# Practical Probabilistic Systems for Satellite Image Segmentation and Classification

Felix McGregor

Department of Electrical and Electronic Engineering

University of Stellenbosch

Study leader: Prof. J A du Preez

Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Research) in the Department of Electrical and Electronic Engineering at Stellenbosch University

*MEng Electrical and Electronic*

March 2020

# Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

# Abstract

This thesis undertakes the task of satellite image classification from a probabilistic perspective. Our probabilistic approach is motivated by using uncertainty to address the lack of data and variability in satellite image data. In the interest of producing accurate models, we adopt Bayesian neural networks (BNNs) as the primary focus for classification models which offer a way of combining uncertainty estimation with the expressive power of deep learning. Furthermore, due to the limited communication bandwidth of a satellite, we require the model to run on-board a satellite which introduces major computational constraints. BNNs can also be designed to introduce sparsity providing a computationally efficient solution. Despite these advantages, BNNs are rarely used in practice as they are difficult to train. We discuss the most recent advances in variational techniques, including Monte-Carlo variational inference, stochastic optimisation, the reparametrisation trick, and local reparametrisation trick. However, even with these advances BNNs often still suffer from crippling gradient variance. In an attempt to understand this we study the relationship between probabilistic modelling and stochastic regularisation techniques, setting the foundation for practical uncertainty estimators, compression techniques and a signal propagation analysis of BNNs. Using this understanding we present an innovation using signal propagation theory to propose a *self-stabilising prior* that improves robustness in training. We then discuss techniques for incorporating spatial information making use of probabilistic graphical models (PGMs). We connect the output of pixel classifications of a BNN to a PGM, developing a *probabilistic system*. This uses the uncertainty of the classifier, together with the contextual information of neighbouring pixels, to have a de-noising effect on the classifier output. Finally, we experimentally evaluate a series of Bayesian and deterministic models for satellite image classification. We see that Bayesian methods excel in situations where data is scarce. We also see that BNNs are able to achieve levels of accuracy comparable to modern deep learning while either remaining well-calibrated in comparison to deterministic methods, or able to yield extremely sparse solutions requiring only 3 % of the original weights. In addition, we qualitatively illustrate the value of models that recognise their fallibility and incorporating them into probabilistic systems which can reason automatically and dynamically incorporate information from different sources depending on the certainty of each source.

# Uittreksel

Hierdie tesis onderneem satelliet beeld klassifikasie vanuit 'n probabilistiese benadering. Ons probabilistiese benadering is gemotiveer deur die gebruik van onsekerheid om die gebrek aan en veranderlikheid in satelliet data te adresseer. Om akkurate modelle te verseker maak ons hoofsaaklik gebruik van "Bayesian neural networks" (BNNs). BNNs verskaf 'n manier om onsekerheid skatting met die modellering krag van "deep learning" te kombineer. Daarbenewens, weens beperkte kommunikasie bandwydte van 'n satelliet, behoort die model op die te kan satelliet opereer wat groot rekenkundige beperkings voorstel. BNNs kan ook ontwerp word om parameters te verwyder wat gevolglik koste effektiewe oplossings verskaf. Ten spyte van hierdie voordele word BNNs selde gebruik want in praktyk kan die opleiding van die modelle geweldig moeilik wees. Ons bespreek onlangse vernuwings in variasionele tegnieke, wat "Monte-Carlo variational inference", "stochastic optimisation", die "reparametrisation trick" en "local reparametrisation trick" insluit. Ons bestudeer ook die verwantskap tussen BNNs en stogastiese regularisering tegnieke wat die fondament vir praktiese onsekerheid skatters, kompressie tegnieke en 'n sein voortplanting analise van BNNs lê. Hierdie tegnieke het Bayesiese diep-leer moontlik gemaak, maar die tegnieke ly steeds aan skadelike gradiënt variansie. Ons spreek hierdie aan met 'n innovasie met die gebruik van sein voortplanting teorie om 'n *self-stabiliserende prior* voor te stel wat opleiding robuust maak. Daarna bespreek ons die gebruik van probabilistiese grafiese modelle (PGMs) om ruimtelike inligting te inkorporeer. Ons verbind die uitset van die klassifikasie model aan 'n PGM, om 'n *probabilistiese stelsel* te ontwikkel. Dit gebruik die onsekerheid van die klassifiseerder in kombinasie met die kontekstuele inligting van die naburige pixels wat die uitset skoon maak. Laastens maak ons 'n eksperimentele evaluering van 'n reeks van Bayesiese en deterministiese modelle op satelliet beeld klassifikasie. Ons neem waar dat Bayesiese modelle presteer in situasies waar data skaars is. Ons sien ook dat BNNs diep-leer vlakke van akkuraatheid bereik terwyl hulle óf, goed gekalibreer bly in vergelyking met deterministiese metodes, óf in staat is om uiters koste effektiewe oplossings te lewer, wat net 3 % van die oorspronklike parameters vereis. Daarbenewens, ondersoek ons die waarde van modelle wat hul feilbaarheid kan herken wat stelsels gee wat dinamies inligting van verskeie bronne kan inkorporeer en outomaties redeneer.

# Acknowledgements

Firstly I would like to thank my study leader, Prof du Preez, whose experience and guidance was invaluable. I would like to acknowledge the willingness to always allow freedom and encourage the joy of exploration. I was also always assured and secure in knowing that you always had my best interests at heart.

Very importantly, I would like to thank my family Gregor, Marié and Ross. Thank you for your wise counsel and sympathetic ears. Writing this thesis was challenging and a journey of many obstacles, major growth and development. I would like to express gratitude for you all being my unshakeable foundation and a great source of uncompromising support and empowerment. Thank you for catching me when I pushed too hard. I would like to express a sincere appreciation, without you, this would not have been possible.

I have received a great deal of support in writing this thesis but for the person who had to listen to my thesis rants the most, Erin, a special thank you for your relentless encouragement, support and belief. I would like to acknowledge you and express that I have so much gratitude for the absolute pillar of strength you were for me in this journey.

I would like to thank my lab-mates, Elan and Cornel, and my flatmates, Alan and Brad, for fun, positive and encouraging day to day interactions, being my friends and community, and making an effort to at least try and understand what my thesis is about.

To my colleague and co-author, Arnu, thank you for a wonderful collaboration and being a true mentor.

I also gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used in this research.

# Contents

# List of Figures

# List of Tables

# List of Acronyms

| | | |
|---|---|---|
| BNN | - | Bayesian Neural Network |
| ELBO | - | Evidence Lower Bound |
| MAP | - | Maximum a Posteriori |
| DNN | - | Deep Neural Network |
| PGM | - | Probabilistic Graphical Model |
| EB | - | Empirical Bayes |
| KL Divergence | - | Kullback-Leibler divergence |
| CLT | - | Central Limit Theorem |
| ReLU | - | Rectified Linear Unit |
| MC | - | Monte-Carlo |
| MCMC | - | Markov chain Monte-Carlo |
| MCVI | - | Monte-Carlo Variational Inference |
| adMCVI | - | adaptive Monte-Carlo Variational Inference |
| RT | - | Reparametrisation Trick |
| LRT | - | Local Reparametrisation Trick |
| LBP | - | Loopy Belief Propagation |
| LBU | - | Loopy Belief Update |
| RIP | - | Running Intersection Property |
| CPD | - | Conditional Probability Distribution |
| HMC | - | Hamiltonian Monte-Carlo |
| NDVI | - | Normalized Difference Vegetation Index |
| PCA | - | Principle Component Analysis |
| SNR | - | Signal to Noise Ratio |
| SVM | - | Support Vector Machine |
| MRF | - | Markov Random Field |
| CRF | - | Conditional Random Field |

# List of Notations

| | | |
|---|---|---|
| $x$ | - | scalar |
| $\boldsymbol{x}$ | - | vector |
| $X$ | - | matrix |
| $P(x)$ | - | probability of x |
| $p(x)$ | - | probability distribution over x |
| $\boldsymbol{I}$ | - | identity matrix |
| $\mathcal{N}(\mu, \sigma^2)$ | - | Gaussian in covariance form with mean $\mu$ and variance $\sigma^2$ |
| $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ | - | multivariate Gaussian in covariance form with mean $\boldsymbol{\mu}$ and covariance matrix $\Sigma$ |
| $\sigma(a)$ | - | Sigmoid function of $a$ |
| $A \odot B$ | - | element-wise multiplication of $A$ and $B$ |
| $\|\|\boldsymbol{x}\|\|$ | - | $L^2$ norm or Euclidean of $\boldsymbol{x}$ |
| $\mathrm{KL}(q\|\|p)$ | - | Kullback Leibler divergence between $q$ and $p$ |
| $\mathbb{E}_q$ | - | Expectation with respect to $q$ |
| $q_\phi(w)$ | - | Distribution $q$ over random variable $w$. Distributional parameters defined by $\phi$ |
| $\nabla_\phi y$ | - | partial derivative of $y$ with respect to $\phi$ |
| $\epsilon \sim p(\epsilon)$ | - | random variable $\epsilon$ is distributed as defined by distribution $p(\epsilon)$ |
| $\mathcal{H}$ | - | cross entropy |

# List of Symbols

| | | |
|---|---|---|
| $D$ | - | dimension |
| $\mathcal{D}$ | - | data |
| $c$ | - | constant |
| $b$ | - | bias |
| $\boldsymbol{w}$ | - | weight vector |
| $W$ | - | weight matrix |
| $\theta$ | - | random variable |
| $p(w\|\mathcal{D})$ | - | true posterior distribution |
| $q(w\|\mathcal{D})$ | - | approximate posterior distribution |
| $\mathcal{L}$ | - | ELBO |
| $h_j^l$ | - | hidden unit $j$ at layer $l$ |
| $q(W)$ | - | approximate posterior or variational distribution |
| $p(W)$ | - | prior distribution |
| $\tilde{q}(W)$ | - | product distribution of prior and approximate posterior |
| $p(\epsilon)$ | - | auxiliary noise distribution that does not depend on parameters |
| $\mu$ | - | Gaussian mean |
| $\sigma^2$ | - | Gaussian variance |
| $B$ | - | pre-activation matrix |
| $\nu$ | - | empirical variance at pre-activation |
| $\tau$ | - | empirical mean at pre-activation |
| $Z$ | - | normalisation constant |
| $\delta_{s\to t}$ | - | message from sender $s$ to target $t$ |
| $\Psi$ | - | cluster belief |

# Chapter 1

# Introduction

This thesis investigates satellite image classification. We study machine learning methods to assign labels to pixels of a satellite image. This corresponds to identifying what type of crop is growing or effectively locating and mapping farms represented on earth. The images are hyper-spectral that provide an information-rich measurement at each pixel suitable for allowing machine algorithms to identify complex discriminating features. These models can be used to monitor farming, making it possible to farm more productively and efficiently. Given that we can identify farms, we can also reason further to identify areas that need intervention due to disease, drought etc. in farms. Frequent monitoring of agriculture is also in high demand for monitoring and identifying crops to estimate crop yields, expected food supply and geographical change [1].

The main challenges in satellite image classification are: (1) there are extremely few labelled training examples due to an expensive labelling process; (2) a very limited computational budget since models are required to run on-board a satellite due to limited communication bandwidth. This thesis focuses on the Bayesian approach to address the aforementioned challenges. Compared to standard machine learning, Bayesian methods offer better uncertainty estimation relative to the data a model has seen, as well as automatic model regularisation vital for reducing overfitting, particularly in settings where data is scarce. Another key advantage is that the Bayesian framework can flexibly introduce prior information. This can take the form of inducing sparsity into models, resulting in reduced computational cost useful for embedded applications with limited computational resources.

Our approach involves exploring classical and proven pattern recognition models, in logistic regression and neural networks, from a Bayesian perspective. These models are trained to recognise

the hyper-spectral signature of a pixel and assign each individual pixel to a particular class. The predictions of these models produce a *noisy* image representing the mapping of farms or estimated pixel classes. We then use probabilistic graphical models (PGMs) to process this image. We use this to reason about the class of a particular pixel in the context of its neighbouring pixels, thus integrating spatial information that has a de-noising effect.

## 1.1 Problem Background and Overview

Hyper-spectral satellite images contain a substantial amount of information to analyse crops. A widely used metric for analysing vegetation is the normalized difference vegetation index (NDVI) [2], [3], which is comprised of key spectral bands of a hyper-spectral fingerprint. The NDVI index is used frequently to highlight vegetation and can be used by expert analysts to roughly interpret the health of a plant. However, with machine learning we can build models to automatically make predictions and classifications about crops. Early machine learning methods were developed and applied in the form of classical statistical classifiers such as logistic regression, random forests [4] and, amongst the most successful, support vector machines (SVMs) [5].

Recently, deep learning has become the predominant approach to satellite image classification [6], [7]. These methods have shown potential for learning better feature representations in classifying satellite images and markedly improved predictive performance. With deep learning, however, performance is commensurate to the amount of data available. Data is inherently scarce in remote sensing applications as the labelling process is expensive. Labelling usually requires intensive human attention and is time-consuming. Another consideration of deep learning is that larger, more complex or deeper models are usually associated with increased success and improved performance. We require the model to make predictions on-board a satellite (discussed further in the project objectives). Due to energy and computational constraints, the excessive size of many modern neural networks precludes it from being realistically deployed on a satellite.

The main focus of this thesis is on Bayesian neural networks (BNNs) as they allow us to make use of the predictive performance of neural networks with the benefits of the Bayesian approach. Following the Bayesian approach makes it possible to coherently reason and train models in uncertain conditions. This makes these techniques robust to overfitting and variations in data, thereby making them adept to training with little data. BNNs outperform standard networks when data is extremely limited, even with proper regularisation [8], [9]. We also intend to use a classifier in conjunction with other probabilistic models to build a *probabilistic system*. In this

setting it is useful for models to be able to accurately estimate their uncertainty, as Bayesian models do, such that we can reason in context of their confidence.

The recent resurgence of BNNs is due to a host of recent advancements in approximate Bayesian inference, making inference more scalable, efficient, accurate and faster [10], [11], [12], [13], [14]. This thesis concentrates on and discusses relevant methods that allow scalable and practical inference that will allow us to employ the principles of Bayesian modelling to deep neural networks. Despite recent advances, large gradient variance still remains an issue and scaling BNNs to larger, more expressive models is still a challenging task [15]. Addressing this, we present novel self-stabilising priors, inspired by signal propagation theory [16], [17], [18], that allow us to more robustly scale BNNs. We will see that BNNs with stabilising priors outperform deterministic neural networks and other BNNs in satellite image classification in terms of accuracy and quality of uncertainty estimation as we are able to make use of larger models from a probabilistically principled paradigm.

Another advantage of BNNs we investigate is the ability to compress models by imposing sparsity inducing priors. We investigate heuristic compression techniques as well as *variational dropout* [19], stemming from recent advances in interpreting stochastic regularisation techniques as Bayesian inference [20], to sparsity models. We will see that the size of BNNs can be greatly reduced without a decline in predictive accuracy. We are able to discard 97% of the original weights for satellite image classification using variational dropout. By sparsifying the model, we can deploy a powerful yet compact model that avoids unnecessary computation and resources.

Remote sensing applications usually require additional means to supplement the classification models as hyper-spectral images are highly susceptible to noise and classes can prove difficult to distinguish. Even for models trained on large amounts of data, the observation noise typically causes predictions to be noisy and output images or farm mappings to be speckled. This elicited research on filters in combination with machine learning for satellite image classification [4], [21]. A widely used class of filters are extended morphological profiles (EMP) which are based on morphological transformations [22]. Filters are applied to suppress or reduce noise or enforce spatial smoothness.

Traditional filters, however, do not fully capture contextual relationships and are usually characterised by a series of hard-coded transformations. Probabilistic approaches, such as Markov random fields (MRFs) [23], have been employed to more accurately model local pixel interactions. In this thesis we investigate a more general probabilistic approach, using cluster graphs

[24], allowing us to flexibly design models to address noise and inject knowledge about how nearby pixels influence each other.

## 1.2 Project Objectives

**Accuracy:** A fundamental requirement for our system to be of any use in practice is that it should be accurate. The model should be capable of learning underlying patterns in complex satellite data to produce mostly correct classification assignments. Furthermore, since the task is to deploy a system that participates in a world where it is exposed to a myriad of situations, the system must be robust and generalise to unseen observations. We adopt the deep learning approach as it has proven tremendously successful at various complex classification tasks. Deep learning, however, presents difficulties in the context of satellite image classification as it can be very computationally expensive and is particularly prone to overfitting. This leads us to our other objectives.

**Uncertainty-Aware or Calibrated Models:** Many machine learning algorithms, particularly deep learning, require a large amount of data in order to generalise well. While there is a vast amount of satellite data available, there are very few labelled examples. Images that have labels for crop classification are particularly scarce and are often only partially labelled. This is because labelling satellite images is very expensive and time consuming. In situations where data is scarce, overfitting is an important concern. Models may specialise on peculiarities present in small subsets of data that may not be representative of the true underlying patterns. This is a particularly relevant concern in the domain of land cover classification as a particular crop may exhibit many variations due to season, water content, stage of growth etc. and there is a lot of variation in hyper-spectral measurements from variations in angle, cloud cover, sun angle etc. We thus require that the models we develop be well suited to small data regimes and training regimes to be very robust to overfitting. We undertake this by mandating that a model be aware of when it is uncertain. Preserving uncertainty relative to the amount of observation noise or data observed is an effective strategy to avoid overfitting. A model should be more confident in its correct predictions and less confident in its erroneous predictions (a model like this is said to be calibrated). With the Bayesian framework we can also supplement uncertain predictions with alternative sources of information such as a prior, or defer decisions on uncertain predictions to a human expert.

**Computationally efficient:** The system is required to run on-board a satellite as communi-

cation between a satellite and ground station is extremely restricted. Communication is often only possible for short periods on occasions few and far between. With limited communication bandwidth, it is desirable to process images on-board and only relay results which, compared to sending a raw hyper-spectral image, is significantly more efficient. Considering the limited energy and computational resources on-board a satellite, prioritising computational efficiency is crucial. We thus focus on reducing model size, complexity and computation time of predictions. Note that the computational constraints apply only to predictions and training procedures are unrestricted.

We address these objectives by adopting the Bayesian framework for modelling. Bayesian methods excel in settings where data is scarce and have principled methods for modelling data under uncertainty. Using BNNs, we can utilise the acclaimed predictive performance of deep learning while accurately estimating uncertainty yielding models that are robust to overfitting as well as capable of modelling highly complex functions. In addition, we also investigate using the Bayesian framework to compress neural networks for huge computational savings without a reduction in accuracy. We also investigate Bayesian logistic regression as a baseline representing a computationally efficient solution with uncertainty but not as powerful as a neural network.

## 1.3 Outcomes and Contributions

**Using uncertainty in a probabilistic system:** Often Bayesian methods are not considered because they are too computationally expensive and non-trivial to implement. We successfully implement Bayesian versions of proven models. Furthermore, we integrate these models into a probabilistic system that uses uncertainty to assist in reasoning about pixel classes. This is done by combining Bayesian models that recognise the spectral signature of a pixel with a PGM to integrate spatial information. The outcome is a probabilistic system that uses uncertainty to dynamically rely more on either the hyper-spectral information in the pixel itself or on contextual information from neighbouring pixels. The value of these methods lie in their ability to remain uncertain so as to reduce purporting false positives in sequential decision making systems. This is advantageous in scenarios with little data as we can understand the reasoning of the system and build expert knowledge into the system.

**Implementing advanced variational techniques and applying Bayesian neural networks to satellite image classification:** BNNs have been applied to very few problems as they are difficult to train and challenging to build a stable implementation. We explore and

implement leading-edge advances in variational inference and deep learning and apply them to satellite image recognition. We explore the nascent field of Bayesian deep learning to utilise deep learning as well as uncertainty for satellite image classification. By studying and implementing advances in variational inference, we are able to scale BNNs to deeper and more powerful models, yielding a useful application of BNNs to satellite image classification.

**Theoretical contribution: Self-stabilising priors**

- The following contribution was work done in preparation for a conference in collaboration with Arnu Pretorius. The role of Pretorius was more of a supervisory nature and he appeared as second author. In particular, Pretorius contributed substantially with advice and several discussions around developing derivations in the domain of signal propagation theory, building on his previous work on signal propagation in deterministic networks [16].

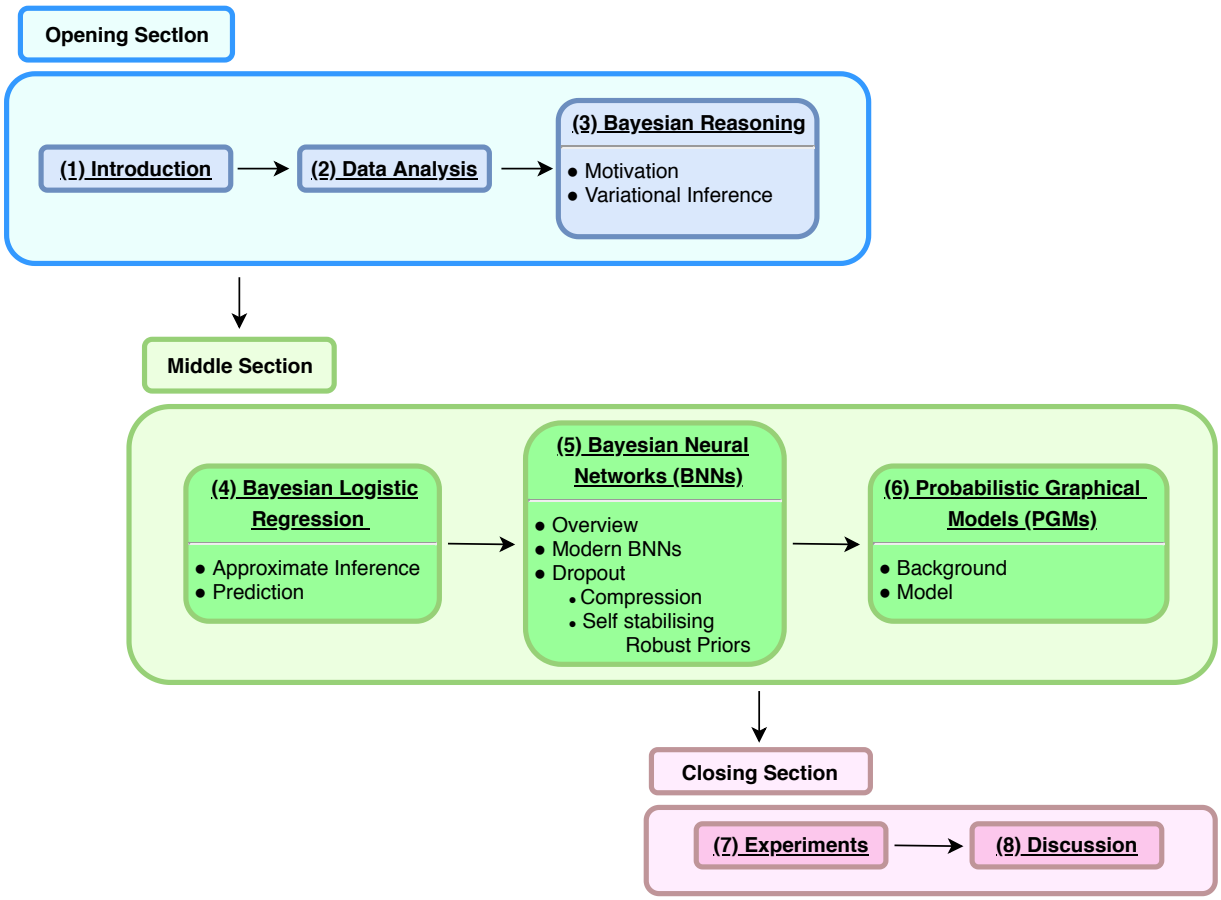
Although BNNs have enjoyed a resurgence in modern Bayesian deep learning, BNNs have yet to reach the level of success of modern deep learning. Stochastic optimisation methods, which makes inference scalable, exhibit high variance, resulting in BNNs being very sensitive to small changes in hyper-parameters, architecture, choice of prior, and it is widely accepted that BNNs are effectively untrainable beyond a certain depth. With signal propagation theory we can quantify this. Inspired by signal propagation theory in deep neural networks [17], [18], [16] we derive a novel prior to preserve the variance of signals propagating through a BNN. By choosing a prior that optimises signal propagation behaviour, it allows us to effectively train deeper BNNs than otherwise possible while also resulting in improved convergence. Included in this approach, we derive a novel evidence lower bound (ELBO) objective to enable the prior to be able to influence the network on the forward pass.

Our work extends initialisation techniques [25] [18], [16] to an iteratively updating prior to allow more stable flow of information through the network *throughout* training. This defends against poor signal propagation associated with vanishing or exploding signals and poor network performance. We further note that this is the first application of signal propagation theory outside of initialisation schemes for deterministic networks that we are aware of.

**Sparsifying Bayesian neural networks:** The final outcome is a successful sparsification of neural networks. We prune down large models to require a fraction of the original weights without a reduction in accuracy. In particular, we reduce a neural network with 5 layers of width 512 to require 3 % of the original weights. This is a major result in the context of our computational

constraints. This drastically reduces storage space and the amount of computation required, making it feasible to deploy on a satellite.

## 1.4 Project Summary




This project essentially undertakes two tasks: (1) Classification of the hyper-spectral signature of a pixel that assigns each pixel to a class. We investigate the use of classifiers in Bayesian logistic regression and BNNs for this task and attempt to accomplish the aforementioned objectives. BNNs constitute a large technical focus of this thesis. (2) Integrating spatial information with the use of PGMs. This combines the probabilistic classifiers from (1) into a probabilistic system that actively relies more on either the hyper-spectral information in the pixel itself or on contextual information from neighbouring pixels based on uncertainty.

Following the introduction we begin with exploring hyper-spectral satellite image data. The discussion is intended to familiarise ourselves with, and gain insight into the data and serves as a preface to our modelling approach. We explore the Indian Pines dataset that defines and guides our model design and is the subject of our experiments. The dataset consists of a single

image where each pixel contains 220 spectral bands representing a 20 $\times$ 20m square on the earth's surface. Despite this high resolution measurement, classification remains challenging as the dataset consists of a single image and thus contains very few labelled examples for pixels to conduct training and testing. We then investigate and plot the spectral profiles contained in a pixel considering examples that exhibits the diversity of intra class variation as well as spectra from different classes with very similar attributes. We also present a classification baseline with logistic regression, observing a noisy mapping relative to the ground truth, demonstrating and advocating the need to incorporate spatial information. Thus, in the context of our data, we motivate our approach of using both spatial and spectral information. This includes BNNs, capable of modelling complex spectral patterns under uncertainty, in combination with PGMs. We then move our discussion to modelling and introduce the Bayesian framework.

As we have established, in satellite image classification, data is scarce and there is a large amount of random variation in the data. This brings about our discussion of the Bayesian framework to address this inherent randomness. Using Bayesian probability theory we are able to use proven machine learning techniques and reason under uncertainty. Bayesian methods also have the ability to introduce prior information or priors that encourage the model to conform to reflect our beliefs about the world. Priors are often successfully implemented in the form of model regularisation (overfitting is a major concern when data is scarce) and sparsification (useful for embedded applications such as on a satellite).

In the Bayesian paradigm, parameters are considered random variables and are modelled as distributions as they are unknown quantities. With parameters no longer representing finite point estimates, parameters express variability or uncertainty with distributions. *Bayesian inference* is performed by simple applications of the rules of probability theory. The goal is to infer a posterior distribution over the parameters once we have seen some data or given evidence. We use the posterior to make predictions about new observations. The posterior reflects how certain we are about parameters relative to the data we have seen and we make predictions by integrating over the posterior. This considers all possible weights dictated by the posterior distribution weighted by their probability. This, in effect, automatically regularises models and quantifies uncertainty about predictions. This is very useful as we can tell whether a model is making informed predictions or guessing at random that is desirable in building connected probabilistic systems. Uncertainty also has applications in low-resource settings such as active learning, which makes the problem of data acquisition more efficient, as we can use uncertainty to identify which labels we should acquire in an image that would be most informative.

Bayesian reasoning is a principled framework to reason about uncertainty, but for many complex models, exact inference can come at a prohibitive computational cost. For many models, inference is intractable meaning no analytic expressions exist, therefore we have to employ approximate inference techniques. In this thesis we aim to develop scalable algorithms such that we are able to employ the modelling power of deep learning with advantages of probabilistic modelling in BNNs. Inference for BNNs is intractable that brings us to *variational inference*. Variational inference involves approximating the true posterior with some approximating variational posterior. This approximate posterior belongs to a family of tractable distributions that is easy to manipulate. We make use of Gaussian distributions that allows us to compute and represent distributions over parameters using only mean and variance statistics. The posterior is estimated by minimising the distance between the approximating posterior and the true posterior according to some metric that measures the distance between probability distributions. We then optimise the parameters of the approximating posterior distribution by calculating gradients with respect to these parameters such that it minimises this distance. We focus on Monte-Carlo variational inference (MCVI) to approximate the true posterior which forms the basis inference technique our discussion of BNNs.

We briefly discuss Bayesian logistic regression that serves as a baseline with which we can reliably compare more complex approaches. Logistic regression is an established classifier often used for its interpretability, but is limited in that it is capable of only representing linear decision boundaries. Nevertheless, Bayesian logistic regression represents a simple baseline and a reliable method to yield uncertainty estimates. Its simplicity also amounts to a computationally efficient solution. We introduce and discuss the Laplace approximation for inference of the posterior and probit approximation for prediction. This serves as a demonstration of applying approximate inference and an introduction to applying Bayesian reasoning to a simple model preparing us for the following chapter in which we discuss BNNs.

We then turn our attention towards neural networks and deep neural networks that have proven to be very successful in modelling input-output relationships with high predictive accuracy. However, these models require huge amounts of labelled data to generalise well and are computationally expensive. Thus, we introduce BNNs and discuss the variational interpretation of these tools such that we can apply deep learning in small data regimes. In doing so, the Bayesian framework also allows us to include prior knowledge that we develop to allow us to obtain accurate confidence estimates and model compression.

The first part of our discussion of BNNs revolves around methods that play a central role in

Bayesian deep learning and constructing practical inference techniques that scale well to large models with many parameters. We begin with casting BNN training to a variational inference objective using MCVI and stochastic optimisation. We then discuss the *reparametrisation trick* [12] which plays a critical role in modern Bayesian deep learning. Its significance lies in that the reparametrisation allows us to pass gradients through stochastic nodes or random variables. This allows us to employ gradient optimisers, so successfully used in deep learning, for variational inference. In BNNs, the weights are stochastic, thus with stochastic optimisation and the reparametrisation trick, we are able to calculate gradients with respect to the variational parameters of the weights. We discuss the reparametrisation trick for a Gaussian followed by evaluating its efficiency as an estimator.

We also discuss the *local reparametrisation trick* [20] which improves on the reparametrisation trick. The core idea is that instead of sampling weights and multiplying them with the inputs to obtain a pre-activation matrix, we calculate the distribution of the pre-activation matrix analytically and sample from the pre-activation directly. This effectively gives us independent samples of the weights for each data point in the mini batch, whereas before we only had one weight sample per mini batch. This leads to less gradient variance and more efficient training.

We end the introduction of BNNs by briefly discussing prediction. Since it it not possible to analytically calculate the predictive probability for a new observation, we often estimate prediction by sampling. We sample a set of weights and compute a forward pass, yielding a sample prediction for a given an input and repeat this many times and average over the predictions. We call this "test-time averaging". While this produces satisfactory performance, requiring many forward passes may be too computationally expensive for deployment on a satellite. To address this, we discuss an alternative method called "distillation" [26]. This involves training a deterministic neural network to mimic the behaviour of a BNN. This distills the behaviour of integrating over the posterior distribution of the weights into a different neural network from which it is cheaper to make predictions. We then demonstrate the application of these techniques to a BNN with fully factorised Gaussian priors and posteriors constituting non-conjugate inference following [27],[11]. We show experiments on MNIST demonstrating robustness to overfitting and that the model provides good uncertainty estimation.

We then turn our attention to the recent theoretical links between stochastic regularisation techniques and Bayesian inference [28], [20]. Specifically, we consider dropout, which injects noise into the model as a means of regularisation. This is done by dropping out or ignoring random units in a network or injecting multiplicative noise to corrupt the weights. The key insight is that

dropout, by training a network with noise injection, accomplishes a form of ensembling which resembles the Bayesian approach of asserting distributions over weights.

This leads the discussion to Monte-Carlo dropout (MC dropout), which casts dropout training in deep neural networks as approximate Bayesian inference in deep Gaussian processes [28] . This method uses dropout during training *as well as test time*. We do not fully explore the theoretical argument but discuss the general interpretation that suggests that dropout approximately integrates over a model with distributions over its weights. This method is widely used for uncertainty estimation in the deep learning community but has attracted some criticisms. We discuss the criticisms but find MC dropout to be a practical and efficient method to obtain a deep network capable of uncertainty estimation. We then discuss the work in [20] relating Gaussian dropout to variational inference in BNNs. Under a specific constraint of the variance parameters we see that Gaussian dropout corresponds precisely to training a BNN. Thus, injecting weights with multiplicative Gaussian noise is equivalent to maintaining a Gaussian posterior distributions over the weights in a variational framework. This understanding sets the foundation for work in model compression as well as signal propagation analysis of BNNs and self-stabilising priors for robust Bayesian deep learning.

We then discuss using Bayesian methods to compress or sparsify neural networks. Neural networks are heavily overparametrised and thus use more memory and computation time than necessary. They can be pruned significantly without any loss in accuracy. This is done by using priors that induce sparsity which urges the model to remove parameters during the learning process. We first explain heuristic ways of pruning down weights. This involves selecting weights of which a large portion of the probability mass lies on zero and pruning these parameters by setting them to zero. Alternatively, we can select to prune weights of which the variance of a parameter is large compared to the mean. We can think of these weights as having a low signal to noise ratio (SNR ratio) and do not contribute to the predictions of our model. We then set weights with low signal to noise ratios to zero. We also discuss automatic relevance determination (ARD) priors for BNNs [9] that automatically determines the degree to which inputs are relevant to the performance. ARD priors can be used in conjunction with either of the aforementioned criteria to promote sparsification. We then demonstrate these techniques with an experiment on MNIST where we see that we are able to achieve the same accuracy as with all the parameters by only using 10 % of the weights. We then discuss the work of [19] that follows the previously discussed variational dropout [20]. It turns out that the prior that is implied by variational dropout (by interpreting training with Gaussian dropout as training a BNN [20]), implicitly describes a

sparsity-inducing prior. Then, with the use of an *additive reparametrisation trick*, variational dropout naturally sparsifies the model. The additive reparametrisation trick effectively replaces multiplicative noise with additive noise which yields more stable gradients. This allows for a method that sparsifies the model during the optimisation process. With variational dropout we are able to prune a neural network, reducing parameters to 3 % of the original number of weights.

Until this point, we have introduced BNNs as useful tools for leveraging the expressive power of deep learning with benefits of probabilistic modelling. However, BNNs have not yet reached the level of success of modern deep learning because of their limited practicality. In practice, deep BNNs are brittle and hard to train. Due to the stochastic nature of the optimisation, deep architectures suffer from crippling variance and often require careful tuning of hyper-parameters for any training to occur. We thus present *adaptive Monte-Carlo variational inference* (adMCVI) with self-stabilising priors for robust training of BNNs. Using a signal propagation analysis of BNNs [17], [18], [16], we design a prior with parameters derived to ensure stability of a signal propagating through the network. This allows more stable flow of information through the network throughout training. Signal propagation in BNNs is determined by the parameters of the weight distribution in BNNs and we find conditions that allow us to adjust these parameters and promote stable signal propagation.

Traditionally, the prior impacts the variational objective or the *evidence lower bound* (ELBO) through a regularising additive term, affecting the weights at update time with backpropagation. In this setting, the prior has no effect on the signal propagation dynamics of the network. We suggest that priors exert their influence during the forward pass, so as to make them capable of promoting stable signal propagation. We thus present a novel alternative variational objective to allow the prior to influence the network on the forward pass. This is essential if any training is to occur in deep networks, i.e. this enables the signal to reach the outputs. With this objective, we develop a self-stabilising prior, where the parameters of the prior are adjusted at each forward pass to preserve the variance of signals propagating forward. This approach to variational inference stabilises network dynamics during training and leads to improved convergence and robustness. This makes it possible to train deeper networks and in more noisy settings. We demonstrate the effectiveness of adMCVI with stabilising priors in several experiments on MNIST, CIFAR-10 and synthetic data.

The discussion until this point has been focused on classification models or BNNs to classify the spectra of each individual pixel. The resulting image from the classifier output, where each pixel has been assigned a class, is most often noisy, speckled and disjoint due to variation in

the data, measurement noise as well as an undersupply of data. We then turn our attention towards processing the output of the classifier and creating a connected probabilistic system. Our system constitutes inputting the outputs of the classification model into a probabilistic graphical model (PGM) that de-noises the image. The ability of Bayesian models to include prior information offers an advantage in low data situations as we are able to express where we believe there to be some relationship between variables. PGMs are practical and interpretable, making it easy to translate prior knowledge meaningfully and allow us to assert certain beliefs when it cannot be established from the data. We wish for our resulting image to more closely resemble real-world farms and be more continuous or smoother in shape. We make use of PGMs to explicitly incorporate this prior knowledge into our system. Our modelling approach follows the assumption that strong correlations exist between neighbouring pixels. We thus incorporate spatial information by allowing neighbouring pixels to influence the probability of a particular pixel.

Our work with PGMs focuses on cluster graphs. We introduce fundamental background information by discussing concepts of representation and inference in these graphs. Briefly, inference is done by communicating evidence between variables with an algorithm called belief propagation. This involves an iterative "message-passing" algorithm that updates beliefs about variables given evidence and relationships with other variables. Following the introduction of these, and other fundamental concepts necessary to understand PGMs, we design a model where we integrate spatial information into the per-pixel classification result. We discuss describing a continuous relationship between pixels with a PGM by encoding this knowledge in the graph structure. Pixels are configured to communicate their beliefs about what class they belong to and how it may affect their neighbours. We can then reason about a pixel in context of its predicted class as given by the classifier and the adjacent pixels.

We demonstrate the effect of our PGM model on some small examples but find that inference in PGMs is expensive. The problem we face is that scaling our model to multiclass situations grows exponentially in storage space with the number of classes. To counter this we configure the PGM in a specific way, representing the PGM more compactly, involving a different factor configuration. This makes it possible to scale inference to many classes and greatly reduce the amount of storage space required. We also discuss an augmentation to our PGM where we use the confidence of our classification models as a prior belief. This can be interpreted as the likelihood of an error which in turn allows the PGM to probabilistically reason whether or not it should change the belief that a pixel belongs to a specific class. These models can then function

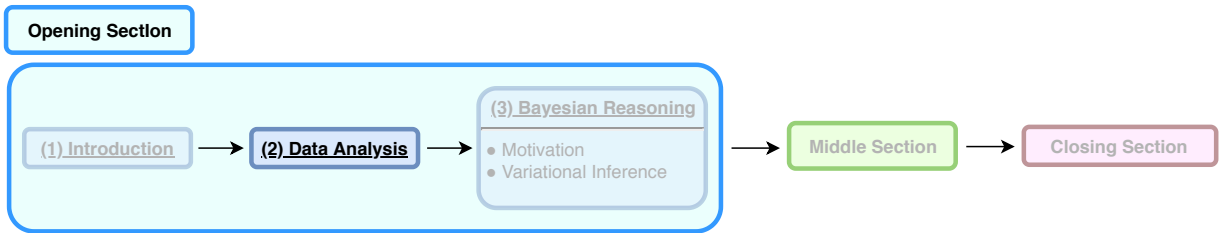
in a system that assesses when a pixel may need to rely more on the information supplied from adjacent pixels or from the classification model.

We then come to the experiments chapter, considering a series of experiments investigating whether the methods we developed are accurate and able to generalise. We compare logistic regression and Bayesian logistic regression as well as BNNs with self-stabilising priors, normal BNNs with Gaussian priors, MC dropout and deterministic neural networks. We see that Bayesian methods excel in situations where data is scarce. It is also evident that BNNs achieve good accuracy while remaining aware of its uncertainty in predictions. We qualitatively analyse the effect of using a PGM to integrate spatial information or de-noise images showing various output examples. We also study how uncertainty aids the PGM in reasoning about pixel classes. Finally, we compare model compression techniques using BNNs such that we can feasibly deploy these models on a satellite.

The thesis concludes with a summarised account of the work followed by a discussion of the most consequential results based on the experiments. We discuss how BNNs offer a flexible solution that yield accurate models under uncertainty while also being capable of reducing computational cost. Logistic regression offers a simple modelling procedure with efficient inference, but may produce less meaningful decision boundaries on small data and not capture true relationships. We review generalisation in the context data scarcity and variability and review our approach using probabilistic systems and uncertainty to address this. Finally, we make recommendations for satellite image classification and suggestions for future work.

# Chapter 2

# Data Exploration




Our discussion commences with exploring hyper-spectral satellite image data. Hyper-spectral images contain more spectral bands than regular images, representing a rich source of information for classification. The discussion aims largely to gain insight into the data and frame our modelling approach for the forthcoming chapters. We explore the *Indian Pines* dataset which is the focal point of our experiments and the subject of our model design. The dataset consists of a single image illustrating the relative paucity of data in satellite image classification. We investigate and plot the spectral profiles or *fingerprints* of representative pixel samples to investigate and demonstrate the inter-class similarity and intra-class variability of crop classes. Lastly, we present a classification baseline with logistic regression observing a noisy mapping relative to the ground truth, demonstrating and advocating the need to incorporate spatial information.

## 2.1 Hyper-spectral satellite images

Hyper-spectral images are a major source of land cover information and a rich source of information for monitoring and characterising agriculture [1]. This data is acquired from satellites that capture images with a spectral resolution of hundreds of bands of the electromagnetic spectrum. Compared to regular RGB images, containing 3 bands in the visual spectrum, these images

carry vastly more information in the infra-red and x-ray spectrum and enable more comprehensive analysis of the Earth's surface. Land cover classification is one of the most prolific uses of hyper-spectral data. This involves training a model on a collection of pixels with known labels to recognise and classify new pixel observations. Despite the large amount of information present in hyper-spectral images, it can be difficult to classify due to the intra-class variability, inter-class overlap and limited number of training samples. Furthermore, classes are usually manually annotated, thus suffering from human biases and varying accuracy.

## 2.2 Indian Pines Dataset

We make use of the Indian Pines dataset that is widely used in land cover classification research and benchmarking [29]. It contains a single $145 \times 145$ satellite image of agricultural land where each pixel is labelled as belonging to one of 16 crop classes or as part of a 17-th background or other class. Each pixel represents a $20 \times 20$ m patch of land on earth and contains 220 spectral bands. Of the satellite image datasets available for research, only the Indian Pines and Salinas datasets contain labelled hyper-spectral images for crop classification. We focus on Indian Pines because the Salinas dataset was captured from a satellite with a much closer orbit than the satellites we consider, representing different communication constraints as well as higher spatial resolution with pixels representing 3.7 m for which the task and objectives will deviate from those we set out. In Figure 2.1 we show the Indian Pines dataset image and the ground truth labels. Machine learning models are typically trained on a training set, consisting of a subset of randomly shuffled pixels and using the remaining pixels as a test set.

Figure 2.2 shows examples of the spectral fingerprints contained in a pixel. We show the variability of a particular class in Figure 2.2 (b) illustrating the challenge of capturing all the fluctuations of a single class. We also see in Figure 2.2 (c) that the differences between classes may be slender and classes may overlap making it difficult to distinguish between classes. Note that the variability presented here is contained in a single image where crops are relatively homogenous and classification models do not contend with factors such as measurement noise between images and seasonality. This illustrates that we do not expect that any simple procedure for recognising spectral signatures of crops exists, and models will always have to concern themselves with erratic variability always present in the data. This motivates our approach for investigating BNNs to allow complex modelling in uncertain conditions.

As a benchmark, we present the predictive performance of a logistic regression model in Fig-

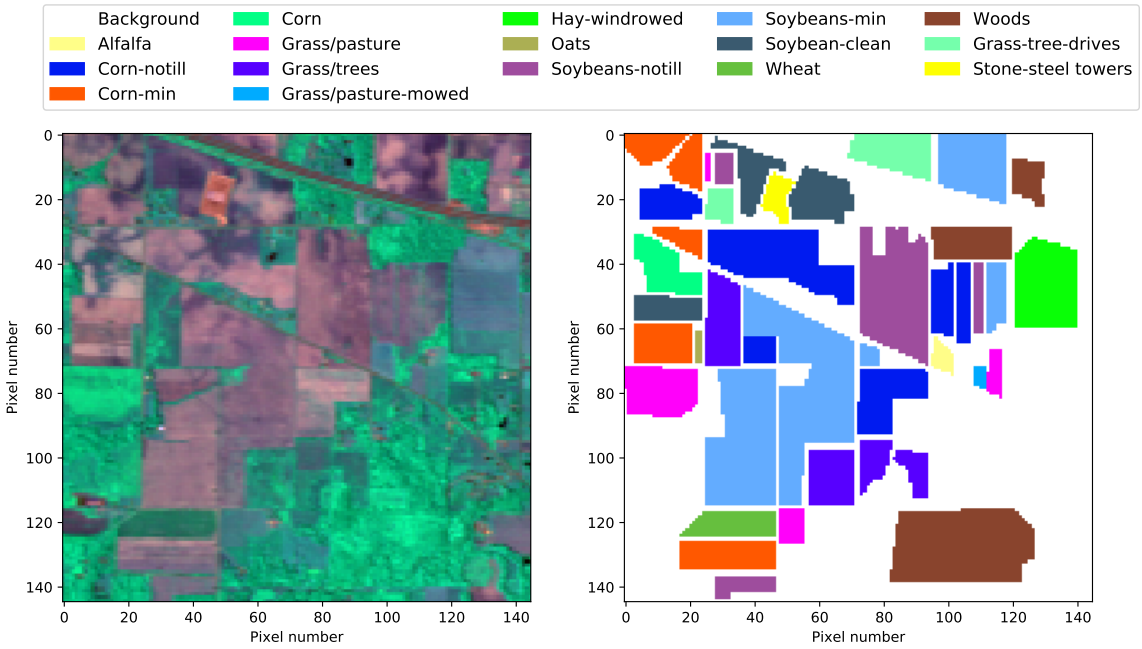



Figure 2.1: Indian Pines dataset labelled ground truth and RGB image.

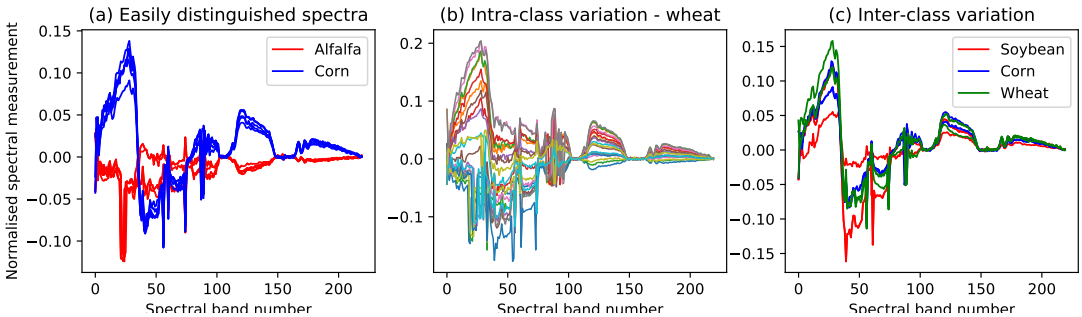



Figure 2.2: Spectral fingerprints of different sampled crops to demonstrate proximity of inter-class variability and intra-class fluctuation. (a) Represents a few samples from classes that can easily be distinguished while (c) represents easily confused classes. (b) Demonstrates the wide range of variability in a single class.

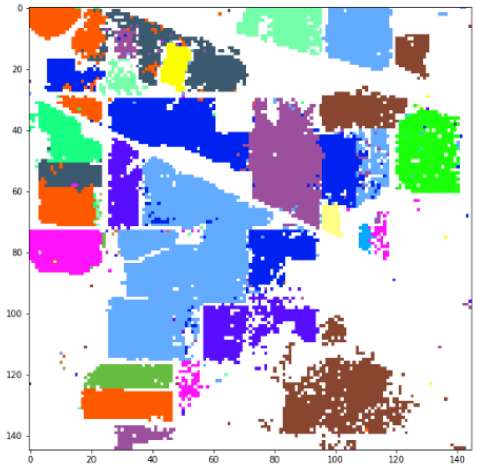


Figure 2.3: Logistic regression benchmark on Indian Pines.

ure 2.3. This particular model is trained on 100 % of the pixels and only reflects training accuracy and not the ability to generalise. Acquiring accurate benchmarks is challenging as training data and test data are scarce since the dataset is comprised of a single image. Moreover, another difficulty of using a single image is that it is not possible to learn contextual relationships from data. This would present test leakage and would not truly reflect the model's ability to generalise. From observation of the ground truth in Figure 2.1, we deduce that farms generally occur in unbroken clusters or patches and nearby pixels are correlated. As shown in Figure 2.3, per-pixel land cover classification techniques typically produce noisy estimates and could benefit from incorporating spatial information. Remote sensing techniques have, as a result, identified strongly with filters and building models that incorporate spatial information. This supports our approach to use PGMs in combination with a classifier allowing us to insert prior knowledge to model spatial relationships.

## 2.3 Conclusion

In this chapter we explored the dataset we will be using in our experiments. We discussed the nature of the data establishing that we have few labelled examples as well as classes with very similar attributes and classes with large fluctuations, making the classification task challenging. Thus, in the context of our data, we motivated our approach of using both spectral and spatial information. This includes BNNs, capable of modelling complex spectral patterns under uncertainty, in combination with PGMs. Next we move our discussion to modelling and begin by introducing the Bayesian framework.

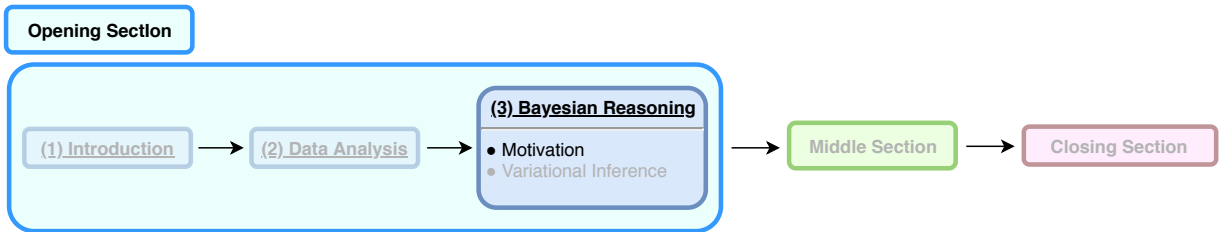
# Chapter 3

# Bayesian Reasoning

*"To know that you do not know is the best. To think you know when you do not is a disease. Recognizing this disease as a disease is to be free of it."*

— Lao Tzu

## 3.1 Motivation




Satellite image classification is intrinsically ensnared with randomness and uncertainty arising from the large amount of variation in the data. A hyper-spectral signature in a particular pixel representing a crop may look different depending on the season, stage of growth, angle, sensor noise etc. A lack of information and randomness is inherent and it will never be possible to fully observe all the variables. Amongst this randomness, however, there is still a comprehensible pattern and large degree of predictability if we can reason under uncertainty.

Machine learning models are capable of finding structure in data, recognising patterns and making predictions on future observations. However, most models have trouble dealing with uncertain situations and have difficulty with problems involving little data. In these cases, oftentimes we cannot tell whether a model is making intelligible predictions or guessing at random. Our aim is

to develop models that are able to reason automatically in uncertain conditions and generalise deterministic models to be able to represent uncertainty.

We formalise our discussion of uncertainty by introducing the Bayesian perspective for probabilistic modelling. Bayesian probability theory is a firmly established tool that offers flexible modelling [30], [31], [24]. It presents a principled language to reason clearly amongst uncertainty in terms of belief and probability. By treating unknown quantities probabilistically, Bayesian methods manage uncertain situations naturally. Inference and learning is performed by simple applications of the rules of probability theory where in general, inference refers to reasoning about unknown probability distributions. Instead of a single variable or weight value, Bayesian modelling treats parameters as distributions, thereby modelling all possibilities of parameters according to a probability distribution. This distribution expresses our beliefs regarding to how likely particular parameter values are relative to data and prior information. Treating parameters in this way generally allows models to generalise better.

We introduce a prior to relate a likelihood function to a probability distribution. The ability of Bayesian models to include prior information exhibits an advantage. We can flexibly insert expert knowledge about a problem to introduce some inductive bias to place more probability mass to better express where we believe there to be some relationship between variables. Priors in BNNs are also commonly applied to achieve model compression [19], [32], regularisation [31], transfer learning with informed priors [33], hierarchical models for complex and abstract variable interactions [34] and to reflect subjective uncertainty preferences for safety-critical applications [35].

The prior is also a common focus for criticism of the Bayesian approach because its subjectivity. This criticism overlooks the fact that any model is subjective according to the assumptions made by the model, citing the popular aphorism in statistics "All models are wrong, but some are useful" [36]. It is true that a misspecified prior could lead to highly erroneous predictions in situations where data is limited. In these situations it is usually better to reduce the sensitivity to the prior by using an uninformative prior which allows the data to unveil patterns and relationships. In these cases, most often the data is sufficiently informative that we can accurately reason despite the vagueness of the prior.

Injecting prior knowledge may be considered a function of the amount of data available. In situations where data is scarce we may need to rely more on prior knowledge as the data may not be fully representative of the patterns manifesting into issues with generalisation. In low data

applications we can encode beliefs and relationships between variables to express our knowledge. In situations where we have a very large amount of data we might find that our prior assumptions of how the world works falls short of how it really works and the data will be sufficiently informative. In this case we may prefer to use black box models like deep learning to leverage its powerful modelling capabilities. Thus, broadly speaking, we can approach probabilistic modelling in two ways: (1) small and focused models built in a principled and well-understood way to gain insight about relationships in our data; (2) black box modelling like deep learning that is very heuristic driven to train highly complex models with excellent predictive performance. It has only been until relatively recently that innovations in variational inference have allowed probabilistic modelling to scale model complexity as well as to larger datasets.

By employing a BNN on the spectra of pixels and a PGM to model contextual relationships, we combine these approaches in a *probabilistic system* where these models interact. In satellite image classification, we do not expect that there is any simple procedure for recognising the spectral fingerprints, advocating for a BNN approach. However, in relating information regarding neighbouring pixels, we may express some knowledge of how noise generally occurs, making use of a PGM. The merits of an interpretable model seem clear in this case as we understand and model a structure in which pixels interact. The ability to quantify uncertainty is essential if we have models that interact in a probabilistic system. Understanding what a model does not know is a critical part in dynamically allowing the system to rely more on information coming from other sources. If our models are able to allocate a high level of uncertainty to their incorrect predictions, the system is able to stop propagating false positives and reason in the context of this uncertainty to make better predictions.

Apart from the discussion of how we intend to address satellite image classification, we briefly discuss a further motivation for our choice of the Bayesian approach. An interesting study [37] showed that in an experiment where labels are assigned completely randomly to a dataset, a neural network consistently obtains 100 % training accuracy and 10 % test accuracy. This shows that neural networks are capable of entirely memorising randomly labelled data rather than finding the true dependence in the data. Dropout or regularisation did not prevent this either. This is *catastrophic* over-fitting and demonstrates that we should be very careful when making use of neural networks. The study was replicated using BNNs and random labelling [19] and showed that BNNs obtain 10 % training accuracy and 10 % test accuracy. While these are only specific case studies, this supports the idea that BNNs are less likely to overfit and likely to reflect that labels are random and there is no underlying pattern.

## 3.2 Bayesian Inference

Following our motivation for the Bayesian approach, we formally discuss the process of Bayesian inference. In the inference process we observe data and infer new posterior distributions given the evidence. We do not choose an optimal set of parameters $w$; we construct a distribution and infer the most probable parameters giving the posterior distribution of the parameters $p(w|\mathcal{D})$, given data $\mathcal{D}$, made up of observations $x$ with labels $y$. We construct a distribution over the weights by defining a prior distribution, $p(w)$. For posterior inference, or learning, we observe new data and incorporate new evidence to update the distribution over weights $p(w|\mathcal{D})$. The influence of new data is captured by the likelihood function $p(\mathcal{D}|w)$. This allows us to calculate the posterior as a function of the unknown model parameters. Bayesian inference of the posterior is derived from Bayes theorem written as

$$p(w|\mathcal{D}) \propto p(\mathcal{D}|w)p(w), \tag{3.1}$$

where it is written as a proportionality as it is not normalised. We are calculating the relative values of the likelihood and omit the normalisation as it is generally not possible to calculate $p(\mathcal{D})$. This procedure estimates the distribution over $w$ that maximises the likelihood in combination with the prior. This allows the prior to influence the posterior and may be designed to regularise or have some other effect. As the amount of data increases, and tends towards infinity, we expect the posterior to concentrate around a point estimate and place all its probability mass on a certain value of the parameter. This in effect washes out the effect of the prior.

For many models, inference of the posterior is intractable and no analytic expressions exist. The difficulty arises when the likelihood function involves some non-linear mapping that makes it impossible to find analytic solutions for the product with a distribution. Inference of the posterior can be done analytically for some models, such as linear regression, where the likelihood is conjugate to the prior. However, all the models we consider do not have a closed-form solution. Exact Bayesian inference is not always possible. Among the techniques most used to overcome this are approximation techniques are sampling techniques and variational inference. Sampling methods are computationally very expensive and we therefore employ variational inference (discussed in the next section).

We are typically interested in the value of some quantity observed in the future and making predictions about it. We make predictions by integrating over the posterior and integrating out uncertainty in parameters to account for all possible settings of parameters and how likely they

are after we have seen the data. This automatically regularises predictions of a model while also enabling accurate uncertainty quantification. For a new observation $x'$, we wish to calculate the make a prediction $y'$ or calculate the predictive probability distribution, $p(y'|w, x', \mathcal{D})$ given by

$$p(y'|x', \mathcal{D}) = \int P(y'|x', w)P(w|\mathcal{D})dw \tag{3.2}$$

which marginalises out the posterior parameters or weights. This is often also intractable, but we can compute its unbiased estimate by sampling and averaging outputs. The sampling estimate usually requires fairly few samples to yield an acceptable estimate.

As an aside, we can interpret probability theory as model agnostic. We may extend the equations presented above to express the implications of a particular model $M$. The posterior is then calculated as

$$P(w|\mathcal{D}, M) = \frac{P(\mathcal{D}|w, M)P(w|M)}{P(\mathcal{D}|M)} \tag{3.3}$$

and the predictive distribution as

$$P(y'|x', \mathcal{D}, M) = \int P(y'|x', w, M)P(w|\mathcal{D}, M)dw. \tag{3.4}$$

This intuitively illustrates how we may train a particular model such as logistic regression or a neural network in a Bayesian way. We can also compute a posterior over models for model selection with

$$P(M|\mathcal{D}) = \frac{P(\mathcal{D}|M)P(M)}{P(\mathcal{D})}. \tag{3.5}$$

This allows us to determine whether we have successfully introduced some inductive bias with a particular model or model architecture. Alternatively, we can find whether some models apply better to a specific dataset. Importantly, this illustrates the idea that we are not calculating the likelihood of data. The likelihood is the function that describes the ability of parameters to explain data and are functions of the model we have selected and its parameters. If we misspecify the model, no amount of data will let us explain the data. We introduced this notation to discuss this interpretation but will make use of the standard notation that omits the model $M$ for the remainder of this thesis.

## 3.3 Advantages and Uses of Bayesian Learning

Now that we have introduced some of the fundamental concepts of Bayesian inference, we can discuss some of the potential benefits that come with using the Bayesian framework. We will not explicitly make use of all the applications we mention in this thesis, but we discuss it nonetheless as it is relevant and useful in the context of satellite image classification.

### 3.3.1 Automatic Regularisation

A Bayesian model inherently incorporates uncertainty in its inference that naturally leads to smoothed estimates and better generalisation. It maintains uncertainty relative to the data it has seen as to not make overconfident predictions. This automatically regularises the model that addresses the problem of overfitting, where the model learns particular patterns specific to a dataset and does not generalise well. In contrast, maximum likelihood estimation provides a point estimate of the optimal set of weights of the classifier. Maximum likelihood estimation is prone to overfitting and often requires regularisation techniques such as penalising model complexity or noise injection. Traditional regularisation, such as L2 regularisation, adds some penalty term to constrain parameters to smaller values such that the parameter values do not grow exorbitantly large which allows the model to generalise better. From a Bayesian perspective, we can show that the L2-penalised solution is equivalent to selecting the mode of the posterior distribution if a Gaussian prior is placed on the parameters that we show next.

In a standard setting our objective is to estimate the likelihood of a model $p(\mathcal{D}|w)$ and find the optimal weights $w_{\text{MLE}}$ with maximum likelihood. Generally, we optimise the negative log likelihood given by

$$w_{\text{MLE}} = \underset{w}{\text{argmin}} - \log p(\mathcal{D}|w). \tag{3.6}$$

Instead, we can adopt the Bayesian framework and aim to find the posterior over the parameters and introduce a prior. Using Bayes' rule we optimise $p(w|\mathcal{D}) \propto p(\mathcal{D}|w)p(w)$ to find the posterior. Adhering to frequentist estimation, we optimise this to obtain finite optimal parameters, $w_{\text{MAP}}$, that corresponds to a maximum a posteriori (MAP), instead of the posterior distribution, $p(w|\mathcal{D})$.

The MAP estimate yields

$$w_{\text{MAP}} = \underset{w}{\text{argmin}} - \log\left(p(\mathcal{D}|w)p(w)\right) \tag{3.7}$$

$$= \underset{w}{\text{argmin}} - \log p(\mathcal{D}|w) - \log p(w). \tag{3.8}$$

The $\log p(w)$ term represents our prior and acts as a regularisation term. Choosing a Gaussian distribution with mean 0 and variance $\lambda$ as the prior gives

$$w_{\text{MAP}} = \underset{w}{\text{argmin}} - \log p(\mathcal{D}|w) - \frac{1}{2\lambda}w^2 + c \tag{3.9}$$

that is equivalent to L2 regularisation. MAP estimation can be seen as a reduced form of Bayesian modelling, compromising between the optimal setting of the weights while fitting prior beliefs. The prior can be interpreted as desiring solutions close to zero as dictated by the variance or certainty of the prior Gaussian variance.

This regularises the model but still yields a point estimate. Anything that is not observed should be integrated out to maintain uncertainty. Our parameters are not observed so we should integrate over them. Only in the case where the amount of data increases to a very large degree, where we expect the posterior to concentrate around some finite parameter, can we be absolutely certain of the value of a parameter. In the context of the vast amount of variance and noise present in satellite image classification, it is naive to assume that one set of parameters can adequately capture and describe the underlying pattern.

Recent work has also related stochastic regularisation techniques in neural networks to Bayesian inference [20], [28], [35]. These noise-based regularisation techniques include variants of dropout [38], [39] that corrupt either the inputs or weights by injecting noise. The result of [35] asserts that for almost any network trained with a stochastic regularisation technique, we can obtain a predictive mean and variance, or confidence, related to some deep Gaussian process. In addition, [20] showed that training a BNN with variational inference, under some restrictions, is precisely the same as training a network with Gaussian dropout. Stochastic regularisation techniques have achieved great success in reducing overfitting and have become customary in deep learning practice. However, the regularising effects of noise regularisation are automatically encompassed by a Bayesian model. Essentially, from a Bayesian point of view the introduction of noise is interpreted as learning a distribution over the parameters, instead of a point estimate parameter. With dropout, instead of searching for optimal parameters, the parameters are effectively sampled from a distribution, thus training an ensemble of networks.

### 3.3.2 Sparsifying Models

Bayesian methods have often been used to compress models [19], [32], [9], [40]. We can prune redundant parameters to reduce computational complexity and storage space. Reducing the number of parameters is desirable in situations where there is limited storage space and energy like onboard a satellite. We do this by using priors that induce sparsity that urges the model to remove irrelevant parameters.

An example of a sparsity-inducing prior includes placing a Laplace distribution over the weights. If the mode of the posterior is used, i.e. the MAP estimate, this corresponds to the L1 solution similar to the Gaussian and L2 regularisation. The Laplacian distribution has a very sharp peak compared to a Gaussian. If we place this peak on zero, it strongly encourages the model to set parameters that do not contribute towards prediction to zero, thus resulting in a sparse solution. We later investigate sparsification of neural networks, making use of variational dropout to sparsify BNNs [19].

### 3.3.3 Online Learning

Bayesian methods naturally learn continually. This is useful for streaming data as it is not required to retrain the entire model each time we receive more data. Data can be used to update the beliefs or the parameters then discarded and never revisited [41], [42].

To illustrate this, we assume that the dataset arrives in $M$ independent different parts. This gives a dataset as

$$\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 ... \cup \mathcal{D}_M. \tag{3.10}$$

We infer the posterior given the first batch of data, $\mathcal{D}_1$, using Bayes rule:

$$p(w|\mathcal{D}_1) = \frac{p(\mathcal{D}_1|w)p(w)}{\int p(\mathcal{D}_1|w)p(w)dw}. \tag{3.11}$$

As we continue to receive data we can continue to train sequentially to update our posterior. Assuming the data $\mathcal{D}_1$ and $\mathcal{D}_2$ is conditionally independent given $w$ we can use the previously

obtained posterior as the prior for the next batch of data:

$$p(w|\mathcal{D}_1, \mathcal{D}_2) = \frac{p(\mathcal{D}_2|w)\Big[p(\mathcal{D}_1|w)p(w)\Big]}{\int p(\mathcal{D}_2|w)\Big[p(\mathcal{D}_1|w)p(w)\Big]dw} \tag{3.12}$$

$$= \frac{p(\mathcal{D}_2|w)p(w|\mathcal{D}_1)}{\int p(\mathcal{D}_2|w)p(w|\mathcal{D}_1)dw}. \tag{3.13}$$

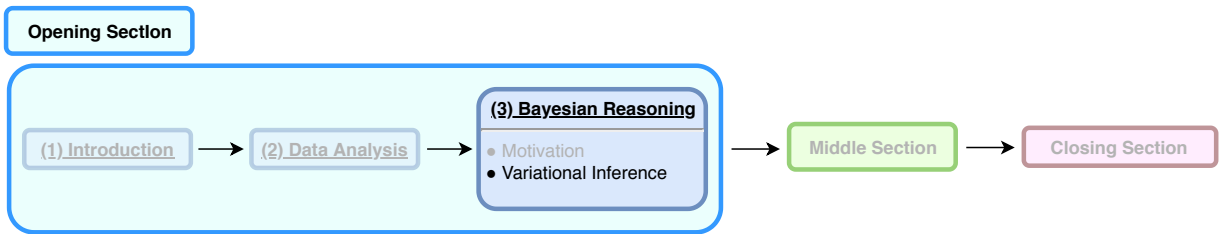
The original prior is always incorporated and we can naturally learn incrementally without requiring to retrain with the arrival of each new batch of data. Doing this for all $M$ parts is then the same as $p(w|\mathcal{D})$.

### 3.3.4 Active Learning

Deep learning usually requires large amounts of labelled data to generalise well. Labelling satellite images is very expensive and time consuming. One could approach the problem with an active learning approach [43] that we expand on next.

One can use the uncertainty estimates in a Bayesian system to identify what unlabelled data would be most informative for the model. This learning system actively proposes which pixels to label by a human annotator in order to improve performance. This allows the system to efficiently explore the variation in the data and gradually increase confidence as more data is observed. The data points to be labelled are selected through an acquisition function, which determines how informative a data point may be. Many different acquisition functions exist, such as expected improvement, which consider the exploration exploitation trade-off. Using active learning significantly decreases the amount of training data required in achieving good performance.

## 3.4 Variational Approximations




Now that we have studied Bayesian inference and considered its benefits, we discuss a family

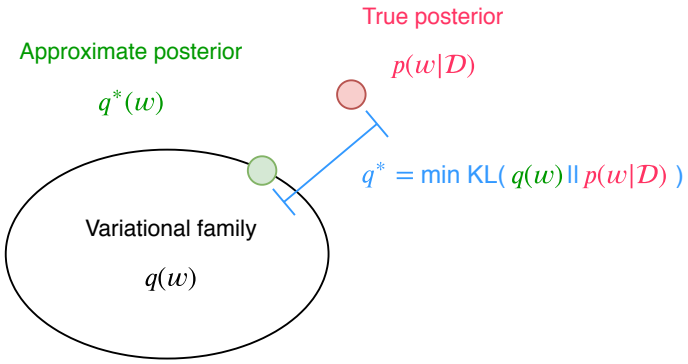



Figure 3.1: Variational inference as optimising the distance between the true posterior $p(w|\mathcal{D})$, and our approximate posterior $q(w)$. We choose an approximating family of distributions that may closely resemble the true posterior. This introduces bias as the approximating distribution will never exactly capture the posterior. However, this allows us the convenience of representing and computing distributions over variables using only a few statistics that describe the distribution.

of practical Bayesian approximate inference algorithms called variational inference. For all the models we consider in this thesis, computing the exact posterior distribution is intractable. In variational inference we project the posterior onto an approximating family of tractable distributions. This allows us to easily manipulate, compute and represent distributions over parameters using only a few sufficient statistics. We then estimate the posterior through optimisation. We use gradient-based procedures to optimise for parameters of an approximating posterior distribution such that it approaches the true posterior. Computing derivatives is often much easier than computing integrals that makes it possible to scale models while maintaining distributions over parameters. Furthermore, we are able to leverage the optimisation tools and methods used in deep learning to make scaling possible.

The true posterior $p(w|\mathcal{D})$, over parameters $w$, usually cannot be evaluated analytically. We approximate this with a variational distribution $q(w)$, which usually has convenient properties, such as simple representation and simple calculation of integrals. The aim is for our approximating distribution to be as close as possible to the true posterior distribution. We measure the distance between the distributions with the Kullback-Leibler (KL) divergence given by

$$\mathrm{KL}[q(w)||p(w|\mathcal{D})] = \int_{-\infty}^{\infty} q(w) \log \frac{q(w)}{p(w|\mathcal{D})} dw. \tag{3.14}$$

We use this to optimise the parameters of our variational distribution to get closer to the posterior. This entails computing gradients with respect to the parameters that define $q(w)$.

## 3.5 Bias Variance trade-off

The traditional view of overfitting is seen as a trade-off between bias and variance of models. Both bias and variance cause error in predictions. Bias is the tendency of a model to be incorrect or the average difference between predictions and the true value. Variance is the variability of model predictions based on how sensitive the model is to random variation. Simple models have high bias but lower variance. Complex models are flexible and are capable of representing the true relationships in data, thereby achieving low bias. This flexibility, however, allows the model to fit and memorise the random variation in the training data and suffer from high variance.

In the context of Bayesian inference, the bias variance trade-off exists in terms of the method in which we do inference, by either sampling methods or variational methods. Sampling methods are unbiased, i.e. we can model more complex probability distributions, but this comes at the price of high variance. For infinite samples, the variance would be zero but in practice we need to determine some acceptable number of samples. Different samples will give different answers which introduces sampling error resulting in fluctuations in our answer. Sampling methods do not make assumptions about the posterior distribution, whereas variational methods restrict us to a set of distributions which we understand. These restrictions or assumptions allows simple representation of the variational distribution and simplifies expressions, often making it possible to analytically calculate approximations of integrals. The variational approximation, however, introduces bias by projecting the posterior on this domain. The true posterior typically lies outside our chosen set of distributions and we attempt to find the best approximating distribution as we show in Figure 3.1. Even if we train our model for an infinite number of iterations with infinite data we will have an irreducible error between our approximation and the true posterior. The projection makes assumptions we cannot overcome.

While variational inference is more limited or restricted than the full set of all probability distributions, it is practical, stable and has low variance. The bias we incur in variational inference is often much less costly than the variance we incur with sampling. Sampling methods are usually only used in smaller models as the number of samples required to counter the variance grows substantially with the number of parameters. We investigate variational methods as they are the only practical methods of scaling probabilistic models to allow Bayesian deep learning.

## 3.6 Formulation of Variational Objective

Probabilistic models have only recently been scaled to large models and datasets with the introduction of stochastic optimisation for variational inference [14]. We now consider the objective for stochastic optimisation of probabilistic models (broadly following the common modern variational formulation of BNNs [10], [11]). Our goal is to approximate the true intractable posterior distribution of our parameters or weights $p(w|\mathcal{D})$, given data $\mathcal{D}$, with an approximating distribution $q_\phi(w)$, from a tractable family parametrised by variational parameters $\phi$. The goal is to find the values of the parameters $\phi$ that minimises the KL divergence. We write this as

$$\begin{aligned}
\phi^* &= \underset{\phi}{\operatorname{argmin}} \quad \mathrm{KL}[q_\phi(w)||p(w|\mathcal{D})] && (3.15)\\
&= \underset{\phi}{\operatorname{argmin}} \int q_\phi(w) \log \frac{q_\phi(w)p(\mathcal{D})}{p(\mathcal{D}|w)p(w)} dw && (3.16)\\
&= \underset{\phi}{\operatorname{argmin}} \int q_\phi(w) \log \frac{q_\phi(w)}{p(w)} dw - \int q_\phi(w) \log p(\mathcal{D}|w) dw && (3.17)\\
&= \underset{\phi}{\operatorname{argmin}} \quad \mathrm{KL}[q_\phi(w)||p(w)] - \mathbb{E}_{q_\phi(w)} \log p(\mathcal{D}|w). && (3.18)
\end{aligned}$$

For discriminative models, where we are interested in modelling $p(y|x, w)$, we can show

$$\phi^* = \underset{\phi}{\operatorname{argmin}} \quad \mathrm{KL}[q_\phi(w)||p(w)] - \int q_\phi(w) \log p(y|x, w) dw. \tag{3.19}$$

This minimises the distance up to some constant of the evidence bounded by the *evidence lower bound* (ELBO). The bound represents a constant irreducible distance between true posterior and the approximating posterior. Thus KL divergence minimisation to the true posterior is equivalent to maximising the ELBO or $\mathcal{L}_q$ which we define as

$$\mathcal{L}_q := \int q_\phi(w) \log p(y|x, w) dw - \mathrm{KL}[q_\phi(w)||p(w)]. \tag{3.20}$$

This defines the objective we will refer to henceforth that we separate and rewrite as

$$\begin{aligned}
\mathcal{L}_q &= L_\mathcal{D}(\phi) - \mathrm{KL}[q_\phi(w)||p(w)] && (3.21)\\
L_\mathcal{D}(\phi) &= \mathbb{E}_{q_\phi(w)}[\log p(y|x, w)]. && (3.22)
\end{aligned}$$

The term $L_\mathcal{D}(\phi)$ is the expected log-likelihood of the data under $q_\phi(w)$. Maximising this encourages the approximate posterior to explain the data well. The second term in (3.21) is the KL divergence between the approximate posterior and the prior over the weights and can usually be

calculated analytically. This introduces some regularisation as it encourages the posterior to be resemble the prior. Thus, in variational inference, we maximise this objective to find the best fit for the variational parameters $\phi$, defining the distribution over our weights, that will explain the data while being close to the prior distribution.

We extend this framework and define $q_\phi(W)$, a probability distribution over any set of parameters represented by tractable probability distributions, such as a Gaussian, and $p(y|x, W)$ any parametric model, such as a neural network, which maps input $x$ to the probability of $y$. Instead of optimising the network's weights $W$ directly, we use gradient-based procedures to optimise their variational parameters, $\phi$. As part of the optimisation objective we require an estimate of (3.22). We discuss how to obtain a stochastic estimate of this expectation with the reparametrisation trick.

### 3.6.1 Monte-Carlo Estimators in Variational Inference

We cannot calculate the expectation in (3.22) in general. We use what is known as Monte-Carlo variational inference (MCVI) with doubly stochastic optimisation to estimate the integral.

There are three main Monte-Carlo estimation techniques, of which we use path-wise gradient estimators [44]. This method has shown success in modern stochastic optimisation [27], [13], [12], offers low variance and is easy to implement. It essentially entails pushing the parameters of the variational distribution into the objective function and then finding the derivatives of the new objective function (this is also called the "push in" method and we discuss how we push in the gradient function in Appendix A). We do this by making use of the "reparametrisation trick" [12] that we discuss in much greater detail in Section 5.4.1 in the context of BNNs.

Essentially we move the expectation from approximate posterior $q(w)$, to auxiliary distribution $p(\epsilon)$. This distribution is transformed using a function $\xi$, that is differentiable in the parameters $\phi$, such that $w = \xi(\epsilon, \phi)$. We then rewrite our expected log likelihood as

$$L_{\mathcal{D}}(\phi) = \mathbb{E}_{q(w)} \log p(y|x, w) \tag{3.23}$$

$$= \mathbb{E}_{p(\epsilon)} \log p(y|x, w = \xi(\epsilon, \phi)) \tag{3.24}$$

where have moved or "pushed in" the dependence of the variational parameters to inside the expectation. The auxiliary distribution $p(\epsilon)$ is easy to sample from and usually a standard Gaussian. Since the expectation is intractable, we approximate this with a Monte-Carlo estimate

given by

$$L_{\mathcal{D}}(\phi) \approx \sum_{n=1}^{N} \log p(y_n|x_n, w = \xi(\epsilon, \phi)). \tag{3.25}$$

This allows us to sample from our distribution $q_\phi(w)$ and efficiently calculate derivatives directly with respect to the variational parameters $\phi$. We are now able to move the gradient operation inside the expectation, leading to less gradient variance, and find the optimal parameters $\phi^*$.

### 3.6.2 Doubly Stochastic Optimisation

We optimise the expectation in (3.25) using stochastic optimisation constituting stochastic variational inference [14]. We take the gradient with respect to the parameters of the distribution $\phi$. We write this as

$$\nabla_\phi L_{\mathcal{D}}(\phi) \approx \sum_{n=1}^{N} \nabla_\phi \log p(y_n|x_n, w = \xi(\epsilon, \phi)) \tag{3.26}$$

Specifically the gradient is with respect to the distribution $q$, so that we optimise the parameters of its distribution to minimise the distance between the approximate posterior and the true posterior.

The second source of stochasticity is introduced by sampling mini batches. Sub-sampling the data provides better generalisation and it is also useful when it is too costly to perform computations over the entire dataset. For a specific mini-batch of size $M$ we estimate the integral as

$$\nabla_\phi L_{\mathcal{D}}(\phi) \approx \frac{N}{M} \sum_{m=1}^{M} \nabla_\phi \log p(y_m|x_m, w = \xi(\epsilon, \phi)) \tag{3.27}$$

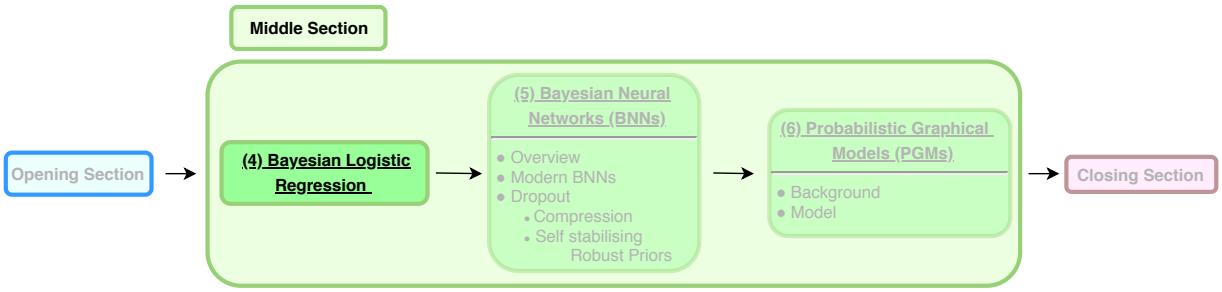
randomly selecting mini-batches. This approximation forms an unbiased stochastic estimator to (3.26). This describes doubly stochastic MCVI and the objective function for BNN training which we return to in a later chapter.

# Chapter 4

# Bayesian Logistic Regression

We briefly discuss Bayesian logistic regression that serves as a baseline with which we can reliably compare more complex approaches. Logistic regression is a well established classifier often used for its interpretability. The Bayesian implementation of logistic regression offers a simple solution to obtaining a model with uncertainty, requiring little computational cost.

This chapter also provides an introduction to applying approximate inference methods to classification models in preparation of our discussion of BNNs. In Bayesian logistic regression, both the posterior inference and prediction is intractable. We follow [31] in discussing the Laplace approximation for approximate inference of the posterior as well as the probit approximation for prediction.




## 4.1 Logistic Regression

First, we briefly introduce logistic regression as a binary classifier. The logistic sigmoid function can be interpreted as the probability of an input vector, $\boldsymbol{x}$, belonging to a class labelled $y = 0$,

of two possible classes with labels 0 or 1, given parameters or weights, $\boldsymbol{w}$, written as

$$P(y|\boldsymbol{w},\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{x}) \tag{4.1}$$

$$= \frac{1}{1+e^{\boldsymbol{w}^T\boldsymbol{x}}}. \tag{4.2}$$

This establishes a classifier with a linear decision boundary at $\boldsymbol{w}^T\boldsymbol{x} = 0$ where the probability of $\boldsymbol{x}$ belonging to either class is 0.5. The classifier assigns a binary outcome to an input according to where it falls in relation to the decision boundary. The likelihood then follows a Bernoulli distribution and, over a set of observations $\mathcal{D} = (\boldsymbol{x}_i, y_i)$ where $n = \{1, ..., N\}$, can be expressed as

$$p(\boldsymbol{y}|\boldsymbol{w},\boldsymbol{x}) = \prod_{i=1}^{N} p(y_i|\boldsymbol{x}_i,\boldsymbol{w}) \tag{4.3}$$

$$= \prod_{i=1}^{N} \left[ \left(\sigma(\boldsymbol{w}^T\boldsymbol{x}_i)\right)^{y_i} + \left(1-\sigma(\boldsymbol{w}^T\boldsymbol{x}_i)\right)^{1-y_i} \right]. \tag{4.4}$$

Training a logistic classifier usually entails taking the log of the likelihood function and setting the derivative, with respect to the parameters $\boldsymbol{w}$, to zero. This defines the objective we minimise given by

$$\nabla_{\boldsymbol{w}} \ln p(\boldsymbol{y}|\boldsymbol{w}) = \sum_{i=1}^{N} (\sigma(\boldsymbol{w}^T\boldsymbol{x}_i) - y_i)\boldsymbol{x}_i = 0. \tag{4.5}$$

## 4.2 Bayesian Logistic Regression

Bayesian logistic regression faces the problem that the posterior is intractable. We introduce a prior probability over $\boldsymbol{w}$ and combine it with the likelihood in (4.3) which gives the posterior as

$$p(\boldsymbol{w}|\mathcal{D}) \propto p(\boldsymbol{w}) \prod_{i=1}^{N} p(y_i|\boldsymbol{x}_i,\boldsymbol{w}). \tag{4.6}$$

Due to the non-linearity introduced by our likelihood in (4.4), inference can no longer be carried out analytically. Thus, we consider approximate approaches to inference.

#### 4.2.0.1 Laplace Approximation

The Laplace approximation capitalises on underlying simplicity of the log likelihood function of logistic regression. The log likelihood is concave that suggests that a Gaussian approximation of the posterior distribution is appropriate. The Laplace approximation exploits this concavity by finding a Gaussian approximate posterior $q(\boldsymbol{w})$, centered at the mode of the true posterior $p(\boldsymbol{w}|\mathcal{D})$. The covariance matrix is calculated based on the Taylor expansion at this mode.

In finding the posterior, we recall the expression for the posterior in (4.6) and we substitute the the likelihood and a Gaussian prior, $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{\mu}_0, \Sigma_0)$. This gives

$$\ln p(\boldsymbol{w}|\mathcal{D}) = -\frac{1}{2}(\boldsymbol{w} - \boldsymbol{\mu}_0)^T \Sigma_0 (\boldsymbol{w} - \boldsymbol{\mu}_0) \tag{4.7}$$

$$+ \sum_{i=1}^{N} \Big( y_i \ln \sigma(\boldsymbol{w}^T \boldsymbol{x}_i) + (1 - y_i) \ln(1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}_i)) \Big) + c. \tag{4.8}$$

We optimise this expression to yield the MAP solution, defining the mean of the approximating distribution $\boldsymbol{w}_{\text{MAP}}$. We obtain the covariance from the inverse of the Hessian (discussed in [31]) given as

$$\Sigma_N^{-1} = -\nabla\nabla \ln p(\boldsymbol{w}|\mathcal{D}) \tag{4.9}$$

$$= \Sigma_0 + \sum_{n=1}^{N} \sigma(\boldsymbol{w}^T \boldsymbol{x_n})(1 - \sigma(\boldsymbol{w}^T \boldsymbol{x_n})) \boldsymbol{x_n} \boldsymbol{x_n}^T. \tag{4.10}$$

The posterior given by the Laplace approximation is then given by

$$q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{w}_{\text{MAP}}, \Sigma_N). \tag{4.11}$$

### 4.2.1 Prediction

Using the posterior we obtained, we can make predictions $y'$ on a new observation $\boldsymbol{x}'$ by integrating out the effect of the parameters given by

$$p(y'|\boldsymbol{x}', \mathcal{D}) = \int p(y'|\boldsymbol{x}', \boldsymbol{w}) p(\boldsymbol{w}|\mathcal{D}) d\boldsymbol{w}. \tag{4.12}$$

Instead of taking a single set of weights this integration considers all possible parameters weighted by their probability defined by the posterior. For logistic regression we make use of the probit approximation as in [31] that provides closed-form approximations to the predictive distribution.

We approximate the predictive distribution as

$$p(y'|\boldsymbol{x}', \mathcal{D}) \approx \sigma(\kappa(\sigma_a^2)\mu_a) \tag{4.13}$$

where

$$\kappa(\sigma^2) = \left(1 + \frac{\pi\sigma^2}{8}\right)^{-\frac{1}{2}}, \tag{4.14}$$

$$\mu_a = \boldsymbol{\mu}^T \boldsymbol{x}', \tag{4.15}$$

$$\sigma_a = \boldsymbol{x}'^T \Sigma \boldsymbol{x}'. \tag{4.16}$$

This gives us a method to efficiently average over the posterior in closed form. We can then accurately estimate uncertainty based on the full posterior and, as shown in Figure 4.1, find certain and uncertain regions relative to the amount of data seen. The predictive distribution under the probit approximation is shown in Figure 4.1. This illustrates how the approximation allows consideration of the full posterior distribution in making predictions.

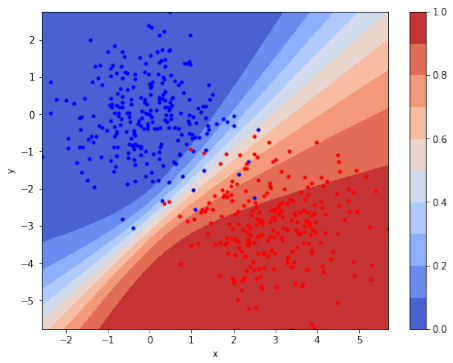



Figure 4.1: Bayesian logistic regression predictions with the probit approximation. This illustrates the effect of a closed-form approximation to incorporate the full posterior. This produces uncertain regions near where the model is yet to see data.

## 4.3 Conclusion

This chapter introduced logistic regression and explicated a Bayesian implementation of the classifier. We introduced practical methods of evaluating intractable integrals with approximations in the Laplace approximation of the posterior distribution as well as the probit approximation for the predictive distribution. These approximations represent closed form solutions, which is possible due to the simplicity of logistic regression. For the remainder of this thesis we consider more complicated models, in neural networks, for which closed-form approximations become

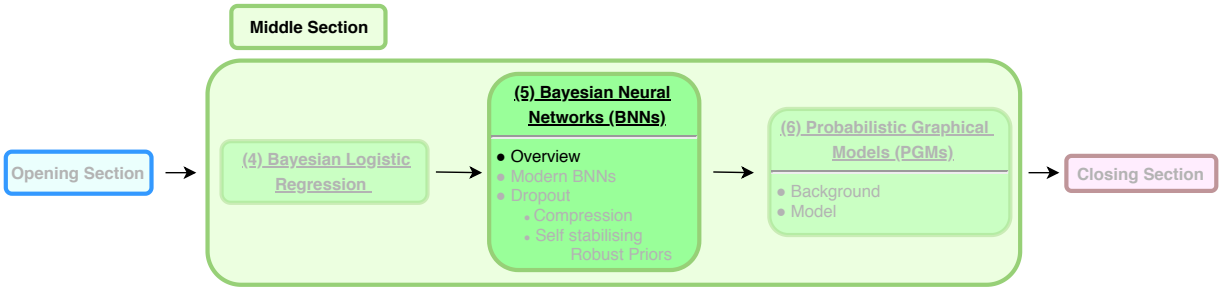
prohibitively complex.

Bayesian logistic regression represents a simple and reliable baseline that capable of estimating uncertainty. Due to its simplicity, it also amounts to a very efficient solution in terms of computation time and memory. Logistic regression, however, is very limited in that it is able to only represent linear decision boundaries between features. In the upcoming chapter we discuss BNNs that are capable of much more complex modelling.

# Chapter 5

# Bayesian Neural Networks

In this chapter we introduce and discuss Bayesian neural networks (BNNs). BNNs have emerged as useful tools for leveraging the modelling capacity of deep learning while offering the flexibility of incorporating prior information and principled parameter estimation of the Bayesian framework. We develop BNNs for two scenarios: (1) modelling data under uncertainty and (2) model compression or sparsification.




We discuss a short history of BNNs followed by a brief review of neural networks. We then discuss recent advances in variational inference, including the reparametrisation trick and the local reparametrisation trick, that have made it possible to train BNNs at scale. Subsequently, using these techniques we introduce and discuss the training procedure of a standard modern BNN with fully factorised Gaussian priors and posteriors. This model is able to accurately estimate uncertainty that we demonstrate with an experiment on MNIST, where we corrupt the inputs. A discussion then follows of work casting stochastic regularisation techniques in neural networks as Bayesian inference that lays the foundation for model compression techniques, signal propagation analysis of BNNs and robust priors. The chapter concludes by discussing prediction with distillation methods.

## 5.1 Motivation

Deep neural networks are powerful tools for modelling highly complex functions, but tend towards overconfidence. As a result, their confidence estimates are not accurate, particularly for inputs that vary from the training data distribution. Furthermore, since these models are able to easily memorise random patterns in data [37], they are prone to overfitting, resulting in poor generalisation. This in effect makes them unsuitable for problems with small datasets such as in satellite image classification. Additionally, this may be troublesome for informing downstream decision tasks in a connected probabilistic system. To address this shortcoming we make use of BNNs. This framework allows us to capture uncertainty in the neural network with a posterior distribution over the parameters. By integrating over this distribution we obtain better uncertainty about the predictions of the model. This also brings about automatic model regularisation and improved ability to learn from small datasets.

A further trait of deep neural networks is that they regularly contain millions of parameters and require considerable computation time and memory. Deep learning is generally implemented on GPUs or power-hungry specialised hardware, making it inappropriate for embedded applications with limited power such as on a satellite. Neural networks, however, are overparametrised and contain redundant parameters and can be pruned significantly without any loss in accuracy. The Bayesian paradigm presents a principled procedure to prune these models. By introducing sparsity-inducing priors, we can easily discard more than 90% of the original weights. By sparsifying the model, we can deploy a powerful yet compact model that avoids unnecessary computation and resources.

## 5.2 Brief Overview of Bayesian Neural Networks

Early work considering the task of approximate inference for BNNs followed [45], using the Laplace approximation to provide a deterministic approximation to the posterior. Much like in our discussion of logistic regression, it is easily obtained by approximating a Gaussian centred at the MAP estimate of the parameters. It requires inverting the Hessian of the log-likelihood that limits its scalability. It is possible to relax some dependencies with approximations to scale the model, but this generally weakens performance. First variational inference methods for neural networks were presented in [46], which later formed the basis for variational inference in modern BNNs. Hamiltonian Monte-Carlo was later developed in [9] as an efficient gradient-based Monte-

Carlo method that was considered the choice inference method for BNNs for many years. These methods are, however, also difficult to scale and, as a sampling method, it may be difficult to assess convergence.

Only recently, with the introduction of the Monte-Carlo variational inference (MCVI) type BNN estimators in [10], have variational methods been made practically scalable allowing the enhanced performance of larger models. With the introduction of stochastic optimisation techniques [14] and improved approximate inference in probabilistic modelling [27], accompanied by practical and efficient estimation tricks in the reparametrisation trick [12], [13], new developments in BNNs were driven and "Bayes by backprop" was introduced [11] which has led to successful large-scale applications.

Probabilistic backpropagation [47] was also proposed to address scalability that uses message passing techniques (widely used in PGMs), specifically expectation propagation [48]. It showed good results and is easily parallelisable but require custom implementations. Other alternative inference methods have since been proposed such as [49], [50]. We implement and build on the large body of work based on MCVI and Bayes by backprop. Our aim is to leverage the modelling power of deep learning, requiring the ability to easily scale inference techniques. MCVI allows us to make use of the scalability, flexibility and applicability of methods in modern Bayesian deep learning techniques such as the reparametrisation trick that simplifies the training of BNNs. This allows training BNNs to become a simple stochastic optimisation task. MCVI also allows training of non-conjugate models [27], allowing versatility and potential for designing many possible combinations of priors and posteriors.

## 5.3 Review of Neural Networks

We briefly review neural networks following **Bishop:2006:PRM:1162264** which serves mainly as an introduction to the notation used in this chapter. The network takes in a set of input vectors or observations $\mathcal{D} = (\boldsymbol{x}_n, \boldsymbol{y}_n)$ where $n = \{1, ..., N\}$ with a parameter matrix $W$ of dimension $D_{l-1} \times D_l$. We compute the output $\boldsymbol{y}(\boldsymbol{x}, W)$ as a function of hidden layers $\boldsymbol{h}$ of which each layer is made of a vector $\boldsymbol{h}^{(l)}$ of hidden unit pre-activations. Individual units, with an input $\boldsymbol{x} = \{h_0^{l-1}, ..., h_{D_{l-1}}^{l-1}\}$, are defined by

$$h_j^{(l)}(\boldsymbol{x}) = \sum_{i=1}^{D_{l-1}} w_{i,j}^{(l)} x_i + b_j^{(l)}. \tag{5.1}$$

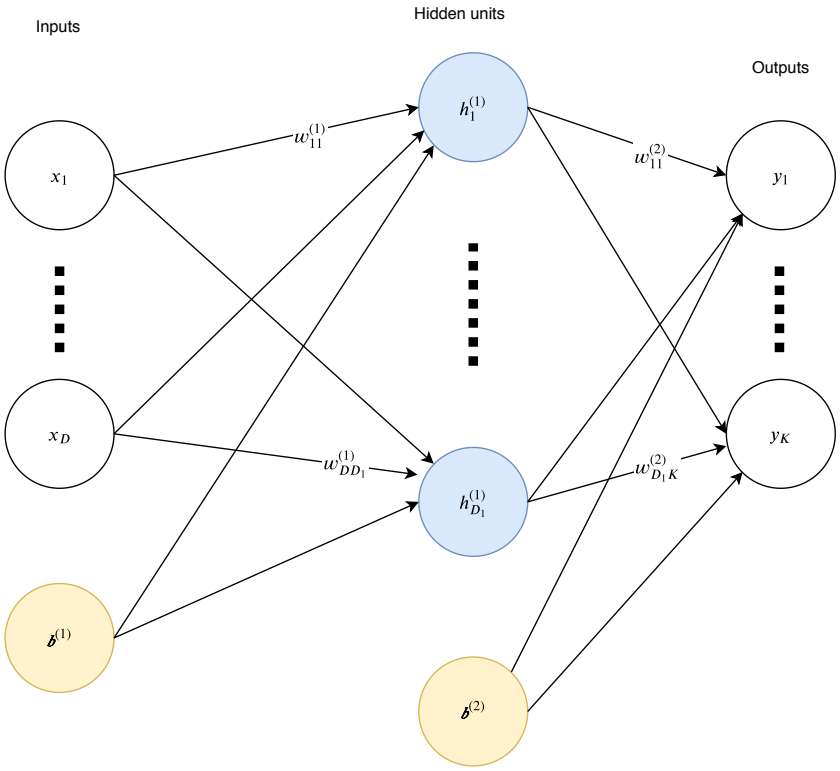



Figure 5.1: Diagram describing a two-layer neural network.

Here, $w_{i,j}^{(l)}$ is the weight on the connection from the input $x_i^{(l)}$ to hidden unit $j$. The biases and hidden unit output feeding into $h_j^{(l)}$ are denoted with $b_j^{(l)}$ and $x_i$ respectively. Each output value is a weighted sum of hidden units with the addition of a bias which then passes through a non-linear activation function denoted by $g(.)$. This gives the value of a hidden unit post activation after a non-linearity is introduced $g(h_j^{(l)}(\boldsymbol{x}))$. An activation function defines the output of a node and can be seen as extracting intermediate features and representing them in the hidden units as "hidden features" that are useful in modelling input-output relationships.

A neural network can be defined as a series of transformations given in (5.1). For classification, we use the softmax function $\sigma(.)$ at the output layer which yields a vector of probabilities. In Figure 5.1 we illustrate a typical neural network with one hidden layer with a weight matrix $W$, of which the output is computed as follows:

$$\boldsymbol{y}(\boldsymbol{x}, W) = \sigma\left(\sum_{j=1}^{D_{l-1}} w_{k,j}^{(2)} g\left(\sum_{i=1}^{D} w_{j,i}^{(1)} x_i + \boldsymbol{b}^{(1)}\right) + \boldsymbol{b}^{(2)}\right). \tag{5.2}$$

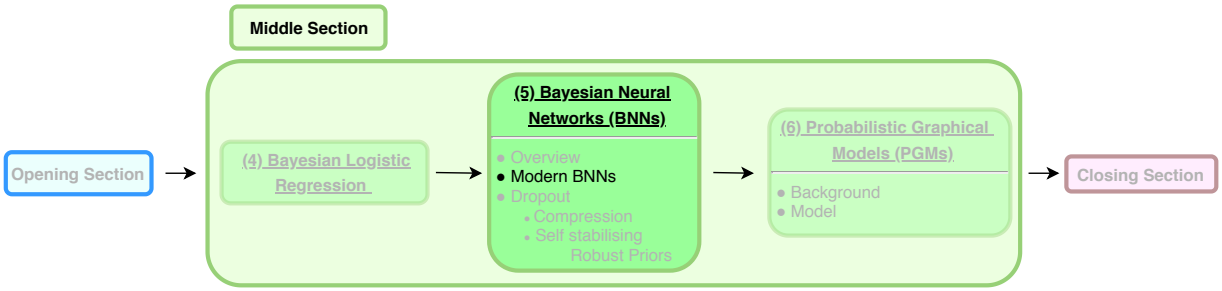
We will omit the superscript $l$ denoting the layer until our discussion of signal propagation in BNNs.

It was shown that a neural network with one hidden layer can approximate any function arbi-

trarily closely, if there are a sufficient number of hidden units [51], [52]. However, modern deep learning makes use of deeper and more complex architectures with great success. This usually includes adding more layers or including a variety of specialised layers, such as convolutional layers. We make use of the ReLU activation function, $g(x) = max(0, x)$, in this thesis.

Training involves using *backpropagation*, a gradient-based algorithm to adjust or tune the parameters, made up of our weights and biases. There are many backpropagation-based optimisation algorithms to minimise the difference between the measured error between the network outputs and targets on the training set. We make use of the Adam optimiser [53].

## 5.4 Variational Inference for Bayesian Neural Networks




We now formulate the objective for training BNNs using MCVI (introduced in Section 3.4) most commonly used in modern BNNs [10], [11]. Considering a set of weight matrices $W = \{W^1, .., W^L\}$ for $L$ layers and weights $w^l_{i,j}$ constituting matrices $W^l \in \mathbb{R}^{D_l \times D_{l-1}}$, we aim to approximate the true intractable posterior distribution $p(W|\mathcal{D})$, over our parameters, with an approximating distribution $q_{\boldsymbol{\phi}}(W)$, from a tractable family parametrised by our variational parameters $\boldsymbol{\phi}$. Our goal is to find the values of our parameters $\phi$ that minimises the ELBO or $\mathcal{L}$. We write this as

$$\mathcal{L}(\boldsymbol{\phi}) = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \quad E_{q_{\boldsymbol{\phi}}(W)}[\log p(\boldsymbol{y}|W, \boldsymbol{x})] - \mathrm{KL}[q_{\boldsymbol{\phi}}(W)||p(W)], \tag{5.3}$$

where the first term is the expected loss or expected likelihood of our data with respect to $q_{\boldsymbol{\phi}}(W)$. This is our data fitting term where we can use any deep neural net or parametric model which maps $\boldsymbol{x}$ to the probability of $\boldsymbol{y}$. For example, with classification it would be the cross entropy loss averaged over all the possible settings of our weights. The expectation of this model is taken with respect to the distribution over the parameters $q_{\boldsymbol{\phi}}(W)$. Calculating the expectation over a deep neural network is intractable so we require some approximation. We can obtain an accurate and efficient stochastic estimate of this expectation with the reparametrisation trick which we

discuss in the next section. The second term is the distance between the variational distribution and the prior distribution which can usually be calculated analytically. This introduces some regularisation as it forces the posterior to be close to the prior.

### 5.4.1 The Reparametrisation Trick

In this section we make use of the reparametrisation trick, a recent innovation in variational inference to allow for tractable and scalable inference and parameter learning in probabilistic models. The reparametrisation trick was introduced in [12] forming a fundamental component of variational auto-encoders. In short, the aim is to, instead of learning a mapping of a compressed representation onto a fixed vector as with auto-encoders, mapping a compressed latent representation onto a *probability distribution* instead. The reparametrisation trick's significance lies in that the reparametrisation allows gradients to pass through stochastic nodes or random variables. This allows training BNNs end to end with gradient optimisers. In BNNs the weights are stochastic and in training we sample one set of weights with each forward pass. In this thesis, all weights are independent distributions that we can reparametrise individually such that we can calculate gradients with respect to their variational parameters. We first discuss the reparametrisation for Gaussians, then in general followed by evaluating its efficiency.

We formulated a probabilistic model in the previous section, thus unlike conventional neural networks, the output is non-deterministic, i.e. the output will differ every forward pass through the network for the same data. We are not optimising the likelihood, we optimise the expected likelihood. Because we are treating the weights as random variables, we optimise the expectation of the likelihood function with respect to $q_\phi(W)$. We cannot optimise $q_\phi(W)$ directly with a gradient estimator because weights are random variables and represent stochastic nodes in the computation graph. The reparametrisation trick, as we will see, allows us to sample only once and get an unbiased estimate of the expectation allowing us to train BNNs with gradient optimisers. It allows us to, instead of optimising the network's weights, rather optimise their variational parameters directly. In other words, the trick allows us to obtain an unbiased differentiable Monte-Carlo estimator of the expected log-likelihood.

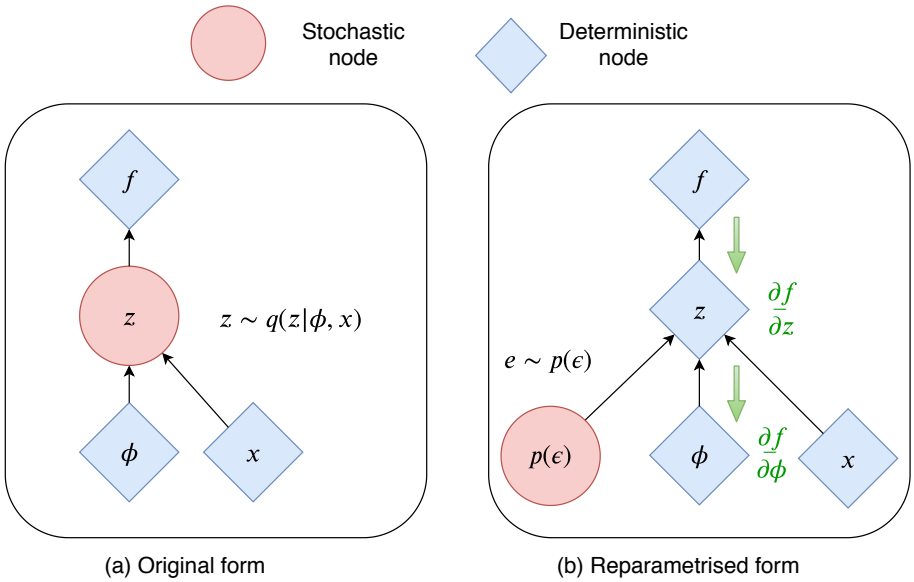



Figure 5.2: The reparametrisation trick in a computation graph. In the first image we have a stochastic node that is blocking all of our gradients because we cannot backpropagate through a stochastic node. In the second image we reparametrise to move the stochastic part outside the path from which we wish to calculate the gradient. This allows us to use backpropagation to optimise the variational parameters of our weight distribution.

#### 5.4.1.1 Gaussian Example

In the more general context of the reparametrisation trick we consider a random node $z$ (representing a stochastic variable such as a weight $w_{i,j}$ in BNN). Let us look at the computation graph in Figure 5.2 (a). Considering a Gaussian distribution, that we want to sample from on the forward pass, but we cannot calculate the gradient of the sampling operation on the backward pass. To overcome this obstacle, we replace the random node with a reparametrised version as seen in the computation graph in Figure 5.2 (b). What this reparametrisation has done is move the source of noise outside of the main flow of the network.

For a Gaussian random variable $z = \mathcal{N}(\mu, \sigma^2)$, we can write the reparametrised node or random variable that we are sampling as

$$z = \mu + \sigma\epsilon \tag{5.4}$$

where the distribution $p(\epsilon)$ is a standard Gaussian $\epsilon \sim \mathcal{N}(0, 1)$. This is equivalent to sampling from $\mathcal{N}(\mu, \sigma^2)$. Here, $\mu$ and $\sigma$ represent parameters of our variational distribution that we wish to learn with backpropagation and the variable $\epsilon$ is a fixed stochastic node, we do not need to learn anything about it as it does not depend on any parameters. In a forward pass we draw a sample from $p(\epsilon)$, multiply with our variance, add our mean and we have a sample of our random

node. On the backward pass it is now possible to calculate the partial derivatives with respect to $\mu$ and $\sigma$, making it possible to optimise these parameters.

#### 5.4.1.2 General Form

We introduced a way to get samples from a Gaussian distribution and its derivatives with respect to the parameters of the distribution, but it is not limited to Gaussians. Almost every continuous distribution can be decomposed to a different form where we sample from a simpler distribution $p(\epsilon)$. We can employ the reparametrisation trick if there exists a deterministic function that will transform samples from $\epsilon$ into samples of $z$. In general, for an approximate posterior $q_{\phi_{i,j}}(w_{i,j})$ we can reparametrise the random variable $w_{i,j} \sim q_{\phi_{i,j}}(w_{i,j})$ using a differentiable transformation $g(\epsilon, \phi_{i,j})$. We write this as

$$w = g(\epsilon; \phi_{i,j}) \quad \text{where} \quad \epsilon \sim p(\epsilon). \tag{5.5}$$

Our random variable $w_{i,j}$, becomes a function of $\phi_{i,j}$ and $\epsilon$ where $p(\epsilon)$ is distributed according to a distribution that does not depend on any parameters and $g$ is the function that transforms samples while making it possible to maintain a path or flow of partial derivative for backpropagation.

It is not always possible to reparametrise for any distribution, but for our interests, it is always possible to do so with Gaussians. We can always express an arbitrary normal distribution with a standard normal distribution, as it is just a linear operation to shift the mean and scale the variance. Then the gradient with respect to the mean and variance can easily be computed or gradients are able to flow through the computation graph. Instead of taking the derivative with respect to sampled weights $w$ from $q_\phi(w)$, where gradients will vary with each sample, we can say $w$ is a function that takes parameter $\phi$ and some noise, $p(\epsilon)$, and we can calculate gradients with respect to $\phi$. This results in much lower gradient variance that we discuss next.

#### 5.4.1.3 Examining the Variance

In stochastic variational inference, a major goal is to reduce variance and thus increase the efficiency of the estimator. We can then obtain more accurate inference with fewer samples or scale to larger models and bigger datasets. We now examine how the reparametrisation trick helps in efficiently calculating gradients with respect to parameters.

For a straightforward scalar objective that resembles the ELBO (in the sense of calculating the gradient of a variational parameter in expectation), let us consider the task of finding a parameter $\mu$ with the objective

$$\underset{\mu}{\operatorname{argmin}} \mathbb{E}_{q_\mu(z)}[z] \tag{5.6}$$

where we assume $q_\mu$ to be normally distributed $q_\mu = \mathcal{N}(\mu, \sigma^2)$ and our aim is to estimate the parameter $\mu$. In optimising for $\mu$ we consider the quantity $\nabla_\mu \mathbb{E}_{q_\mu(z)}[z]$. Without the use of the reparametrisation trick we can calculate the quantity as follows

$$\nabla_\mu \mathbb{E}_{q_\mu(z)}[z] = \int z \nabla_\mu q_\mu(z) dz \tag{5.7}$$

$$\nabla_\mu \mathbb{E}_{q_\mu(z)}[z] = \int z \frac{q_\mu(z)}{q_\mu(z)} \nabla_\mu q_\mu(z) dz \tag{5.8}$$

then using the log derivative trick gives

$$= \int z q_\mu(z) \ln q_\mu(z) dz \tag{5.9}$$

$$= \mathbb{E}_{q_\mu(z)}[z \ln q_\mu(z)]. \tag{5.10}$$

For the Gaussian $q_\mu = \mathcal{N}(\mu, \sigma^2)$ this becomes

$$\nabla_\mu \mathbb{E}_{q_\mu(z)}[z] = \mathbb{E}_{q_\mu(z)}\left[ z \left( -\frac{1}{2\sigma^2}(z-\mu)^2 - \frac{1}{2} \ln 2\pi\sigma^2 \right) \right]. \tag{5.11}$$

We can then estimate this drawing $S$ samples of $z$ as $z_s$. Ignoring the terms that do not depend on $\mu$ gives

$$\approx \frac{1}{S} \sum_s z_s \left( -\frac{(z_s - \mu)^2}{2\sigma^2} \right). \tag{5.12}$$

Our estimate of the gradient will vary and introduce a sampling error for each different occasion that we draw samples. We may require many samples as this exhibits a large amount of variance and computing this multiple times over a dataset of samples $s$ will yield very different answers.

For the reparametrised version we can rewrite the expectation such that the distribution with respect to which we take the gradient is independent of the parameter $\mu$. We can directly calculate the gradient and it is not necessary to introduce the ratio $\frac{q_\mu(z)}{q_\mu(z)}$ as in (5.6). Compared to the previous approach, by applying the reparametrisation trick, we exchange the distribution

with respect to which we are taking the expectation with $p(\epsilon)$ as follows

$$\nabla_\mu \mathbb{E}_{q_\mu(z)}[z] = \nabla_\mu \mathbb{E}_{p(\epsilon)}[(\mu + \sigma\epsilon)] \quad \text{where} \quad \epsilon \sim \mathcal{N}(0,1) \tag{5.13}$$

which allows us to push in the gradient operator since it is independent of $p(\epsilon)$. This gives

$$\nabla_\mu \mathbb{E}_{q_\mu(z)}[z] = \mathbb{E}_{p(\epsilon)}[\nabla_\mu(\mu + \sigma\epsilon)] \tag{5.14}$$

that for $S$ samples, the gradient with respect to $\mu$ of the expression $\mathbb{E}_{q(z)}[z]$ gives

$$= \frac{1}{S}\sum_s 1 \tag{5.15}$$

$$= 1. \tag{5.16}$$

We see that in estimating the quantity $\nabla_\mu \mathbb{E}_{q(z)}[z]$ there is no variance when using the reparametrisation trick and the gradient is independent of the parameter $\mu$. This reduction of variance is a huge benefit as each estimate of the gradient is closer to the true expectation. With the reparametrisation trick, stochastic optimisation is more feasible as gradient estimates are accurate despite being with respect to stochastic variables. This allows us to train significantly more efficiently.

### 5.4.2 Monte-Carlo Estimator for Bayesian Neural Networks

Let us examine the reparametrisation trick in the context of optimising BNNs. We recall the need to estimate the first term of (3.27) that is an expectation taken with respect to a distribution that depends on parameters $\boldsymbol{\phi}$ or

$$E_{q_{\boldsymbol{\phi}}(W)}[\log p(\boldsymbol{y}|\boldsymbol{x}, W)]. \tag{5.17}$$

The goal is to optimise this expectation with respect to the variational parameters $\phi$. Taking the gradient gives

$$\nabla_{\boldsymbol{\phi}} \int q_{\boldsymbol{\phi}}(W) \log p(\boldsymbol{y}|\boldsymbol{x}, W) dW. \tag{5.18}$$

In order to take the derivative we require an estimate of $\int q_{\boldsymbol{\phi}}(W) \log p(y|x, W) dW$. To obtain an unbiased estimation for this expression we take a Monte-Carlo estimate given by

$$\approx \frac{1}{n} \sum_{i=1}^{n} \log p(\boldsymbol{y}|\boldsymbol{x}, W_i) \quad \text{where} \quad W_i \sim q_{\boldsymbol{\phi}}(W) \tag{5.19}$$

that will have good accuracy for sufficient $n$. The problem, however, is that we are sampling many weights to get a Monte-Carlo estimate and taking derivatives with respect to this estimate. We are trying to optimise $\boldsymbol{\phi}$ but we can only calculate gradients with respect to samples of $W$ and use that to update $\boldsymbol{\phi}$. This has very high variance as sampling first then differentiating ignores that the distribution of the weights $q_{\boldsymbol{\phi}}(W)$ depends on $\boldsymbol{\phi}$. Derivatives will vary greatly for each set of sampled weights. For example, if we take a sample of one set of weights and take the gradient, then we take another sample of weights and take the gradient, the two gradients may point in vastly different directions.

We want to move the dependence of $\boldsymbol{\phi}$ from the probability distribution into the integral function. We can also think of this as trying to move the derivative operator to inside the integral to avoid sampling *then* differentiating. So we reparametrise $q_{\boldsymbol{\phi}}(W)$ to give

$$\nabla_{\boldsymbol{\phi}} \int p(\epsilon) \log p(\boldsymbol{y}|\boldsymbol{x}, f(\epsilon, \boldsymbol{\phi})) d\epsilon \tag{5.20}$$

This allows us to move differentiation inside the expectation giving us

$$\int p(\epsilon) \nabla_{\boldsymbol{\phi}} \log p(\boldsymbol{y}|\boldsymbol{x}, f(\epsilon, \boldsymbol{\phi})) d\epsilon. \tag{5.21}$$

We can now rewrite the Monte-Carlo integration to compute the expectation as

$$\approx \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\phi}} \log p(\boldsymbol{y}|\boldsymbol{x}, f(\epsilon_i, \boldsymbol{\phi})) \quad \text{where} \quad \epsilon_i \sim p(\epsilon). \tag{5.22}$$

Here we get an efficient unbiased estimator of the expectation of our neural network and we can calculate the gradient of our variational parameters $\boldsymbol{\phi}$ directly. We can move the derivative operator into our Monte-Carlo estimate and remove sampling variance. In essence our estimate of $\int q_{\boldsymbol{\phi}}(W) \log p(\boldsymbol{y}|\boldsymbol{x}, W) dW$, of which we require to take the derivative, we write as

$$\nabla_{\boldsymbol{\phi}} \int q_{\boldsymbol{\phi}}(W) \log p(\boldsymbol{y}|\boldsymbol{x}, W) dW \approx \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\phi}} \log p(\boldsymbol{y}|\boldsymbol{x}, f(\epsilon_i, \boldsymbol{\phi})). \tag{5.23}$$

This is very efficient as we usually need just one sample per iteration. By doing this it is

guaranteed that our stochastic gradient estimator will have the least possible variance.

#### 5.4.2.1 Conclusion

In this section we saw that the reparametrisation trick is a tool by which we substitute random variables of some known distribution with a deterministic transformation of another random variable. This greatly reduces the variance in our gradient and we can calculate derivatives with respect to our variational parameters directly. In a BNN we use the reparametrisation trick for all of our weights individually such that we can learn the posterior variational parameters with gradient descent. What we observe in practice, however, is that while this dramatically reduces variance, our estimator still struggles with high variance in larger models, we thus implement the local reparametrisation trick which we will discuss next.

### 5.4.3 Local Reparametrisation Trick

The local reparametrisation trick was introduced in [20], which builds on the reparametrisation trick, to further reduce variance in BNNs that use batches during training. The core idea is instead of sampling weights and multiplying them with the inputs to get a pre-activation matrix, we calculate the distribution of the pre-activation matrix analytically and sample from this directly. We first discuss its implementation, then how it affects efficiency during training.

Given that our weights are drawn from independent Gaussian distributions with means $\mu_{i,j}$, and variance $\sigma^2_{i,j}$, we write individual weights of weight matrix $W$ as $w_{i,j} \sim \mathcal{N}(\mu_{i,j}, \sigma^2_{i,j})$ . We define a new pre-activation matrix, $B$, after the weights have been multiplied as

$$B = XW, \tag{5.24}$$

where $X$ is the input batch. Matrix $B$ is still Gaussian as it is only shifted and scaled by the input batch. We calculate the mean and variance of the elements $b_{i,j}$ of this matrix to sample from by taking the moments. At an arbitrary layer $l$ of width $D_l$ with an input batch of size $M$ yielding $\boldsymbol{x}$ of size $M \times D_{l-1}$ and weight matrix $D_{l-1} \times D_l$ with a layer of size $D_l$, the moments

yields

$$\mathbb{E}[b_{i,j}] = \gamma_{m,j} \qquad \text{where} \qquad \gamma_{m,j} = \sum_{i=1}^{D_{l-1}} x_{m,i}\mu_{i,j}, \tag{5.25}$$

$$\text{Var}[b_{i,j}] = \delta_{m,j} \qquad \text{where} \qquad \delta_{m,j} = \sum_{i=1}^{D_{l-1}} x_{m,i}^2\sigma_{i,j}^2, \tag{5.26}$$

where $\boldsymbol{x}_m$ is the $m$-th element of the input batch $X$. Then we can define the distribution of $b_{i,j}$ as

$$b_{i,j} \sim \mathcal{N}(\gamma_{m,j}, \delta_{m,j}). \tag{5.27}$$

This is the reparametrised distribution at a pre-activation from which we sample with the local reparametrisation trick.

Returning to matrix notation, we consider $\mu_{i,j} \in \boldsymbol{\theta}$ and $\sigma_{i,j}^2 \in \Sigma$ defining matrices of the independent weights parameters. To simplify notation, we regard the square root in the following equation as element wise. We can then sample a matrix of pre-activations with

$$B = X \odot \boldsymbol{\theta} + \sqrt{X \odot X \odot \Sigma \odot \Sigma} \odot E \tag{5.28}$$

and $\odot$ is the element-wise product. Here, $E$ is a $M \times D_l$ matrix with elements $\epsilon_{m,j} \sim \mathcal{N}(0, 1)$. We thus only require only $M \times D_l$ samples, yielding $D_{l-1}$ fold savings. We sample directly from this distribution with the reparametrisation trick rather than the weight distribution. This defines the core computation of a Bayesian layer in modern Bayesian deep learning.

Sampling directly from this distribution results in independent weight samples for each data point, whereas if we sample from $q_{\boldsymbol{\phi}}(W)$ we only have one weight sample per mini batch. Previously, we had just one sample of weights per batch which all used the same $\epsilon$. The noise terms were shared across a mini batch in the weight sample which cause predictions to become correlated and therefore gradients to vary. Moving the noise injection from the parameters to the pre-activations gives us more independent parameter samples with each forward pass at the same computational cost. Effectively, we translate the global noise in the weights into local noise that is not correlated between data points in the mini batch.

A single noise sample per mini batch causes predictions to tend to deviate from their expected values as predictions become correlated. Let us examine $L_i$, the contribution of the $i$-th data point in the mini batch $M$ given by $\log p(\boldsymbol{y}_i|\boldsymbol{x}_i, W = f(\boldsymbol{\phi}, \epsilon))$. We estimate $L_{\mathcal{D}}(\phi)$ of the entire dataset

with a Monte-Carlo estimator $L_{\mathcal{D}}(\phi) \approx \hat{L} = \frac{N}{M}\sum_{i=1}^{M} L_i$. Here the variances and covariances are only determined by the mini batch, $x_i$ and $y_i$, and $\epsilon$ which is a shared noise sample, i.e.

$$\text{Var}[L_i] = \text{Var}[\log p(\boldsymbol{y}_i|\boldsymbol{x}_i, W = f(\boldsymbol{\phi}, \epsilon))]. \tag{5.29}$$

We can then compute the empirical variance of (5.29) over a mini batch by taking the second moment, the sum of the variances plus the sum of the covariances. Because different estimates are identically distributed, they have the same variances and covariances, we can write the variance of the Monte-Carlo estimator for a mini batch as

$$\text{Var}[\hat{L}] = \frac{N^2}{M^2}\left(\sum_{i=1}^{M}\text{Var}[L_i] + 2\sum_{i=1}^{M}\sum_{j=1}^{M}\text{Cov}[L_i, L_j]\right) \tag{5.30}$$

$$= N^2\left(\frac{1}{M}\text{Var}[L_i] + \frac{M-1}{M}\text{Cov}[L_i, L_j]\right). \tag{5.31}$$

The key observation here is that when our mini batch size $M$ is not equal to 1, the covariance dominates the variance. This means for even moderately large mini batches there will be high variance in our estimator and thus large gradient variance.

As discussed in [20], the weights $W$, and noise $\epsilon$, influence the expected log likelihood $L$, only through the neuron pre-activations $B$. If we can therefore sample the random pre-activations $B$ directly, without sampling $W$ or $\epsilon$, we obtain an estimator where $\text{Cov}[L_i, L_j] = 0$ and at a much lower cost than drawing separate samples for weight matrices for each training example.

In summary, we saw that the local reparametrisation trick leads to less gradient variance because it draws independent noise samples for each weight entry in the mini-batch. This reparametrisation simply requires analytically calculating the mean and variance of the pre-activations and sampling from that distribution. Before we discuss applying thes reparametrisation tricks to a BNN and training these models we briefly cover prediction in the next section.

## 5.5 Prediction

Our introduction of BNNs ends with a consideration of prediction methods. To predict the label $\boldsymbol{y}$ from an input $\boldsymbol{x}$, we have to average over the posterior $q_\phi(W)$. We can make predictions on

new data given a new observation $x$ with

$$p(\boldsymbol{y}|\boldsymbol{x}) = \int p(\boldsymbol{y}|\boldsymbol{x}, W) q_{\boldsymbol{\phi}}(W) dW \tag{5.32}$$

and average the outputs. This is intractable and in practice we take many samples as an estimate. We sample a set of weights and do a forward pass, yielding a sample prediction for a given input and repeat this many times and average over the predictions. We call this test-time averaging. The accuracy saturates relatively quickly and may require very few samples; usually 20 samples are adequate to acquire reliable estimates.

### 5.5.1 Distillation

An alternative is to distill the ensemble behaviour into a different deterministic neural network [26]. In the case when taking many samples or requiring many forward passes of the network is too computationally expensive or slow, we may train a neural network to mimic the behaviour of a BNN.

We train a neural network, which is referred to as the student network, to minimise the distance between predictions of the ensemble, or the teacher network. Over a dataset with $X = [\boldsymbol{x_0}, ..., \boldsymbol{x_N}]$ and $Y = [\boldsymbol{y_0}, ..., \boldsymbol{y_N}]$ we optimise

$$L(W_{\text{student}}) = \mathbb{E}_{q_\phi(W_{\text{teacher}})} \Big[ \mathcal{H} \left[ (p(Y|X, W_{\text{student}}), p(Y|X, W_{\text{teacher}})) \right] \Big] \tag{5.33}$$

where $\mathcal{H}$ represents the cross entropy between the outputs of the student and the teacher network. We propagate the samples through both then use the cross entropy to update the student. This typically does not do as well as an ensemble, but does better than a single network.

We first train the *teacher* BNN and obtain an approximate posterior $q_{\boldsymbol{\phi}}(W)$. To train the *student* deterministic network, we sample a minibatch and sample the predictions of the teacher. The softmax output of the teacher is used as soft labels labels for the student. The student network minimises the distance between the output of the teacher network and its outputs start to resemble the teacher. Next we will apply the methods we have discussed, applying both reparametrisation tricks to a BNN with fully factorised Gaussian posteriors.

## 5.6 Bayesian Neural Networks with Fully Factorised Gaussian Priors and Posteriors

We now discuss the training procedure of a BNN with fully factorised Gaussian priors and posteriors, similar to [10] and [11], as we will be using for many of our experiments. This follows [27] in that inference is non-conjugate. Fully factorised Gaussian posteriors are robust to overfitting and provide accurate uncertainty estimates. For a particular layer $W^l$ for all layers $l \in 1, ..., L$ we approximate the posterior distribution over our weights to be one-dimensional Gaussians, yielding

$$q_{\boldsymbol{\phi}}(W^l) = \prod_{\forall i \in W} \mathcal{N}\left(w_i^l | \mu_i^l, (\sigma_i^l)^2\right). \tag{5.34}$$

For brevity we write $W$ as the set of weights containing all layers $\{W^1, ..., W^L\}$ We apply the local reparametrisation trick, where we sample from the pre-activation matrix $B$ before the non-linearities, given in (5.28) as our expression for our weights. This ensures that there is no correlation from a shared noise sample from our sampled weights. Using the local reparametrisation trick, using (5.25) and (5.26), to obtain a weight sample at the pre-activation we compute

$$b_{i,j} = \gamma_{m,i} + \sqrt{\delta_{m,j}}\epsilon \quad \text{where} \quad \epsilon \sim \mathcal{N}(0, 1). \tag{5.35}$$

We choose our prior to be fully factorised Gaussians with a prior mean $\mu_{\text{prior}}$ and variance $\sigma^2_{\text{prior}}$ given by $p(w_{i,j}) = \mathcal{N}(\mu_{\text{prior}}, \sigma^2_{\text{prior}})$. The objective function is then derived from our MCVI objective in (5.3), which we recall as

$$\mathcal{L}(\phi) = \underset{\phi}{\text{argmin}} \quad E_{q_\phi(W)} \log p(y|W, x) - \text{KL}[q_\phi(W)||p(W)]. \tag{5.36}$$

Replacing the first term with the reparemtrised version and the KL term as the summation of the KL distance, in closed form for Gaussians, between the independent posteriors and the prior, the objective is then given by

$$\mathcal{L}(\theta, \Sigma) = \mathbb{E}_{p(E)}\left[\log p(y|x, W = \boldsymbol{\theta} + \Sigma \odot E) + \sum_{\forall i \in W} \log \frac{\sigma_i^2}{\sigma^2_{\text{prior}}} + \frac{\mu_i^2 - \mu^2_{\text{prior}} + \sigma_i^2 - \sigma^2_{\text{prior}}}{2\sigma^2_{\text{prior}}}\right]. \tag{5.37}$$

During training, when using reparametrisation tricks, one forward pass with one sample per

weight is sufficient to estimate the first term, $p(y|x, W = \boldsymbol{\theta} + \Sigma \odot E)$. We optimise this with respect to $\boldsymbol{\theta}$ and $\Sigma$. The first term represents an MCVI estimator with the reparametrisation trick. The second part is the KL distance between two Gaussians that we have calculated in closed form as a function of our prior and variational parameters. This adds some regularisation to the mean vector to keep it close to $\mu_{\text{prior}}$ and also some regularisation to the variance that encourages posterior variances close to prior variances. Typically it is better to use small values for $\sigma_{\text{prior}}$ and initialise $\sigma$ to start small for stability.

From here it is simple gradient-based optimisation to find $\boldsymbol{\theta}$ and $\Sigma$. For simplicity we treat non-expressive parameters such as biases as deterministic parameters. It is possible to place priors over them by augmenting the input at each layer with an additional column of ones and adding an extra column of weights to treat biases as random variables. However, these parameters are not likely to overfit and this brings a degree of complexity introducing even more randomness in a forward pass of a network when using stochastic optimisation methods.

We implement a BNN of this kind trained on MNIST and demonstrate an experiment showcasing uncertainty estimation using this network and noisy inputs in Figure 5.3. Using entropy of the softmax output as a measure of confidence, the confidence for predictions on unseen or majorly distorted digits is drastically reduced. This demonstrates how BNNs naturally account for uncertainty when faced with observation noise. Inputs that no longer resemble a handwritten digit is assigned high entropy reflecting uncertainty.

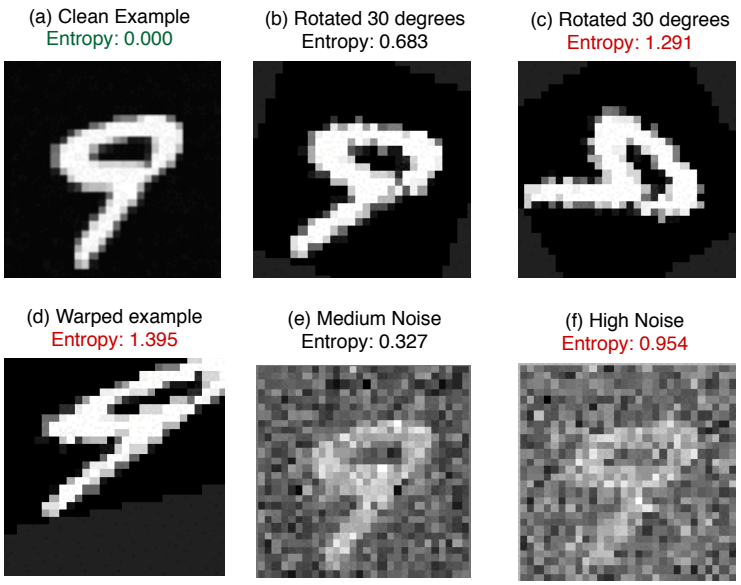
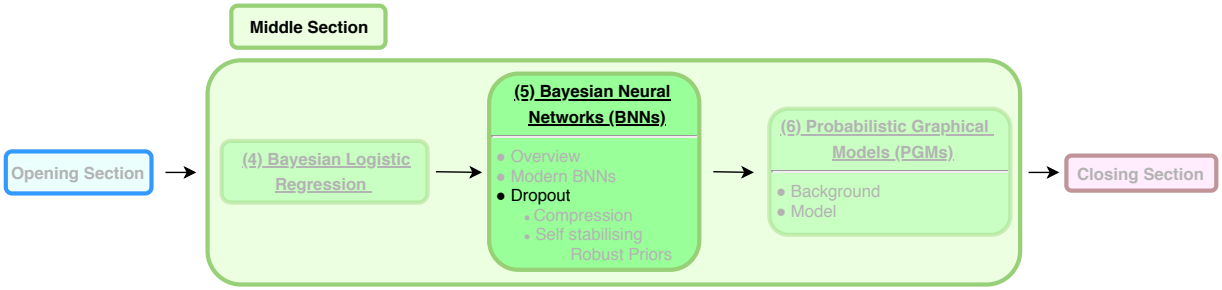



Figure 5.3: We show BNN predictions on noisy MNIST inputs. We measure the entropy as a representation of uncertainty. We distort a clean example given in (a) with rotation, warping and adding noise. We see that entropy rises significantly when the inputs no longer resemble a handwritten digit.

## 5.7 Dropout as Bayesian Inference




Now that we have introduced BNNs, we discuss recent work [28], [20] relating stochastic regularisation techniques to Bayesian inference. We introduce MC dropout that relates binary dropout to Bayesian inference in Gaussian processes that, despite drawing some criticisms, we find be a practical method for uncertainty estimation. We also discuss variational dropout which precisely relates Gaussian dropout to training BNNs. This interpretation sets the foundation for the following sections.

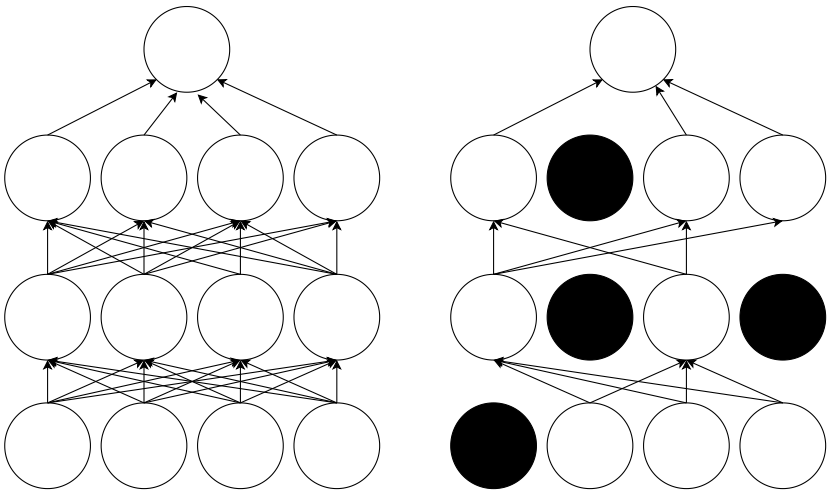
### 5.7.1 MC Dropout

Here we discuss work [28] casting dropout training in deep neural networks as approximate Bayesian inference in deep Gaussian processes [54]. This method is referred to as Monte-Carlo

dropout (MC dropout) and makes use of dropout during training *as well as* test time. Dropout was introduced as a regularisation technique used to avoid overfitting in neural networks [55], [38]. We do not fully explore the theoretical link between Gaussian processes and dropout, but discuss the interpretation suggesting that dropout approximately integrates over a model with distributions over its weights. This method is widely used in uncertainty estimators in the deep learning community but has attracted some criticisms. We discuss these criticisms but find it to be practical and efficient.

#### 5.7.1.1 Dropout

Dropout is a stochastic regularisation technique introduced in [38], [55] that ignore or "drop out" random units or neurons or in a neural network. At each training step we temporarily set random sets of neurons to zero and these are not considered during a particular forward or backward pass. As seen in Figure 5.4 this results in a sparse net at each training step. The result is that we train many of these sparse networks and approximately combine many different neural network architectures.



(a) Standard neural network.

(b) Neural network applying dropout.

Figure 5.4: Illustration of dropout randomly switching off nodes.

Performing dropout essentially adds Bernoulli noise to the inputs of hidden layers of a normal feed-forward neural network. We multiply the inputs to a hidden layer $X$ with a random noise matrix $R$ where elements $r_{i,j}$ are drawn from a Bernoulli distribution with a dropout probability $p$ given by $r_{i,j} \sim \text{Bernoulli}(p)$ . Then given weight matrices $W$ and bias vector $\boldsymbol{b}$, the output of

a neural network layer becomes

$$\boldsymbol{h} = g(W(X \odot R) + \boldsymbol{b}) \tag{5.38}$$

and only non-zero weights are updated.

In each training step we train a sparse model that causes the network not to rely too heavily on one set of node features. It forces the network to learn various discriminating features and independent features. This has proven to be very successful in regularising neural networks and avoid overfitting.

#### 5.7.1.2 Dropout as Approximate Bayesian Inference

In [28], it was shown that a neural network with dropout applied is equivalent to an approximation to a deep Gaussian process [54]. This allows a method of estimating uncertainty using dropout with the MC dropout procedure. With the derivations being non-trivial, we only discuss an example of how we may specify our neural network as to gain some intuition of how dropout is linked to probabilistic modelling. We define a new weight matrix $\hat{W}$, as

$$\hat{W} = W \odot R \quad \text{where} \quad r_{i,j} \sim \text{Bernoulli}(p) \tag{5.39}$$

$$\tag{5.40}$$

where we inject Bernoulli noise. If we maintain the noise in the prediction, we can interpret $\hat{W}$ as a random variable similar to BNNs. We can envisage $\hat{W}$ as sampled from a probability distribution $q(W)$, construed as a posterior distribution over the weights and similar to training an ensemble of networks. From this point of view we are not training a fixed set of weights $W$ but rather having weights as random variables or maintaining a distribution of weights $q(W)$. This suggests that sampling or injecting noise approximately integrates over model parameters instead of choosing one set of weights. Notice that the regularising effect of noise regularisation is naturally encompassed by a Bayesian model by integrating over uncertainty.

To apply MC dropout in practice, we compute stochastic forward passes at test time to average over the random units or use sample predictions to find the moments. The method differs from training a neural network only by using dropout in the network at test time. We repeat this $T$ times with dropping out different units every forward pass and aggregate the predictions. We

calculate an estimate mean, $\boldsymbol{y}^*$, of the prediction samples $\hat{\boldsymbol{y}}_t$ by taking the first moment,

$$\mathbb{E}[\boldsymbol{y}^*] \approx \frac{1}{T}\sum_{t=1}^{T}\hat{\boldsymbol{y}}_t. \tag{5.41}$$

This represents an estimate with $T$ samples of approximate predictions, $\hat{\boldsymbol{y}}_t$, sampled from a network with dropout in the forward pass. As $T$ goes to infinity, we should get the true expectation. Similarly, we use the second moment for the outcomes

$$\mathrm{Var}[\boldsymbol{y}^*] \approx \frac{1}{T}\left(\sum_{t=1}^{T}\hat{\boldsymbol{y}}_t^T\hat{\boldsymbol{y}} - \mathbb{E}[\boldsymbol{y}^*]^T\,\mathbb{E}[\boldsymbol{y}^*]\right) \tag{5.42}$$

that represents our uncertainty for regression (note this is a simplified version of the original variance proposed in [28]). For classification we average the outputs and measure the entropy of the softmax output as our uncertainty.

A direct result of this that we can apply dropout at inference time and easily obtain an uncertainty estimator. With MC dropout we acquire practical uncertainty estimates while using well-established deep learning methods without needing to change the models or optimisation. We alleviate the problem of representing uncertainty without incurring a great deal more of computational complexity or sacrificing predictive performance.

Some concerns have been raised about this method, particularly that the model's uncertainty is not well calibrated [56]. The predictive variance seems to be too heavily influenced by the dropout probability and [57] showed that for a small neural network, the predictive uncertainty does not decrease with more data that brings the approximation into question. However, due to its ease of implementation, it is still widely used and in our own experiments we are able to produce satisfactory results.

### 5.7.2 Variational Dropout

Here we discuss variational dropout [20], which interprets Gaussian dropout, multiplicative noise injection, as variational inference. This interpretation underpins the development of techniques for model compression as well as robust Bayesian deep learning. We begin with interpreting the noise injection as sampling from Gaussian weights. This allows us to write the objective function of Gaussian dropout in a similar way to MCVI to relate the objective of Gaussian dropout training to the objective of variational inference. This then allows us to work backwards

to relate a specific prior that is consistent with Gaussian dropout and Bayesian inference.

Gaussian dropout is a stochastic regularisation method [58], where, instead of dropping out units with Bernoulli noise, Gaussian noise is injected by means of multiplication. The noise distribution does not change the mean value of the weights, it only slightly corrupts the weights with noise. The Gaussian noise distribution, given by $\epsilon$, with noise parameter $\alpha$, is given by

$$\epsilon \sim \mathcal{N}(1, \alpha) \tag{5.43}$$

that is then multiplied with a weight of a neural network. This is also called fast dropout, because relative to binary dropout this results in faster convergence. This is because there is less stochasticity in the computation of the gradient when units are slightly corrupted by noise rather than being dropped out. Gaussian dropout was shown to be related to binary dropout [58], where the parameters of the different noise distributions are related by

$$\alpha = \frac{p}{1-p}. \tag{5.44}$$

Variational dropout extends Gaussian dropout by defining an approximate posterior distribution $q(W|\boldsymbol{\theta}, \alpha)$ where $\boldsymbol{\theta}$ is construed as either the weight matrix of a normal neural network with Gaussian dropout, or the set of mean parameters of Gaussian posteriors. To be consistent with dropout, one scalar value, $\alpha$, defines a shared variance. Since Gaussian dropout does not affect the expected value of a weight, the sampling distribution for a particular weight $\hat{w}_{i,j} \in W$, on which Gaussian dropout of rate $\alpha$, can be written as

$$\hat{w}_{i,j} \sim \mathcal{N}(w_{i,j}, \alpha w_{i,j}^2) \tag{5.45}$$

or similarly the posterior distribution as

$$q(w_{i,j}) = \mathcal{N}(\theta_{i,j}, \alpha\theta_{i,j}^2) \tag{5.46}$$

where $w_{i,j}$, of a deterministic network, is analogous to $\theta_{i,j}$ the mean parameter of a posterior distribution and $\alpha$ is shared amongst all weights and represents the dropout rate. This is treated as a noise hyper-parameter, which determines the spread around the mean. We then define our variational parameters as the set $\boldsymbol{\phi} = (\boldsymbol{\theta}, \alpha)$ allowing us to write our posterior over a particular weight as $q_{\phi_{i,j}}(w_{i,j})$. Any multiplicative noise that affects our weights we may call a posterior distribution in this way. We illustrate this idea in Figure 5.5.

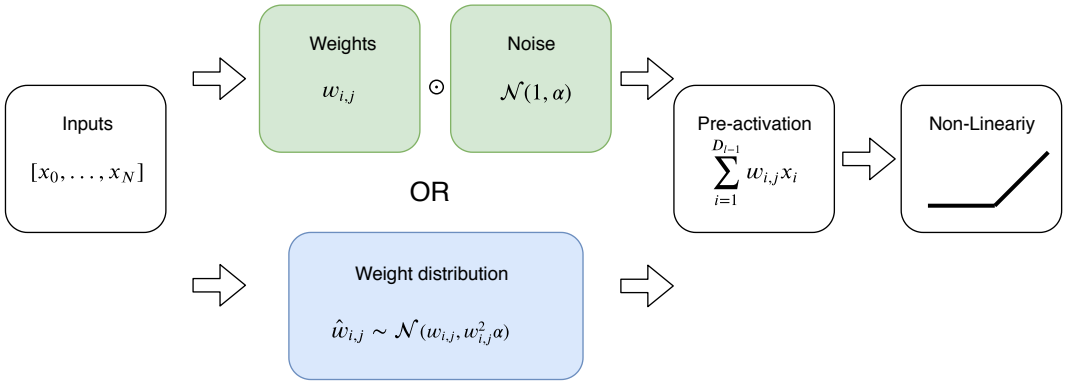



Figure 5.5: Gaussian dropout as a posterior distribution over weights. The noise distribution does not change the mean value of the weights, it only adds multiplicative noise. We can thus interpret multiplicative noise either as Gaussian dropout or training a BNN with Gaussian posteriors.

We now look to relate the procedure of Bayesian inference to the posterior we just discussed to Gaussian dropout. The procedure of training networks with Gaussian dropout is a stochastic optimisation task as we optimise the expected log likelihood. Thus we write the training objective of Gaussian dropout, making use of the reparametrisation trick, giving

$$\mathbb{E}_{p(\epsilon)\sim\mathcal{N}(0,I)}[\log p(y|x, W = \theta \odot (1 + \sqrt{\alpha}E))]. \tag{5.47}$$

During dropout training, $W$ or $\boldsymbol{\theta}$ is adapted to optimise this objective. This is equivalent to the likelihood term in variational inference of BNNs. Thus, the optimisation process of a neural network with dropout is the equivalent to the optimisation process of finding variational parameters of a BNN without incorporating a prior.

We require some restrictions on the prior for the optimisation to be congruous with the optimisation of a variational lower bound. We work backwards from the objective to find a prior to allow this to be consistent with Bayesian inference. The objective for variational dropout, which includes a prior $p(W)$, can be written as

$$\mathbb{E}_{q_\phi(W)} \log p(y|x, W) - \mathrm{KL}(q_\phi(W)||p(W)). \tag{5.48}$$

If we can find $p(W)$ consistent with dropout optimisation, Gaussian dropout and variational inference become equivalent. To determine the prior, as discussed in [20], the KL divergence $\mathrm{KL}(q_\phi(W)||p(W))$ should not depend on $\theta$. If we constrain our prior, $p(W)$, to depend only on $\alpha$ and we fix $\alpha$, the only prior that meets this requirement is the scale-invariant log-uniform

distribution:

$$p(w_{i,j}) \propto \frac{1}{|w_{i,j}|}. \tag{5.49}$$

The KL divergence term is then given by

$$-\mathrm{KL}(q_\phi(W)||p(W)) = \frac{1}{2}\alpha - \mathbb{E}_{\epsilon\sim\mathcal{N}(1,\alpha_{i,j})} \log|\epsilon| + C. \tag{5.50}$$

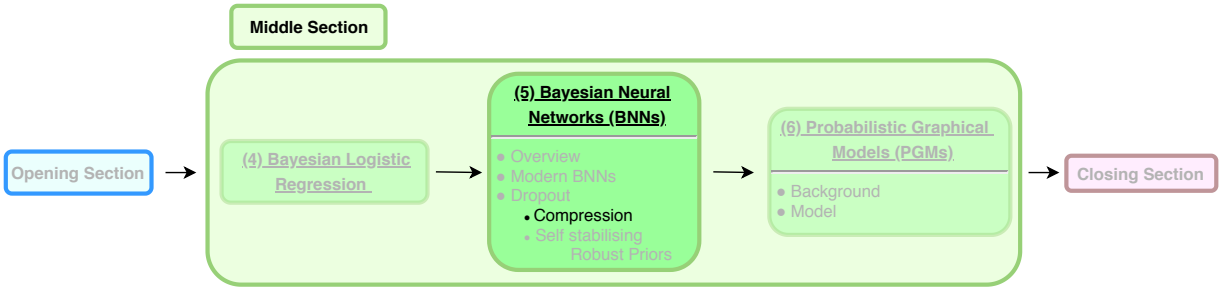
where $C$ is approximated with a straight line function (see Appendix B for more details; we do not make use of this particular prior in any of our implementations).

The result of this is that there is some prior that casts Gaussian dropout as exactly equivalent to a special case of training BNNs. This allows us to train individual dropout rates $\alpha_{i,j}$ for each weight. Later we discuss how an improvement on this, with an additive reparametrisation trick, which allows us to sparsify models. We also make use of this new interpretation to relate signal propagation theory for deterministic neural networks with stochastic regularisation to signal propagation for BNNs in the following chapter.

# Chapter 6

# Bayesian Neural Network Priors and Applications

## 6.1 Model Compression




Our discussions until now have only considered BNNs as uncertainty estimators. Here we extend the discussion to the use of Bayesian methods to compress neural networks. Compressing neural networks to reduce parameters is advantageous in situations where there is limited storage space and energy such as onboard a satellite. We do this by using priors that induce sparsity that urges the model to remove parameters in the learning process and prune the weights. We first explain heuristic ways of pruning down weights followed by variational dropout, which proves to automatically sparsify the model during the optimisation process.

In neural networks, a large number of parameters are present that supposedly learn feature representations that could be included to improve predictive performance. Neural networks, however, are heavily overparametrised, having many redundant parameters, and thus use more memory and computation than necessary. We show that they can be pruned significantly without any

loss in accuracy. However, we cannot be sure which parameters are more closely associated with the targets and are truly relevant in making predictions. Neural networks are not interpretable, the underlying mechanisms are not well understood, so we cannot confidently determine which attributes are relevant. Priors can be designed to induce sparsity or for automatic relevance determination (ARD). The model can then automatically determine the degree to which inputs of unknown relevance are in fact relevant. Accordingly, inference is based on a combined objective of improving the model's ability to explain the data as well as limiting the number of parameters.

### 6.1.1 Pruning Neural Networks with Gaussian posteriors

The simplest form of compression can be done by heuristically pruning the number of parameters in the network. We start removing parameters according to some criteria until the accuracy on some held-out data starts to decline.

In the case of Gaussian posteriors, the variance parameter for our weight can serve as parameter uncertainty. Uncertainty in this context describes how certain we are about the values of the parameters or to what extent they contribute to the predictions of our model. We decide to prune a weight if the variance of a parameter is large compared to the mean. We interpret this similar to a signal to noise ratio (SNR) given by

$$\text{SNR} = \frac{\mathbb{E}[w]}{\sqrt{\text{Var}[w]}}. \tag{6.1}$$

If the expected value is much less than the variance, the SNR is high and we remove the weight.

Alternatively, the metric we can use to prune a parameter is simply the probability that it will be zero given its probability distribution. We order our parameters by the likelihood that a particular parameter will take the value of zero, according to its distribution, and start removing them until the validation accuracy starts to fall below an acceptable level. We show an experiment demonstrating pruning with this criteria in Figure 6.1.

We can also encourage this by using sparsity-inducing or ARD priors as in [9], [27]. Considering a BNN with fully factorised Gaussian posteriors, we define a Gaussian prior $p(w_{i,j}) = \mathcal{N}(0, \sigma^2_{\text{prior}})$. The variational lower bound objective with this prior is then given by

$$\mathcal{L}(\theta, \Sigma) = \mathbb{E}_{p(E)}\left[\log p(y|x, W = \theta + \Sigma \odot E) + \sum_{\forall i \in W} \log \frac{\sigma_i^2}{\sigma^2_{\text{prior}}} + \frac{\mu_i^2 + \sigma_i^2 - \sigma^2_{\text{prior}}}{2\sigma^2_{\text{prior}}}\right]. \tag{6.2}$$

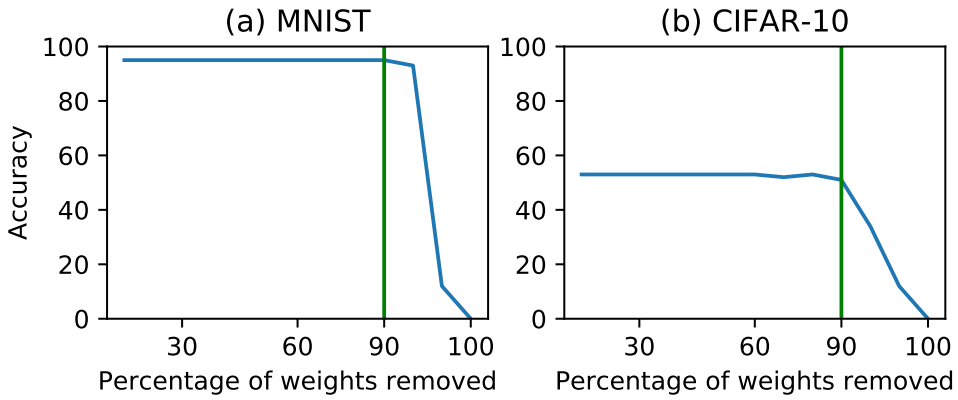



Figure 6.1: Pruning a Bayesian neural network on MNIST and CIFAR-10. We see that we require only 10 % of the weights without any decline in accuracy. This experiment uses the criteria of pruning weights in order of probability that a particular parameter will take the value of 0 given the weight distribution. We train a 2-layer BNN and sort weights by this criteria and incrementally remove parameters, evaluating accuracy each step that parameters are removed.

Similar to logistic regression in [27], we can find the optimal value for each prior hyper-parameter by taking the derivative with respect to $\sigma^2_{\text{prior}}$ and setting it to zero. For a matrix $W^l$ of size $D^l$ by $D^{l-1}$ this gives $\sigma^2_{\text{prior}^l} = \frac{1}{D^l \times D^{l-1}} \sum_{\forall i \in W^l} (\mu_i^l)^2 + (\sigma_i^l)^2$. Substituting the optimal parameters back into the objective we can obtain a simplified objective in

$$\mathcal{L}(\theta, \Sigma) = \mathbb{E}_{\epsilon \sim \mathcal{N}(0,I)} \left[ \log p(y|x, W = \theta + \Sigma \odot E) + \frac{1}{2} \sum_{\forall i \in W} \log \frac{\sigma^2_{\text{prior}}}{\mu_i^2 + \sigma_i^2} \right]. \tag{6.3}$$

This encourages the posterior to have a high density around zero and removing parameters with high likelihood of zero, yields sparse solutions and has an ARD effect. Since we choose hyper-parameters that optimise the likelihood of the data, this is considered and empirical Bayes (EB) procedure. The legitimacy of this is often questioned from a pure Bayesian perspective as it is possible to overfit by having the data influence the prior. However, EB is often used in practice with great success [59]. Next, we discuss variational dropout and its function as a sparsity-inducing prior.

#### 6.1.1.1 Variational Dropout for Sparsification

Here we continue our discussion of relating dropout to Bayesian inference and discuss model sparsification using variational dropout presented in [19]. Due to an additive reparametrisation trick, which reduces the variance of variational dropout, it is possible to more stably train, as well as sparsify, BNNs. We will see that variational dropout automatically yields sparse solutions by naturally encouraging the removal of weights.

#### 6.1.1.2 Additive Reparametrisation Trick

In the original paper presenting variational dropout [20], the authors reported trouble with training models with large dropout rates $\alpha$, and recommended constraining dropout rates restricted to $\alpha \leq 1$. As discussed in [19] the concern is due to $\alpha$ corresponding to multiplicative noise that grows rapidly and results in gradients with high variance for large $\alpha$. The additive reparametrisation trick effectively replaces multiplicative noise with additive noise that yields more stable gradients.

To implement this additive reparametrisation, we consider an individual weight where the variational parameters for a weight $w_{i,j}$ are defined by the set $\phi_{i,j} = (\theta_{i,j}, \alpha_{i,j})$ and $\alpha_{i,j}$ represents the dropout rate. In variational dropout, our weights are computed as

$$w_{i,j} = \theta_{i,j}(1 + \sqrt{\alpha_{i,j}}\epsilon_{i,j}) \quad \text{where} \quad \epsilon_{i,j} \sim \mathcal{N}(0, 1). \tag{6.4}$$

The additive reparametrisation involves introducing a new parameter $\sigma_{i,j}$ so that we can write (6.4) as

$$w_{i,j} = \theta_{i,j} + \sigma_{i,j}\epsilon_{i,j}, \tag{6.5}$$

and for this to be true we require

$$\sigma_{i,j}^2 = \alpha_{i,j}\theta_{i,j}^2. \tag{6.6}$$

The optimisation process now involves three interdependent parameters, $\sigma, \alpha$ and $\theta$. The objective function and the posterior distribution remain the same with this parametrisation. We only adjusted the parametrisation of the approximate posterior and the computation of a forward pass in a network that reduces variance in the gradient.

The success of this parametrisation is attributed to greatly reducing the variance of the gradient for large $\alpha$. Using this trick, we can train the model within the full range of $\alpha_{i,j}$ i.e. $\alpha$ is no longer restricted to $\alpha \leq 1$. It is clear when we look at the partial derivative with respect to the parameters $\theta_{i,j}$ given by

$$\frac{\partial \mathcal{L}}{\partial \phi_{i,j}} = \frac{\partial \mathcal{L}}{\partial w_{i,j}} \frac{\partial w_{i,j}}{\partial \theta_{i,j}}. \tag{6.7}$$

The usual form that we calculate from (6.4) is given by

$$\frac{\partial w_{i,j}}{\partial \theta_{i,j}} = 1 + \sqrt{\alpha_{i,j}} \tag{6.8}$$

and with the additive reparametrisation calculated from (6.5) we get

$$\frac{\partial w_{i,j}}{\partial \theta} = 1. \tag{6.9}$$

We can see that the gradients of $\theta$ no longer vary depending on $\alpha$. Large values for $\alpha$ correspond to random or noisy weights when sampled, which corrupts the feed-forward signal. Considering the computation of a weight in (6.5), large values of $\alpha$ correspond to a node that is absolutely random and contributes nothing to the signal propagating through the network.

To gain some intuition, we compare Gaussian dropout with ordinary binary dropout. We recall from (5.44) that binary and Gaussian dropout are related by

$$p = \frac{\alpha}{1 + \alpha} \tag{6.10}$$

where $p$ is the probability we switch off the neurons in binary dropout and $\alpha$ the variance of the Gaussian noise in Gaussian dropout. If $\alpha = 1$, the dropout probability is 0.5 that corresponds to a frequently used dropout rate in practice. If $\alpha$ is very small, there is little Gaussian noise and $p$ goes zero. This corresponds to a small chance that a neuron is to be switched off or dropped out. If $\alpha$ is very large, then there is a lot of dispersion around the weight mean and becomes indistinguishable from random noise. In this case $p$ is high and there is a high probability the weight is to be switched off or dropped out. Or as $\alpha \to \infty$ then $p = 1$ which effectively means that the corresponding weight or neuron is always dropped out and can be removed from the model. A random weight will reduce the likelihood of the data and we thus remove it from the network.

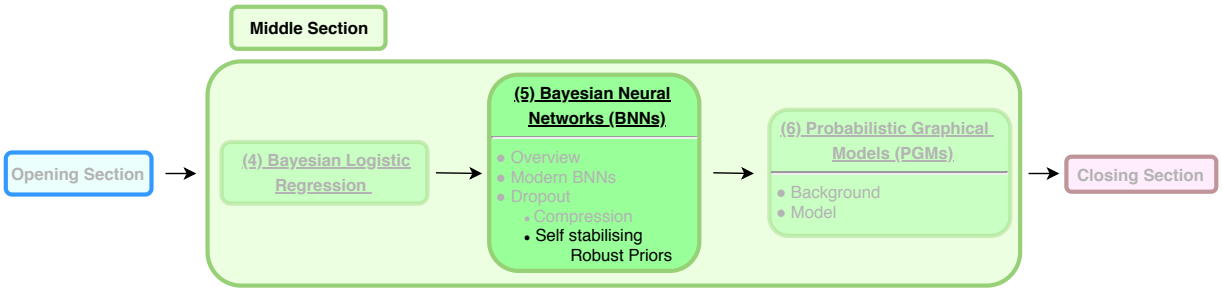
It turns out that if we put no restrictions on $\alpha$ it leads to very large values of $\alpha$ and very sparse solutions. This will become clear by examining a simplified ELBO for intuition given by

$$\mathbb{E}_{W \sim q(W|\theta,\alpha)} \log p(y|x, W) - \mathrm{KL}(\alpha) \tag{6.11}$$

where for our interest $\mathrm{KL}(\alpha)$ is the KL divergence and is some function of the value $\alpha$. The optimisation attempts to maximise this function, and in doing so, increase the negative KL divergence. The negative KL divergence grows as $\alpha$ grows, so a larger $\alpha$ maximises the ELBO. The

objective thus naturally favours large dropout rates. As discussed previously, large dropout rates correspond to absolutely random weights that we select to remove. Thus variational dropout with the additive reparametrisation trick leads to very sparse solutions. This effect is similar to ARD effect as it automatically decides which weights are irrelevant and to remove. It automatically prunes down the network with no accuracy degradation. This is our final sparsification technique and concludes our discussion of compression. The succeeding section continues to build on variational dropout and the interpretation of stochastic regularisation techniques as training BNNs without the additive reparametrisation.

## 6.2 Self-Stabilising Robust Bayesian Neural Networks




**Contribution statement:** As discussed in the contributions sections, this section is joint work with Arnu Pretorius.

BNNs have emerged as useful tools for modelling data under uncertainty due to advances in variational inference, making it possible to train BNNs at scale. However, despite these advances, BNNs remain brittle and hard to train, especially in the following two instances: (1) when using deep architectures consisting of many hidden layers and (2) in situations of large weight variance. Here we introduce *adaptive Monte-Carlo variational inference* (adMCVI) with *self-stabilising priors*, which makes it possible to successfully train BNNs in both (1) and (2). This prior is derived and inspired by a signal propagation analysis of deep BNNs. The effectivity of the stabilising prior depends on reformulating the ELBO objective such as to allow the prior to influence the network during the forward pass. Then, by allowing the prior to influence the network during the forward pass, we develop a self-stabilising prior, where the distributional parameters of the prior are adjusted at each forward pass to ensure stability of a propagating signal. This approach to variational inference stabilises network dynamics during training and leads to improved convergence and robustness. This makes it possible to train deeper networks and in more noisy settings. We demonstrate the effectiveness of adMCVI with stabilising priors

in several experiments on MNIST, CIFAR-10 and synthetic data.

Inspired by signal propagation theory in deep neural networks [17], [18], [16], we propose a heuristic similar in nature to an iterated application of empirical Bayes (EB) for setting prior hyper-parameters, as is common in BNN training ([10], [50]) and [27] propose closed-form updates for prior hyper-parameters). While EB chooses hyper-parameters that optimise the likelihood of the data, our approach instead chooses prior hyper-parameters for each forward pass that attempt to optimise signal propagation behaviour in the network. This allows us to effectively train deeper BNNs than otherwise possible, but we also observe improved performance in the previously trainable regime. We further note that this is the first application of signal propagation theory for neural networks outside of initialisation schemes that we are aware of.

### 6.2.1 Reformulating the ELBO: Adaptive MCVI

The prior $p_\alpha(W)$, with hyper-parameters $\alpha$, in the variational objective as in (5.3) impacts the ELBO through the KL term, affecting the weights at update time with backpropagation. Note, however, that these updates only take place after a completed forward pass, having no effect on the signal propagation dynamics of the network. We instead argue for priors that exert their influence *during* the forward pass, so as to simultaneously promote stable signal propagation, and improve robustness in deep BNNs. It is essential that a stabilising prior be able to influence the network on the forward pass if any training is to occur in deep networks i.e. to enable the signal to reach the outputs. To be able to achieve this effect, we make two innovations: (1) we reformulate the ELBO to allow the prior to influence the signal propagation of the network and (2) we derive a novel prior, specifically for ReLU networks, that stabilises the signal during each forward pass during training.

We reformulate the ELBO by lower bounding the log marginal likelihood of the data as follows

$$\log p(\boldsymbol{y}|\boldsymbol{x}) = \log \int p(\boldsymbol{y}|\boldsymbol{x}, W) p_\alpha(W) dW \tag{6.12}$$

$$= \log \int p(\boldsymbol{y}|\boldsymbol{x}, W) p_\alpha(W) \frac{q_\phi(W)}{q_\phi(W)} dW, \tag{6.13}$$

where we combine the prior and approximating posterior as $\tilde{q}_{\{\alpha,\phi\}}(W) = p_\alpha(W) q_\phi(W)/Z$. The combined approximating posterior $\tilde{q}_{\{\alpha,\phi\}}(W)$, requires the normalisation constant $Z$ to be a valid probability distribution. We thus multiply the ratio $Z/Z$ into (6.13). Then, we can construct a

lower bound making use of Jensen's inequality which gives

$$\log p(\boldsymbol{y}|\boldsymbol{x}) = \log \int p_\alpha(W) q_\phi(W) \frac{Z}{Z} \frac{p(\boldsymbol{y}|\boldsymbol{x}, W)}{q_\phi(W)} dW \tag{6.14}$$

$$= \log \int \tilde{q}_{\{\alpha,\phi\}}(W) Z \frac{p(\boldsymbol{y}|\boldsymbol{x}, W)}{q_\phi(W)} dW \tag{6.15}$$

$$\geq \int \tilde{q}_{\{\alpha,\phi\}}(W) \log \frac{Z p(\boldsymbol{y}|\boldsymbol{x}, W)}{q_\phi(W)} dW$$

$$= \mathbb{E}_{\tilde{q}_{\{\alpha,\phi\}}(W)} \left[\log p(\boldsymbol{y}|\boldsymbol{x}, W)\right] - \mathbb{E}_{\tilde{q}_{\{\alpha,\phi\}}(W)} \left[\log \frac{q_\phi(W)}{Z}\right]. \tag{6.16}$$

By reformulating the ELBO in this way, we can estimate the above expectations using a Monte-Carlo estimator with samples drawn from $\tilde{q}_{\{\alpha,\phi\}}(W)$ instead of $q_\phi(W)$. This ensures that the sampled weights of the network are being influenced by the current prior $p_\alpha(W)$ during the forward pass. Finally, we define our new variational objective as

$$\mathcal{L}_{\tilde{q}} \coloneqq \mathbb{E}_{p(\epsilon)} \left[\log p(\boldsymbol{y}|\boldsymbol{x}, W = \xi(\epsilon, \alpha, \phi))\right] - \mathrm{KL}(\tilde{q}_{\{\alpha,\phi\}}(W)||p_\alpha(W)), \tag{6.17}$$

where $\epsilon \sim \mathcal{N}(0, I)$. This reformulation of the ELBO allows the prior to exercise its stabilising effect during the forward pass as well as adaptively changing the expectation of the likelihood in MCVI, leading to more efficient estimates and stable training.

Next, we derive the signal propagation dynamics of a BNN that enables us to find a stabilising prior by finding optimal prior parameters $\alpha$ as a function of the approximate posterior parameters $\phi$. The prior is updated after every gradient update to $\phi$ to ensure it adapts to the current setting of the posterior (see Algorithm 1 for simple sequence of updates). These prior parameters will turn out to be optimal in the sense that together with the reformulated ELBO in (6.17), the sampled weights from $\tilde{q}_{\{\alpha,\phi\}}(W)$ will have a self-stabilising effect on the signal propagation dynamics of a deep BNN.

### 6.2.2 Signal Propagation in BNNs

Our work follows from signal propagation being extended to include noise regularisation [16] and BNNs being related to stochastic regularisation techniques [28], [20]. We analyse BNNs from a signal propagation theoretic perspective in an attempt to understand how to scale BNNs and make them more robust during training. We know that at a certain depth neural networks lose the ability to propagate discriminatory information about their inputs [18], moreover, stronger noise regularisation reduces this depth [16]. Thus it was shown in [16] that the distribution of the

parameters with which one randomly initialises a deterministic network should be determined by the strength of the noise regularisation. In BNNs, signal propagation is determined by the parameters of the weight distribution. We look to find conditions that allow us to adjust these parameters to promote stable signal propagation. Our work extends initialisation techniques in deterministic networks to an iteratively updating prior to allow more stable flow of information through the network *throughout* training. This defends against poor signal propagation associated with vanishing or exploding signals and poor network performance.

Our approach to deriving a stabilising prior is inspired by recent work in signal propagation theory for infinite width neural networks [17], [18], [16]. Specifically, we make use of the *mean-field* assumption [60], [17]. The mean-field assumption allows for the components of the pre-activation vectors $\mathbf{h}^l$ to be treated as independent Gaussian random variables, fully characterised by their second-order statistics. This assumption is supported by the following observation: sampling weights i.i.d. for an infinite width neural network means that the pre-activations in a given layer each consist of an infinite sum of i.i.d. random variables (the incoming connections from the previous layer). Then, according to the central limit theorem, these pre-activations are Gaussian distributed. Although we work in the limit of infinite width setting, there is empirical evidence that the assumption is accurate even in neural networks of reasonable finite width (e.g. $D_l = 256$ or 512) [61],[18],[62],[63],[64],[16].

In signal propagation we study the statistics of a signal $\boldsymbol{x}^l$ through the layers of the network. In order to do this in a BNN we recursively define layers, given an input $\boldsymbol{x}^0 \in \mathbb{R}^{D_0}$, as

$$\boldsymbol{h}^l = W^l \boldsymbol{x}^l + \boldsymbol{b}^l, \qquad \text{for } l = 1, ..., L \tag{6.18}$$

where the weights $W^l \in \mathbb{R}^{D_l \times D_{l-1}}$ and the biases $\boldsymbol{b}^l \in \mathbb{R}^{D_l}$ constitute the model parameters $\theta = \{W^l, \boldsymbol{b}^l\}_1^L$. This is a recursive sequence of operations as the previous layer feeds into the current layer, feeding into the next etc. At any layer in a deep network the signal propagation dynamics is recursively is captured by

$$\boldsymbol{x}^l = g(\boldsymbol{h}^{l-1}). \tag{6.19}$$

We denote the dimensionality of the hidden layers using $D_l$ and compute activations at each layer element-wise using an activation function $g(\cdot)$.

We can analyse the signal propagation dynamics of BNNs by recursively examining the statis-

tics of an input at each layer and in expectation of the parameters over infinitely wide layers. Considering factorised Gaussian posterior distributions over the weights and focussing on ReLU activations, i.e. $g(a) = \max(0, a)$, we consider a single scalar hidden unit $h_j^l$ at an arbitrary layer $l$ in the network. We calculate the mean and variance of a propagating signal at $h_j^l$ for a ReLU network in expectation over the weights and biases under the mean-field assumption. We make the assumption, as in [9], that for networks of infinite width, individual contributions by the weights feeding into a hidden unit, are roughly equal. Thus, for a hidden unit $j$ at layer $l$, weights determining signal propagation are identical with means $\mu_{q_j}^l$ and variances $(\sigma_{q_j}^l)^2$. The relevant statistics governing signal propagation in the forward pass are thus given by Lemmas 1 & 2:

**Lemma 1**

$$\begin{aligned}\mathrm{Var}[h_j^l] = \nu_j^l &= \mathbb{E}\left[\left(\sum_{i=1}^{D_{l-1}} w_{i,j}^l g(h_j^{l-1} + b_j)\right)^2\right] - \mathbb{E}\left[\sum_{i=1}^{D_{l-1}} w_{i,j}^l g(h_j^{l-1} + b_j)\right]^2 \qquad (6.20)\\ &= (\mu_{q_j}^l)^2\left[\frac{(\tau^{l-1})^2}{4} + \tau^{l-1}\sqrt{\frac{\nu^{l-1}}{2\pi}} + \left(1 - \frac{1}{\pi}\right)\frac{\nu^{l-1}}{2}\right]\\ &\quad + (\sigma_{q_j}^l)^2\left[\frac{(\tau^{l-1})^2}{2} + 2\tau^{l-1}\sqrt{\frac{\nu^{l-1}}{2\pi}} + \frac{\nu^{l-1}}{2}\right] + \sigma_b^2, \qquad (6.21)\end{aligned}$$

and

**Lemma 2**

$$\begin{aligned}\mathbb{E}[h_j^l] = \tau^l &= \mathbb{E}\left[\sum_{j=1}^{D_{l-1}} w_{i,j}^l g(h_j^{l-1} + b_i)\right]\\ &= \mu_{q_j}^l\left(\frac{\tau^{l-1}}{2} + \sqrt{\frac{\nu^{l-1}}{2\pi}}\right) + \mu_b^l \qquad (6.22)\end{aligned}$$

where $\tau^{l-1}$ and $\nu^{l-1}$ are the mean and variance of the incoming signal to hidden layer $l$ respectively. This defines how the signal will progress through the layers. The output mean $\tau^l$, and variance $\nu^l$, of a layer feeds into the next layer and grows or shrinks relative to weight mean $\mu_{q_j}^l$ and variance $(\sigma_{q_j}^l)^2$. We will use the equations in Lemmas 1 & 2 above and some assumptions to design a prior that will preserve the variance of a signal in a BNN. This ensures that an input signal is not destroyed by the variance either vanishing or exploding.

**Proof: Lemma 1 & 2**

We consider the quantity $\mathbb{E}[(h_j^l)^2]$ during a forward pass. This quantity is the second moment of a single hidden unit $h_j^l$ in layer $l$, consisting of $D_{l-1}$ incoming connections, where the expectation taken is over the network parameters and is given by

$$\mathbb{E}[(h_j^l)^2] = \mathbb{E}\left[\left(\sum_{i=1}^{D_{l-1}} w_{i,j}^l x_i^l + b_j^l\right)^2\right] \tag{6.23}$$

$$= \sum_{i=1}^{D_{l-1}} \mathbb{E}[(w_{i,j}^l)^2]\mathbb{E}[(x_i^l)^2] + \mathbb{E}[(b_j^l)^2]. \tag{6.24}$$

We define

$$\sigma_{\tilde{q}_j}^l = \sum_{i=1}^{D_{l-1}} (\sigma_{i,j}^l)^2 \qquad \text{and} \qquad \mu_{\tilde{q}_j}^l = \sum_{i=1}^{D_{l-1}} \mu_{i,j}^l \tag{6.25}$$

to use as the statistics to describe $\sum_{i=1}^{D_{l-1}} w_{i,j}$. Note that while it is true that we can write

$$\sum_{i=1}^{D_{l-1}} w_{i,j}^l \sim \mathcal{N}\left(\mu_{\tilde{q}_j}^l, (\sigma_{\tilde{q}_j}^l)^2\right) \tag{6.26}$$

at initialisation, in our analysis of the network at an arbitrary stage of the stage of training the i.i.d. assumption of the central limit theorem (CLT) does not strictly hold. As in [50], we empirically find that some form of the CLT holds for the hidden units during training. We thus continue to approximate the expectation with a Gaussian according to the CLT. Next, in designing $\tilde{q}_{\{\alpha,\phi\}}(W)$ we scale the variance and impose $\mathbb{E}[(w_{i,j}^l)^2] = \frac{(\mu_{\tilde{q}}^l)^2 + (\sigma_{\tilde{q}}^l)^2}{D_{l-1}}, \forall i, j$ to ensure that the variance is bounded in the infinite width limit [18]. This also allows the variance propagated forward to be independent of the layer width. We now have

$$\mathbb{E}[(h_j^l)^2] = ((\mu_{\tilde{q}_j}^l)^2 + (\sigma_{\tilde{q}_j}^l)^2)\frac{1}{D_{l-1}} \sum_{i=1}^{D_{l-1}} g(h_i^{l-1})^2 + ((\mu_b^l)^2 + (\sigma_b^l)^2). \tag{6.27}$$

As $D_{l-1} \to \infty$, $h_j^l$ becomes an infinite sum of i.i.d. random variables and becomes Gaussian distributed according to the CLT. We can thus write

$$\mathbb{E}[(h_j^l)^2] = ((\mu_{\tilde{q}}^l)^2 + (\sigma_{\tilde{q}_j}^l)^2)\mathbb{E}_z[\phi(\tau^{l-1} + \sqrt{\nu^{l-1}}z)^2] + (\mu_b^l)^2 + (\sigma_b^l)^2 \tag{6.28}$$

where $z \sim \mathcal{N}(0, 1)$, and $\tau^{l-1}$ and $\nu^{l-1}$ are the incoming signal to layer $l$'s mean and variance

respectively. If we use the ReLU activation, i.e. $g(a) = \max(0, a)$, then

$$\begin{aligned}\mathbb{E}[(h_j^l)^2] &= ((\mu_{\tilde{q}}^l)^2 + (\sigma_{\tilde{q}_j}^l)^2)\left\{\int_{-\infty}^{\infty} \Phi(z)\phi(\tau^{l-1} + \sqrt{\nu^{l-1}}z)^2 dz\right\} + (\mu_b^l)^2 + (\sigma_b^l)^2 \\ &= ((\mu_{\tilde{q}}^l)^2 + (\sigma_{\tilde{q}_j}^l)^2)\left\{\int_0^{\infty} \Phi(z)\left((\tau^{l-1})^2 + 2\tau^{l-1}\sqrt{\nu^{l-1}}z + \nu^{l-1}z^2\right) dz\right\} \\ &\qquad + (\mu_b^l)^2 + (\sigma_b^l)^2 \\ &= ((\mu_{\tilde{q}}^l)^2 + (\sigma_{\tilde{q}_j}^l)^2)\left[\frac{(\tau^{l-1})^2}{2} + \frac{2\tau^{l-1}\sqrt{\nu^{l-1}}}{\sqrt{2\pi}} + \frac{\nu^{l-1}}{2}\right] + (\mu_b^l)^2 + (\sigma_b^l)^2 \end{aligned} \tag{6.29}$$

where $\Phi(z) = \frac{e^{-z^2/2}}{\sqrt{2\pi}}$.

Similarly, we can show that the relevant statistics governing signal propagation in the forward pass are given by

$$\begin{aligned}\mathbb{E}[h_j^l] &= \mathbb{E}\left[\sum_{j=1}^{D_{l-1}} w_{i,j}^l g(h_j^{l-1} + b_i)\right] \\ &= \mu_{\tilde{q}_j}^l\left(\frac{\tau^{l-1}}{2} + \sqrt{\frac{\nu^{l-1}}{2\pi}}\right) + \mu_b^l \end{aligned} \tag{6.30}$$

and

$$\begin{aligned}\nu_j^l &= \mathbb{E}\left[\left(\sum_{j=1}^{D_{l-1}} w_{i,j}^l g(h_j^{l-1})\right)^2\right] - \mathbb{E}\left[\sum_{j=1}^{D_{l-1}} w_{i,j}^l g(h_j^{l-1})\right]^2 \\ &= (\mu_{\tilde{q}_j}^l)^2\left[\frac{(\tau^{l-1})^2}{4} + \tau^{l-1}\sqrt{\frac{\nu^{l-1}}{2\pi}} + \left(1 - \frac{1}{\pi}\right)\frac{\nu^{l-1}}{2}\right] \\ &\quad + (\sigma_{\tilde{q}_j}^l)^2\left[\frac{(\tau^{l-1})^2}{2} + 2\tau^{l-1}\sqrt{\frac{\nu^{l-1}}{2\pi}} + \frac{\nu^{l-1}}{2}\right] + (\sigma_b^l)^2. \end{aligned} \tag{6.31}$$

It is widely accepted that random networks and BNNs are effectively untrainable beyond a certain depth. With signal propagation theory we quantify this. Signal propagation has already been applied to quantify random initialisation in deterministic networks and design initialisation schemes to make it possible to train deeper networks. Stable signal propagation for ReLU networks requires pre-activations of variance 2 in the infinite width limit (only half the signal propagates through the activation). This corresponds to the He initialisation widely used for ReLU networks [25]. In ReLU networks this is more appropriate than the Xavier/Glorot [65] initialisation that suggests unit pre-activation variances (half the He initialisation that leads to vanishing signals in ReLU networks).

### 6.2.3 Self-stabilising Prior

For models with few parameters, there are principled ways to select priors and get sensible posteriors, however, in the context of BNNs, selecting meaningful priors seems obscure. In addition, BNNs are sensitive to the choice of prior [9]. Our approach is to design a prior to optimise stable signal propagation. In effect we use knowledge about the network architecture and activations and how signals propagate in the infinite width limit to inform our prior.

To begin, we make the following distributional assumptions for the prior, approximating posterior and their product. Importantly, we define a shared prior variance for weights feeding into the same hidden unit following from our assumption that in the infinite width limit contributions to the variance feeding into a single hidden unit are roughly equal. Specifically, for every element $w^l_{i,j}$ of the weight matrix $W^l$, we assume

$$\begin{aligned} p_\alpha(w^l_{i,j}) &= \mathcal{N}(\mu^l_{p_{i,j}}, (\sigma^l_{p_j})^2), & \alpha^l_{i,j} &= \{\mu^l_{p_{i,j}}, \sigma^l_{p_j}\} \\ q_\phi(w^l_{i,j}) &= \mathcal{N}(\mu^l_{q_{i,j}}, (\sigma^l_{q_{i,j}})^2), & \phi^l_{i,j} &= \{\mu^l_{q_{i,j}}, \sigma^l_{q_{i,j}}\} \\ \tilde{q}_\beta(w^l_{i,j}) &= \mathcal{N}(\mu_{\tilde{q}_{i,j}}, (\sigma^l_{\tilde{q}_{i,j}})^2), & \beta^l_{i,j} &= \{\alpha^l_{i,j}, \phi^l_{i,j}\} \end{aligned}$$

where the parameters for the joint distribution $\tilde{q}_\beta(w)$ are determined by $\alpha$ and $\phi$ as follows

$$\mu^l_{\tilde{q}_{i,j}} = \frac{\mu^l_{q_{i,j}}(\sigma^l_{p_{i,j}})^2 + \mu^l_{p_{i,j}}(\sigma^l_{q_{i,j}})^2}{(\sigma^l_{p_j})^2 + (\sigma^l_{q_{i,j}})^2}, \qquad \sigma^l_{\tilde{q}_{i,j}} = \sqrt{\frac{(\sigma^l_{p_j})^2(\sigma^l_{q_{i,j}})^2}{(\sigma^l_{p_j})^2 + (\sigma^l_{q_{i,j}})^2}}. \tag{6.32}$$

We also define

$$\mu^l_{\tilde{q}_j} = \sum_{i=1}^{D_{l-1}} \mu^l_{\tilde{q}_{i,j}}, \qquad (\sigma^l_{\tilde{q}_j})^2 = \sum_{i=1}^{D_{l-1}} (\sigma^l_{\tilde{q}_{i,j}})^2 \tag{6.33}$$

and similarly for $p$ and $q$.

Our aim is to extend initialisation techniques and design $\tilde{q}_{\{\alpha,\phi\}}(W)$ to preserve variances in a BNN, presented in (6.21), *throughout* training. Focusing on the signal propagation dynamics for a BNN with ReLU activations described in equation (6.21) and the recursive definition of a deep neural network, we can derive $\alpha$ to preserve the variance of a signal propagating through a BNN. We assume zero-mean inputs at each layer, a somewhat unrealistic assumption (further discussed in Discussion 1), allowing us to maintain $\tau^{l-1} = 0$ during training. Under these conditions the

variance of the signal in equation (6.21) reduces to

$$\nu_j^l = \left[\left(1 - \frac{1}{\pi}\right)(\mu_{\tilde{q}_j}^l)^2 + (\sigma_{\tilde{q}_j}^l)^2\right]\frac{\nu^{l-1}}{2}. \tag{6.34}$$

**Discussion 1: Assumptions for mean-preserving prior**

Previously described is the signal propagation dynamics in general. With a mean preserving prior we can only control variance by multiplicatively expanding or squeezing $\sigma_{\tilde{q}_j}^l$. We can only design for conditions that set $\tau^{l-1} = 0$ and $\sigma_b = 0$, which is true at initialisation but starts to break down during training. In order to continue, we thus implicitly assume that during training: (1) the mean of the summed weights' means across a hidden layer's pre-activation remain mean zero i.e. $\mathbb{E}[(\sum_{j=1}^{D_l} \mu_{q_j}^l)] = 0$; (2) biases are zero (note, it is possible to absorb the biases by augmenting the input at each layer with an additional column of ones, this yields more stable signal propagation. We find that treating biases as deterministic parameters aids in training and outweighs the minor gain in stabilising propagation). This allows us to write the variance $\nu_j^l$ as

$$\nu_j^l = \left[\left(1 - \frac{1}{\pi}\right)(\mu_{\tilde{q}_j}^l)^2 + (\sigma_{\tilde{q}_j}^l)^2\right]\frac{\nu^{l-1}}{2}. \tag{6.35}$$

We find forcing our assumptions and setting parameter means and biases to zero does not allow for any training.

To stabilise the signal, we look for conditions that preserve the variance during the forward pass, more specifically, we want to ensure $\text{Var}[h_j^l] = \text{Var}[h_{j'}^{l-1}]$ or $\nu_j^l = \nu_{j'}^{l-1}$, where $j' \in \{1, ..., D_{l-1}\}$. Setting the variances equal in such a way leads (6.34) to yield

$$(1 - 1/\pi)(\mu_{\tilde{q}_j}^l)^2 + (\sigma_{\tilde{q}_j}^l)^2 = 2, \tag{6.36}$$

defining the condition for BNN pre-activations with stable signal propagation.

Since the forward pass makes use of $\tilde{q}_{\{\alpha,\phi\}}(W)$ with the reformulated ELBO, we can solve prior parameters $\alpha$ using (6.36) to yield a self stabilising-prior. We first choose our prior mean $\mu_{p_{i,j}}^l$ to preserve the mean during the forward pass and set it equal to the approximate posterior mean $\mu_{q_{i,j}}^l$. Secondly, we find we find $\sigma_{p_j}^l$ in (6.32), to satisfy the condition in (6.36). This gives the

stabilising prior parameters $\alpha$ as

$$\mu^l_{p_{i,j}} = \mu^l_{q_{i,j}}, \qquad \sigma^l_{p_j} = \sqrt{\frac{(\sigma^l_{\tilde{q}_j})^2\gamma}{(\sigma^l_{\tilde{q}_j})^2 - \gamma}}, \tag{6.37}$$

where $\gamma = |2 - (1 - 1/\pi)(\mu^l_{q_j})^2|$ and we take the absolute value to ensure positive variances (see Discussion 2 for an alternative formulation to avoid negative variances). Note that we can apply the result in (6.34) recursively for all layers $l = 1, ..., L$, with base case $\text{Var}[\boldsymbol{x}^0] = \frac{1}{D_0}(\boldsymbol{x}^0)^T \cdot \boldsymbol{x}^0$. Therefore, sampling the elements of $W$ as $w \sim \tilde{q}_\beta(w)$ at each forward pass, while setting the prior $p_\alpha(w_{i,j}) = \mathcal{N}(\mu^l_q, |(\sigma^l_q)^2\gamma/((\sigma^l_q)^2 - \gamma)|/D_{l-1})$, enables the network to simultaneously update our current posterior as well as promote stable signal propagation to improve robustness. Algorithm 1 gives an overview of the training procedure.

**Algorithm 1:** Algorithm to train BNN with signal stabilising priors.

1. Initialize $q_\phi(W)$ ;
2. Calculate $\alpha$ based on $q_\phi(W)$ to ensure stable signal propagation of $\tilde{q}_{\{\alpha,\phi\}}$ using equation (6.37);

**while** *Training* **do**

3. Compute forward pass for minibatch ;
4. Update $\phi$ using backpropagation on $\nabla_\phi \mathcal{L}_{\tilde{q}}$ (equation (6.17));
5. Calculate $\alpha$ based on $q_\phi(W)$ to ensure stable signal propagation of $\tilde{q}_{\{\alpha,\phi\}}$ using equation (6.37);

**end**

**Discussion 2: Alternative formulation augmenting Empirical Bayes**

We consider an alternative formulation of the prior in which we augment empirical Bayes (EB) [27] to eliminate situations yielding negative variances for the prior. We set the parameters of the distribution according to EB and exponentiate our prior distribution with $\beta$. Omitting the superscript $l$ for all parameters at the current layer we write the prior as

$$p(w_{i,j}) = \mathcal{N}(\mu_{q_{i,j}}, \sigma^2_{q_{i,j}})^\beta. \tag{6.38}$$

Thus on a forward pass the sampling distribution becomes

$$\tilde{q}(w_{i,j}) = \mathcal{N}(w_{i,j}|\mu_{q_{i,j}}, \sigma^2_{q_{i,j}})\mathcal{N}(w_{i,j}|\mu_{q_{i,j}}, \sigma^2_{q_{i,j}})^{\beta}. \tag{6.39}$$

Using the result that $\sigma^2_{\tilde{q}} = \frac{\sigma^2_q}{\beta+1}$ when the variances are equal, we find $\beta$ satisfying the condition $(1 - 1/\pi)(\mu^l_{\tilde{q}_j})^2 + (\sigma^l_{\tilde{q}_j})^2 = 2$ yielding

$$\beta = \frac{\sigma^2_{q_j}}{\gamma} - 1, \tag{6.40}$$

where $\gamma = 2 - (1 - 1/\pi)\mu^2_{q_j}$. We constrain $\beta \in [1, \infty)$ to yield a prior that default parameters are determined by EB, while in situations of larger variance $\beta$ increases and the prior becomes more active in encouraging sampling closer to the mean to stabilise signal propagation.

### 6.2.4 Discussion

The effect of the stabilising prior is shown in Figure 6.2. Intuitively, the larger the mean and variance of the incoming weights, the more likely it is to destroy the signal, whereas if the means are close to zero, it is not likely to add noise to the signal. The prior becomes active when the second moment of the distribution is large. Once active, the prior urges the weight distribution to sample closer to its means.

We examine signals in a ReLU network in Figure 6.3 to analyse the effect of the prior on the network signal propagation. We monitor the variance dynamics of the same data point throughout training by calculating the empirical variance of the vector of pre-activations at each layer during a forward pass. In Figures 6.3 (a) and (b) we show a controlled example, where we force our assumptions, setting biases and parameter means to zero and see that the prior preserves the signal, whereas in a standard BNN it explodes. Furthermore, we show a typical training scenario in Figures 6.3 (c) and (d), where we see that our assumptions hold in the early stages of training and start to break down later in training, yielding less stable signal propagation.

We opt to always include the prior during test time i.e. we sample from $\tilde{q}(W)$ instead of $q(W)$ because we find that the training accuracy of the model progresses faster requiring fewer training epochs. Since we require adjustment during the forward pass throughout training to ensure stable

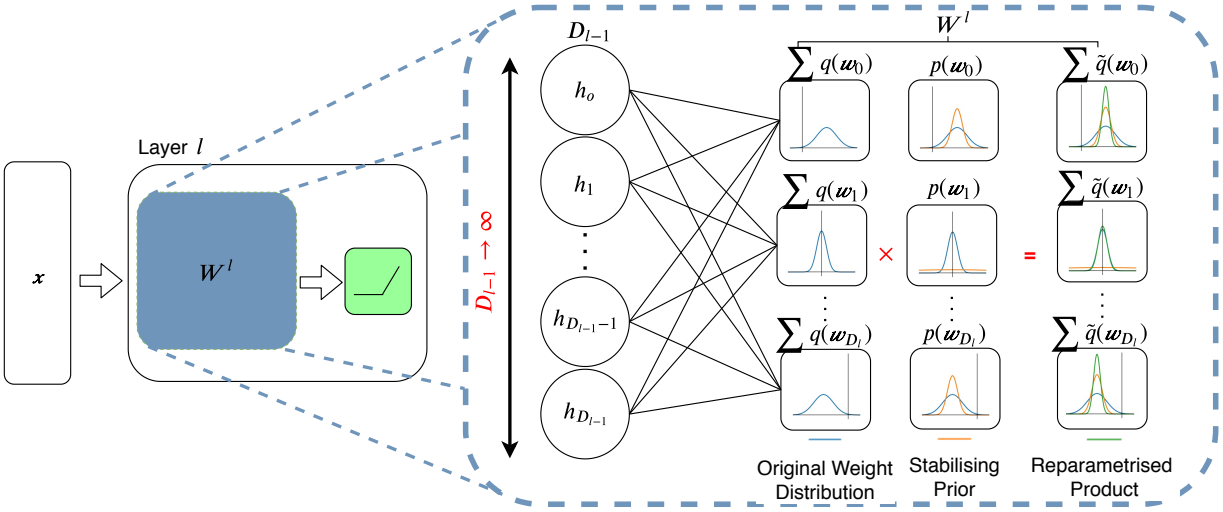



Figure 6.2: *Adaptive self stabilising prior.* At a layer we replace the original weight distribution, $q(W)$, with $\tilde{q}(W)$. This incorporates knowledge of how signals propagate through a network in the infinite width limit through the prior. The prior is optimal in the sense that it preserves the variance of signals being propagated forward through the network. It forms part of $\tilde{q}(W)$ that allows its signal-stabilising effect to be exerted on the forward pass that is beneficial for deep networks. The prior adapts based on the sum of weights feeding into the same hidden unit. The effect of this is that when the second moment of the weight distribution around the origin becomes large, it is likely to significantly contribute noise to the signal propagating through the network and destroy the signal. The prior becomes active when the variance increases and encourages the weight distribution to sample closer to its mean.

signal propagation, it seems reasonable to include it in the forward pass at test time. Note that the adjustment to the variational posterior for signal propagation is in this sense unlike the use of EB to choose the hyper-parameters maximising the likelihood, in such a case the prior should not be factored in and $q(W)$ would be appropriate. Using only $q(W)$ at test time exhibits performance similar to networks with unstable signal propagation. We investigate what effect including the prior at test time has on the quality of prediction uncertainty in the subsequent experiments section.

#### 6.2.4.1 Experiments

We have proposed a prior intending to stabilise the signal propagation in deep Bayesian ReLU networks. For this prior to be effective we make use of the reparametrisation $\tilde{q}(W)$ at each layer and investigate the impact with a series of experiments. The goal in our work is to develop a training scheme that is robust to a wide range of network depths and widths, with respect to the initial specification of the variational posterior parameters. This allows more freedom of choice of architecture and reduces the need to tune hyper-parameters. No tuning is necessary for the width, since it is incorporated in the derivation of the technique. We consider classification on

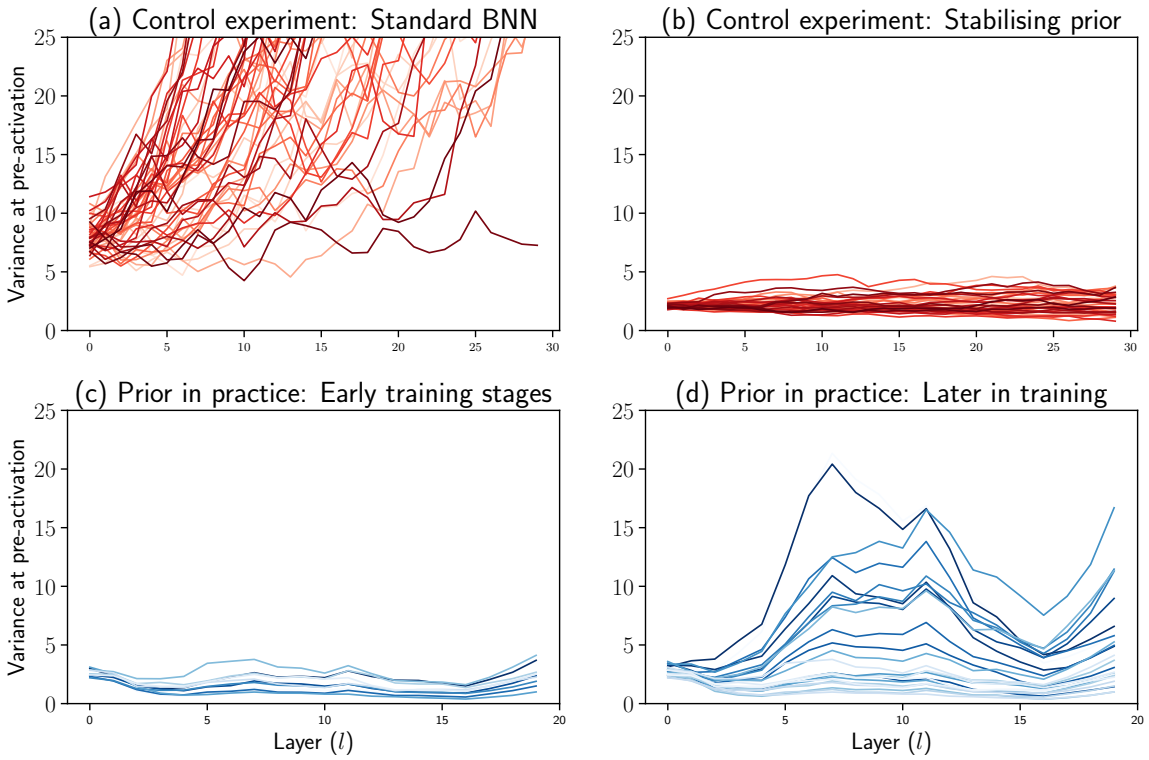



Figure 6.3: Signal propagation dynamics of the same signal propagated through different networks. We track the variance of a data point throughout training by calculating the empirical variance of the vector at the pre-activation at each layer. Lines are shaded from lighter, in early epochs, to darker in later epochs. In a controlled setting we achieve perfectly stable signal propagation. In practice our assumptions hold for the early stages of training.

MNIST and CIFAR-10 and also study the quality and calibration of the uncertainty estimates. We restrict ourselves to BNNs with fully-connected layers with a specified number of hidden layers, all of constant width. We initialise all networks with the He initialisation [25] and use a batch size of 1000. See Appendix C for a detailed discussion of reproducing these experiments.

**Limits of trainability.** We investigate performance at extremities by training a series of networks with hidden layer widths of 256 with varying depths and initial variances using 50 training epochs. We compare a series of networks with our proposed stabilising prior incorporated on the forward pass with a standard non-conjugate Gaussian prior [27] with small variance (the prior variance does not have much effect on the limits of training and only affects the extent to which the weights are regularised) that we report in Figures 6.4 (a) to (d). We observe our stabilising prior makes it possible to train deeper BNNs and in more noisy conditions. We further note that the signal explodes in standard BNNs deeper than 30 layers, failing to train.

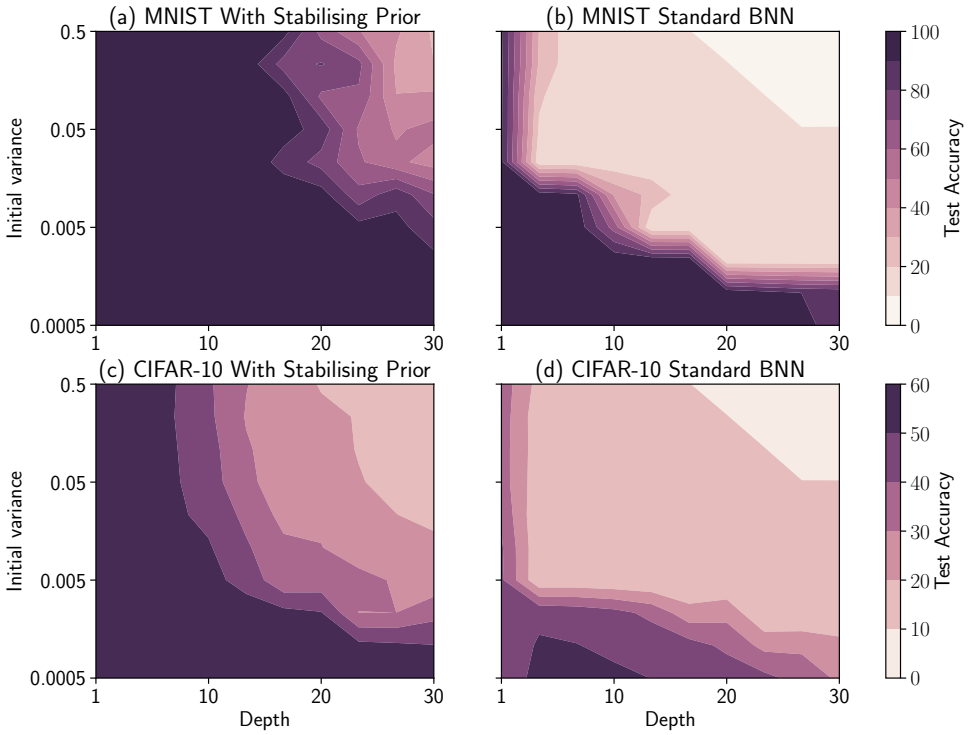



Figure 6.4: MNIST and CIFAR-10 large scale experiments. Classification accuracy grid of ReLU networks trained on MNIST and CIFAR-10 with varying depths and initial variance conditions.

**Accelerated training.** In general, we also observe that our prior increases the training speed as demonstrated in Figure 6.5. In this experiment, we compare with EB as in [27] that uses the gradient to find optimal hyper-parameters for the prior. We compare these priors with a regularising Gaussian prior and report their results for both the reparametrisation trick (RT)

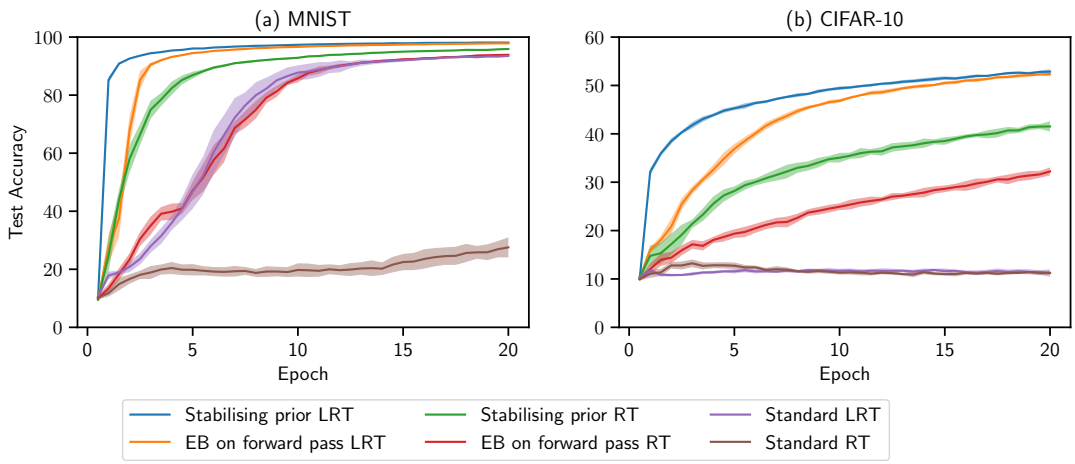



Figure 6.5: Progression of test accuracy for various networks through training averaged over 10 runs for a 5 layer deep 512 wide network with an initial variance of 0.001.

[12] and local reparametrisation trick (LRT) [20]. We observe empirically that incorporating the EB prior on the forward pass also accelerates training relative to leaving the influence of the prior to a KL term added to the loss. Our prior only outperforms EB in some settings. We find an advantage over EB with settings involving deeper networks, higher initial variance and wider networks, where the central limit theorem more strongly holds (see Discussion 2 for an alternative formulation of the prior augmenting EB).

**Quality of uncertainty.** Finally, we turn to the issue of what effect this prior has on uncertainty and calibration. We measure calibration with the Brier score and, similar to [56], the accuracy of predictions above 50% and 90% confidence to see whether our models tend towards overconfidence. We monitor the progression of these metrics of models with different priors through 100 epochs reported in Figure 6.6. As with any iteratively updating prior, we expect that it may adapt to the dataset and overfit, as is shown to be true of our stabilising prior and EB in Figure 6.6. As an answer to this we explore combining a regularising and stabilising prior that trains faster and results in a well-calibrated model with better Brier scores than any solitary prior.

### 6.2.5 Conclusion

We have used signal propagation theory to derive priors for BNNs that promote stable signal propagation. The prior incorporates knowledge of model architecture and activation function, ReLU in particular, derived from how signals propagate in the network in the infinite width limit. We showed that these priors, when their effect is exerted in the forward pass, makes it

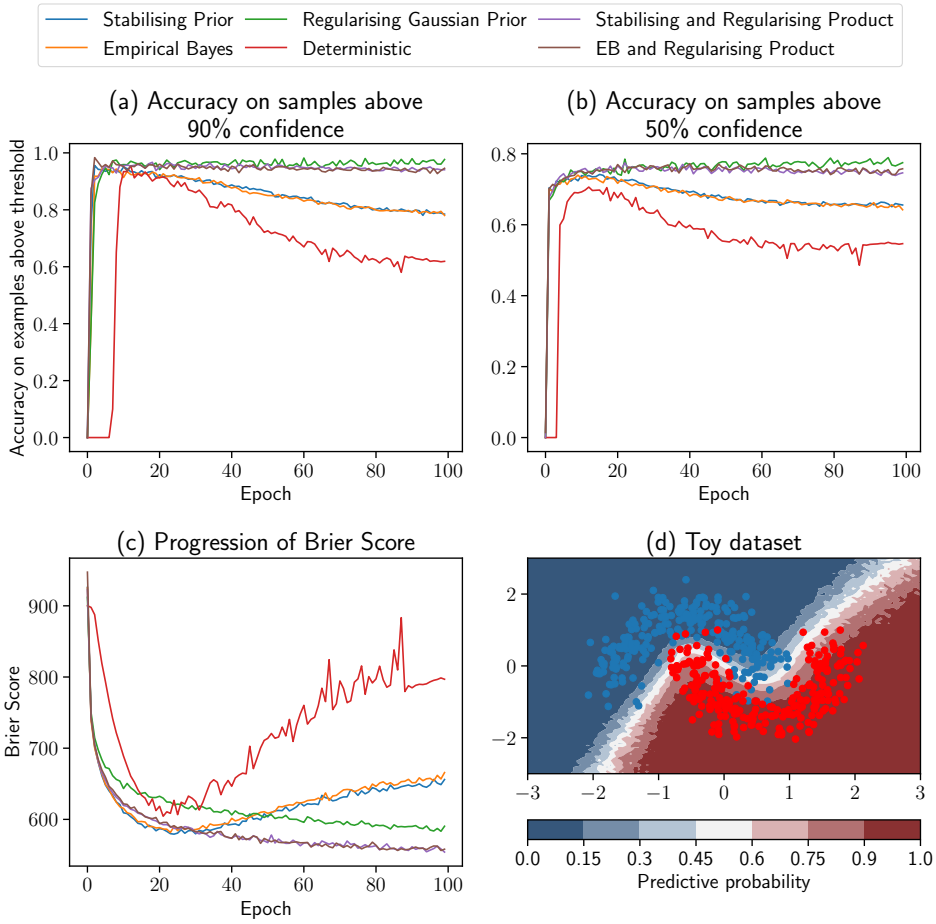



Figure 6.6: Uncertainty and calibration experiments on CIFAR-10. Iteratively updating priors overfit, however, we can combine regularising and optimal priors to maintain calibrated confidence and better Brier scores. In (d) we also see we are able to get reasonable uncertainty estimates with a deep neural network on a toy dataset.

possible to train deeper networks and in more noisy conditions. This alleviates the need to tune hyper-parameters and extends BNNs to deeper architectures. We also observe in general that stable signal propagation accelerates training, which we attribute to cleaner signals and gradients being propagated through the network with more efficient expectations.

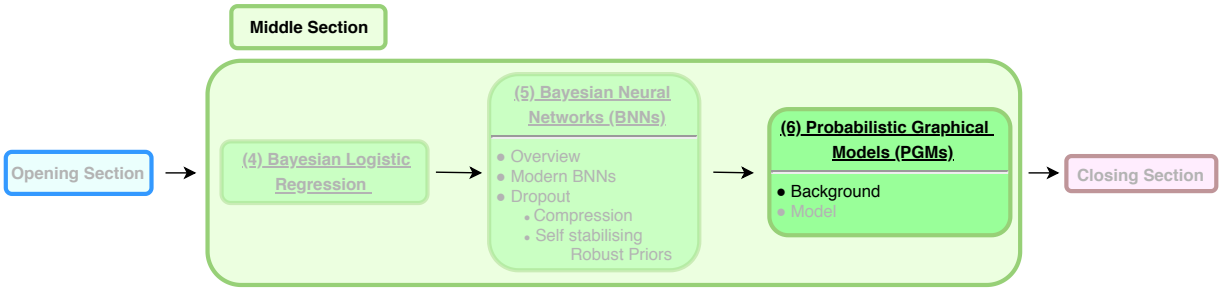
## 6.3 Conclusion

We have explored BNNs which are powerful estimators that leverage the predictive performance of modern deep learning with reliable uncertainty estimation or to sparsify models for more computationally efficient predictions. We discussed the advances in variational inference that have made it possible to scale Bayesian deep learning in the reparametrisation trick, local reparametrisation trick, the additive reparametrisation trick, MCVI and stochastic variational inference. We also presented our work on self stabilising priors which improves robustness in Bayesian deep learning.

# Chapter 7

# Spatial Integration with Probabilistic Graphical Models

Thus far we have discussed classification models to model the hyper-spectral signature of a pixel and assign each pixel to a class. We now turn our attention to the second task this thesis undertakes: integrating spatial information into the system. We explore combining contextual information, from neighbouring pixels, with hyper-spectral information in the pixel itself, from the predictions of the classifiers.




## 7.1 Introduction

In this chapter we make use of probabilistic graphical models (PGMs) to explicitly incorporate domain or prior knowledge into the system. PGMs offer a powerful framework for dealing with complex problems in Bayesian inference. Our approach follows the assumption that strong correlations exist between neighbouring pixels and measurement noise is commonly observed. We thus incorporate spatial information by allowing neighbouring pixels to influence the probability of a particular pixel. This is motivated by the observation that the output from our classification

models are often noisy mappings or flecked images. We may expect more continuous or smoother shapes as to more closely resemble real-world farms. Our system constitutes inputting the output probabilities of our classification model into the PGM that de-noises the image, i.e. inputting the noisy image where each pixel has been assigned a class into a PGM. This approach is common in image processing and plays a large role image segmentation and image de-noising [24]. Note this thesis is focused on the application of PGMs and we make use of the EMDW library [66], which automatically constructs and handles many of the requirements of cluster graphs, therefore we only discuss concepts key to understanding the mechanics and behaviour of PGMs before focussing on application and modelling.

## 7.2 Probabilistic graphical models (PGMs)

PGMs are graphical representations of our interpretations or models of the world, representing conditional dependencies among random variables. Statistical relationships between random variables are encoded by the graph as a structured probabilistic model. We represent a joint probability distribution and the structure represents how variables factorise with independence assumptions, i.e. what variables are independent of other variables given some information. This allows us to more efficiently calculate conditional probability distributions deriving from the full joint distribution. We can thus efficiently reason about large probabilistic systems. PGMs offer a modular language for representing probability distributions in an interpretable manner and allow us to use graph algorithms for inference and learning. This makes it possible to model and express many different dependencies between variables and automatically reason about them.

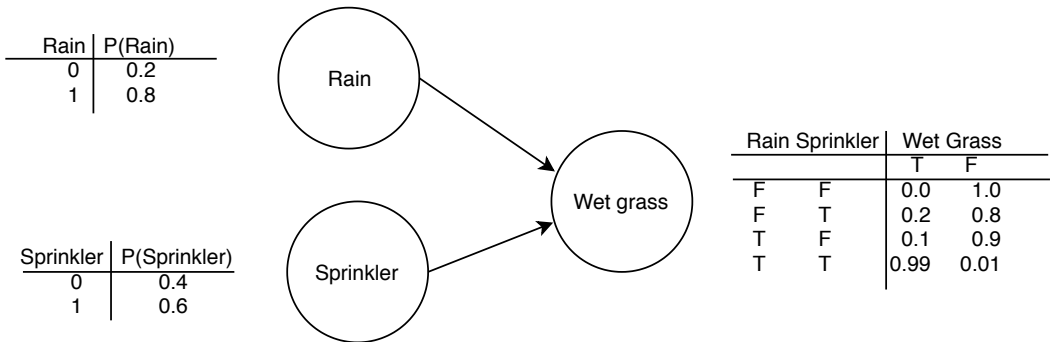



Figure 7.1: An introductory example to PGMs. The graph encodes the relationship between random variables as whether the grass is wet certainly depends on the weather and whether the sprinkler is on.

The nodes in PGMs correspond to random variables, and edges to the relationship between the random variables. In Figure 7.1 we show an example of a PGM, specifically a Bayesian net-

work, that encodes the local interactions in the form of conditional probability densities (CPDs). Briefly, Bayesian networks allow us to easily summarise more direct or causal relationships between variables. Bayesian networks do not necessarily always represent causal links but typically simplifies the understanding and parametrisation of the model. The joint probability is factorised into a product of CPDs using the product rule $p(A, B) = p(A|B)p(B)$ or can be written as $P(\text{Rain, Sprinkler, Wet Grass}) = P(\text{Rain})P(\text{Sprinkler})P(\text{Wet Grass}|\text{Rain, Sprinkler})$ and the resulting PGM is shown. We later make use of Markov networks, considered more general than Bayesian networks in the sense that they are not required to be acyclic. Markov networks are undirected and thus allow us to represent random variables with inter-dependent relationships, rather than causal relationships as with Bayesian networks, useful for the model we present later where we encode correlations between random variables and neighbouring pixels may influence and be influenced by each other. We only use Bayesian networks and Markov networks to aid in representation and use cluster graphs for inference. To reason about these variables we require *inference* techniques. We will discuss message passing techniques, where we pass messages between variables to disseminate evidence amongst correlated variables. These algorithms exploit the graph structure, allowing efficient inference consisting of smaller systems communicating about their combined outcome.

### 7.2.1 Cluster Graphs

Here we introduce a specific type of PGM called *cluster graphs* and the concepts necessary for representation and inference. These graphs are known for their ability to perform inference over problems with many inter-dependant random variables and excel at solving combinatorial type problems. We use cluster graphs as they are generally simple to construct and generalise other types of PGMs. Under the condition that the graph satisfies the running intersection property (RIP) (briefly discussed later), cluster graphs are not restricted to being acyclic and may contain loops, which is usually a concern in message passing algorithms.

A cluster graph is made up of *factors* that describe the relationships between random variables. This enables a cluster graph to compactly represent a total joint probability as the product of smaller factors. A factor is defined over a cluster of variables and determine how those variables are related. These factors are constructed to represent our assumptions or how we believe there to variable relationships. We only make use of discrete factors in this thesis. A discrete factor is represented by a table containing all possible combinations that the random variables can assume and their respective probabilities. An example can be seen in Table 7.1. The nodes of a

cluster graph consists of one or more factors. These nodes are connected into a graph structure by connections holding a subset of variables, called a sepset. Information about the random variables in the cliques is communicated through the sepsets. Next we discuss how cliques pass messages to their neighbours with messages, also encoded as factors, which produce approximate marginals.

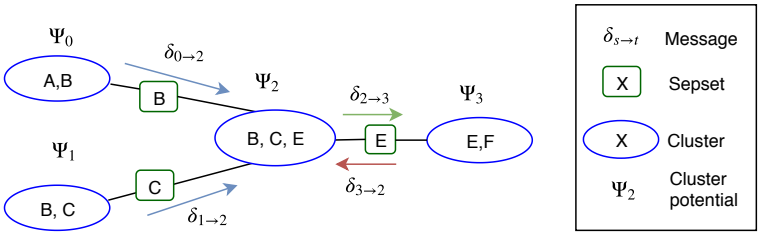



Figure 7.2: Cluster graph representing message passing. The message being calculated is from cluster 2 to cluster 3 represented by a green arrow. The messages included in the product are represented as blue arrows and the message from the target that needs to be excluded is shown as a red arrow.

### 7.2.2 Inference

Inference is done in PGMs by passing information between factors through connecting sepsets with one of the many PGM inference techniques. We first discuss the operations necessary to do inference, after which we discuss belief propagation.

#### 7.2.2.1 Factor Operations

In order to perform inference or pass messages in PGMs with the discrete factors, we require the following operations:

**Multiplication:** The product of two discrete probability tables involves calculating the product of the probabilities where the values of the shared random variables match. In cases when there are random variables that are not shared, we expand the table to include all the variables. The new entries create a Cartesian product (see [24] for more examples and details).

**Division:** Much like multiplication, we divide the probabilities of the random variables where the shared variables match and do not typically divide factors that do not share all the same variables.

**Marginalisation:** Similar to the continuous version of integrating out a variable, we sum out all entries of a particular variable. We sum the probabilities associated with the random variable

being marginalised such that there is one entry for all combinations of the other variables.

**Normalisation:** Normalising involves dividing all the probabilities in the table by the sum of the probabilities. This ensures that the entries sum to 1 and represents a valid probability table. Messages should also be normalised as the values can become extremely small due to many product operations between probabilities in the computation.

#### 7.2.2.2 Belief Propagation

We now introduce the sum-product belief propagation algorithm that is one of many variants of belief propagation. It is an iterative message passing algorithm introduced in [67] used to update beliefs or do inference on a loopy graph. It consists of a series of local message-passing rounds that change each clique's belief about its variables.

To compute an outgoing message $\delta_{s\to t}$ from a sending cluster, $s$, to a target, $t$, containing sets of corresponding variables $x_s$ and $x_t$, we multiply all the incoming messages from the neighbouring nodes and the cluster potential of the node itself $\Psi$. We then marginalise out the variables that are not shared by the receiving node so that they match the variables of the sepset. It is important to note that the message excludes the message $\delta_{t\to s}$ from the node that is receiving the message. We denote this with the backslash where the set of neighbours $s^{\setminus t}$ does not contain $\delta_{t\to s}$ in the set of neighbours of $s$. We can then write an outgoing message as

$$\delta_{s\to t} = \sum_{x_s} \left( \Psi_{st}(x_s, x_t) \prod_{i \in \text{neighbours } s^{\setminus t}} \delta_{i\to s} \right). \tag{7.1}$$

By excluding the receiving node's message, an outgoing message does not contain the information from the cluster that it is sending its message to such as to avoid a positive feedback loop. We may think of a message as containing all the information a sender can get from its neighbours about the target's probability, but leaving out the message that the target is sending to the sender so as to not become self-affirming. In general a cluster can only send a message when it has received all the messages from its neighbours. We show an example in Figure 7.2 where we show a message being sent from the cluster 2 to cluster 3.

We also make use of a variant called the belief update (BU) algorithm [68]. There is a minor difference in that instead of excluding the message from the target we include it in the multiplication and subsequently dividing it out. The difference is subtle as they appear as though they

should compute to the same value, we observe, however, that we often obtain slightly better results with the BU algorithm. We write these messages as

$$\delta_{s\to t} = \left[\sum_{x_s}\left(\Psi_{st}(x_s,x_t)\prod_{i\in\text{neighbours}}\delta_{i\to s}\right)\right]/\delta_{t\to s}. \tag{7.2}$$

Marginals obtained from message passing algorithms on tree structured graphs converge to the exact marginals. However many useful and meaningful PGMs contain loops. In graphs with loops, we can use loopy belief propagation (LBP) or update (LBU). These algorithms pass messages between clusters just like in BP, but do not require all incoming messages before sending messages. They iteratively pass messages until convergence which lead to approximate marginals. Loopy graphs have no guarantee of convergence [24], but are still widely and successfully used [69].

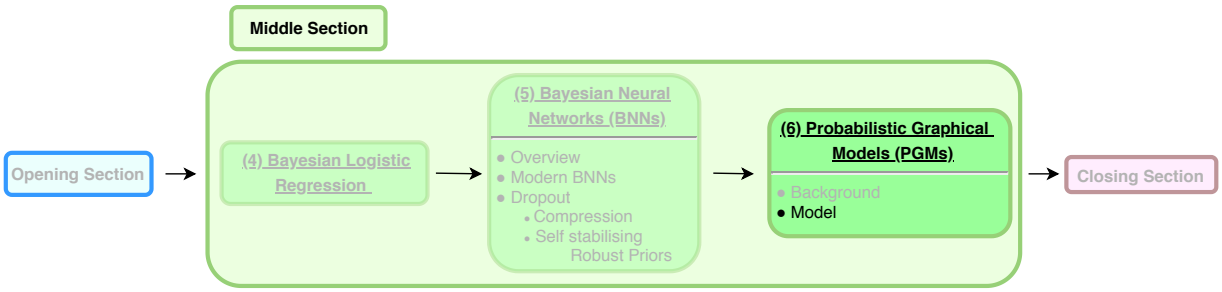
Once we have a graph that has converged, we may want to find the belief $B(x_i)$ of a specific variable $x_i$. This involves integrating out all the other variables from the cluster belief cluster containing $x_i$. Cluster potentials are defined as the product of all incoming messages from each neighbouring cluster with the cluster potential $\Psi_i(x_i)$ given as

$$B(x_i) = \Psi_i(x_i)\prod_{n\in\text{neighbours}}\delta_{n\to i}. \tag{7.3}$$

As the messages are passed in a cluster graph, they change the belief of the particular clusters. Convergence occurs when the belief about all variables of each cluster is equal to the belief about those variables of neighbouring clusters at each edge. This represents the shared belief of a variable by all the clusters once all the information has circulated. If the cluster graph satisfies the running intersection property (RIP), we ensure that there are no positive feedback loops. RIP requires that there may only be one unique direct path for which information about a random variable can take between pairs of clusters [24]. We make use of a C++ library, EMDW, which automatically constructs a cluster graph from a given list of factors and ensures RIP while also handling message scheduling and convergence [66].

## 7.3 Model

In our model we aim to incorporate our belief that there is a relationship with adjacent pixels using a PGM. A firm relationship is observed in data as we generally observe that farms are




clustered patches and have defined boundaries. Our model integrates spatial information into the per-pixel hyper-spectral classification result allowing us to infer what crop is growing where, based on each pixel's spectral fingerprint in the context of its neighbours.

We consider a grid of pixels, where each pixel is a random variable denoted by $X_{i,j}$ where $i$ and $j$ denote the row and column respectively. These represent discrete-valued random variables that reflect the true class of crop that the pixel belongs to. The initial value of a variable $X$ is defined by the class prediction made by a neural network or BNN for a particular pixel and the neural network is in this sense "connected" to the PGM model. We discuss how we can account for errors made by the neural network by augmenting the model in section 7.3.1. An example of pixels as random variables and the relationship between such a grid of random variables is shown with a Markov network in Figure 7.3 for a variable $X_{i,j}$ and its neighbours.

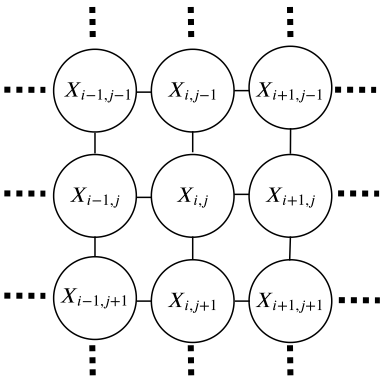
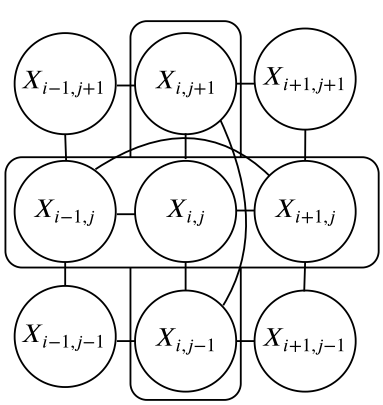


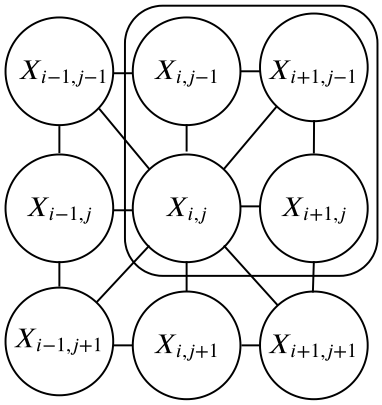

Figure 7.3: Pixels as random variables. We focus on a specific pixel or random variable $X_{i,j}$ in a grid of pixels as in an image.

We found it safe to assume that pixels more than one pixel apart have a negligible influence on each other. We thus consider a variable $X_{i,j}$ and only its adjacent neighbours. The relationship between the random variables is dictated by the graph structure and how we connect nodes. In figure 7.4 we represent a Markov network overlayed with two different methods with which we may select our factors to construct a cluster graph from which to do inference. In Figure 7.4 (a) we show how we might have two "triplet" factors for each variable $X_{i,j}$. One factor takes into account neighbours from above and below and the other left and right. Note that there are two

factors for each pixel in the image and the factor containing variables $X_{i,j}, X_{i,j-1}$ and $X_{i,j+1}$ would overlap with the set of neighbouring factors associated with $X_{i,j+1}$ and $X_{i,j-1}$ i.e. the factor for $X_{i,j-1}$ contains the variables $X_{i,j}, X_{i,j-1}$ and $X_{i,j-2}$. Each of these factors represent a probability table as given in Table 7.1. In Figure 7.4 (b) we introduce an "alternative" factor setup. In this configuration there are 4 factors associated with each variable $X_{i,j}$ including 4 variables each and differs from (a) by defining a relationship with diagonally adjacent pixels. The probability table for the alternative setup is constructed in a similar nature to triplet factor and is shown in Table D.1.




(a) Triplet factors. This configuration yields 2 factors per pixel including 3 variables each. Factors supplying the centre pixel with information from neighbouring pixels above and below and left and right. Probability table shown in Table 7.1.

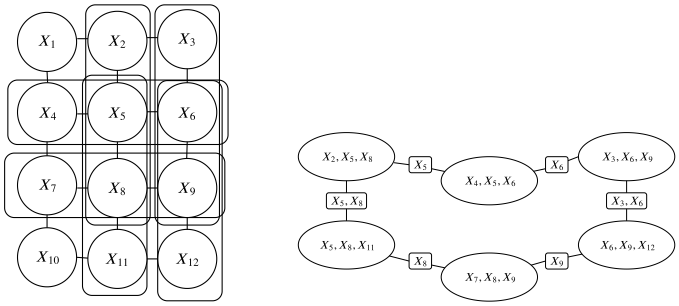



(b) Alternative factor setup. This configuration yields 4 factors per pixel including 4 variables each. Factors allowing diagonal neighbours to also influence the centre pixel. Probability table shown in D.1.

Figure 7.4: Alternative factor setups overlayed on Markov network. We mostly make use of the triplet factor setup but the alternative factor setup produces very different behaviour, enforcing very strong smoothing, which we compare in the experiments section.

In Figure 7.5 we show a simplified example of how one would construct a cluster graph given a list of factors for a largely reduced model similar to the setup way we have described. Given a grid of variables as shown in Figure 7.5 (a) we show the corresponding cluster graph in Figure 7.5 (b). Note that, with the use of EMDW, the cluster graph is constructed automatically while satisfying RIP. We believe Markov networks are more intuitive and interpretable in representing relationships between grids of variables and we can return to Markov networks for representation with either factor setup.

The probability table for our factors, as in Table 7.1 for the triplet factor and Table D.1 in Appendix D, represent our prior beliefs about the relationship between a pixel of interest and its neighbours either above and below or left and right. For our example in Figure 7.4 (a) we




(a) Grid of random variables with reduced example of how factors make up relationships between random variables.

(b) Example converted to cluster graph.

Figure 7.5: Constructing cluster graph from list of factors. A list of factors, such as the groupings in the example in (a), would be provided to EMDW which automatically constructs a valid cluster graph as shown in (b).

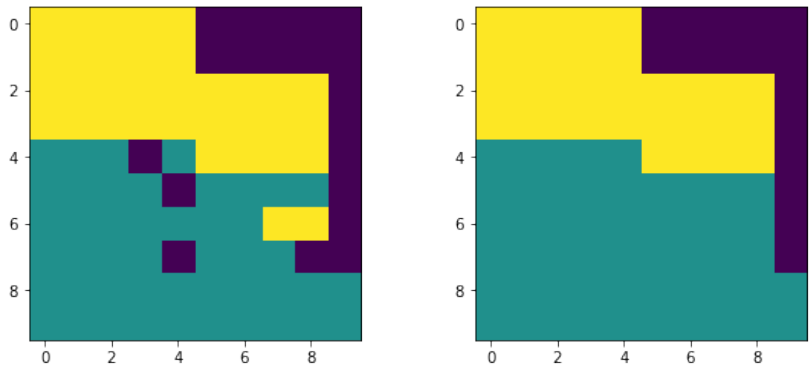
represent the probability of a particular pixel, $X_{i,j}$, given information from two adjacent pixels.

We may interpret our the probabilities in the factor table as one of three cases: (1) the current pixel belongs to the same class as both its neighbours, or it agrees with both of its neighbours that we believe to have a high probability; this may reflect a pixel in the middle of a field or a mono-culture entity; (2) the current pixel belongs to the same class as one of its neighbours, this may represent a border and is reasonably probable; (3) the current pixel does not belong to the same class as either of its neighbours; this we believe to be unlikely and a result of noise in the system. The same reasoning follows for the factor configuration Figure 7.4 (b) by including a corner pixel, extending it to four variables, which slightly increases probabilities when the diagonally adjacent pixel is also in agreement with the pixel of interest shown in Table 7.4. Once we have the factors, we use loopy belief propagation to infer a new image. Figure 7.6 demonstrates the PGM applied on a toy example and the intended de-noising effect.

It is possible to learn the factor probabilities from data if we had access to more data. However, we expect that the relationships to vary greatly between images and the ability to select the prior probabilities allows flexibility for the designer. The probabilities should, however, reflect the belief of the amount of noise in the system as well as the confidence we have in the probabilities of our predictions from our classification model. We find that substituting the exact values, i.e. empirically calculated percentages of how many pixels agree from the ground truth image, does not have a noticeable impact.

| $X_{i,j}$ | $X_{i,j-1}$ | $X_{i,j+1}$ | $\phi(X_{i,j}, X_{i,j-1}, X_{i,j+1})$ |
|---|---|---|---|
| 0 | 0 | 0 | 0.35 |
| 0 | 0 | 1 | 0.1 |
| 0 | 1 | 0 | 0.1 |
| 0 | 1 | 1 | 0.05 |
| 1 | 0 | 0 | 0.05 |
| 1 | 0 | 1 | 0.1 |
| 1 | 1 | 0 | 0.1 |
| 1 | 1 | 1 | 0.35 |

Table 7.1: Factor containing agreement probabilities normalised over the whole table. Each random variable is assigned the probability given that it belongs to a certain class. The first and last rows represent where the middle pixel belongs to the same class as its neighbouring pixels. The second and third rows represent where the centre pixel agrees with or belongs to the same class as one of the adjacent pixels. This represents a border or boundary and is not as likely to be true as represented 0.1 probability. The fourth and fifth rows represent where the middle pixel believes it belongs to a different class than both of its neighbours, this is not as likely but because we have set up our factors only vertical or horizontal this may still come up. This can be extended to multi-class where $X_{i,j}$ has a belief that it belongs to any of the $K$ classes.




(a) Noisy example before PGM.

(b) Cleaned example after PGM.

Figure 7.6: PGM applied to a toy example. Factors for each each pixel are constructed as discussed and inference is done noisy image such as in (a) to result in a cleaned image (b).

### 7.3.1 Augmentation for Calibrated Inputs

If we can be sure that we can trust the confidence of our predictions of our classification models we can augment our models with another variable that we show in Figure 7.7. As previously discussed $X_{i,j}$ represents the true class or crop type that a pixel represents that we wish to infer. The initial value this variable takes is assigned by the classification of a neural network. We now introduce a new variable $Y_{i,j}$ for each pixel that represents the observed pixel. Our classifier then estimates $P(X_{i,j}|Y_{i,j})$. This represents the classifier's confidence or belief and can also be interpreted as the prior error rate or the belief of the likelihood of an error by the neural network. Using the softmax probability of the neural network to estimate $P(X_{i,j}|Y_{i,j})$ represents the probability of a prediction for $X_{i,j}$ is erroneous and allows the PGM to probabilistically reason whether or not it should change the belief that a pixel belongs to a specific class. Having calibrated models is useful as the system is then able to determine when a pixel may need to rely more on the information supplied from adjacent pixels. In general we would follow an approach similar to this augmentation where we may specify some prior over each variable that reflects our belief that the classifier is correct. In this general approach, using the probability assigned by the classifier as a prior, when the model is not calibrated, may lead to errors downstream in the decision making process as we will see in the experiments chapter. In this case we may opt to use the training accuracy as a blanket prior belief of an error rate accounting for incorrect predictions instead. This augmentation only makes sense when a model is calibrated and differs slightly to better incorporate uncertainty from the classifier where we can trust the uncertainty estimates.

Usually, the graph structure indicates the relationship to some underlying true value, $X$, of which we are only able to observe via some measurement $Y$. More formally, we observe $Y$ and normalise obtaining

$$P(X|Y = y) = P(Y = y|X)P(X)/Z, \tag{7.4}$$

which translates to a likelihood and prior over $X$. However, with a classifier we are now simply *directly* estimating $P(X|Y = y)$ and implicitly includes the prior over $X$.

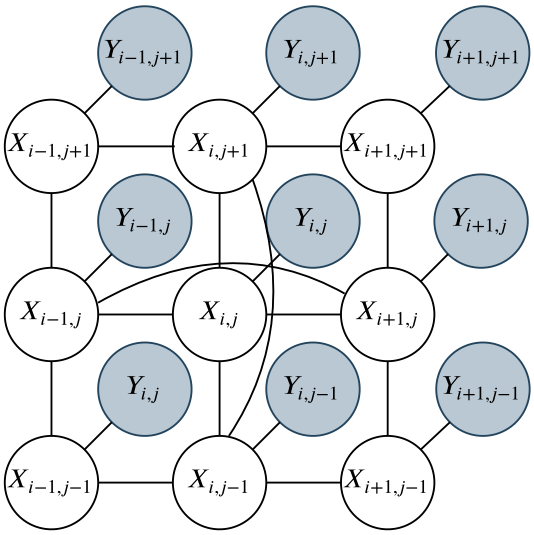



Figure 7.7: Classifier output as estimating $P(X|Y = y)$. The classifier models this conditional probability directly and is connected via an undirected link to the associated latent pixel class variable. Shaded nodes represent observed variables in, $Y_{i,j}$ which represents some unreliable measurement process to the true variable $X_{i,j}$. The factor for relating $X_{i,j}$ and $Y_{i,j}$ by observing $Y_{i,j}$ is shown in Table 7.2

| $X_{i,j}$ | $P(X_{i,j}\|Y_{i,j})$ |
|---|---|
| 0 | classifier$(P(X_{i,j} = 0\|Y_{i,j}))$ |
| 1 | classifier$(P(X_{i,j} = 1\|Y_{i,j}))$ |
| 2 | classifier$(P(X_{i,j} = 2\|Y_{i,j}))$ |
| $\vdots$ | $\vdots$ |
| $K$ | classifier$(P(X_{i,j} = K\|Y_{i,j}))$ |

Table 7.2: Probability table of an augmented model for calibrated inputs observing $Y_{i,j}$ the feature vector or spectral fingerprint associated with pixel $X_{i,j}$. The conditional takes the predictive probability of the classifier, in effect observing the variable $Y_{i,j}$ and making a prediction, serving as a noisy measurement to the true value $X_{i,j}$.

### 7.3.2 Graph structure

We can arrange the factors in many ways to suit different purposes but we are concerned with accurate and realistic de-noising and, since inference can be very computationally expensive, computationally efficient solutions. With these goals in mind we experimented with the factor setup shown in Figures 7.4 (a) and (b).

#### 7.3.2.1 Factor Setup for Loopy Graphs

Considering the two factor configurations shown in Figures 7.4 (a) and (b), we might expect that (b) offers an advantage by allowing us to incorporate knowledge from pixels diagonally adjacent. However, the variable overlap is large, creating many small loops in the graph structure. For (a),

the number of messages sent between when a cluster is scheduled to send messages is larger than (b). This makes for more variable interactions or space between loops and reduces the severity that introducing loops into the graph has on inference and convergence. In other words more messages are sent between sending a message and receiving a message again. This allows a greater opportunity for the evidence and information to disseminate through the graph. This presents favourable conditions for loopy graphs and aids in convergence and approximation accuracy.

#### 7.3.2.2 Factor trick

If each pixel can assume $K$ discrete values, or $K$ different classes, the factor $P(X_{i,j}, X_{i,j-1}, X_{i,j+1})$ would have $K^3$ different probabilities. For the 17 classes of Indian Pines, this scales unmanageably for most computers. It is possible to split the problem into smaller sections to be processed separately. Alternatively, we may reduce the problem to a binary class inference problem or a one-vs-all approach, whereby we run inference on a map with the aim of classifying, for example, wheat farm or not wheat farm. This may still be useful for some applications but we present a simple and effective strategy involving a different factor configuration to scale to many classes.

We aim to describe the factors more compactly while maintaining the same net effect by introducing two new latent variables. We introduce the variable $A_{i,j}$ which represents the agreement between two pixels next to each other. This variable can only take the value 1 if the classes of $X_i$ and $X_j$ are the same as shown in Table 7.3. For all combinations where neighbours do not agree $A_{i,j}$ will have value 0. This implies a latent variable interpreted as representing agreement when adjacent pixels belong to the same class and takes the value 1 or "agree", and "disagree" when adjacent class labels differ and $A_{i,j}$ takes the value 0. We then reason about $X_{i,j}$ in terms of $A_{i,j}$ and $A_{j,k}$ for which the probability table is shown in 7.4. The variable $A_{j,k}$ acts similar to a "flag variable" or latent "on or off" variable. We show a Bayesian network representation in Figure 7.8 to represent these "agreement variables" and the relationship with the factors originally posed. The directed line from $X_i$ and $X_j$ to $A_{i,j}$ indicates the probability $P(A_{i,j}|X_i, X_j)$.

The probability setup now contains factors of size $2K^2$ and $2^2K$ resulting in a more compact representation. Furthermore, we also do not store values with probability zero resulting in a sparser representation and saving even more space. The new graph with factors $P(A_{i,j}|X_i, X_j)$ and $P(A_{i,j}, A_{j,k}, X_j)$ contain only $2K^2 + 2^2K$ different probabilities sized factor per pixel instead of $K^3$ scaling well for an arbitrary number of classes with analogous behaviour to Table 7.1.

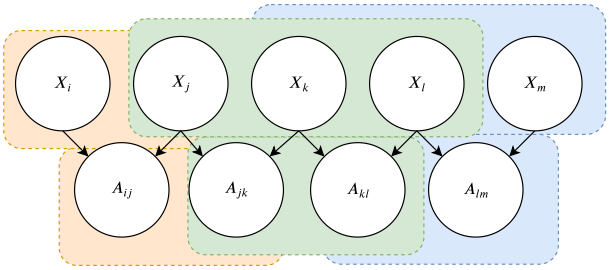



Figure 7.8: Latent agreement variable for saving space for multi-class variables. The variables $A_{i,j}$ and $A_{j,k}$ represent the agreement between $X_i$ and $X_j$ and takes either a value of 1 if they belong to the same class or 0 if neighbouring pixels do not belong to the same class. A new factor to reason about $X_j$ is shown in Table 7.4 which depends on $A_{i,j}$ and $A_{j,k}$. The colours show the variables related to the original triplet factor setup.

| $X_i$ | $X_j$ | $A_{i,j}$ | $P(A_{i,j}\|X_i, X_j)$ |
|---|---|---|---|
| 0 | 0 | 1 | 0.111 |
| 1 | 1 | 1 | 0.111 |
| 2 | 2 | 1 | 0.111 |
| 0 | 1 | 0 | 0.111 |
| 1 | 0 | 0 | 0.111 |
| 0 | 2 | 0 | 0.111 |
| 2 | 0 | 0 | 0.111 |
| 1 | 2 | 0 | 0.111 |
| 2 | 1 | 0 | 0.111 |
| others | | | 0.0 |

Table 7.3: Probability tables of latent agreement variables for 3 classes normalised over the whole table. The agreement variable takes the value 1 when the classes of $X_i$ and $X_j$ agree, as shown in the first 3 entries, and 0 when they are different.

| $A_{i,j}$ | $A_{j,k}$ | $X_j$ | $P(A_{i,j}, A_{j,k}, X_j)$ |
|---|---|---|---|
| 0 | 0 | 0 | 0.35 |
| 0 | 1 | 0 | 0.1 |
| 1 | 0 | 0 | 0.1 |
| 1 | 1 | 0 | 0.05 |
| 0 | 0 | 1 | 0.35 |
| 0 | 1 | 1 | 0.1 |
| 1 | 0 | 1 | 0.1 |
| 1 | 1 | 1 | 0.05 |
| 0 | 0 | 2 | 0.35 |
| 0 | 1 | 2 | 0.1 |
| 1 | 0 | 2 | 0.1 |
| 1 | 1 | 2 | 0.05 |

Table 7.4: Table of agreement using latent variables for 3 classes. Unnormalised to show correspondence to Table 7.1. This configuration scales better for multiclass problems yielding tables of size $2^2K$ for $K$ classes.

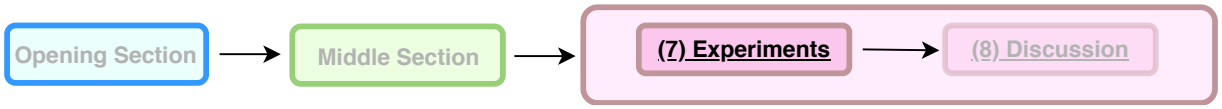
## 7.4 Conclusion

We showed how to implement a PGM to incorporate spatial information in a satellite image. We built a model that, for each pixel, we can reason about its probability given neighbouring pixels and the belief that those pixels belong to a specific class. We also saw that we can use the confidence of our classification models as an indication on whether we should rely more on spectral or spatial information in determining what class a pixel belongs. We can thus leverage models which preserve uncertainty to offer an advantage over standard filter techniques.

These methods are relatively computationally expensive and are likely not feasible for on-board satellite computation. This framework may be more useful for analysis by offering flexible and interpretable modelling, providing a means of inserting domain specific knowledge. We could increase the probability of a specific class given that we know what region an image is taken from and what typically grows there. We may also have some idea of the shape of a farm and we can adjust that area of individual priors to reflect this.

# Chapter 8

# Experiments




We now consider a series of experiments investigating whether the methods we developed are in alignment with our objectives of designing accurate models that generalise well, while remaining aware of uncertainty and considering limited computational capacity. We compare logistic regression and Bayesian logistic regression as well as deterministic neural networks with a series of BNNs and observe that Bayesian methods excel in situations where data is scarce. We then study the quality of uncertainty of the different methods measuring calibration and accuracy as a function of confidence. We qualitatively analyse the effect of using a PGM to integrate spatial information or de-noise images and study how uncertainty aids the PGM in reasoning about pixel classes. Finally, we compare model compression techniques using BNNs such that we can feasibly deploy these models on a satellite.

## 8.1 Method

We make use of the Indian Pines dataset and compare: (1) BNNs with self-stabilising priors; (2) normal BNNs with Gaussian priors and posteriors; (3) MC dropout, each layer with a binary dropout rate of 0.5; (4) deterministic neural networks; (5) Bayesian logistic regression and (6) deterministic logistic regression. In the compression section we compare the various BNN compression techniques we have discussed. We evaluate accuracy, quality of uncertainty, a qualitative assessment of PGM de-noising and computational complexity. In all the experiments we collect

5 different samples over 5 different folds of the data and present the average. If not specified, models are trained on a 75 % training and 25 % test split.

The goal is to verify claims of contrasting methods rather than searching optimal model configurations, so for the neural network models we restrict ourselves to fully-connected layers with three hidden layers of width 512 to facilitate the comparison. We use the ReLU activation, He initialisation [25], the Adam optimiser with a learning rate of 0.001, a batch size of 100 and train for 20 epochs unless otherwise specified.

## 8.2 Accuracy

We compare the accuracy and data efficiency of various classifiers discussed in this thesis by measuring the prediction accuracy on the test set with various test and train splits of the data. We report the average and variability over 5 different folds in Figure 8.1. Accuracy represents the modelling power and expressivity of a model whereas comparing different test-train splits reflects a model's capacity to efficiently learn from data and attempts to investigate the model's ability to generalise. As seen in Figure 8.1 neural network models generally achieve significantly better predictive performance, while Bayesian methods excel when data is scarce.

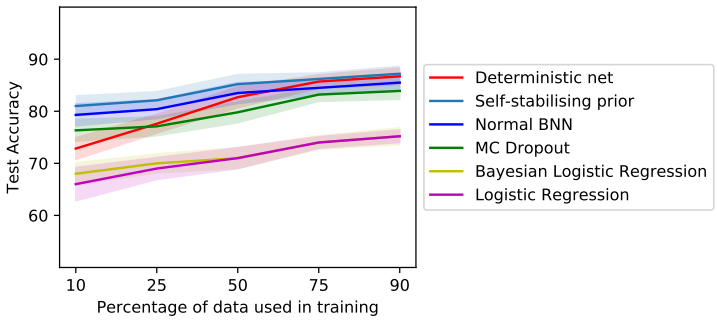



Figure 8.1: Data efficiency of various models. Accuracy measured of different models with different train-test splits averaged over 5 different folds. We plot the mean performance with shading depicting the standard deviation.

As a further investigation, we inspect the quality of performance that different accuracies represent in comparison with the ground truth in Figure 8.2. In addition, while not of critical importance, we show rates of convergence for different models in Figure 8.3, demonstrating again the improved convergence of our proposed self-stabilising prior relative to other BNNs.

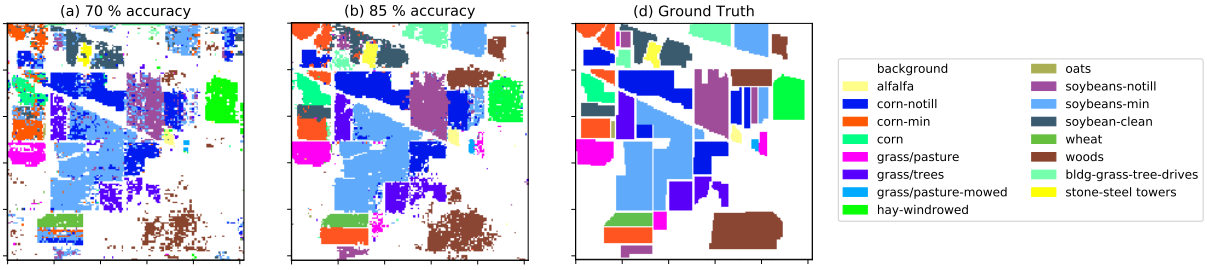



Figure 8.2: Qualitative assessment of accuracy. Different accuracies achieved by providing a neural network with varying amounts of exposure to the training data to intuitively demonstrate the usefulness of models of certain accuracies. Each pixel is assigned a class where the colour represents a type of crop and white represents background or no particular crop.

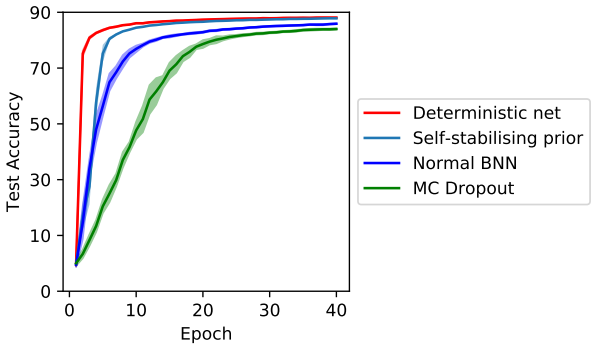



Figure 8.3: Convergence of different neural networks. Progression of test accuracy of different neural networks through epochs averaged over 10 runs. The self-stabilising prior presents the best convergence of probabilistic techniques.

### 8.2.1 Quality of Uncertainty

In order to quantify the quality of uncertainty of different models, we use calibration as a measure of uncertainty. Accurate calibration represents a model that can correctly assign more confidence in its correct predictions and less confidence in its errors. This gives us a model that is robust to overfitting and able to generalise well. We compare Brier scores that is general measure of accuracy of probabilistic predictions for categorical outcomes, and the accuracy of a model for its predictions above certain confidence thresholds similar to [56] to measure whether models tend towards overconfidence. As we will see, the Bayesian methods are better calibrated than their deterministic counterparts.

Uncertainty, in the context of classification, refers to the output softmax probabilities. This describes the certainty with which a particular input is assigned to a class. A model is usually assumed to be calibrated when predictions with prediction probability $p$ are correct $p$ percent of the time. The predictive probability then accurately reflects the accuracy of the model. We measure accuracy as a function of confidence, similar to [56], to evaluate the usefulness of predictive uncertainty. In this experiment the model is evaluated only on cases where the model's confidence is above a threshold. Given a prediction $p(y = k|\boldsymbol{x})$, we define the prediction as $\hat{y} = \underset{k}{\text{argmax}} \quad p(y = k|\boldsymbol{x})$ and confidence as $p(y = \hat{y}|\boldsymbol{x})$. We then measure the accuracy of predictions above 50%, 70% and 90% confidence thresholds. We also measure the Brier score, assuming one-hot encoding, given as

$$\text{BS} = \frac{1}{N}\sum_{n=1}^{N}(p(\boldsymbol{y}_n|\boldsymbol{x}_n, W) - \boldsymbol{y}_n)^2 \tag{8.1}$$

where $p(\boldsymbol{y}_n|\boldsymbol{x}_n, W)$ is the probability forecast and $\boldsymbol{y}_n$ is the actual label and lower scores are more accurate with 0 being perfect. We report the Brier score as a percentage i.e. $BS \times 100$, since it will always be less than 1. This measures the accuracy of probabilistic predictions in a classification setting.

The calibration metrics of the various models are reported in Table 8.1. We see that Bayesian methods are generally well calibrated. BNNs with self-stabilising priors exhibit the best scores among most metrics that we attribute to these measures having a strong connection with accuracy, i.e. an incorrect prediction strongly negatively impacts the Brier score. The calibration and accuracy of the deterministic network suffers in comparison with BNNs as it tends to overfit and requires regularisation. We also observe in general experimentation that the larger the size

| **Neural Networks** | Accuracy above confidence threshold | | | Brier Score | Prediction Accuracy |
|---|---|---|---|---|---|
| | 50 % | 70 % | 90 % | | |
| Regular BNN | 85.4 | 92.2 | 94.4 | 22.8 | 84.1 |
| BNN with self-stabilising priors | 86.2 | **93.7** | **98.1** | **21.9** | **85.0** |
| MC Dropout | **88.5** | 91.4 | 92.2 | 26.3 | 80.9 |
| Deterministic Neural Net | 84.3 | 85.5 | 95.0 | 28.5 | 84.5 |

| **Logistic Regression** | Accuracy above confidence threshold | | | Brier Score | Prediction Accuracy |
|---|---|---|---|---|---|
| | 50 % | 70 % | 90 % | | |
| Bayesian Logistic Regression | **89.6** | **91.0** | 93.1 | **33.4** | **75.2** |
| Logistic Regression | 81.7 | 88.8 | **94.4** | 33.9 | **75.2** |

Table 8.1: Measuring the quality of uncertainty of classifiers.

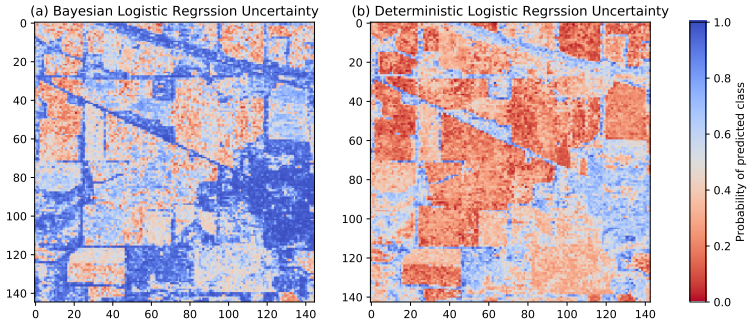



Figure 8.4: Qualitatively analysis of uncertainty output of logistic regression compared to Bayesian logistic regression with 25 % of the data used in training.

of a deterministic network, the less calibrated the network, while the calibration of BNNs do not deteriorate, but rather begin to suffer from more gradient variance. Interestingly, MC dropout achieves the highest accuracy for predictions above 50 % confidence, although it achieves the lowest overall accuracy of neural network models. We suppose this to be due to the noise injection causing the model to struggle to concentrate on a sensible posterior.

We also qualitatively show the uncertainty output of logistic regression compared to Bayesian logistic regression with 25 % of the data used in training in Figure 8.4. We see that Bayesian logistic regression, by maintaining a distribution over the parameters, appears to better capture and explain patterns in this low data setting.

## 8.3 Spatial Information Integration

Here we investigate integrating spatial information into the system with a PGM and compare the effect that different PGM configurations have on smoothing or de-noising an image. We also

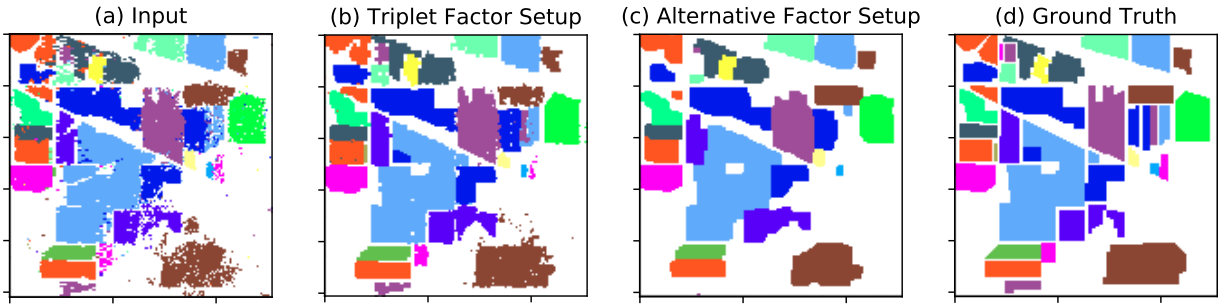



Figure 8.5: Comparison of PGM performance.

investigate the role of uncertainty in the reasoning process of a PGM.

In Figure 8.5 we show representative examples of how the different factor configurations differ (the factor configurations discussed in Section 7.3 and illustrated in Figure 7.4). We compare the triplet factor setup, where each pixel bears a factor allowed to communicate information with neighbouring pixels above, below, left and right; and the alternative factor setup that allows diagonal neighbours to influence the centre pixel. The triplet configuration fails to completely clean the image and some speckles remain but provides useful spatial integration. Alternatively, we see that the alternative factor setup produces very smooth images, but often connects gaps or completely discards small patches.

Next, we compare the outputs of the augmented PGM for calibrated inputs (discussed in Section 7.3.1). The classifier makes predicts the probability of each pixel belonging to each of the classes. The PGM takes this vector of probabilities and assigns them to the probabilities of factors before inference is done. This considers using the probability of the classifier as the likelihood of an error allowing the PGM to probabilistically reason whether or not it should change the belief that a pixel belongs to a specific class. We compare this using the same augmented PGM with inputs being generated by either deterministic or probabilistic models. We train a neural network and BNN on 90 % of the training data for 50 epochs such as to encourage overfitting to demonstrate extremes of contrasting approaches. We show the outputs of the augmented PGM with a triplet factor setup in Figure 8.6. We see that the PGM struggles to clean the output of the deterministic network as the network starts to grow overconfident in its erroneous predictions. The PGM assumes that predictions with high accuracy are correct and does not change their class despite its neighbours belonging to a different class. Observing the output from a probabilistic model, we conclude that a well-calibrated model with this augmentation assists the PGM to dynamically rely more on either spatial or spectral information. Furthermore, we show in Figures 8.7 and 8.8 how the system makes use of uncertainty in reasoning by observing how uncertainty changes from the output of the classifier to after inference of the PGM. We

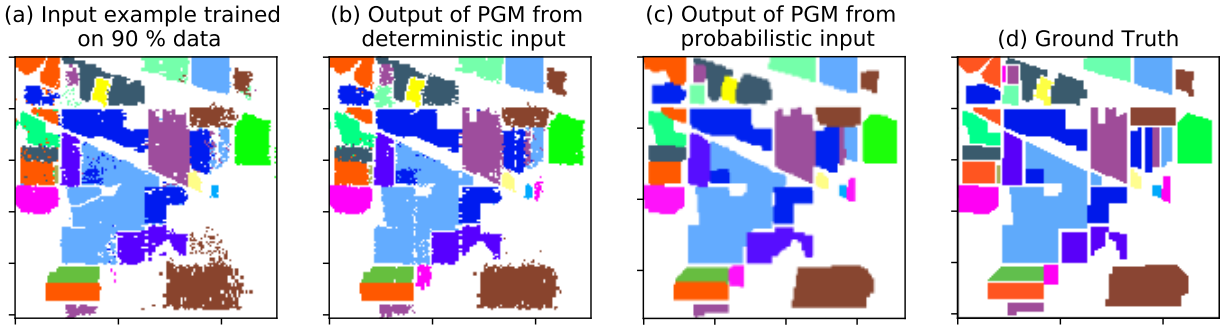



Figure 8.6: Comparison of augmented PGM performance. (a) shows an example of predictions made by a model trained on 90 % of the available data. We compare performance of the PGM when given class probabilities generated from a deterministic neural network in (b), where predictions tend to be overconfident, with a BNN in (c). We see that uncertainty awareness helps the PGM reason.

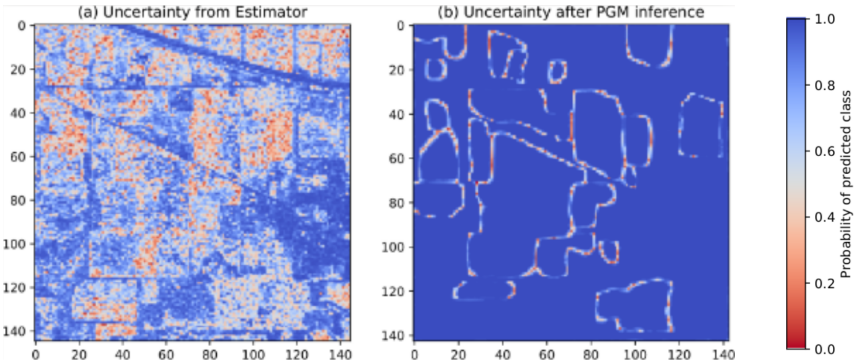



Figure 8.7: Uncertainty of Bayesian logistic regression before and after PGM inference.

see that once PGM inference is complete, the uncertainty lies on the boundaries between farms, offering some transparency to the reasoning process of the PGM.

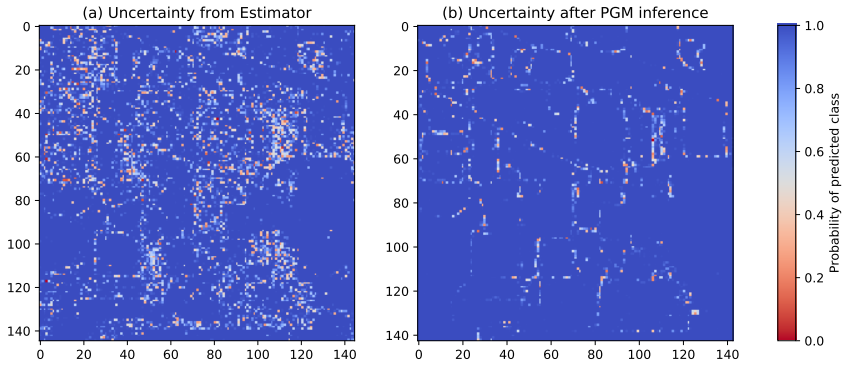



Figure 8.8: Uncertainty of a BNN before and after PGM inference.

## 8.4 Computational Complexity

Here we compare the BNN compression techniques studying: (1) SNR pruning, removing weights with a low signal to noise ratio or weights with high variance relative to their means; (2) pruning weights with a high likelihood of being zero; (3) variational dropout with the additive reparametrisation trick. We measure the percentage of remaining weights after weights deemed to be irrelevant have been removed. This translates directly into required storage space and the amount of computations necessary to make predictions. We see that variational dropout achieves the most compression with only 3 % of the original weights remaining. Variational dropout, however, sacrifices a small amount of accuracy while SNR pruning achieves the highest accuracy and calibration, but the least compression at 14 %.

For pruning weights with a high probability of zero we sort weights by how likely each weight is to be zero calculating $p(w = 0|\mu, \sigma^2)$ according to Gaussian probability density function. Then we incrementally remove the weights, starting with those with most mass on zero until we observe a decline in prediction accuracy. Similarly, for SNR pruning, we sort by lowest SNR ratio to highest and remove parameters until predictive performance declines. For both these models we used non-conjugate Gaussian BNNs with either (1) ARD priors or (2) priors with zero means and small initial variance, $\sigma_p^2 = 0.0001$, such that the model is encouraged to resemble a prior and strongly drawn towards zero. We found that pruning weights with a high probability of 0 benefited more from an adaptive ARD prior while SNR pruning performed better with a specified prior.

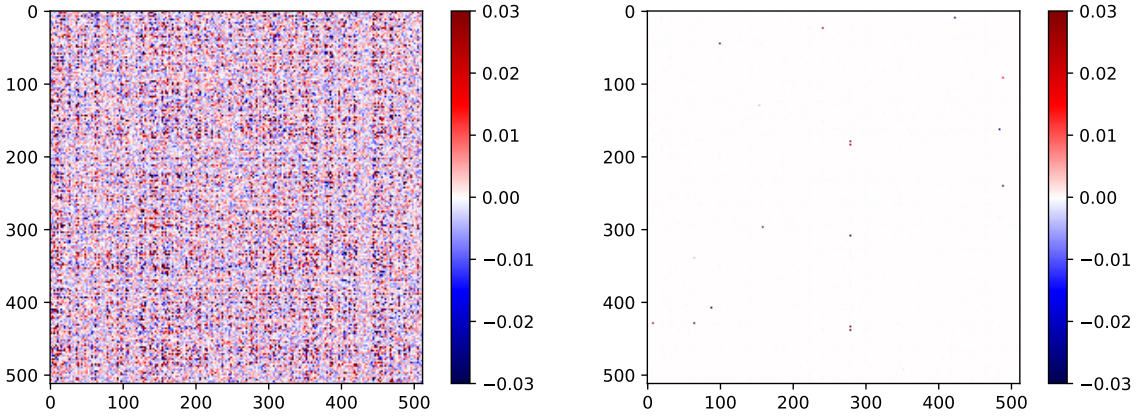
For variational dropout, as in [19] we use the additive reparametrisation trick and pre-train the network before applying variational dropout. This can be seen as a warm-up period such that the network can establish which weights may be relevant for prediction instead of immediately

| | Acc. | Brier Score | Percentage of removed weights per layer | | | | | Total remaining weights percentage |
|---|---|---|---|---|---|---|---|---|
| | | | Input Layer | L1 | L2 | L3 | Output Layer | |
| Variational Dropout | 82.7 | 34.4 | **75.0** | **97.7** | **99.1** | **98.9** | **72.2** | **3.05** |
| SNR Pruning | **84.1** | **31.1** | 71.1 | 92.4 | 98.8 | 94.1 | 67.0 | 14.11 |
| Pruning of weights with high probability of zero | **84.1** | 32.8 | 73.3 | 94.5 | 99.0 | 94.8 | 68.9 | 11.4 |

Table 8.2: Measuring effectivity of sparsification techniques.

pushing weights to zero. We anneal after this pre-training has occurred and then activate the KL term to encourage sparsification. This significantly increases the training phase, but since we are only interested in test-time efficiency, this is not a concern. We clip weights with variance values $\sigma^2 \geq 20$ for numerical stability. This value also acts as the criteria to prune weights and these weights are considered random and do not contribute towards prediction.

In Table 8.2 we report the sparsity in each layer, the overall sparsity as well as the accuracy and Brier score. We see that variational dropout achieves the best compression while SNR pruning achieves the highest accuracy and calibration with the least compression. Pruning weights with a high probability of zero offers a compromise between accuracy and compression. The table shows a relationship between compression ratio and the ability to express uncertainty. In heavily parametrised models, such as deep neural networks, it appears only a few of the parameters are responsible for the predictive performance, while the remaining parameters contribute to explaining uncertainty. We illustrate the extreme sparsity of variational dropout in Figure 8.9.




(a) Weight matrix at initialisation.

(b) Weight matrix after 20 epochs with variational dropout.

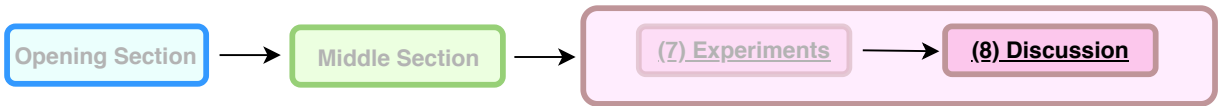
Figure 8.9: Visualisation of compression for variational dropout. We show a weight matrix of dimension 512 × 512 using a heatmap to visualise the values of weights at initialisation and in variational dropout training. (a) Represents initialisation and (b) corresponds to 99 % compression.

## 8.5 Discussion

In this chapter we evaluated a series of Bayesian and deterministic models and studied accuracy, data efficiency, calibration, de-noising in combination with a PGM, and compression techniques. We saw that neural network architectures produce the highest accuracy while logistic regression offers a computationally efficient solution but may produce less meaningful decision boundaries. We also saw that Bayesian methods excel in situations where data is scarce and deterministic networks are often less calibrated. We demonstrated the utility of BNNs which offer a way of incorporating uncertainty estimation while leveraging the expressive power of deep learning. Much of the value of the Bayesian methods discussed lie in their ability to remain uncertain and "know what they do not know" when making predictions. If the confidence estimates are well calibrated, one can trust the model's prediction probabilities, allowing applications such as detecting out-of-distribution inputs, building labelling systems with a human in the loop etc. We demonstrated using uncertainty in a probabilistic system, using a PGM to integrate spatial information or de-noise images and aid in reasoning about pixel classes. Alternatively, we showed that BNNs can be designed to introduce sparsity, providing a computationally efficient solution with the modelling power of deep learning. These models, however, were seen to sacrifice their uncertainty and calibration in removing parameters.

# Chapter 9

# Summary and Discussion




## 9.1 Summary

This thesis provided a probabilistic solution to satellite image classification motivated by using uncertainty to combat the lack of data and variability in satellite images. Additionally, in the interest of improving accuracy whilst under computational constraints, we adopted BNNs as the primary focus for classification of hyper-spectral signatures of pixels. The use of BNNs allowed us to leverage the modelling capacity of deep learning while permitting the preservation of uncertainty or the flexibility of introducing priors to induce sparsity in neural networks. We then developed a probabilistic system, connecting the output of the per-pixel classification with a PGM, to incorporate spatial information, having a de-noising effect on the classifier output. Hence, we were able to automatically reason about pixel labels using hyper-spectral information and uncertainty in classifiers in tandem with contextual information from PGMs.

Focussing largely on BNNs required the discussion of advanced variational techniques, including recent innovations such as stochastic variational inference, MCVI, the reparametrisation trick and local reparametrisation trick. We also investigated the relationships between stochastic regularisation and variational inference and saw that with dropout we can easily obtain a deep neural network with practical uncertainty estimates in MC dropout. Furthermore, we saw that variational dropout corresponds exactly to training a BNN that set the foundation for compression techniques and signal propagation analysis of BNNs for robust priors.

In the context of our limited computational budget, we considered Bayesian methods to compress models while attempting to preserve predictive performance. With the use of the additive reparametrisation trick, we were able to make use of variational dropout, which implicitly induces sparsity, to compress neural networks to 3 % of the original number of weights. We compared this to heuristic procedures for pruning BNNs and found variational dropout to yield extremely sparse solutions while sacrificing some calibration compared to these methods.

The Bayesian interpretation of noise regularisation also inspired our work extending noisy signal propagation theory to BNNs. We used signal propagation to analyse BNNs and develop techniques to improve robustness in Bayesian deep learning. We first presented the signal propagation properties of BNNs then derived an alternative ELBO to allow the prior to exert its stabilising effect on the forward pass. Using these results we provided a novel prior that stabilises the signal propagation dynamics of a BNN during training. This allowed us to train deeper networks than previously possible and exhibited improved convergence properties.

We then combined our classification approaches with a PGM model to incorporate spatial information. Observing noisy output mappings from our classifiers testified to the need to incorporate information from neighbouring pixels to influence a pixel's class. Because of the lack of data, we are not able to learn contextual relationships, we made use of PGMs to encode our prior beliefs into graphical representations. Using cluster graphs we proposed two alternative factor configurations with varying behaviour and some further model augmentations to improve performance.

In our experiments we tested various neural network and logistic regression models on the Indian Pines dataset. We saw that Bayesian methods excel in situations where data is scarce. We qualitatively analysed the effect of using a PGM to integrate spatial information or de-noise images and saw how uncertainty aids the PGM in reasoning about pixel classes. Finally, we compared model compression techniques using BNNs, demonstrating the ability of models to accurately model the relationships with a fraction of the parameters. Of the classification models investigated, BNNs offer a flexible solution that yield accurate models under uncertainty while also being capable of reducing computational cost. Logistic regression offers a simple modelling procedure with efficient inference but may produce less meaningful decision boundaries on small data and not capture true relationships.

## 9.2 Discussion

Understanding issues of generalisation is still highly immature and very troublesome to evaluate in situations where data is so scarce as well as erratic such as in satellite image classification. In order to improve generalisation, we investigated the value of using probabilistic systems and classifiers that recognise their fallibility to improve generalisation. This yields a system which can dynamically incorporate information from different sources depending on the certainty of each source. Each part can contribute meaningfully, while being aware of its shortcomings, to reason automatically about a subject and reach a general agreement or consensus.

Bayesian deep learning is a promising emergent field of development for informing downstream decision making tasks or safety-critical applications, but face issues of practicality. We contributed to the development of Bayesian deep learning both by examining BNN signal propagation dynamics, and using this result in combination with a novel ELBO, that allows the prior to influence the network during a forward pass, to derive a self-stabilising prior. Using such a prior makes it possible to train deeper architectures and exhibits improved convergence properties.

The self-stabilising prior is designed considering signal propagation in the infinite width limit. This allows us to utilise the prior knowledge of the activation function and architecture to promote stable signal propagation and robustness. This prior offers an attractive alternative prior for neural networks, particularly deep BNNs, where designing meaningful priors is invariably obscure.

## 9.3 Future Work

This thesis focussed mainly on making use of uncertainty to improve predictive performance. However, remote sensing could benefit from further applications of uncertainty. For example, we can use uncertainty to build a system to aid in labelling. We can actively estimate what unlabelled data would be most informative for the model. The system then actively proposes which pixels to label by a human annotator in order to improve performance [43]. This allows efficient exploration of the variation in the data and reduces the amount of data required in training useful models.

While we focussed on land-cover classification in this thesis, once we know the locations or positions of farms, we can reason further to provide more useful information to governments

or farmers. Given that we know what crop is growing at a particular pixel, we can use this to monitor change and detect abnormalities. This pertains to anomaly detection techniques [70]. Alternatively, another promising avenue for further research is time-series prediction for monitoring the health of crops or yield estimation [71]. However, this requires the acquisition of a large amount of labelled data over a long period of time. Nevertheless, this is highly valuable for helping farmers improve and manage crop production and planning for food security.

Alternatively, another advantage of Bayesian modelling is the ability to express different forms of uncertainty. We can group uncertainty into types of uncertainty being aleatoric and epistemic uncertainty [72]. Aleatoric uncertainty accounts for noise innate to the observation process. This is ingrained in satellite image data as they capture information on a very large scale and sensor noise or motion noise as well as variations in the data due to seasonality etc. is always present. Aleatoric uncertainty cannot be reduced even when given infinite data. We could apply this by employing aleatoric uncertainty to obtain specific noise models related to observation noise. Alternatively, epistemic uncertainty captures the uncertainty in the model. This reduces as more data is collected. This is useful for explaining variation in data or identifying out-of-data examples and more robustly identify anomalies, which can be useful for detecting sick crops or unidentified classes.

# References


[1] L. Martino and M. Fritz, "New insight into land cover and land use in europe", *Statistics in focus*, vol. 33, 2008.

[2] T. N. Carlson and D. A. Ripley, "On the relation between ndvi, fractional vegetation cover, and leaf area index", *Remote sensing of Environment*, vol. 62, pp. 241–252, 1997.

[3] N. Pettorelli, J. O. Vik, A. Mysterud, J.-M. Gaillard, C. J. Tucker, and N. C. Stenseth, "Using the satellite-derived ndvi to assess ecological responses to environmental change", *Trends in ecology & evolution*, vol. 20, pp. 503–510, 2005.

[4] D. Lu and Q. Weng, "A survey of image classification methods and techniques for improving classification performance", *International journal of Remote sensing*, vol. 28, pp. 823–870, 2007.

[5] J. Knorn, A. Rabe, V. C. Radeloff, T. Kuemmerle, J. Kozak, and P. Hostert, "Land cover mapping of large areas using chain classification of neighboring landsat satellite images", *Remote Sensing of Environment*, vol. 113, pp. 957–964, 2009.

[6] N. Kussul, M. Lavreniuk, S. Skakun, and A. Shelestov, "Deep learning classification of land cover and crop types using remote sensing data", *IEEE Geoscience and Remote Sensing Letters*, vol. 14, pp. 778–782, 2017.

[7] E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez, "Convolutional neural networks for large-scale remote-sensing image classification", *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, pp. 645–657, 2016.

[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[9] R. M. Neal, "Bayesian learning for neural networks", PhD thesis, 1994.

[10] A. Graves, "Practical variational inference for neural networks", in *Advances in Neural Information Processing Systems*, 2011, pp. 2348–2356.

[11] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks", *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

[12] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes", *International Conference on Learning Representations*, 2013.

[13] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models", *Proceedings of the 31st International Conference on Machine Learning*, 2014.

[14] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference", *The Journal of Machine Learning Research*, vol. 14, pp. 1303–1347, 2013.

[15] C. Zhang, J. Butepage, H. Kjellstrom, and S. Mandt, "Advances in variational inference", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

[16] A. Pretorius, E. Van Biljon, S. Kroon, and H. Kamper, "Critical initialisation for deep signal propagation in noisy rectifier neural networks", in *Advances in Neural Information Processing Systems*, 2018, pp. 5722–5731.

[17] B. Poole, S. Lahiri, M. Raghu, J. Sohl-Dickstein, and S. Ganguli, "Exponential expressivity in deep neural networks through transient chaos", in *Advances in Neural Information Processing Systems*, 2016, pp. 3360–3368.

[18] S. S. Schoenholz, J. Gilmer, S. Ganguli, and J. Sohl-Dickstein, "Deep information propagation", *Proceedings of the International Conference on Learning Representations*, 2017.

[19] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks", in *Proceedings of the 34th International Conference on Machine Learning*, JMLR. org, 2017, pp. 2498–2507.

[20] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick", in *Advances in Neural Information Processing Systems*, 2015, pp. 2575–2583.

[21] X. Kang, S. Li, and J. A. Benediktsson, "Spectral–spatial hyperspectral image classification with edge-preserving filtering", *IEEE transactions on geoscience and remote sensing*, vol. 52, pp. 2666–2677, 2013.

[22] J. A. Benediktsson, J. A. Palmason, and J. R. Sveinsson, "Classification of hyperspectral data from urban areas based on extended morphological profiles", *IEEE Transactions on Geoscience and Remote Sensing*, vol. 43, pp. 480–491, 2005.

[23] Y. Yuan, J. Lin, and Q. Wang, "Hyperspectral image classification via multitask joint sparse representation and stepwise mrf optimization", *IEEE transactions on cybernetics*, vol. 46, pp. 2966–2977, 2015.

[24] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification", in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[26] A. K. Balan, V. Rathod, K. Murphy, and M. Welling, "Bayesian dark knowledge", *Advances in Neural Information Processing Systems*, 2015.

[27] M. Titsias and M. Lázaro-Gredilla, "Doubly stochastic variational Bayes for non-conjugate inference", in *Proceedings of the 31st International Conference on Machine Learning*, 2014, pp. 1971–1979.

[28] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning", in *International Conference on Machine Learning*, 2016, pp. 1050–1059.

[29] Y. Tarabalka, M. Fauvel, J. Chanussot, and J. A. Benediktsson, "Svm-and mrf-based method for accurate classification of hyperspectral images", *IEEE Geoscience and Remote Sensing Letters*, vol. 7, pp. 736–740, 2010.

[30] J. Pearl, *Causality: models, reasoning and inference*. Springer, 2000.

[31] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[32] C. Louizos, K. Ullrich, and M. Welling, "Bayesian compression for deep learning", 2017, pp. 3288–3298.

[33] A. Atanov, A. Ashukha, K. Struminsky, D. Vetrov, and M. Welling, "The deep weight prior", *International Conference on Learning Representations*, 2019.

[34] L. Fei-Fei and P. Perona, "A Bayesian hierarchical model for learning natural scene categories", in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE, vol. 2, 2005, pp. 524–531.

[35] Y. Gal, "Uncertainty in deep learning", PhD thesis, University of Cambridge, 2016.

[36] G. E. Box, "Science and statistics", *Journal of the American Statistical Association*, vol. 71, pp. 791–799, 1976.

[37] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization", *International Conference on Learning Representations*, 2016.

[38] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors", *Proceedings of the 29th International Conference on Machine Learning*, 2012.

[39] L. Wan, M. D. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, "Regularization of neural networks using DropConnect.", 2013, pp. 1058–1066.

[40] D. P. Wipf, B. D. Rao, and S. Nagarajan, "Latent variable Bayesian models for promoting sparsity", *IEEE Transactions on Information Theory*, vol. 57, pp. 6236–6255, 2011.

[41] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, *et al.*, "Overcoming catastrophic forgetting in neural networks", *Proceedings of the national academy of sciences*, vol. 114, pp. 3521–3526, 2017.

[42] H. Ritter, A. Botev, and D. Barber, "Online structured laplace approximations for overcoming catastrophic forgetting", in *Advances in Neural Information Processing Systems*, 2018, pp. 3738–3748.

[43] B. Settles, "Active learning literature survey", University of Wisconsin-Madison Department of Computer Sciences, Tech. Rep., 2009.

[44] S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih, "Monte carlo gradient estimation in machine learning", *arXiv preprint arXiv:1906.10652*, 2019.

[45] D. J. C. MacKay, "Bayesian interpolation", *Neural Computation*, pp. 415–447, 1992.

[46] G. Hinton and D. Van Camp, "Keeping neural networks simple by minimizing the description length of the weights", in *In Proceedings of the 6th Annual ACM Conference on Computational Learning Theory*, Citeseer, 1993.

[47] J. M. Hernandez-Lobato and R. Adams, "Probabilistic backpropagation for scalable learning of Bayesian neural networks.", in *Proceedings of the 32nd International Conference on Machine Learning*, 2015, pp. 1861–1869.

[48] T. P. Minka, "Expectation propagation for approximate Bayesian inference", in *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence*, 2001, pp. 362–369.

[49] A. Kucukelbir, D. Tran, R. Ranganath, A. Gelman, and D. M. Blei, "Automatic differentiation variational inference", *The Journal of Machine Learning Research*, vol. 18, pp. 430–474, 2017.

[50] A. Wu, S. Nowozin, E. Meeds, R. E. Turner, J. M. Hernandez-Lobato, and A. L. Gaunt, "Deterministic variational inference for robust Bayesian neural networks", in *International Conference on Learning Representations*, 2019.

[51] K.-I. Funahashi, "On the approximate realization of continuous mappings by neural networks", *Neural networks*, vol. 2, pp. 183–192, 1989.

[52] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators", *Neural networks*, vol. 2, pp. 359–366, 1989.

[53] D. Kingma and J. Ba, "Adam: A method for stochastic optimization", *International Conference on Learning Representations*, 2014.

[54] A. Damianou and N. Lawrence, "Deep gaussian processes", in *Artificial Intelligence and Statistics*, 2013, pp. 207–215.

[55] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting.", *Journal of Machine Learning Research*, pp. 1929–1958, 2014.

[56] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles", in *Advances in Neural Information Processing Systems*, 2017, pp. 6402–6413.

[57] I. Osband, "Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout", in *NIPS Workshop on Bayesian Deep Learning*, 2016.

[58] S. Wang and C. Manning, "Fast dropout training", in *Proceedings of the 30th International Conference on Machine Learning*, 2013, pp. 118–126.

[59] B. P. Carlin and T. A. Louis, *Bayes and empirical Bayes methods for data analysis*. Chapman and Hall/CRC, 2010.

[60] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks", *Proceedings of the International Conference on Learning Representations*, 2014.

[61] B. Poole, J. Sohl-Dickstein, and S. Ganguli, "Analyzing noise in autoencoders and deep networks", *Advances in Neural Information Processing Systems*, 2014.

[62] J. Pennington, S. Schoenholz, and S. Ganguli, "Resurrecting the sigmoid in deep learning through dynamical isometry: Theory and practice", in *Advances in Neural Information Processing Systems*, 2017, pp. 4788–4798.

[63] L. Xiao, Y. Bahri, J. Sohl-Dickstein, S. S. Schoenholz, and J. Pennington, "Dynamical isometry and a mean field theory of CNNs: How to train 10,000-layer vanilla convolutional neural networks", *Proceedings of the International Conference on Machine Learning*, 2018.

[64] M. Chen, J. Pennington, and S. S. Schoenholz, "Dynamical isometry and a mean field theory of RNNs: Gating enables signal propagation in recurrent neural networks", *Proceedings of the International Conference on Machine Learning*, 2018.

[65] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks", in *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[66] S. Streicher and J. du Preez, "Graph coloring: Comparing cluster graphs to factor graphs", in *Proceedings of the ACM Multimedia 2017 Workshop on South African Academic Participation*, ACM, 2017, pp. 35–42.

[67] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

[68] S. L. Lauritzen and D. J. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems", *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 50, pp. 157–194, 1988.

[69] K. P. Murphy, Y. Weiss, and M. I. Jordan, "Loopy belief propagation for approximate inference: An empirical study", in *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, Morgan Kaufmann Publishers Inc., 1999, pp. 467–475.

[70] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey", *ACM computing surveys (CSUR)*, vol. 41, p. 15, 2009.

[71] M. Adami, B. F. T. Rudorff, R. M. Freitas, D. A. Aguiar, L. M. Sugawara, and M. P. Mello, "Remote sensing time series to evaluate direct land use change of recent expanded sugarcane crop in brazil", *Sustainability*, vol. 4, pp. 574–585, 2012.

[72] A. Kendall and Y. Gal, "What uncertainties do we need in Bayesian deep learning for computer vision?", in *Advances in neural information processing systems*, 2017, pp. 5574–5584.

[73] J. Salvatier, T. V. Wiecki, and C. Fonnesbeck, "Probabilistic programming in Python using PyMC3", *PeerJ Computer Science*, vol. 2, 2016.

# Appendix A

# Path-wise Estimator

This follows the derivation of the path-wise derivative estimator as in [35]. We introduce an auxiliary variable $\epsilon$ that is usually a standard Gaussian which we can transform into any Gaussian. We assume we can write $\int q_\phi(w,\epsilon)d\epsilon = \int q_\phi(w|\epsilon)p(\epsilon)d\epsilon$ with $q_\phi(w|\epsilon) = \delta(w - g(\phi,\epsilon))$ the dirac delta for a given $\epsilon$. We apply this transformation to the stochastic optimisation expectation and get

$$\nabla_\phi \int f(w)q_\phi(w)dw = \nabla_\phi \int f(w)\left(\int q_\phi(w,\epsilon)d\epsilon\right)dw \tag{A.1}$$

$$= \nabla_\phi \int f(w)q_\phi(w|\epsilon)p(\epsilon)d\epsilon dw \tag{A.2}$$

$$= \nabla_\phi \int \left(\int f(w)\delta(w - g(\phi,\epsilon))dw\right)p(\epsilon)d\epsilon \tag{A.3}$$

which simplifies from the dirac delta having a point mass at $w = g(\phi,\epsilon)$ and being zero elsewhere, giving

$$= \nabla_\phi \int f(g(\phi,\epsilon))p(\epsilon)d\epsilon \tag{A.4}$$

Now that the gradient does not depend on the new simple distribution $p(\epsilon)$ we can write

$$\nabla_\phi \int f(w)q_\phi(w)dw = \int p(\epsilon)\nabla_\phi f(g(\phi,\epsilon))d\epsilon \tag{A.5}$$

or

$$\nabla_\phi \,\mathbb{E}_{q_\phi(w)}[f(x)] = \mathbb{E}_{p(\epsilon)}[\nabla_\phi f(g(\phi,\epsilon))]. \tag{A.6}$$

This allows us to sample from our distribution $q_\phi(w)$ and efficiently calculate derivatives directly with respect to the variational parameters $\phi$. The key thing here is that $f(g(\phi,\epsilon))$ is a deterministic function. We only sample once and compute the gradient and we have it with respect to $\phi$.

# Appendix B

# Approximation of Kullback-Leibler Divergence for Variational Dropout

The KL divergence was calculated up to an additive constant $C$ in [20] which was later improved by [19] to allow a better fit for larger values of $\alpha$ which we present here. The KL divergence between the log-scale uniform prior distribution and the true posterior is given as

$$\mathrm{KL}\big(q(w_{i,j}|\theta_{i,j},\alpha_{i,j})||p(w_{i,j})\big) \approx k_1\sigma(k_2 + k_3\log\alpha_{i,j}) - 0.5\log(1+\frac{1}{\alpha_{i,j}}) + C \tag{B.1}$$

$$\tag{B.2}$$

where

$$k_1 = 0.63576 \quad k_2 = 1.87320 \quad k_3 = 1.48695 \tag{B.3}$$

# Appendix C

# Reproducing Self-Stabilising Priors Experiments

Here we describe the procedure used to produce the experiments on self-stabilising priors. All of our experiments make use of ReLU networks, the He initialisation [25] and ADAM optimiser [53] We make use of the reparametrisation $\tilde{q}(W)$ at each layer. We restrict ourselves to BNNs with fully connected layers with a specified number of hidden layers all of constant width. The specified number of hidden layers does not include input layers and output softmax layers. We consider classification on MNIST and CIFAR-10 using a batch size of 1000.

**Limits of trainability.** We investigate performance at extremities by training a series of networks with hidden layer widths of 256 with varying depths and initial variances using 50 training epochs. We compare a series of networks with our proposed stabilising prior incorporated on the forward pass with a standard non-conjugate Gaussian prior [27] with variance 0.00001 which we report in Figures C.1 (a) and (b). This involves training 100 networks of each type. We train 10 networks initialising the approximating posterior $q(W)$ with mean 0 and vary the variance from 0.0005 to 0.5 multiplying intermediate variances with 5. For each of these 10 networks we train 10 more at the given variance adjusting the depth or amount of hidden layers from 1 hidden layer to 30, incrementing the depth with 3 for each successive network.

**Accelerated training.** We observe that our prior increases the training speed in general as demonstrated in Figure C.2. This experiment is repeated and averaged over 10 runs. We compare with EB as in [27] which uses the gradient to find optimal hyper-parameters for the prior. Our prior only outperforms EB in some settings. We find an advantage over EB with settings involving deeper networks, higher initial variance and wider networks. We show a representative example of architecture in this experiment of 5 hidden layers and 512 layer width.

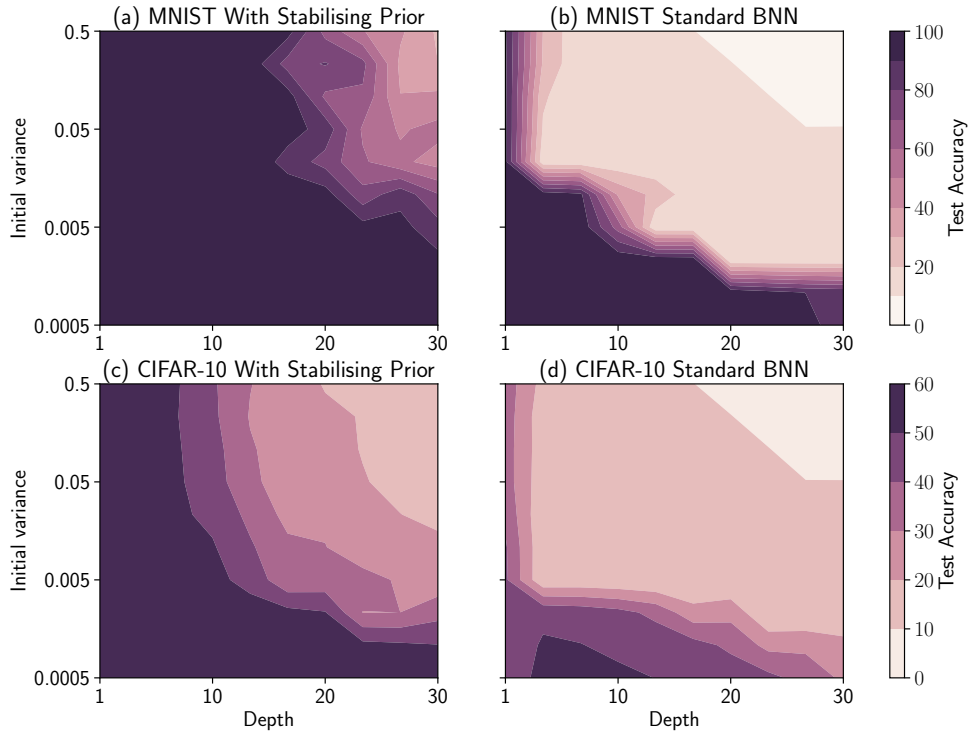



Figure C.1: MNIST and CIFAR-10 large scale experiments. Classification accuracy grid of ReLU networks trained on MNIST and CIFAR-10 with varying depths and initial variance conditions.

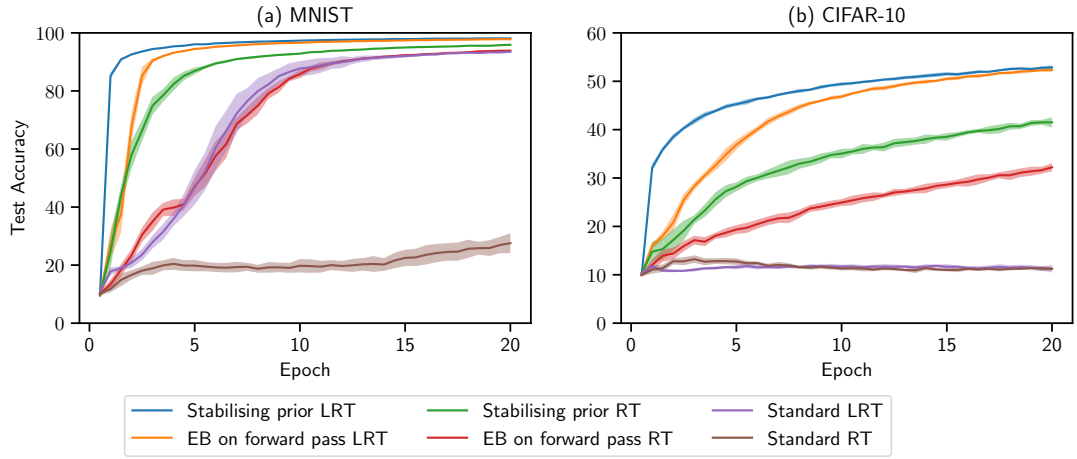



Figure C.2: Progression of test accuracy for various networks through training averaged over 10 runs for a 5 layer deep 512 wide network with an initial variance of 0.001.

Note that because of the scaling of the variance of the dimension of the incoming hidden layer $D_l$ being incorporated in the derivation, the stabilising excels with wider networks. This also dictates that the prior tends to resemble deterministic networks in the infinite width limit if means grow large. We also initialise the posterior with an initial variance of 0.001. The larger the variance the better the relative performance of the stabilising prior. We compare these priors with a regularising Gaussian prior and report their results for both the reparametrisation trick (RT) [12] and local reparametrisation trick (LRT) [20]. As expected, the local reparametrisation trick converges faster than the reparametrisation trick for all versions. We observe empirically that incorporating the EB prior on the forward pass also accelerates training relative to leaving the influence of the prior to a KL term added to the loss. We also find that when we combine EB with our stabilising prior we see an even further improvement in training speed. But the focus of our work is analysing the signal propagation dynamics and performance of the proposed prior.

**Quality of uncertainty.** We measure calibration with the Brier score for which there are many libraries available online that can compute this. The Brier score effectively measures discrepancy, between for categorical probabilities. A lower Brier score represents better calibrated predictions. We also measure the accuracy of predictions above 50 % and 90 % , similar to [56]. This is to present an alternative measurement to analyse whether models become overconfident. We only present experiments on CIFAR-10 as it presents many cases where models require to be aware of their uncertainty. MNIST is too simple and models easily achieve 95 % accuracy. We monitor the progression of these metrics of models with different priors through 100 epochs reported in Figure C.3. As with any iteratively updating prior, we expect that it may adapt to the dataset and overfit, as is shown to be true of our stabilising prior and EB in Figure C.3. However, relative to a deterministic network these methods excel. As an answer to this we explore combining a regularising and stabilising prior which trains faster and results in a well calibrated model with better Brier scores than any solitary prior.

The experiment in Figure C.3 (d) is trained on the "half moons" dataset generated from the PyMC3 library [73]. The network is a 20 layer deep, 512 wide network with a self-stabilising prior, showing that it is possible to obtain reasonable uncertainty estimates with very deep BNNs. Half moons is a common dataset used to visualise uncertainty of a model.

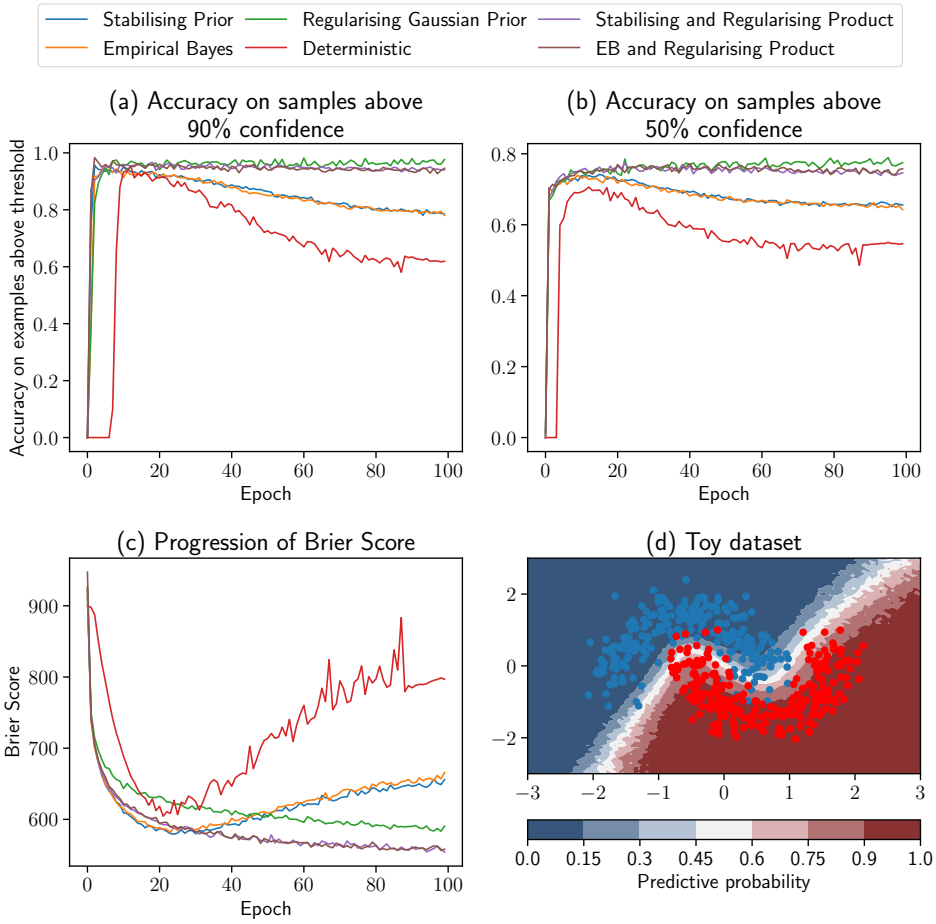



Figure C.3: Uncertainty and calibration experiments on CIFAR-10. Iteratively updating priors overfit, however, we can combine regularising and optimal priors to maintain calibrated confidence and better Brier scores. In (d) we also see we are able to get reasonable uncertainty estimates with a deep neural network on a toy dataset.

# Appendix D

# Probability table for alternative factor setup

| $X_{i,j}$ | $X_{i+1,j-1}$ | $X_{i+1,j}$ | $X_{i,j-1}$ | $P(X_{i,j}, X_{i+1,j-1}, X_{i+1,j}, X_{i,j-1})$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0.35 |
| 0 | 0 | 0 | 1 | 0.1 |
| 0 | 0 | 1 | 0 | 0.1 |
| 0 | 0 | 1 | 1 | 0.05 |
| 0 | 1 | 0 | 0 | 0.03 |
| 0 | 1 | 0 | 1 | 0.08 |
| 0 | 1 | 1 | 0 | 0.08 |
| 0 | 1 | 1 | 1 | 0.32 |
| 1 | 0 | 0 | 0 | 0.32 |
| 1 | 0 | 0 | 1 | 0.08 |
| 1 | 0 | 1 | 0 | 0.08 |
| 1 | 0 | 1 | 1 | 0.03 |
| 1 | 1 | 0 | 0 | 0.05 |
| 1 | 1 | 0 | 1 | 0.1 |
| 1 | 1 | 1 | 0 | 0.1 |
| 1 | 1 | 1 | 1 | 0.35 |

Table D.1: Probability table that represents a factor containing agreement probabilities for alternative factor setup shown again in Figure D.1. The probabilities are chosen to be very similar to the triplet factors as in Table 7.1, as well as being left unnormalised to show relationship, with the addition of having the diagonally adjacent or corner pixel slightly reduce probabilities when it not in agreement. This is seen for example with rows 1-4, which corresponds to the same as the triplet factors, compared to rows 5-8 where the corner pixel is not in agreement and slightly reduces probability.

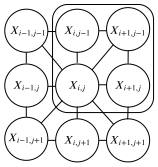



Figure D.1: Alternative factor setup.