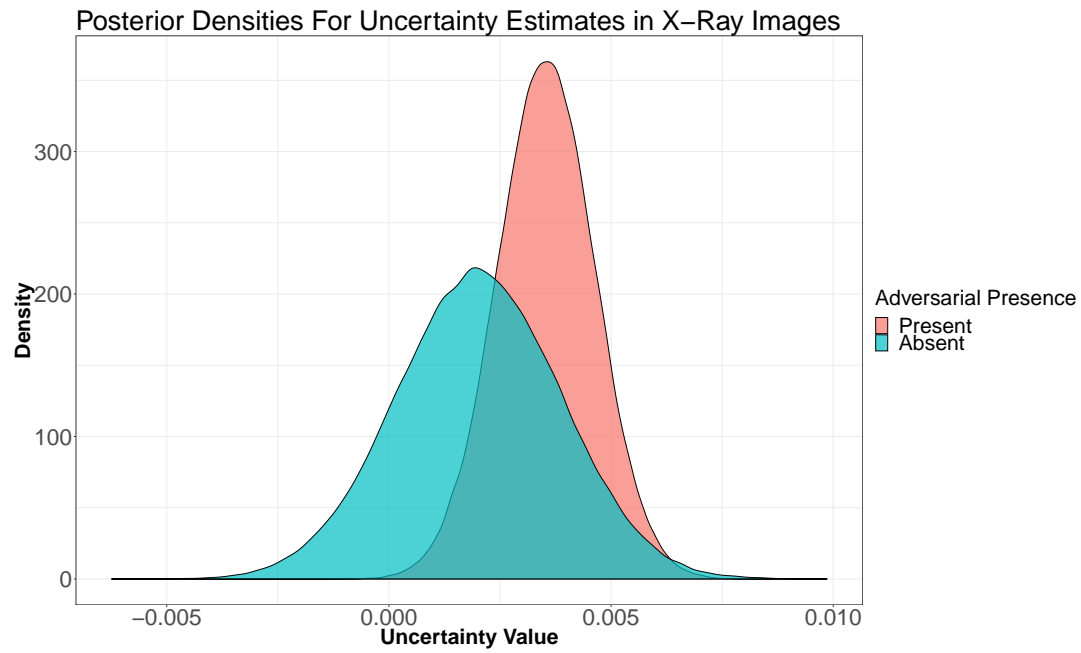


FIGURE 5.13: Posterior distributions for uncertainty estimates originating from BCNN predictions made on adversarially perturbed X-Ray images and their unperturbed counterpart.

estimate.

5.3 Training A Deep Q-Network With Bayesian Neural Network Functional Approximation

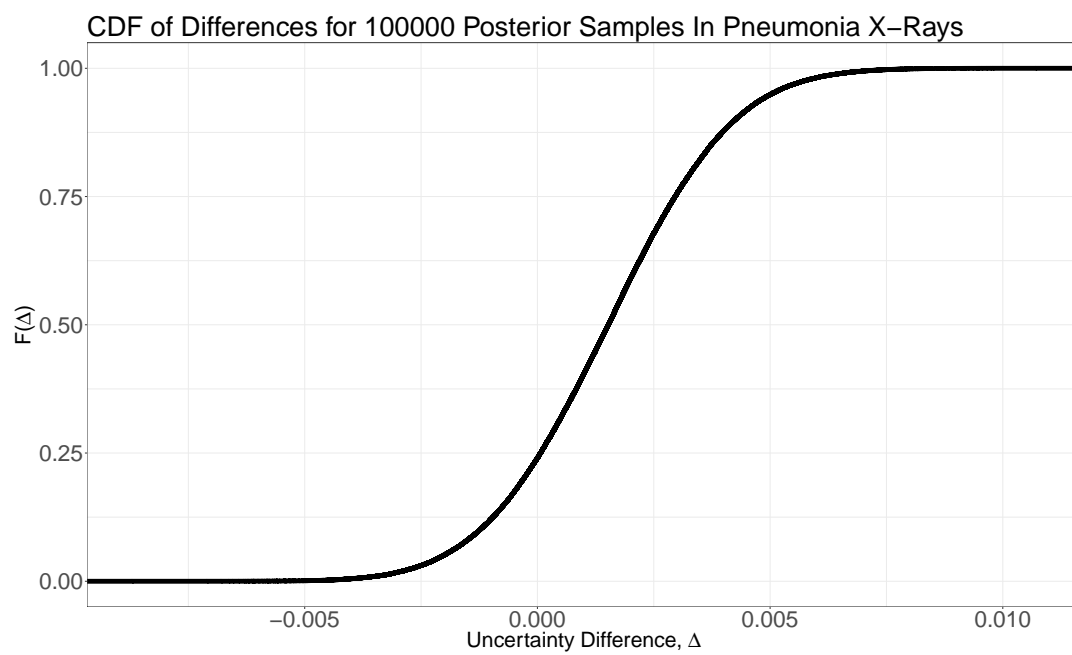


FIGURE 5.14: Empirical CDF, F , for the probability a posterior sample of adversarial uncertainty is greater than an uncertainty estimate original from an unperturbed image.

Chapter 6

Discussion

6.1 Discussion

The hypothesis behind the BCNN and CNN's accuracy drop to below random guessing and then a *recovery* to random guessing for the X-Ray images in Section 5.2.6 is a observation made by Szegedy et al. In this work they found that images undergoing random distortion¹ reduced a neural network's accuracy to random guessing, however, adversarial examples were able to reduce the classifier's accuracy to a value worse than random guessing [53]. With this work and that of Szegedy et al., observing this outcome, the destruction caused by adversarial examples can be seen to go beyond just random guessing, even without crafting targetted attacks.

6.2 Further Work

Detecting and mitigating the effects of adversarial examples is an incredibly popular area of research within the machine learning community. Consequently, new methods and theories appear very frequently in the literature. In this section, three potential ideas for extending the work presented here are given, however, it is worth noting that many more ideas exist.

6.2.1 Conditionally Classifying Nearest Neighbouring Observations

An immediate extension to the work presented here is to analyse the suitability of BCNNs in detecting adversaries within reinforcement learning (RL) problems whereby images are used to make future actions. Huang et al. were able to show that adversaries were devastating within RL, however, the work presented here motivates the idea that through BCNNs, evasive measures could be made by the agent when an adversary is detected. In a contextual setting, these evasive methods could be domain specific, such as parking the car in self-driving cars.

¹Random distortion is different to an adversarial perturbation as it is simply Gaussian noise overlaid onto an image.

Building on this idea, it would of interest to test the efficacy of nearest neighbour algorithms, such as KNN, to be activated when an adversary is detected. Throughout the agent's training phase when we often assume the model is void of any adversaries, we can build a set observations that we know to *unperturbed*. Through a dimensionality reducing algorithm, such as Principal Component Analysis or t-Distributed Stochastic Neighbour Embeddings, a lightweight set of observations would be collected. Upon conclusion of the agent's training, these 2-D or 3-D image representation points could be modelled using KNN. With this trained KNN present in the background, it could be activated when the presence of an adversary is detected and instead of determining a next action based on the current adversarial example, the agent could instead find the image's nearest-neighbour from the underlying KNN model and determine a next action based upon this. To more verbosely explain this idea, see Algorithm 1

Result: Write here the result

Train DQN with BCNN, $h(\cdot)$;

Train KNN $K(\cdot)$ Throughout Training;

```

for  $i \leftarrow 0$  to  $gameCount$  by 1 do
    observation = current state;
     $Q(s, a)$ , uncertainty =  $h(\text{observation})$  if  $uncertainty > threshold$  then
        observation =  $K(\text{observation})$ ;
         $Q(s, a) = h(\text{observation})$ ;
    else
        pass;
    end
end

```

Algorithm 1: KNN as a contingency classifier in RL problems.

6.2.2 Re-Calibrating A Model To Refine Uncertainty Estimates

A secondary line for potential future work could concern improving the final model's calibration and sharpness, an idea drawn from meteorology and the work of Kuleshov, Fenner, and Ermon. For a model to be well calibrated, an event estimated to occur with probability p must occur $(p * 100)\%$ of the time. Intuitively, we can think of perfectly calibrated models in a Bayesian framework as being models whereby the n^{th} percentile credible interval captures the true probability of an event occurring. A *sharp* model on the other hand will output very sure predictions that are very close to 0 or 1, meaning there is little confusion in a models belief of a prediction's probability [26].

Using the notions of sharpness and calibration, a model can be improved post-training through *re-calibration*. In a classification task, re-calibration is accomplished by first training a classifier such as a BCNN, then training an additional regression model on the density produced by the BCNN's predictions and the observation's empirical density. Intuitively this can be thought of as *nudging* the final prediction density in the direction of the true underlying density, thus resulting in a sharper, more calibrated model [26]. Kuleshov, Fenner, and Ermon was able to significantly improve the accuracies of BCNNs, as proposed by Gal and Ghahramani [5], by performing this re-calibration.

The potential advantage in adversary detection is that un-calibrated models underestimate a prediction's uncertainty, thus, using the idea that adversarial examples will surpass some uncertainty threshold, potentially missing some adversarial examples [6]. To summarise this second notion, by re-calibrating our final BCNN, we may be able to produce more accurate uncertainty values, consequently correctly identifying an increased number of adversaries.

6.2.3 Experimenting With Dropout Extensions

A final line of potential further work concerns the recent developments in the differing methods of application for dropout in a neural network architecture. Since, Srivastava et al. along with Hinton et al. first presented Dropout in 2014 and 2012 respectively, as a technique to prevent overfitting in neural networks, several variations of Dropout have appeared in the literature [50, 19]. Two such examples of this are Gaussian dropout [55] and variational dropout [22]. Both methods hinge themselves upon sampling from a Gaussian distribution, rather than a Bernoulli, so significant work would need to be done to investigate the convergence of neural networks to GPs when dropout probabilities are sampled from a non-Bernoulli distribution.

6.3 Limitations And Reflections

Chapter 7

Conclusion

7.1 Contributions

Within this work we have presented and contributed several experiments and ideas, the primary one being the potential uses of BCNNs to detect adversaries. The Bayesian-based approach shown in Sections Finally, we have provided an open source, fully documented code base. Within this, there exists implementations of Bayesian neural networks in both Keras [2] and PyTorch [42], along with a Bayesian CNN based DQN implementation in Python. Significant objects and functions have been fully documented through Sphinx¹ and a thorough README.md file exists in the source code's repository detailing how people can replicate the work carried out here.

7.2 Closing Thoughts

¹<http://www.sphinx-doc.org/en/master/>

Appendix A

Appendices

A.1 Image Pre-processing

In the experiments conducted in Section 5, both the MNIST and X-Ray images' pixel values were normalised to the range of $[0, 1]$. This is common practice in computer vision tasks and therefore allowed our results to be more comparable to that found in the literature. Additionally, the X-Ray images were originally of varied width and height dimensions. Constructing a neural network to accommodate for such non-uniformity is not an easy task and it is not uncommon to crop images to a uniform shape. The decision was therefore made to crop all X-Ray images to 224×224 pixels and then flattened the image from a Red-Green-Blue, 3 channel image down to a single channel greyscale image. This final decision was made as after testing two neural networks, one trained on the 3-channel input, the other on the single greyscale input, the performance was indifferent, however, the computational run-time was significantly less for the single channel network.

Bibliography

- [1] Marc G. Bellemare et al. “The arcade learning environment: An evaluation platform for general agents”. In: *IJCAI International Joint Conference on Artificial Intelligence*. Vol. 2015-Janua. 2015, pp. 4148–4152. ISBN: 9781577357384. DOI: 10.1613/jair.3912.
- [2] François Chollet and others. *Keras*. 2015. URL: <https://keras.io>.
- [3] Dan C. Cireşan et al. “Flexible, high performance convolutional neural networks for image classification”. In: *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Volume Two* (2011), pp. 1237–1242. DOI: 10.5591/978-1-57735-516-8/ijcai11-210. URL: <https://dl.acm.org/citation.cfm?id=2283603>.
- [4] Kevin Eykholt et al. “Robust Physical-World Attacks on Deep Learning Models”. In: (July 2017). DOI: 10.1109/CVPR.2018.00175. URL: <http://arxiv.org/abs/1707.08945>.
- [5] Yarın Gal and Zoubin Ghahramani. “Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning”. In: (June 2015). URL: <http://arxiv.org/abs/1506.02142>.
- [6] Yarın Gal, Jiri Hron, and Alex Kendall. “Concrete Dropout”. In: (May 2017). URL: <http://arxiv.org/abs/1705.07832>.
- [7] Andrew Gelman. *Bayesian Data Analysis*, p. 661. ISBN: 1439840954.
- [8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*, p. 775. ISBN: 0262035618.
- [9] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and Harnessing Adversarial Examples”. In: (Dec. 2014). URL: <http://arxiv.org/abs/1412.6572>.
- [10] Alex Graves. *Practical Variational Inference for Neural Networks*. 2011. URL: <https://papers.nips.cc/paper/4329-practical-variational-inference-for-neural-networks>.
- [11] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. “Speech Recognition with Deep Recurrent Neural Networks”. In: (Mar. 2013). URL: <http://arxiv.org/abs/1303.5778>.
- [12] Robby Haelterman. “Analytical study of the Least Squares Quasi-Newton method for interaction problems”. In: (2009). URL: <https://biblio.ugent.be/publication/720660>.

- [13] Hado van Hasselt et al. "Learning functions across many orders of magnitudes". In: *Neural Information Processing Systems*. 2016. URL: <https://www.semanticscholar.org/paper/Learning-functions-across-many-orders-of-magnitudes-Hasselt-Guez/4931c91f4b30eb122def1e697abc096f14c48987>.
- [14] W. K. Hastings. "Monte Carlo Sampling Methods Using Markov Chains and Their Applications". In: *Biometrika* 57.1 (Apr. 1970), p. 97. ISSN: 00063444. DOI: 10.2307/2334940. URL: <https://www.jstor.org/stable/2334940?origin=crossref>.
- [15] Kaiming He and Jian Sun. "Convolutional neural networks at constrained time cost". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2015, pp. 5353–5360. ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7299173. URL: <http://ieeexplore.ieee.org/document/7299173/>.
- [16] Warren He et al. "Adversarial Example Defenses: Ensembles of Weak Defenses are not Strong". In: (June 2017). URL: <http://arxiv.org/abs/1706.04701>.
- [17] Geoffrey E. Hinton and Drew van Camp. "Keeping Neural Networks Simple". In: *ICANN '93*. London: Springer London, 1993, pp. 11–18. DOI: 10.1007/978-1-4471-2063-6{_}2. URL: http://link.springer.com/10.1007/978-1-4471-2063-6_2.
- [18] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. "A Fast Learning Algorithm for Deep Belief Nets". In: *Neural Computation* 18.7 (July 2006), pp. 1527–1554. ISSN: 0899-7667. DOI: 10.1162/neco.2006.18.7.1527. URL: <http://www.ncbi.nlm.nih.gov/pubmed/16764513>.
- [19] Geoffrey E. Hinton et al. "Improving neural networks by preventing co-adaptation of feature detectors". In: (July 2012). URL: <http://arxiv.org/abs/1207.0580>.
- [20] Matt Hoffman et al. "Stochastic Variational Inference". In: (June 2012). URL: <http://arxiv.org/abs/1206.7051>.
- [21] Sandy Huang et al. "Adversarial Attacks on Neural Network Policies". In: (Feb. 2017). URL: <http://arxiv.org/abs/1702.02284>.
- [22] Diederik P. Kingma, Tim Salimans, and Max Welling. "Variational Dropout and the Local Reparameterization Trick". In: *Advances in Neural Information Processing Systems*. 2015, pp. 2575–2583. URL: <https://papers.nips.cc/paper/5666-variational-dropout-and-the-local-reparameterization-trick>.
- [23] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. 2009. URL: <https://www.semanticscholar.org/paper/Learning-Multiple-Layers-of-Features-from-Tiny-Krizhevsky/5d90f06bb70a0a3dced62413346235c02b1aa086>.

- [24] Alex Krizhevsky, Ilya Sutskever, and Geoffret E Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. 2012. DOI: 10.1145/3065386. URL: <https://doi.org/10.1145/3065386>.
- [25] John K. Kruschke. "Bayesian estimation supersedes the t test." In: *Journal of Experimental Psychology: General* 142.2 (May 2013), pp. 573–603. ISSN: 1939-2222. DOI: 10.1037/a0029146. URL: <http://www.ncbi.nlm.nih.gov/pubmed/22774788><http://doi.apa.org/getdoi.cfm?doi=10.1037/a0029146>.
- [26] Volodymyr Kuleshov, Nathan Fenner, and Stefano Ermon. "Accurate Uncertainties for Deep Learning Using Calibrated Regression". In: *International Conference on Machine Learning*. July 2018, pp. 2796–2804. URL: <http://proceedings.mlr.press/v80/kuleshov18a.html>.
- [27] Solomon. Kullback. *Information theory and statistics*. Dover Publications, 1997, p. 399. ISBN: 0486696847.
- [28] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. ISSN: 00189219. DOI: 10.1109/5.726791. URL: <http://ieeexplore.ieee.org/document/726791/>.
- [29] Lin and Long-Ji. *Reinforcement learning for robots using neural networks*. 1992. URL: <https://dl.acm.org/citation.cfm?id=168871>.
- [30] Yen-Chen Lin et al. "Tactics of Adversarial Attack on Deep Reinforcement Learning Agents". In: (Mar. 2017). URL: <http://arxiv.org/abs/1703.06748>.
- [31] David J. C. MacKay. "A Practical Bayesian Framework for Backpropagation Networks". In: *Neural Computation* 4.3 (May 1992), pp. 448–472. ISSN: 0899-7667. DOI: 10.1162/neco.1992.4.3.448. URL: <http://www.mitpressjournals.org/doi/10.1162/neco.1992.4.3.448>.
- [32] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 0028-0836. DOI: 10.1038/nature14236. URL: <http://www.nature.com/articles/nature14236>.
- [33] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: (Dec. 2013). URL: <http://arxiv.org/abs/1312.5602>.
- [34] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. "DeepFool: a simple and accurate method to fool deep neural networks". In: (Nov. 2015). URL: <http://arxiv.org/abs/1511.04599>.

- [35] Kevin P. Murphy. *Machine learning : a probabilistic perspective*. MIT Press, 2012, p. 1067. ISBN: 0262018020.
- [36] Radford M. Neal. *Bayesian Learning for Neural Networks*. Vol. 118. Lecture Notes in Statistics. New York, NY: Springer New York, 1996. ISBN: 978-0-387-94724-2. DOI: 10.1007/978-1-4612-0745-0. URL: <http://link.springer.com/10.1007/978-1-4612-0745-0>.
- [37] Radford M. Neal. *Bayesian Learning via Stochastic Dynamics*. 1993. URL: <https://papers.nips.cc/paper/613-bayesian-learning-via-stochastic-dynamics>.
- [38] Ian Osband et al. "Deep Exploration via Bootstrapped DQN". In: *Advances in Neural Information and Processing Systems (NIPS)*. 2016, pp. 4026–4034. URL: <https://papers.nips.cc/paper/6501-deep-exploration-via-bootstrapped-dqn>.
- [39] John Paisley, David Blei, and Michael Jordan. "Variational Bayesian Inference with Stochastic Search". In: *ICML (2012)*. URL: <http://arxiv.org/abs/1206.6430>.
- [40] Nicolas Papernot and Patrick McDaniel. "Deep k-Nearest Neighbors: Towards Confident, Interpretable and Robust Deep Learning". In: (Mar. 2018). URL: <http://arxiv.org/abs/1803.04765>.
- [41] Razvan Pascanu and Yoshua Bengio. "Revisiting Natural Gradient for Deep Networks". In: *International Conference on Learning Representations* (Jan. 2013). URL: <http://arxiv.org/abs/1301.3584>.
- [42] Adam Paszke, Sam Gross, and Adam Lerer. "Automatic differentiation in PyTorch". In: *Advances in Neural Information and Processing Systems (NIPS)*. 2017. URL: <https://www.semanticscholar.org/paper/Automatic-differentiation-in-PyTorch-Paszke-Gross/b36a5bb1707bb9c70025294b3a310138aae8327a>.
- [43] Carl Edward. Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. MIT Press, 2006, p. 248. ISBN: 026218253X.
- [44] G.O. Roberts and A.F.M. Smith. "Simple conditions for the convergence of the Gibbs sampler and Metropolis-Hastings algorithms". In: *Stochastic Processes and their Applications* 49.2 (Feb. 1994), pp. 207–216. ISSN: 0304-4149. DOI: 10.1016/0304-4149(94)90134-1. URL: <https://www.sciencedirect.com/science/article/pii/0304414994901341>.
- [45] Andrew Slavin Ross and Finale Doshi-Velez. "Improving the Adversarial Robustness and Interpretability of Deep Neural Networks by Regularizing their Input Gradients". In: (Nov. 2017). URL: <http://arxiv.org/abs/1711.09404>.

- [46] Andras Rozsa et al. "Are Facial Attributes Adversarially Robust?" In: (May 2016). URL: <http://arxiv.org/abs/1605.05411>.
- [47] Stuart J. 1962 Russell and Peter 1956 Norvig. *Artificial intelligence a modern approach*. ISBN: 9781292153964.
- [48] David Silver et al. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm". In: (Dec. 2017). URL: <http://arxiv.org/abs/1712.01815>.
- [49] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (Jan. 2016), pp. 484–489. ISSN: 0028-0836. DOI: 10.1038/nature16961. URL: <http://www.nature.com/articles/nature16961>.
- [50] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: <http://jmlr.org/papers/v15/srivastava14a.html>.
- [51] Jiawei Su, Danilo Vasconcellos Vargas, and Sakurai Kouichi. "One pixel attack for fooling deep neural networks". In: (Oct. 2017). URL: <http://arxiv.org/abs/1710.08864>.
- [52] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning : an introduction*. MIT Press, 1998, p. 322. ISBN: 0262193981.
- [53] Christian Szegedy et al. "Intriguing properties of neural networks". In: (Dec. 2013). URL: <http://arxiv.org/abs/1312.6199>.
- [54] J.N. Tsitsiklis and B. Van Roy. "An analysis of temporal-difference learning with function approximation". In: *IEEE Transactions on Automatic Control* 42.5 (May 1997), pp. 674–690. ISSN: 00189286. DOI: 10.1109/9.580874. URL: <http://ieeexplore.ieee.org/document/580874/>.
- [55] Sida Wang and Christopher Manning. "Fast dropout training". In: *International Conference on Machine Learning*. Feb. 2013, pp. 118–126. URL: <http://proceedings.mlr.press/v28/wang13a.html>.
- [56] Max Welling, D Bren, and Yee Whye Teh. "Bayesian Learning via Stochastic Gradient Langevin Dynamics". In: (2011).
- [57] Christopher K. I. Williams. "Computation with Infinite Neural Networks". In: *Neural Computation* 10.5 (July 1998), pp. 1203–1216. ISSN: 0899-7667. DOI: 10.1162/089976698300017412. URL: <http://www.mitpressjournals.org/doi/10.1162/089976698300017412>.
- [58] W.R.Thompon. "On the Likelihood that one Unknown Probability Exceeds Another in View of the Evidence of Two Samples". In: *Biometrika* 25 (1933), pp. 285–294.

-
- [59] Weilin Xu, David Evans, and Yanjun Qi. "Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks". In: (Apr. 2017). DOI: 10.14722/ndss.2018.23198. URL: <http://arxiv.org/abs/1704.01155><http://dx.doi.org/10.14722/ndss.2018.23198>.
- [60] Matthew D. Zeiler and Rob Fergus. "Visualizing and Understanding Convolutional Networks". In: Springer, Cham, 2014, pp. 818–833. DOI: 10.1007/978-3-319-10590-1_53. URL: http://link.springer.com/10.1007/978-3-319-10590-1_53.