The Practicalities of Scaling Bayesian Neural Networks to Real-World Applications

Adam D. Cobb

Worcester College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2019

Acknowledgements

I thank my supervisors Stephen Roberts and Andrew Markham for their support and guidance throughout my PhD. It has been a privilege to work with them and I am especially thankful for the encouragement I have received that has enabled me to explore a wide range of research areas. I also thank my collaborators: Ahsan Alvi, Atılım Güneş Baydin, Arno Blaas, Paul Duckworth, Richard Everett, Wolfgang Fruehwirt, Yarin Gal and Binxin Ru. In addition, I thank those in the lab who have given me endless support. Special mention goes to Ivan Kiskin who has followed the same path as me from undergrad and has always been there for proofreading and help throughout the process. My PhD has been made possible through my sponsorship by the AIMS CDT (http://aims.robots.ox.ac.uk) and the EPSRC (https://www.epsrc.ac.uk), for which I am grateful. In particular, the opportunities made available through the AIMS CDT were facilitated by the amazing work of Wendy, who has greatly simplified my PhD life. I also thank the NASA Frontier Development Lab for the opportunity to pursue research in the astrophysical domain. It also gave me the chance to meet many wonderful new colleagues and friends.

Finally, I thank my family for all their support and encouragement along the way, where I would not be able to get away without mentioning my mum, my dad, Simon and Ana.

Abstract

In this work, I will focus on ways in which we can build machine learning models that appropriately account for uncertainty, whether with computationally cheap estimates or with more expensive and reliable ones. In particular, I will explore how we can model distributions with Bayesian neural networks and how we can manipulate them depending on the task. The two main techniques for performing inference in Bayesian neural networks are variational inference and Markov chain Monte Carlo. I will look into the advantages and disadvantages of both methods and apply them to real-world problems. The emphasis is on how to achieve calibrated uncertainty estimates without compromising scalability. One contribution of this work is to offer a new method for implementing Bayesian neural networks within the framework of Bayesian decision theory, where Bayesian decision theory is important in all decision-making applications. A further contribution is developing sampling techniques that provide more reliable uncertainties, especially over data that lie outside the training distribution. Finally I also introduce a method for using Bayesian neural networks in an astrophysical application where it is vital that uncertainties are calibrated appropriately for the task.

Contents

List of Figures

List of symbols

All matrices are denoted by bold upper case letters, e.g. $\mathbf{X}$. Vectors are denoted by bold lower case letters, e.g. $\mathbf{x}$.

Data

$D_{\mathtt{in}}$	Input dimension of data
$D_{\mathtt{out}}$	Output dimension of data
N	Size of dataset
$\mathbf{x}$	Input data point
$\mathbf{y}$	Output data point
$\mathbf{X}$	Dataset inputs ($N \times D_{\mathtt{in}}$)
$\mathbf{Y}$	Dataset outputs ($N \times D_{\mathtt{out}}$)
$\mathcal{D}$	Dataset

Model

$\boldsymbol{\omega}$	A set of random variables (referring to model parameters)
$\mathbf{W}$	Weight matrix
$\mathbf{b}$	Bias vector
λ	Likelihood precision
τ	Prior precision

$\boldsymbol{\theta}$	Variational parameters
L	Number of layers (neural network)
Q_l	Number of units in layer l (neural network)
$\mathbf{z}_l$	Layer-wise dropout mask

Bayesian decision theory

$\mathcal{G}$	Gain
u	Utility
$\mathbf{H}$	Decisions (an individual decision is denoted by $\mathbf{h}$)
$\mathcal{H}$	Space of decisions

Hamiltonian Monte Carlo

$\mathbf{M}$	Mass matrix
$\mathbf{G}$	Fisher matrix
$\mathbf{p}$	Momentum vector
H	Hamiltonian
ϵ	Step size
L	Trajectory length (number of leapfrog steps)
α	Smoothness parameter
Ω	Binding term
ϕ^δ	Symmetric mapping (with step size δ)
$\mathbf{J}$	Skew-symmetric matrix

Atmospheric retrieval

$\mathbf{a}$	Atmospheric parameters (single vector)
$\mathbf{A}$	Atmospheric parameters (dataset)
$\mathbf{s}$	Spectrum
$\mathbf{S}$	Spectra (dataset)
κ_0	Grey cloud opacity
R^2	Coefficient of determination

List of acronyms

BLR	Bayesian linear regression
BNN	Bayesian neural network
CLT	Central limit theorem
CNN	Convolutional neural network
CPU	Central processing unit
ELBO	Evidence lower bound
GP	Gaussian process
GPU	Graphics processing unit
HMC	Hamiltonian Monte Carlo
IID	Independent and identically distributed
IOU	Intersection of union
KL	Kullback–Leibler
LCBNN	Loss-calibrated Bayesian neural network
MC	Monte Carlo
MCMC	Markov chain Monte Carlo
MH	Metropolis–Hastings
NN	Neural network
PBP	Probabilistic backpropagation
RF	Random forest
RMHMC	Riemannian manifold Hamiltonian Monte Carlo

S3HMC	Semi-semi-separable Hamiltonian Monte Carlo
SGLD	Stochastic Gradient Langevin Dynamics
SSHMC	Semi-separable Hamiltonian Monte Carlo
SVI	Stochastic variational inference
VI	Variational inference
WFC3	Wide Field Camera 3

Chapter 1

Introduction: striking the balance with uncertainty

In the quest to model the world from a Bayesian perspective, we often find ourselves with a choice. Do we pick a path that results in reliable, yet costly distributions over our variable of interest, or do we take a short-cut and settle for an approximation that may just about be good enough? In the era of big data, where Bayesian modelling can be extremely costly, we must find ways to strike a balance. For example, in tasks where safety is of paramount importance, robust and reliable uncertainty bounds over predictions can be the difference between life and death. Conversely, if the task is to develop a tool for a recreational purpose such as a smartphone app for identifying flowers, then expensive real-time uncertainty estimates may be unnecessary and detrimental to the objective of the application.

Neural networks are traditionally deterministic models that can only provide point estimates over their outputs. Therefore like all parametric models that do not fall within a Bayesian framework, they are unable to appropriately deal with uncertainty. However, despite their inability to capture uncertainty, they have become a useful tool for many machine learning applications due to their ability to scale to high-

dimensional and large data sets. On the other hand, if we are interested in a model that can capture uncertainty, but still takes advantage of the data scalability, then we can place neural networks in a Bayesian framework.

Bayesian neural networks (BNNs) combine the structure of a neural network model with Bayesian methodology to capture uncertainty and provide distributions over their outputs. The challenge with BNNs is ensuring that the benefit of employing Bayesian statistics does not come at the price of poor uncertainty estimates or poor scalability. Indeed, the goal within our community has always been to find ways of pushing algorithms to achieve these two, often competing, objectives.

1.1 Research direction and objectives

Current approaches that find the right balance between uncertainty quantification and scalability are hard to achieve as they often only focus on one of the two goals. In general, they will either become too concerned with uncertainty quantification that they lose the ability to scale, or they are too concerned with traditional performance metrics (e.g. accuracy) that they no longer provide trustworthy uncertainties. Therefore one of the objectives of this thesis is to introduce improvements to current techniques that provide better uncertainty quantification without compromising the ability to scale BNNs to large networks.

Another objective is to find new applications that push Bayesian neural networks into a regime that requires reliable uncertainty in a scalable fashion. We have already seen in domains such as computer vision [Kendall et al., 2015, Kendall and Gal, 2017] and reinforcement learning [Gal et al., 2016] that BNNs are a powerful tool for making predictions with highly complex input data. However, most baselines that exist follow uncertainty-independent metrics, such as accuracy or mean squared error. These metrics are not useful for ensuring real-world applications of BNNs are

performing in a reliable way and while this is an ongoing area of research [Filos et al., 2019], a further aim of this thesis is to demonstrate how one goes about ensuring uncertainty is accounted for when designing machine learning models. This research question is becoming increasingly important as BNNs are becoming more prevalent in applications ranging from diagnosing diabetes [Leibig et al., 2017] to performing end-to-end control in autonomous cars [Amini et al., 2017].

1.2 Main contributions

In light of the objectives highlighted above, the work presented in this thesis aims to tackle these problems. I explore common approaches to performing inference in BNNs, whereby each approach has its own advantages and disadvantages.

In particular, Chapter 4 provides a decision-theoretic perspective for BNN applications and provides new metrics for these tasks. I show how modern variational inference approaches are not necessarily well-suited to Bayesian decision theory and therefore introduce a new way of performing inference in Bayesian neural networks. This new perspective of analysing BNNs for Bayesian decision theory is demonstrated for both simple illustrative examples and for an autonomous driving data set. The contributions of this chapter align with the two main objectives of demonstrating how we can improve on existing methods for uncertainty quantification in BNNs and by showing more suited metrics for measuring performance.

Although scalable, variational approaches can suffer from poor uncertainty quantification. Given this characteristic, in Chapter 5, I introduce new Monte Carlo approaches to inference and show how these approaches can provide better uncertainty estimates due to fewer limiting assumptions regarding the form of the model. However, the challenges associated with scaling these sampling-based techniques can be problematic. I explore various solutions to this problem by starting in Chapter

5 with examples of performing inference in simple Bayesian hierarchical models and derive a new Hamiltonian-based Monte Carlo sampler. Chapter 6 then extends to BNNs, where I show how these Monte Carlo approaches are beneficial and are also underrated given the current advances in software and hardware. The contribution of Chapter 5 is a new integration scheme that is introduced to Riemannian manifold Hamiltonian Monte Carlo and the contribution of Chapter 6 is to provide a new way of performing Riemannian manifold Hamiltonian Monte Carlo over BNNs.

In Chapter 7, I introduce a new loss function for BNNs that is able to capture correlations between outputs by learning an output covariance matrix for each input. I then demonstrate the advantage of this new loss by taking the real-world astrophysical example of exoplanetary atmospheric retrieval and show how this task benefits from a new model with fast and accurate uncertainties. This is the first example of applying Bayesian neural networks for use in atmospheric retrieval. Therefore this chapter offers a new general model as well as an exciting new problem domain for applying machine learning algorithms.

1.3 Thesis structure and publications

The main focus of Chapters 2 and 3 is to provide the machine learning background for the rest of the thesis. Chapter 2 is more general and introduces the notation, models and data, whereas Chapter 3 is focussed on the literature associated with Bayesian neural networks, both the background and modern approaches.

The chapters thereafter follow different research questions on the theme of balancing uncertainty and scalability in BNN applications. Chapter 4 introduces Bayesian decision theory and provides new methodology for combining decision-making tasks with BNNs. This chapter builds on work that has been presented at *'The theory of deep learning workshop, ICML 2018'* [Cobb et al., 2018b]. Chapter 5 offers an alter-

native Monte Carlo approach to performing inference in Bayesian hierarchical models, where a new explicit integration scheme to Riemannian manifold Hamiltonian Monte Carlo is introduced. These ideas are then further extended in Chapter 6 to achieve better uncertainty quantification in BNNs. This work was presented at the *'Bayesian Deep Learning Workshop, NeurIPS 2019'* [Cobb et al., 2019a]. Finally, Chapter 7 focusses on an astrophysical application, where uncertainty and scalability are of the utmost importance. This work has been published in *'The Astronomical Journal'* [Cobb et al., 2019c] and is the first to apply BNNs in this application domain. The last chapter provides concluding comments and offer directions for future work.

Although not directly contributing here, I would like to mention a few pieces of work that have taken place during my DPhil, which have undoubtedly given me a broader perspective on the work presented in this thesis. They consist of:

- **Optimising Worlds to Evaluate and Influence Reinforcement Learning Agents**, R. Everett, **A. D. Cobb**, A. Markham, and S. J. Roberts, *Proceedings of the 18th International Conference on Autonomous Agents and Multi-Agent Systems, 2019* [Everett et al., 2019].

- **Bayesian deep neural networks for low-cost neurophysiological markers of Alzheimers disease severity**, W. Fruehwirt, **A. D. Cobb**, M. Mairhofer, L. Weydemann, H. Garn, R. Schmidt, T. Benke, P. Dal-Bianco, G. Ransmayr, M. Waser, et al., *Machine Learning for Health (ML4H) Workshop, NeurIPS 2018*, [Fruehwirt et al., 2018].

- **Scalable bounding of predictive uncertainty in regression problems with SLAC**, A. Blaas, **A. D. Cobb**, J. Calliess, and S. J. Roberts, *Scalable Uncertainty Management, SUM 2018*, [Blaas et al., 2018].

- **Identifying sources and sinks in the presence of multiple agents with Gaussian process vector calculus**, **A. D. Cobb**, R. Everett, A. Markham,

and S. J. Roberts. *ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2018*, [Cobb et al., 2018a].

Chapter 2

Data, models and uncertainty

At the heart of Bayesian machine learning is the process of combining prior knowledge with observations. In this chapter I will introduce some of the basic building blocks for Bayesian modelling and then investigate model priors and their effect on performance.

2.1 Common machine learning models

In order to introduce the conundrum of balancing uncertainty with scalability, this chapter introduces three common machine learning models that enable function approximation, whilst capturing uncertainty. These models are also used to introduce notation for the rest of the thesis. Two separate case studies are then introduced: one for a regression task and one for a classification task. Details regarding how to perform inference in these models are left to Section 3.2.

Data. A single $D_{\texttt{in}}$-dimensional input point is referred to as $\mathbf{x} \in \mathbb{R}^{D_{\texttt{in}}}$, with its corresponding $D_{\texttt{out}}$-dimensional output point as $\mathbf{y} \in \mathbb{R}^{D_{\texttt{out}}}$. A set of N input-output pairs $\{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$ are often collectively referred to via the notation $\mathbf{X} \in \mathbb{R}^{N \times D_{\texttt{in}}}$, $\mathbf{Y} \in \mathbb{R}^{N \times D_{\texttt{out}}}$, or via the shorthand $\mathcal{D}$.

Probability. Throughout this thesis we rely on probability theory as our machinery for Bayesian modelling. Before we introduce the machine learning models below, we provide a few definitions that will help give context. We define a set of random variables $\boldsymbol{\omega}$, such that $p(\boldsymbol{\omega})$ is the probability distribution over $\boldsymbol{\omega}$. The joint probability of the data and the set of random variables is given by $p(\mathcal{D}, \boldsymbol{\omega})$. The probability distribution over the output conditioned on both the input and the set of random variables is given by $p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega})$.

Bayes' theorem. Bayes' theorem [Bayes, 1763] is at the heart of Bayesian machine learning and is employed to deal with the manipulation of probability distributions in accordance with the following formula:[1]

$$p(\boldsymbol{\omega}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\omega})\, p(\boldsymbol{\omega})}{p(\mathcal{D})}. \tag{2.1}$$

In machine learning, each distribution is commonly referred to as:

- $p(\boldsymbol{\omega}|\mathcal{D})$: the posterior over the set of random variables $\boldsymbol{\omega}$.
- $p(\mathcal{D}|\boldsymbol{\omega})$: the likelihood, which is a function of $\boldsymbol{\omega}$. This is the probability of the data given the set of random variables.
- $p(\boldsymbol{\omega})$: the prior over $\boldsymbol{\omega}$. Core to Bayesian statistics, where the prior encodes all knowledge of $\boldsymbol{\omega}$ before observing any data.
- $p(\mathcal{D})$: the marginal likelihood or the probability of the data. It is referred to in this manner as it is equivalent to marginalising out (integrating over) $\boldsymbol{\omega}$ in the numerator, such that $p(\mathcal{D}) = \int p(\mathcal{D}|\boldsymbol{\omega})\, p(\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega}$.

The implication of Bayes' theorem is that we can combine prior information and observations within our model in such a way as to enable the inference of probability

[1] More recent evidence suggests that Bayes' theorem was first discovered by the mathematician Nicholas Saunderson [Stigler, 1983].

distributions over model parameters.

2.1.1 Bayesian linear regression

Linear regression. Linear regression is one of the main building blocks of many machine learning techniques. It is based on the simple assumption that the output is a weighted combination of its inputs, such that we define the model as $\mathbf{f}(\mathbf{X}) = \mathbf{X}\mathbf{W} + \mathbf{b}$, where $\mathbf{W} \in \mathbb{R}^{D_{\text{in}} \times D_{\text{out}}}$ and $\mathbf{b} \in \mathbb{R}^{D_{\text{out}}}$ are the weight matrix and bias respectively [Bishop, 2006, Chapter 3]. The objective is to set the model parameters, $\{\mathbf{W}, \mathbf{b}\}$, such that they minimise a loss. In regression, a commonly chosen loss is the average squared error loss $\frac{1}{N}\|\mathbf{Y} - \mathbf{f}(\mathbf{X})\|_2^2$.

Bayesian linear regression. When average squared error loss is used to set the model parameters, linear regression is susceptible to over-fitting the data. The addition of a regularisation term can alleviate this issue by penalising weights that are set to be too large. In fact, by introducing this term, we are imposing a prior on the size of the weights. From a Bayesian perspective, the squared error loss with an additional quadratic regularisation term is equivalent to a Gaussian likelihood model $p(\mathbf{Y}|\mathbf{X}, \mathbf{W}, \mathbf{b}, \lambda) = \mathcal{N}(\mathbf{Y}|\mathbf{f}(\mathbf{X}), \lambda^{-1}\mathbf{I})$ with a Gaussian prior over the weights $p(\mathbf{W}, \mathbf{b})$. The Gaussian likelihood is our linear regression model, where the precision term λ encodes the output noise model, such that the likelihood is equivalent to

$$p(\mathbf{Y}|\mathbf{X}, \mathbf{W}, \mathbf{b}, \lambda) = \mathbf{f}(\mathbf{X}) + \mathcal{N}(\mathbf{0}, \lambda^{-1}\mathbf{I}). \tag{2.2}$$

For our prior, we can set

$$p(\mathbf{W}, \mathbf{b}|\tau) = \mathcal{N}(\mathbf{W}|\mathbf{0}, \tau^{-1}\mathbf{I})\mathcal{N}(\mathbf{b}|\mathbf{0}, \tau^{-1}\mathbf{I}),$$

where τ is the prior precision on the model parameters. Then, invoking Bayes' theorem in Equation (2.1) gives the relationship

$$p(\mathbf{W}, \mathbf{b}|\mathbf{X}, \mathbf{Y}, \lambda, \tau) \propto p(\mathbf{Y}|\mathbf{X}, \mathbf{W}, \mathbf{b}, \lambda) \times p(\mathbf{W}, \mathbf{b}|\tau). \tag{2.3}$$

Finally, as both the likelihood and the prior are part of the same exponential family, the prior is conjugate to the likelihood, which means that the posterior over the model parameters will have a closed form. In this case, the posterior will also be a Gaussian distribution, as can be seen from the Gaussian identities (e.g. refer to Williams and Rasmussen [2006, Appendix A2]). Therefore writing the posterior in logarithmic form gives the log-posterior,

$$\log p(\mathbf{W}, \mathbf{b}|\mathbf{X}, \mathbf{Y}, \lambda, \tau) = -\frac{\lambda}{2N}\|\mathbf{Y} - \mathbf{f}(\mathbf{X})\|_2^2 - \frac{\tau}{2}\|\mathbf{W}\|_2^2 - \frac{\tau}{2}\|\mathbf{b}\|_2^2 + \text{const.} \tag{2.4}$$

The form of this log-posterior demonstrates how the log-likelihood incorporates the data, whilst the prior ensures that the knowledge that we had before observing the data is included. In this case, we expect the weights and biases to have a certain variance, which results in limiting their size and ensuring they are regularised. Therefore if we maximise the log-posterior with respect to the parameters $\mathbf{W}, \mathbf{b}$, this results in a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_{\text{pos}}, \boldsymbol{\Sigma}_{\text{pos}})$, where $\boldsymbol{\Sigma}_{\text{pos}}^{-1} = \tau\mathbf{I} + \lambda\mathbf{X}^\top\mathbf{X}$ and $\boldsymbol{\mu}_{\text{pos}} = \boldsymbol{\Sigma}_{\text{pos}}(\lambda\mathbf{X}^\top\mathbf{Y})$ [Bishop, 2006, Chapter 3, Page 153].

In reality, our model may not always have a conjugate prior and inferring the posterior over the model parameters may not always have a closed form. However, the availability of the posterior means we can use it for making predictions over new observations. We will encounter the process of making predictions when Bayesian inference is formally introduced in Section 3.2.

2.1.2 Gaussian processes for regression

In introducing Bayesian linear regression (BLR), we have actually already introduced a Gaussian process, albeit one with limited expressibility. The definition of a Gaussian process can be taken directly from Williams and Rasmussen [2006, Page 13], where it is defined as *'a collection of random variables, any finite number of which have a joint Gaussian distribution'*. Therefore, in taking the expectation of Equation (2.2),

$$\begin{aligned}\mathbb{E}[\mathbf{Y}] &= \mathbf{X}\mathbb{E}[\mathbf{W}] + \mathbb{E}[\mathbf{b}] + \mathbb{E}[\mathcal{N}(\mathbf{0}, \lambda^{-1}\mathbf{I})] \\ &= \mathbf{0}, \end{aligned} \tag{2.5}$$

and the covariance

$$\text{cov}[\mathbf{Y}] = \mathbb{E}[(\mathbf{Y} - \mathbb{E}[\mathbf{Y}])(\mathbf{Y} - \mathbb{E}[\mathbf{Y}])^\top] = \mathbf{X}\mathbb{E}[\mathbf{W}\mathbf{W}^\top]\mathbf{X}^\top + \mathbb{E}[\mathbf{b}\mathbf{b}^\top] + \text{cov}[\mathcal{N}(\mathbf{0}, \lambda^{-1}\mathbf{I})]$$

$$\text{cov}[\mathbf{Y}] = \tau^{-1}(\mathbf{X}\mathbf{X}^\top + \mathbf{I}) + \lambda^{-1}\mathbf{I}, \tag{2.6}$$

we observe that the collection of N output data points have a joint Gaussian distribution.[2] Therefore $p(\mathbf{Y}|\mathbf{X}, \mathbf{W}, \mathbf{b}, \lambda)$ is a Gaussian process. In addition, we can also see that the prior, $p(\mathbf{f})$, is also a Gaussian process, given Equations (2.5) and (2.6). Formally, a Gaussian process is defined by its mean function $\mu(\mathbf{X})$ and its covariance function $\mathbf{K}(\mathbf{X}, \mathbf{X}')$, such that

$$\mathbf{f}(\mathbf{X}) \sim \mathcal{GP}\big(\mu(\mathbf{X}), \mathbf{K}(\mathbf{X}, \mathbf{X}')\big) \tag{2.7}$$

is a distribution over the function space. Therefore a single draw from this Gaussian process provides a single draw of $\mathbf{f}(\mathbf{X})$ from the functional prior $p(\mathbf{f})$.

[2] We have kept the notation in multi-input multi-output form, where each output dimension can be treated independently for ease of understanding. In the Gaussian process literature, there are many ways in ensuring correlations between outputs are learnt such as in learning a coregionalisation matrix [Alvarez et al., 2012].

So far, it has been shown that the Bayesian linear regression model introduced in the previous section is a Gaussian process and this can be viewed as a distribution over functions. The real advantage of Gaussian processes comes from the *kernel trick*, which enables the Gaussian process to be more expressive than the Bayesian linear regression model that relies on a linear kernel of the form $\mathbf{K}(\mathbf{X}, \mathbf{X}') = \tau^{-1}(\mathbf{X}\mathbf{X}^\top + \mathbf{I})$.

To introduce the kernel trick, one must start by transforming $\mathbf{X}$ via a set of basis functions $\boldsymbol{\phi}(\mathbf{X})$, e.g. $\boldsymbol{\phi}(\mathbf{X}) = [\mathbf{1}, \mathbf{X}, \mathbf{X}^2, \dots]$. We can then replace all instances of $\mathbf{X}$ with $\boldsymbol{\phi}(\mathbf{X})$ and allow $\boldsymbol{\phi}$ to absorb the bias component such that $\text{cov}[\mathbf{f}(\mathbf{X})] = \tau^{-1/2}\boldsymbol{\phi}(\mathbf{X}) \cdot \boldsymbol{\phi}(\mathbf{X})\tau^{-1/2}$. We can now write the above Gaussian process in terms of this inner product, which gives us the opportunity to employ the kernel trick, where this inner product can be replaced by the kernel function $\mathbf{K}(\mathbf{X}, \mathbf{X}')$.

Finally, to give a flavour of the new-found expressiveness of using a set of basis functions, we introduce the squared exponential covariance function defined between two input points $\mathbf{X}_1$ and $\mathbf{X}_2$, as

$$k(\mathbf{X}_1, \mathbf{X}_2) = \exp\{-\frac{1}{2}\|\mathbf{X}_1 - \mathbf{X}_2\|_2^2\}. \tag{2.8}$$

This is equivalent to the dot product when $\boldsymbol{\phi}(\mathbf{X})$ is an infinite set of polynomial basis functions, which can be seen from rearranging Equation 2.8 and using a Taylor series expansion of the exponential function. Therefore by defining an appropriate kernel function for a Gaussian process, we no longer limit our model to a linear combination of its inputs, but in fact, open up our model to be able to represent a much larger family of functions.

Just as for Bayesian linear regression, the posterior over the function can be inferred in the same manner by using the conjugate relationship between the prior and likelihood. Therefore once we have inferred the posterior over the function then we can start to make predictions.[3]

[3]A Gaussian process is a non-parametric model as it is defined by its kernel function and mean

2.1.3 Bayesian neural networks for regression

In a similar manner to how we moved from linear regression to Bayesian linear regression, we can also follow a comparable path for neural networks. Therefore I first introduce neural networks before describing Bayesian neural networks.

Neural networks. The step between vanilla linear regression, as introduced in Section 2.1.1, and a simple fully connected feed-forward network (otherwise known as a dense neural network) is minimal. The additional building block that is required is the introduction of a non-linear function, $h : \mathbb{R}^{D_{\text{in}}} \to \mathbb{R}^{D_{\text{out}}}$, which enables the construction of a single neural network layer,

$$\mathbf{u} = h\left(\mathbf{X}\mathbf{W} + \mathbf{b}\right), \tag{2.9}$$

where $\mathbf{u}$ corresponds to the layer's output and the linear regression model, $\mathbf{f}(\mathbf{X}) = \mathbf{X}\mathbf{W} + \mathbf{b}$, corresponds to the layer's input (or activations). The function approximating power of a neural network comes from stacking these layers together by passing a layer's output into the following layer's input. In order to describe this process, we use the subscript, l, to refer to a layer, where each layer contains Q_l hidden units. Each layer applies a non-linear weighted transformation of its input and has its own matrix of weights $\mathbf{W}_l \in \mathbb{R}^{Q_{l-1} \times Q_l}$ and biases $\mathbf{b}_l \in \mathbb{R}^{Q_l}$. For example, in the case of a network with a single hidden layer of 100 units, $Q_0 = D_{\texttt{in}}$, $Q_1 = 100$ and $Q_2 = D_{\texttt{out}}$. In extending this notation we can now define a fully connected dense neural network with L layers and input $\mathbf{X}$ as:

$$\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{X}) = h(\dots h(h\left(\mathbf{X}\mathbf{W}_1 + \mathbf{b}_1\right)\mathbf{W}_2 + \mathbf{b}_2)\dots)\mathbf{W}_{L+1} + \mathbf{b}_{L+1}, \tag{2.10}$$

where from now on we define $\boldsymbol{\omega} = \{\mathbf{W}_l, \mathbf{b}_l\}_{l=1}^{L+1}$ to refer to all the network weights.

function, which can potentially describe an infinite set of parameters.

As is the case for linear regression, the network weights can be learnt via the minimisation of the average squared error loss

$$\mathcal{L}(\boldsymbol{\omega}) = \frac{\lambda}{2N}\|\mathbf{Y} - \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{X})\|_2^2 + \frac{1}{2}\sum_l \tau_l^{(\mathbf{W})}\|\mathbf{W}_l\|_2^2 + \frac{1}{2}\sum_l \tau_l^{(\mathbf{b})}\|\mathbf{b}_l\|_2^2, \tag{2.11}$$

where we have allowed for each weight and bias to have their own corresponding weight decay (or regularisation) term.

As an extension to the fully connected feed-forward network, we can replace the fully connected layers with convolutional filters. These convolutional filters are then learnt in the same way that the weights and biases are learnt for a dense neural network. Convolutional neural networks (or CNNs) are especially suited to computer vision tasks, where the convolutional layers automatically learn which features in the image space are important for tasks such as classification and regression.

Bayesian neural networks. Bayesian neural networks offer a probabilistic alternative to neural networks by specifying prior distributions over the weights [MacKay, 1992, Neal, 1995]. The motivation for working with BNNs comes from the availability of uncertainty in its function approximation, $\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x})$. The placement of a prior $p(\omega_i)$ over each weight $\omega_i \in \boldsymbol{\omega}$ leads to a distribution over a parametric set of functions. Figure 2.1 shows the structure of a BNN, by placing distributions over the connecting weights.

In addition to BNNs being able to provide distributions over functions, we have already seen that Gaussian processes are able to provide this too. Therefore one might ask if there is a relationship between the two models, especially if we define each prior, $p(\omega_i)$, to be Gaussian distributed. In fact, as shown in Neal [1995, Chapter 2], the prior provided by a fully connected neural network with a single hidden layer tends to a Gaussian process prior as the number of neurons of the hidden layer tends to infinity. This highlights the strong relationship between Gaussian processes and BNNs.

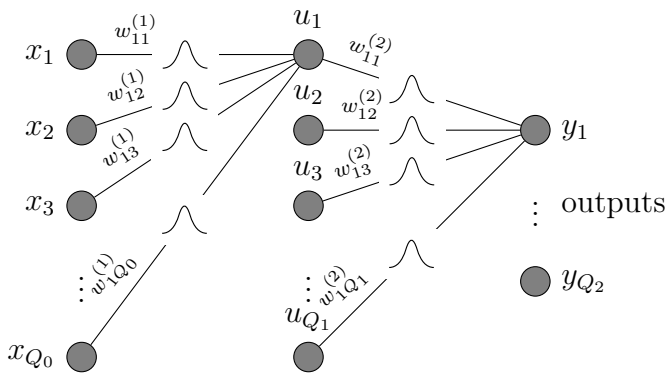


Figure 2.1: A Bayesian neural network with one hidden layer containing Q_0 inputs, Q_1 neurons in the hidden layer and Q_2 in the output layer. All the layers are fully connected, however for clarity only the neurons outputting u_1 and y_1 are shown connected. The distributions over the weights are represented by the 'Gaussian-like' curves drawn over the connections. Biases are left out of this diagram for ease of understanding.

BNN priors converging to Gaussian processes (summary from Neal [1995, Chapter 2]). If we re-write Equation (2.10) in element-wise form for a neural network model with a single hidden layer for a single input we get:

$$f_k(\mathbf{x}) = b_k^{(2)} + \sum_{i=1}^{Q_1} w_{ki}^{(2)} z_i, \quad u_i = h(b_i^{(1)} + \sum_{j=1}^{Q_0} w_{ij}^{(1)} x_j),$$

where superscripts correspond to the layers and subscripts correspond to the elements as in Figure 2.1. The first remark is that we can show that the distribution over the single output value $f_k(\mathbf{x})$ has a mean of zero and a bounded (constant) variance. We let the priors on the hidden unit weights, $w_{ij}^{(1)}$, and hidden-to-output weights, $w_{ki}^{(2)}$ be both independently and identically distributed (IID) with a zero mean and a corresponding standard deviation of $\sigma_{w^{(1)}}$ and $\sigma_{w^{(2)}}$ respectively. The biases, $b_k^{(1)}$ and $b_i^{(2)}$, are also IID according to zero mean Gaussians with standard deviations of $\sigma_{b^{(1)}}$ and $\sigma_{b^{(2)}}$. Therefore, if

we assign these priors, we can show straight away that

$$\begin{aligned}\mathbb{E}[f_k(\mathbf{x})] &= \mathbb{E}[b_k^{(2)}] + \sum_{i=1}^{Q_1} \mathbb{E}[w_{ki}^{(2)}]\mathbb{E}[u_i] \\ &= 0\end{aligned}$$

by using our IID assumption. The variance can be shown as

$$\begin{aligned}\text{var}[f_k(\mathbf{x})] &= \mathbb{E}[f_k(\mathbf{x})^2] \\ &= \mathbb{E}[(b_k^{(2)})^2] + \sum_{i=1}^{Q_1} \mathbb{E}[(w_{ki}^{(2)})^2]\mathbb{E}[(u_i)^2] \\ &= \sigma_{b^{(2)}}^2 + \sigma_{w^{(2)}}^2 \sum_{i=1}^{Q_1} \mathbb{E}[(u_i)^2].\end{aligned}$$

If we constrain the non-linear function, h, to be bounded (e.g. tanh is between -1 and 1) and let the number of hidden units Q_1 be large, then $\text{var}[f_k(\mathbf{x})] < \infty$ allows us to use the central limit theorem (CLT) to give $f_k(\mathbf{x}) \sim (0, \sigma_{b^{(2)}}^2 + \sigma_{w^{(2)}}^2 Q_1 \text{var}[u_i])$. To ensure that there is a limit on the prior distribution, Neal proposes to scale the prior variance of the hidden-to-output weights by Q_1^{-1} to ensure convergence as $Q_1 \to \infty$.

Finally, for a Gaussian process prior, we need to show that $f_k(\mathbf{x}^{(1)}) \dots f_k(\mathbf{x}^{(N)})$ are jointly Gaussian. We can extend the above to the multivariate case of $f_k(\mathbf{x})$ and $f_k(\mathbf{x}')$ where the joint mean remains as zero and the covariance is given by

$$\mathbb{E}[f_k(\mathbf{x})f_k(\mathbf{x}')] = \sigma_{b^{(2)}}^2 + \sigma_{w^{(2)}}^2 k_k(\mathbf{x}, \mathbf{x}'),$$

which constitutes a Gaussian process prior under the same CLT assumptions but for the multivariate case. One interesting result from this derivation is that the independence assumption means that there is no covariance between mul-

tiple outputs (e.g. $f_{k_1}(\mathbf{x})$ and $f_{k_2}(\mathbf{x})$, where $k_1 \neq k_2$). This leads to Neal's observation that *'It makes no difference whether we train one network to produce two outputs, or instead use the same data to train two networks'* as knowledge of one output does not tell us anything about any of the other outputs due to their independence under the prior [Neal, 1995, Page 38].[a] As a result, it may be possible to rely on the multi-input multi-output Gaussian process literature to learn these correlations. As an example, a coregionalisation matrix could be learnt to control interactions between the outputs, where a full review of these techniques can be found in Alvarez et al. [2012].[b] If we were interested in inferring the analytical form of the kernel $k_k(\mathbf{x}, \mathbf{x}')$ such that we could use GP machinery to form an equivalent BNN prior, we could look at work by Williams [1997], who provided two possible analytical forms.

[a] In practice, BNNs do not have an infinite number of neurons and training one network with multiple outputs is both more efficient and sees better performance. However, to ensure direct correlations between outputs, solutions such as in Chapter 7 are necessary.

[b] We will rely on learning a covariance between outputs in Chapter 7.

It is also possible to derive a neural network from a GP, where by defining a GP kernel that is equivalent to a single hidden layer neural network, Gal and Ghahramani [2016] employ modern stochastic variational inference techniques to show how their inference technique is equivalent to an approximation to a deep GP [Damianou and Lawrence, 2013]. This approach will be introduced in Chapter 3.

In the same way that we were able to use Bayes' theorem for Bayesian linear regression in Equation (2.3), we can equally use it here to learn the weights of a Bayesian neural network. The loss in Equation (2.11) is closely related to the log of the likelihood multiplied by the prior, $p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega})p(\boldsymbol{\omega})$ and therefore begs the question of how the posterior over the weights can be inferred. Unfortunately, where we previously relied on conjugacy to infer a closed form solution, BNNs do not have this property as the likelihood now contains non-linear transformations of the weights.

Therefore we require an alternative way of computing the integral in the denominator of Bayes' theorem,

$$p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega})p(\boldsymbol{\omega})}{\int_{\boldsymbol{\omega}} p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega})p(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega}}, \tag{2.12}$$

to infer the posterior distribution over the weights.

The research question of evaluating this integral is key to Bayesian statistics and more specifically one that we will encounter in Chapter 3. The way we choose to evaluate this integral is vital to ensure that BNNs are able to provide reliable uncertainties. Ensuring that these uncertainties are appropriately calibrated to their applications is key for their successful implementation.

2.2 Regression to classification

The machine learning models that have been introduced in Section 2.1 all rely on a variant of the Gaussian likelihood or squared error loss to map from input features to a continuous inferred output. To move from the task of regression to classification, we must now use functions that provide mappings from the feature space to class labels, $c \in \mathcal{C}$, where $\mathcal{C}$ is the set of all possible classes C. In order to achieve this goal we will change the likelihood to one which converts regression outputs to instances of class probabilities. If we set $D_{\texttt{out}}$ to be the number of classes, then the function output $\mathbf{f}(\mathbf{x})$ can be passed through an activation function, which *squashes* the output to an instance of a discrete probability distribution.

In binary classification, unlike multi-class, we only require a one-dimensional output. This is equivalent to scaling the output to lie between 0 and 1, such that we can treat the output as the class probability p_0 of class 0 and $p_1 = 1 - p_0$ as the class probability of class 1. This can be extended to a multi-class problem by introducing a

multi-dimensional output, where a commonly used activation function is the *softmax*,

$$p_{c,n} = \frac{\exp\{\mathbf{f}_c(\mathbf{x}_n)\}}{\sum_{c'} \exp\{\mathbf{f}_{c'}(\mathbf{x}_n)\}}, \tag{2.13}$$

which results in $\sum_{c=1}^{C} p_{c,n} = 1$. The output is now a vector of the form $[p_1, p_2, \ldots, p_C]$. The corresponding likelihood for this multi-class problem is the cross-entropy loss, which can be derived from the categorical distribution, $p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega}) = \prod_n \prod_c y_{c,n}^{p_{c,n}}$ [Bishop, 2006, Page 209, Chapter 4]. As an example, for a neural network we can simply replace the term corresponding to the squared error negative log likelihood in Equation (2.11) with the cross-entropy loss,

$$\mathcal{L}_{\text{CE}}(\boldsymbol{\omega}) = \underbrace{-\sum_{n=1}^{N}\sum_{c=1}^{C} p_{c,n} \log y_{c,n}}_{\text{Cross-entropy term}} + \frac{1}{2}\sum_{l} \tau_l^{(\mathbf{W})} \|\mathbf{W}_l\|_2^2 + \frac{1}{2}\sum_{l} \tau_l^{(\mathbf{b})} \|\mathbf{b}_l\|_2^2, \tag{2.14}$$

whereby $\mathbf{y}_n$ follows the 1-of-K encoding scheme.

Therefore for all the classes of models introduced in Section 2.1, we are now able to apply them to both classification tasks and regression tasks.

2.3 Comparing model priors

When choosing a model and its particular form, we must take into account the capability of the model and the prior assumptions that we want to impose on our task. Figures 2.2 to 2.5 each contain 100 samples from their respective functional priors, as well as the expectation over these priors, given by the solid blue line.[4] It is interesting to see how model selection can have important consequences for the shape of these prior samples, which can be drastically different, even within the same class of

[4] All models have been set to have a zero mean prior. Therefore, given this knowledge, it will be important for practical purposes to ensure that data is normalised such that when we pick a model, the prior covers the possible range of the data.

models.

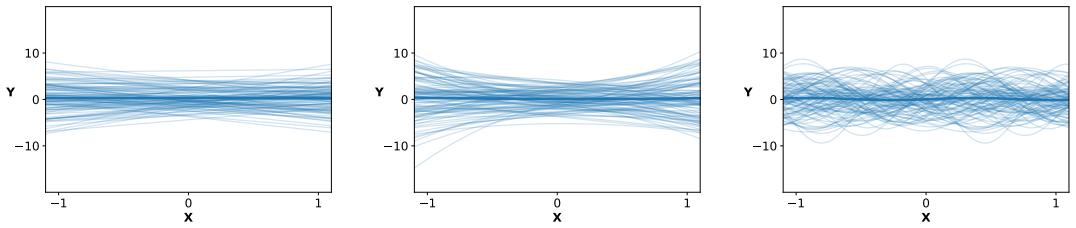
In order to see how the key components of each model affect the functional priors, the samples drawn from the models in this section have certain hyperparameters fixed. Specifically, for Bayesian linear regression, both the prior precision value and the output noise precision are fixed. For Gaussian processes the kernel hyperparameters are fixed and for Bayesian neural networks the IID prior variances over each weight are fixed. This allows us to focus on the influence on the prior when structural changes are made to the model. For example, in Figure 2.3, we see how selecting various basis functions for Bayesian linear regression constrains the model priors. If we pick our basis to be constructed as $\boldsymbol{\phi}(\mathbf{X}) = [\mathbf{1}, \mathbf{X}]$, then our prior samples are constrained to straight lines as shown in Figure 2.2a. However changing the basis to being second order as displayed in Figure 2.2b gives a slightly more flexible function, or selecting a sinusoidal basis as in Figure 2.2c results in the samples' oscillatory behaviour.

If we now look at the Gaussian process priors in Figure 2.3, we see how changing the kernel affects the functional prior. First of all, if we use a kernel constructed from the sum of a linear and a constant kernel, we recover the same functional prior as in Figure 2.2a for Bayesian linear regression. Then, following the reasoning introduced in Section 2.1.2, we can sample infinitely differentiable functions by using a squared exponential covariance function, as seen in Figure 2.3b. In comparison, in Figure 2.3c a Gaussian process with a Matérn $^1/_2$ kernel produces non-differentiable functions that result in very jagged samples.

In Bayesian neural networks, categorising the prior becomes more complicated. Therefore we will focus on how architecture choices, such as the number of neurons and the number of layers affect the prior. First, if we refer to Figure 2.4, we observe the effects of increasing the number of neurons in a single layer. One effect is the noticeable increase in the output scale length (or the output variance). This is the same observation that led to Neal [1995] requiring the prior weight precision to be

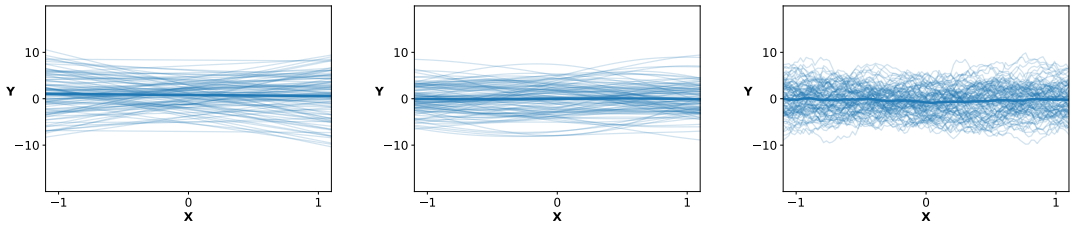
scaled by the number of neurons to control the variance. The other effect of increasing the number of neurons is that it increases the expressiveness of the model, i.e. leading to a wider family of functions that can be represented via the model [Gal, 2016b, Page 5, Chaper 1]. The second observation in relation to changing the architecture is to look at the behaviour of the model as we increase the number of layers. This is shown in Figure 2.5. As is consistent with the deep learning literature, Figure 2.5 shows how increasing the number of layers of a neural network is an effective way of achieving the goal of an expressive functional prior.

In this section we have seen how making assumptions about the structure of a model can affect the functional prior. Therefore it is important to make these decisions carefully, depending on the task. We explore these decisions in the next section.



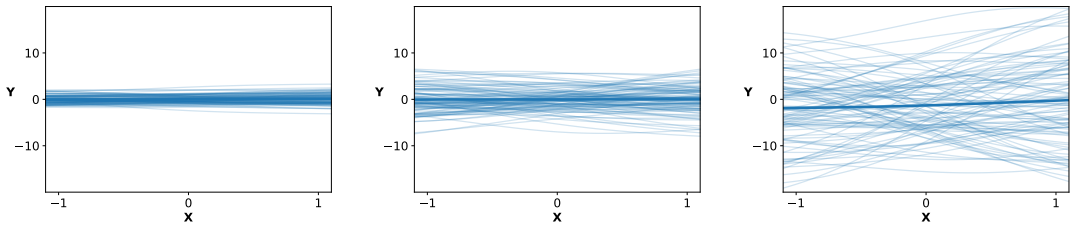
(a) $\phi(\mathbf{X})=[\mathbf{1},\mathbf{X}]$ (b) $\phi(\mathbf{X})=[\mathbf{1},\mathbf{X},\mathbf{X}^2]$ (c) $\phi(\mathbf{X})=[\mathbf{1},\sin(5\mathbf{X}),\cos(5\mathbf{X})]$

Figure 2.2: Each figure contains 100 samples drawn from three different functional priors corresponding to Bayesian linear regression with: **(a)** a linear basis; **(b)** a second order basis; **(c)** a sinusoidal basis.



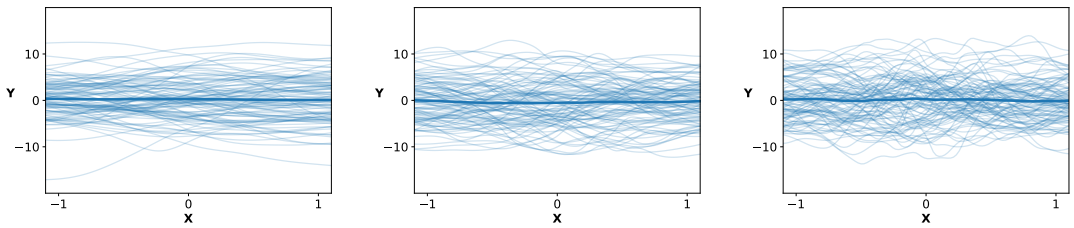
(a) Linear + Constant (b) Squared Exponential (c) Matérn $^1/_2$

Figure 2.3: Each figure contains 100 samples from Gaussian process priors with kernels: **(a)** the sum of a linear kernel and a constant kernel; **(b)** a squared exponential kernel; **(c)** a Matérn $^1/_2$ kernel. The kernel hyperparameters are given by the output scale length σ^2, the input scale length l^2 and σ_c^2 for the bias of the constant kernel.



(a) [1,10,1] (b) [1,100,1] (c) [1,1000,1]

Figure 2.4: Each figure contains 100 samples from different Bayesian neural network priors, where each network differs in architecture by increasing the number of units in a single hidden layer. The network architecture is defined by $[Q_0, Q_1, Q_2]$.



(a) [1,100,100,1] (b) [1,100,100,100,1] (c) [1,100,100,100,100,1]

Figure 2.5: Each figure contains 100 samples from different Bayesian neural network priors, where each network differs in architecture by increasing the number of layers. The network architecture is defined by $[Q_0, \ldots, Q_L]$.

2.4 Incorporating data into our model

Our interest lies in using the aforementioned models to make inferences in the presence of data. Furthermore, we take a special interest in how uncertainty is modelled across

tasks. Here, I introduce two case studies, one for regression and one for classification. The purpose of these case studies is to illustrate how different notions of uncertainty can appear and how we might use uncertainty to inform us. Ensuring that these uncertainties are appropriately calibrated will be an overarching theme, where I use these case studies to set the groundwork that is built on throughout the later chapters.

Before we look at the results, we must concern ourselves with why we require uncertainty and whether the manifestation of the uncertainty takes a form useful to the application. At the most basic level, in most signal processing tasks, the data will consist of noise (which may take different forms) and signal. Our machine learning model must try to explain the data, whilst also being able to understand that some parts of the data may be inherently noisy and not be possible to explain. Different terminology exists across multiple fields for these ideas of uncertainty (Jaynes [2003] provides a detailed analysis). However, here we will start with these two basic terms:

1. *Model uncertainty:* uncertainty that occurs due to some limit in the capability of the model. In this case, the model may not have seen enough data to fully explain a phenomenon, or it is limited in its expressiveness (e.g. see model misspecification [Domingos, 2000]).[5]

2. *Inherent noise:* uncertainty that exists in the data due to a noise process that cannot be decoupled from the problem. For example, for an image segmentation problem, the boundary drawn to separate two objects is inherently noisy, which leads to modelling higher levels of uncertainty at this line.[6]

A good model is able to distinguish between these two types of uncertainties and allow for either in the data.

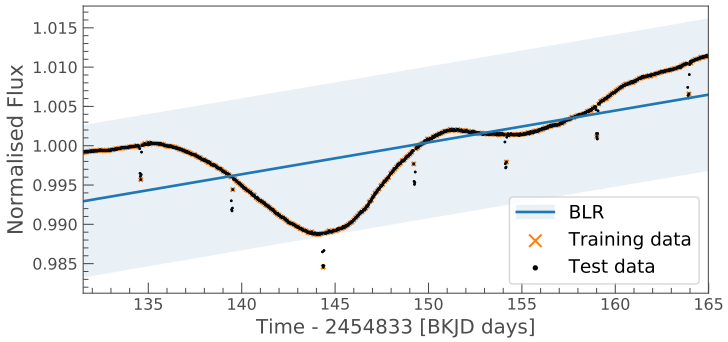
[5]For example, Kendall and Gal [2017] refer to this as *epistemic* uncertainty.

[6]For example, Kendall and Gal [2017] refer to this as *aleatoric* uncertainty, which they further categorise into homescadastic and heteroscedastic uncertainty.

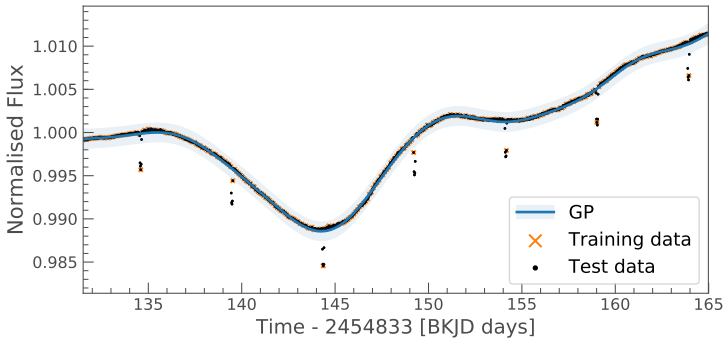
2.4.1 Case study 1: regression

Astronomical data is increasingly seeing a wide variety of interest across the machine learning community due to the need for more data-driven approaches, where traditional approaches may either be too computationally expensive or unable to cope with large data sets that are now collected with modern equipment. One example of such a data set is the collection of light curves made available through the *Kepler Mission* [Borucki et al., 2010], which is a space telescope designed to observe the flux of stars. The mission's aim was to capture periodic dips in the flux (brightness) that correspond to an exoplanet following an orbit that passes between the telescope and the star being observed. There are various reasons why one might be interested in fitting functions through these light curves [Cobb, 2015, Aigrain et al., 2017], however we will treat the example given here as a standard one-dimensional regression task, where the data consists of many frequency components and our interest is in modelling the lower frequency components such that we recover the higher frequency periodic dips, i.e. the exoplanet. Once we have modelled the lower frequency components, we can then remove them using the learnt function. This is with the aim of subtracting any (noisy) behaviour directly related to the star so that we are left with white noise and any patterns in the data that could correspond to an exoplanet. This removal of stellar variability is an important preprocessing step, necessary in exoplanet detection pipelines [Cobb, 2015].

As our interest lies in modelling the lower frequency components of the data, we must expect our model to have uncertainties calibrated accordingly. When a model lacks expressibility, it can often be the case that a model can fail by relying on the wrong kind of uncertainty to help explain the data. As an example, if our model is a simple Bayesian linear regression with a linear basis, then we find ourselves looking for the best fit straight line that explains our signal. In Figure 2.6a, we do precisely this which leaves the model with the only option of optimising the likelihood's variance



(a) Bayesian linear regression



(b) Gaussian process

Figure 2.6: Regression models for the exoplanet 'Kepler 3b'. The inflexibility of the Bayesian linear regression model only finds the linear trend and explains the majority of the data as inherent noise. The Gaussian process is able to fit to the normalised flux of the star, which we could use to remove and retrieve the periodic dips corresponding to the exoplanet passing in front of the star.

parameter to help explain the data. The two standard deviation uncertainty bounds are therefore dominated by the uncertainty component of the model that is supposed to explain the inherent noise in the data. Therefore this model is not suited to the task of detecting the anomalies in the signal.

If we use a more expressive model for this task, such as the Gaussian process in Figure 2.6b, the periodic dips are no longer contained within the two standard deviation uncertainty envelope. We can therefore say that the data points lying outside this envelope are the exoplanets that we are interested in detecting. In the GP's case, the model is confident in its predictions and has learnt to optimise the

likelihood's variance parameter to be small for this task, because the inherent noise is small.

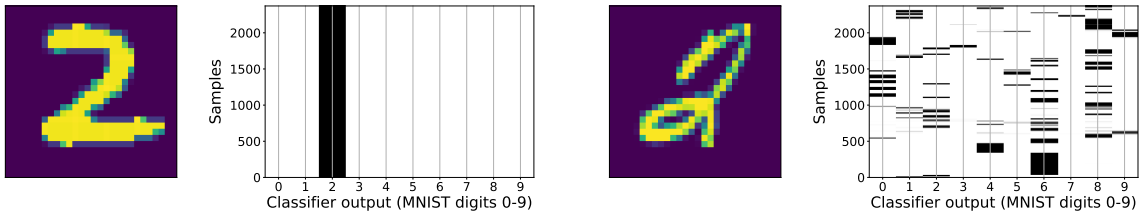
2.4.2 Case study 2: classification

Uncertainty is a key challenge in classification tasks because it is heavily dependent on the uncertainty metric one uses. We will see in this example, how uncertainty might manifest itself within a classification application and how different understandings of uncertainty can be problematic. We will use the canonical *MNIST* data set [LeCun et al., 1998], which is a common classification baseline data set used in the machine learning community. This is a data set composed of images of digits (e.g. $\mathbf{x} \in \mathbb{R}^{28\times28}$) and their one-hot corresponding labels.

In order to set up our classification example, we take take a small subset of 100 *MNIST* digits and use them as the training data for our model. As is common for tasks involving images, we use a CNN for our model structure and assign priors over the weights to result in a Bayesian CNN. This allows us to sample from the network to build a distribution over the output. When we first came across classification in Section 2.2, we used an activation function to ensure that the output was squashed into an instance of a discrete probability distribution. In this case, as there are 10 classes of digits, the output will be 10 dimensional and will sum up to one. The reason for using the phrasing *'an instance'* will become clear here, where we have multiple instances of an output vector for a given input, as we are sampling outputs from the Bayesian neural network model.

In Figure 2.7, we display the results of two digits. Figure 2.7a includes the result for which the classifier is most certain, whereas Figure 2.7b includes the result for which the classifier is least certain.[7] Each figure contains the raw image as well as the predictive output samples taken from the Bayesian CNN. The right-hand plot in each

[7] According to the Bayesian Active Learning by Disagreement (BALD) metric [Houlsby et al., 2011, Gal et al., 2017b].



(a) The *perfect* 2

(b) The *confusing* 2

Figure 2.7: A comparison of the outputs for a Bayesian neural network for two test images. Figure 2.7a displays an MNIST digit for which the model is most confident, which is shown by all the softmax outputs indicating the class 2 with high certainty (all the sample outputs put high probability over class 2). Figure 2.7b displays the digit 2 for which the model is least certain. This is shown by a large distribution of classifier output samples, where there is no consensus on the digit's label. For each classifier output sample in the right plot of each figure, darker pixels represent high probability and lighter pixels represent low probability.

subfigure displays these samples along the vertical axis, where the horizontal axis contains the corresponding 10-dimensional vector in grey-scale. Classes assigned high probabilities within a sample are represented by darker pixels, with 0 corresponding to white and 1 corresponding to black (with values in each row summing to one).

Although both digits are from the same class, the classifier finds the second digit in Figure 2.7b more challenging to classify, which is evident from the diversity in the *softmax* output samples. In comparison, the more *typical* number two in Figure 2.7a, is easier to classify, which we can see from the consistently high predictions across all the samples. Given these two extreme cases for the same digit, one might ask what we can say about uncertainty.

If we stay with the *atypical* number two, there are a few observations that can be made. First of all, if we take the expectation across all the samples, the digit gets classified as a six, with the value for digit eight being the next closest. As our model is basing everything it knows about the data from only 100 images, this expected prediction does not seem unreasonable. However, if we take a single sample from the network, which seems like a probability distribution as it sums up to one, and use that sample to inform us of the models confidence then we may arrive at a rather

different conclusion. For example, the last sampled output for Figure 2.7b is a vector equal to $[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0]$. If we were to take this instance of a probability distribution over the digits to be the actual distribution then the digit would be classified as an eight with the highest possible certainty. This result would clearly be concerning and points to one of the many challenges in reporting uncertainty in classification. Furthermore, this example also highlights that there is a clear risk in using a deterministic classifier and then using the output to say something about the uncertainty. This is a risk that is often taken in practice and can lead to catastrophic classification failures as shown by Nguyen et al. [2015], where deterministic neural networks are shown to provide highly confident incorrect classifications.

2.5 Conclusion

The purpose of this chapter has been to introduce machine learning models that are able to model or incorporate uncertainty into their function approximation. If all of these models already exist and are capable of providing uncertainty, then why is providing appropriate distributions over predictions still an open problem? The answer comes from the inherent challenge of ensuring that uncertainty information is appropriately passed through a model such that we can trust a model when it is confident. In the two case studies from this chapter, we saw examples of how uncertainty can manifest itself in both regression and classification tasks. For regression, selecting an unsuitable model gave poorly calibrated uncertainties that led to the majority of the data being interpreted as inherent noise. In our classification example, treating a single softmax output as a probability distribution led to many examples of misclassifying digits with high confidence. These examples highlight the need for trustworthy calibrated uncertainties. If we are to achieve this and follow the Bayesian methodology, then all models must include appropriate priors over every

parameter. This can be computationally expensive and often manifests itself in the form of an intractable integral as found in Equation (2.12) for BNNs. Since these integrals can be expensive, one might ask why we do not just keep to models such as GPs, which have equivalent tractable integrals (at least for regression)? However, these other models also come with their own limitations, and more importantly, the success of deep learning has meant that neural networks are now being used in many practical real-world applications. However, it is still extremely challenging to ensure that these large neural networks have correctly calibrated uncertainties. Therefore it is important to explore what it means to incorporate uncertainty for applications that require neural networks.

Chapter 3

Inference in Bayesian neural networks

Given the importance of incorporating uncertainty within the ever-expanding deep learning framework, it is no surprise that a large body of literature already exists on this topic. This chapter therefore focusses on the literature surrounding Bayesian neural networks, specifically how one might perform inference in them. The remaining part of the chapter places BNNs within the overarching area of Bayesian deep learning.

3.1 Bayesian deep learning

Bayesian deep learning covers everything from inference in Bayesian neural networks [MacKay, 1992, Neal, 1995] to deep generative models [Goodfellow et al., 2014, Kingma et al., 2014, Rezende et al., 2014]. Therefore Bayesian deep learning is at the intersection of deep learning[1] and Bayesian techniques. Any model that borrows structural components from both sets of methodologies could be considered to fall

[1]Where deep learning in the context of neural networks is defined as any neural network architecture with more than one hidden layer.

under this area. As such, Bayesian deep learning has become a broad area of research. One only needs to look at the most recent list of topics for the *NeurIPS 2018 Workshop on Bayesian Deep Learning* to get a sense of this range [Gal, 2016a].

Bayesian neural networks fall under the umbrella of Bayesian deep learning but are specifically concerned with neural networks that have priors imposed over their weights, as was introduced in Chapter 2. Bayesian neural networks follow and share much of the same history as neural networks.

3.1.1 Early days of neural networks (1943-1992)

Despite the immense success that deep learning has seen in the era of graphics processing units (GPUs) [Krizhevsky et al., 2012, Goodfellow et al., 2016] and widely-available deep learning libraries [Chollet et al., 2015, Abadi et al., 2015, Paszke et al., 2017a] circa 2011 and onwards, the main building blocks have been around since the 1940s. Originally introduced from a theoretical neurophysiology perspective, McCulloch and Pitts [1943] derived logic to describe nervous activity and in doing so defined a *'nervous net'*, the precursor to a neural network. However, despite being the conceptual starting point of the neural network (although shrouded in neurophysiological language), there were a few more steps required to get to a model that more closely represents the modern day neural network.

In 1959 Widrow and Hoff [1960] came up with *Adaline*, an adaptive linear unit, which they had built from *'switches, a battery, potentiometers and a meter'* [Anderson and Rosenfeld, 2000]. This adaptive switching circuit, could learn weights by propagating errors backwards using a least mean squares algorithm. This early form of a neural network can be understood as a linear model, where the output of the linear model is passed through a threshold function (*the quantizer*) that determines whether the final output should be -1 or 1. Importantly, the error is calculated before the threshold unit and therefore updates weights in the same way as in modern lin-

ear regression that relies on gradient-based optimisation. In 1960 when Widrow was demonstrating one of his *'learning machines'* to the press, despite the huge amount of excitement at the time he recalls that *'Nobody knew what the hell it was or what you could do with it. We didn't know what you could do with it.'*[2] [Anderson and Rosenfeld, 2000]. In the end, Widrow decided to stop working on neural networks because of the challenges associated with training multiple layers. In fact, a competitor at the time, Rosenblatt [1958, 1962] also built specific hardware to perform the task of classification. However, unlike Widrow, and more similar to McCulloch and Pitts [1943], Rosenblatt came from a biological background and focussed on building a *'brain model, not an invention for pattern recognition'* [Rosenblatt, 1962]. It was called the *perceptron*, and in Rosenblatt [1962] ideas of backpropagation and multiple layers were discussed. However, again, the challenges of extending to multiple layers and being able to learn the weights proved challenging, despite clearly asking research questions that appeared to be ahead of the times. In 1967, criticisms in a paper by Minsky and Seymour [1969] argued that perceptrons were limited to only solving linearly separable functions, and therefore that neural networks have limited capability.[3] The significance of this work helped contribute to a period where neural network research was mostly stagnant, at least until researchers could solve the problem of optimisation over multiple layers.

It was not until the 1980s that neural network research picked up again. Backpropagation, which solved the problem of multiple layers, was presented by Rumelhart et al. [1986] in a way that established this technique in neural network research. The key was to use differentiable non-linearities, i.e. not the original McCulloch-Pitts

[2]They did in fact manage to set up a company and sell Adalines. They were even on the verge of selling them to a large American telephone company, *General Telephone*, before being told their device could not be built using *'liquid-state electronics'* [Anderson and Rosenfeld, 2000, Page 58].

[3]According to Robert Hecht-Nielson, who made significant contributions to neural network research from the 1980s onwards, Minsky was not a huge fan of the biologically inspired *'soft science'* of the perceptron and and wrote the book 'Perceptrons: an introduction to computational geometry' [Minsky and Seymour, 1969], resulting in the *'the field of neural networks [being] ... discredited and destroyed'* [Anderson and Rosenfeld, 2000].

neurons which were entrenched in their biological beginnings to switch between 0 and 1. Although Rumelhart et al. [1986] can be attributed with popularising back-propagation, the technique has been discovered and rediscovered a few times, first by Werbos [1974] and then by Parker [1982] (see [Widrow and Lehr, 1990]). In fact, Rumelhart admitted that despite not being aware of it at the time, other examples of backpropagation existed a lot earlier in the literature [Anderson and Rosenfeld, 2000, Page 286].[4] However their work both reintroduced it and also gave a demonstration that firmly placed it in the neural network literature.

At this stage, neural networks were mostly concerned with minimising a least squares error, and therefore only driving the weights to converge to their mean (or first moment). In work by Denker and LeCun [1990], they referred to this as a *"lame sort of 'distance between distributions' ... [that] is apparently good enough for many applications"*. Therefore by the end of the 1980s, the ideas of bringing probability distributions to the area of neural networks were beginning to take shape. The initial work, by Denker et al. [1987], introduced the notion of a prior probability over network weights through using multiple realisations of weights from the possible weight-space of a neural network and taking an average, known as the *'ensemble viewpoint'*. This work was key to starting the discussion of what a Bayesian interpretation might look like for layered networks. Building on this, Tishby et al. [1989] used Bayes' theorem to combine this ensemble viewpoint to look at the average statistical prediction error and brought an information-theoretic viewpoint to neural network analysis. Then in a further piece of work coming from *AT&T Laboratories*, Denker and LeCun [1991] also concerned themselves with providing distributions over weights. They built on Denker et al. [1987] and Tishby et al. [1989] by using their terminology, but used a Taylor expansion to approximate the first and second moments of the network

[4]One can also look at contributions to control theory by Kelley [1960] and Bryson [1961] as deriving the early building blocks for backpropagation, where work by Mizutani et al. [2000] shows this relationship.

weights and the outputs. These first steps in the direction of BNNs were important and opened the door for others to build on further down the line.

3.1.2 Building the foundations of Bayesian neural networks (1992-2011)

Despite ideas of probability theory starting to creep into neural network research, it was really the work by MacKay [1992] on formalising a Bayesian framework for neural networks that managed to sew all the previous pieces together. This Bayesian framework focussed on objectively selecting and comparing network hyperparameters and architectures using model evidence, rather than relying just on generalisation error. The challenge was to adapt the Bayesian model comparison framework from Gull [1989] to be suited to the highly multi-modal neural network parameter space. Therefore in building on a similar approximation to that of Denker and LeCun [1991], but without removing the off-diagonal terms of the Hessian,[5] MacKay [1992] was able to build a Bayesian model comparison framework for neural networks by using the *'local'* model evidence for each network. The evidence framework highlighted scenarios where an overcomplicated model would have smaller evidence than a more simple one, even when the generalisation error is comparable, thus following the paradigm of Occam's razor. Finally, MacKay also pointed out how misspecification of the prior can lead to a failure, where the generalisation error and the model evidence are inconsistent.[6]

By the middle of the 1990s, we start to see two familiar paradigms arising in Bayesian neural network research. We see the emergence of variational inference by Hinton and Van Camp [1993] (see Section 3.2 for an introduction to variational

[5]The Hessian of a neural network is a square matrix of second order partial derivatives with respect to the weights of the network. This is utilised extensively in Chapter 5.

[6]This example forced all the prior variances for the weights to be the same. This meant that different layers could be rescaled by arbitrary amounts to achieve the same mappings and led to the inconsistency.

methods) and also the emergence of sampling methods by Neal [1995]. We will start by focussing on variational inference in neural networks before looking at sampling.[7] Without explicitly mentioning it, Hinton and Van Camp [1993] were the first to use a variational approach to Bayesian neural networks. Whilst taking an information-theoretic viewpoint that relied on the minimum description length principle, they took a step away from the local approximations of using Laplace's method [Denker and LeCun, 1991] and instead replaced the entire neural network model with a global approximating distribution. They chose a simple diagonal multivariate Gaussian distribution, such that each weight now had a corresponding mean and variance parameter to be optimised. In choosing to minimise the expected description length, we see the appearance of the Kullback-Leibler divergence as a regularisation term for the weights. Barber and Bishop [1998] extended on this work by deriving analytical solutions for integrals with the approximating distribution as a multivariate Gaussian with a full covariance matrix, rather than a diagonal covariance.[8] They placed Gamma distributions over the hyperparameters and optimised the hyperparameters within their Bayesian framework by using a factorised variational distribution that also included approximating distributions over the hyperparameters. Overall, Barber and Bishop [1998] took important steps in aligning the language of variational inference in Bayesian neural networks with the language used in other probabilistic models at the time, for example that of Jordan et al. [1999].

Rather than replacing posteriors over neural network parameters with approximating distributions, the alternative proposed by Neal [1995] was to directly sample in the weight space by using Monte Carlo methods (see Section 3.2.1). The argument

[7] Variational inference was first referred to as *ensemble learning* as it was thought that the approximating variational distribution was synonymous to learning an approximate ensemble over neural networks [MacKay, 1995].

[8] In order to perform the integrals in the approximating lower bound, [Barber and Bishop, 1998, Appendix A.1] derived a dimensionality reduction trick that resulted in an integration over a unitary normal, rather than a fully parameterised multivariate Gaussian. This is similar to the analytical version of the modern-day reparameterisation trick [Kingma et al., 2014, Rezende et al., 2014].

put forward by Neal, was that there was a need for *'Bayesian learning [of neural network weights] that does not rely on any assumption concerning the form of the posterior distribution'*, i.e. not using any approximating distributions. However to find a suitable Monte Carlo sampling scheme that worked in the high-dimensional space, previous schemes used in statistical methods could not be relied on due to their inefficiencies. In fact, to demonstrate this, in Section 1.2.4 of Neal's thesis, a simple rejection sampling scheme [Devroye, 1986] for a small network required 2.6 million draws from the prior to get 10 accepted samples that fell within range of the six training points (noting the exponential increase in time if the number of training points were to increase). There was clearly a need for a more scalable sampling scheme. Therefore Neal reformulated the *hybrid Monte Carlo* algorithm of Duane et al. [1987] for Bayesian learning of neural network weights. The success of his work in introducing how one would use hybrid Monte Carlo in a Bayesian setting using Hamiltonian dynamics resulted in the more commonly used phrase *Hamiltonian Monte Carlo* (HMC) (refer to Chapter 5 for HMC).

3.1.3 Scaling to modern deep learning architectures (2011-2016)

Up until this point, all variational inference approaches relied on analytical solutions to integrals and were therefore limited to networks with a single hidden layer and linear outputs. Furthermore, even in the case that one did use these analytical approaches, the computational complexity of learning a full covariance matrix for the variational distribution over the weights would limit their utility in practical applications. The real question at this time was how could one scale to multiple layers, which was arguably a similar challenge to that of Rosenblatt, Widrow and Hoff in

the 1960s.[9]

The first major step towards scalability came from Graves [2011], who used a version of stochastic variational inference (SVI) as a practical solution to the previously untenable option of finding an analytical solution to integrals in multiple layers for large networks. Graves [2011] motivated the use of SVI by writing that *'while it may seem perverse to replace one intractable integral (over the true posterior) with another (over the variational posterior), the point is that the variational posterior is far easier to draw probable samples from'*. This quote effectively summarises the reasoning behind SVI (see Section 3.2.4 for SVI). Rather than using analytical derivatives that require performing an intractable integration, the introduction of SVI meant that Monte Carlo (MC) estimates of the derivatives could be used in a way that scaled to much larger networks.

However, the SVI approach of Graves [2011], did not quite achieve what was necessary to make BNNs attractive to practitioners. Despite introducing the vital step of scaling to multiple layers, the variational distribution was the same diagonal multivariate Gaussian that was previously used in Hinton and Van Camp [1993]. This limiting distribution, along with noisy Monte Carlo estimates of the derivatives meant that the problem of performing inference in BNNs had not been solved.[10]

In 2015, two papers Blundell et al. [2015] and Hernández-Lobato and Adams [2015] focussed on further aligning inference in BNNs with the efficient backpropagation algorithm used for optimising deterministic neural networks. Blundell et al. [2015] introduced the reparameterisation trick [Opper and Archambeau, 2009, Kingma et al., 2014, Rezende et al., 2014] to result in performance comparable to deterministic neural

[9] In fact, Graves [2011] notes that it took 18 years to go from the first use of variational inference in neural networks to an actually scalable approach to multiple layers. It took roughly 28 years between Rosenblatt's perceptron in 1958 and backpropagation in 1986.

[10] Blundell et al. [2015] and Louizos and Welling [2017] both reason that the approximation of the diagonal of the Hessian with the negative diagonal of the empirical Fisher led to poor (biased) estimates of the variational variance parameters.

networks and called it *Bayes by Backprop*.[11] However, the uncertainty quantification was not fully explored and the requirement of a neural network to have twice the number of parameters could be argued as a challenge to scalability. Hernández-Lobato and Adams [2015] offered a different approach that also framed itself as being built on backpropagation and is referred to as *probabilistic backpropagation* (PBP). This approach, based on expectation propagation [Minka, 2001], was shown to outperform the SVI approach of Graves [2011] and provided updates for hyperparameters, which were not provided in Blundell et al. [2015]. Hernández-Lobato and Adams [2015] both compare the root mean squared error and (importantly) the test log-likelihood, which was provided as a useful metric for uncertainty quantification. Despite the increased performance in both uncertainty and test error, the largest network that Hernández-Lobato and Adams [2015] used in their paper was a network with 4 hidden layers of 100 units, which was small compared to larger networks that had already been around for a few years (e.g. Krizhevsky et al. [2012]).

The ultimate step in scalability would be to provide an approach to inference in Bayesian neural networks that requires minimal changes to current models, thus opening up Bayesian neural networks to the wider community. In fact, it turns out that one of the key steps in this direction had been introduced as a stochastic regularisation technique for neural networks. Known as dropout [Hinton et al., 2012, Srivastava et al., 2014], the idea is to set a proportion of a neural network's weights to zero using a dropout mask. Drawing a different dropout mask for each forward pass through the network in training results in a better regularised model. However it was also shown to be the case by Gal and Ghahramani [2016] that this was equivalent to performing SVI in Bayesian neural networks. Therefore predictive samples from the BNN could also be drawn via performing the same masking at test time. This is

[11] It is important to highlight that both Graves [2011] and the reparameterisation trick [Kingma et al., 2014, Rezende et al., 2014] use the Gaussian gradient identities from Opper and Archambeau [2009].

referred to as *Monte Carlo (MC) dropout* and we will more formally introduce this in Section 3.2.5. Gal and Ghahramani [2016] showed that MC dropout consistently outperformed PBP in terms of the test log-likelihood and the root mean squared error. Furthermore, they showed that it could scale effectively to any network that used a stochastic regularisation technique. This made it simple for researchers and practitioners alike, to provide uncertainty estimates in tasks such as computer vision [Kendall and Gal, 2017] and medical diagnosis [Leibig et al., 2017]. However, one might ask whether the dropout approximating distribution can always be relied upon for uncertainty quantification, given the limited flexibility of the approximate posterior distribution over the weights.

This time period also saw scalable approaches to sampling. The previous *gold-standard* method for sampling in neural networks until this point was HMC. A key limitation of HMC is that conservation of the Hamiltonian dynamics requires the gradients of the likelihood with respect to the weights to be computed over the entire data set [Neal, 1995]. This prevents HMC from being applied to problems with large data sets. Drawing inspiration from the already existing stochastic optimisation literature [Robbins and Monro, 1951], Welling and Teh [2011] introduced a scalable Monte Carlo method for estimating posterior statistics of probabilistic models. They overcome the limitation of having to evaluate the likelihood of the whole data set at each step by combining ideas from stochastic optimisation with Langevin dynamics [Neal et al., 2011] resulting in Stochastic Gradient Langevin Dynamics (SGLD). This technique allows one to smoothly transition from optimisation to sampling from the posterior of a model. A possible challenge with this technique is ensuring that it does not converge too quickly to a single mode of the distribution, which may require the step size to be fixed after the rejection rate has reached a certain level. Despite Welling and Teh [2011] providing an alternative approach to SVI techniques that does not require an explicit form of the posterior, they described their work as a

'tentative first step' opening the way for more efficient stochastic MCMC sampling techniques (i.e. ones that scale to large data sets and leverage optimisation for fast convergence). Interestingly, they did not directly implement SGLD on a BNN in their work, although it was directly applicable. Finally this work was extended to stochastic gradient HMC in Chen et al. [2014], where this required adding an additional friction term to the formulation of SGLD. However, for BNNs they subsequently found that the performance gain over SGLD and stochastic gradient descent with momentum not to be statistically significant.[12] Furthermore no uncertainty quantification was provided for BNNs.

Data sub-sampling MCMC techniques that rely on Hamiltonian dynamics cannot coherently explore the Hamiltonian flow according to Betancourt [2015], who provides an analysis of how stochastic gradients induce irreducible biases in trajectories. Therefore even when introducing the friction term to account for these biases, such as is found in the formulation of stochastic gradient HMC, this comes at the expense of fully exploring the target distribution. The result of these observations and the performance of these techniques still points to performing HMC over the entire data set if one is concerned with fully exploring the posterior of a BNN. Therefore there is still a need for scalable approaches to HMC, especially as a well-tuned HMC sampler still provides the most reliable uncertainty estimates for BNNs.

3.1.4 Concurrent work and research problems

BNN methods have matured to the stage where it is common to see Bayesian neural networks being applied in various applications from performing end-to-end control in autonomous cars [Amini et al., 2017] to monitoring structures such as bridges [Arangio and Bontempi, 2015]. However, little work focusses on the validity and calibration of the uncertainty measures the network produces. Therefore, although current work

[12] According to an update on their code at https://github.com/tqchen/ML-SGHMC/tree/master/bayesnn.

in BNN research is concerned with scalability, the key is to find scalable approaches that quantify uncertainty in a coherent and trustworthy way. There are various approaches that are becoming increasingly popular in dealing with this challenge, such as normalising flows [Rezende and Mohamed, 2015], deep generative models [Rezende et al., 2014, Kingma et al., 2014, Goodfellow et al., 2014] and Bayesian hypernetworks [Krueger et al., 2017]. This next section will focus on introducing the building blocks of these techniques that will allow us to explore the research questions in the later chapters.

3.2 Inference in Bayesian neural networks

Here, I will formally introduce the challenge of performing inference in Bayesian neural networks. The basic structure of a neural network was previously introduced in Section 2.1.3. We define our model to have likelihood $p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega})$ (e.g. a Gaussian likelihood $\mathcal{N}(\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{X}), \lambda^{-1}\mathbf{I})$ for regression or a categorical likelihood for classification) and a prior over the parameters $p(\boldsymbol{\omega})$ (e.g. a Gaussian prior $\mathcal{N}(0, \tau^{-1}\mathbf{I})$). Our interest lies in inferring the posterior over the weights, $p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})$, as knowledge of this posterior allows us to make predictions over new input points $\mathbf{x}^*$ via the integration,

$$p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) = \int p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega})\, p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}) \mathrm{d}\boldsymbol{\omega}, \tag{3.1}$$

which results in a predictive distribution over the output $\mathbf{y}^*$. In Bayesian modelling, the path to this predictive distribution comes from employing Bayes' theorem:

$$p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega})p(\boldsymbol{\omega})}{p(\mathbf{Y}|\mathbf{X})}, \tag{3.2}$$

where the marginal likelihood, $p(\mathbf{Y}|\mathbf{X})$, is the final term that needs to be inferred in order to obtain the predictive distribution in Equation (3.1). This marginal likelihood

also requires integration with respect to $\boldsymbol{\omega}$:

$$p(\mathbf{Y}|\mathbf{X}) = \int p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega}) p(\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega}. \tag{3.3}$$

If we can infer $p(\mathbf{Y}|\mathbf{X})$, we can evaluate the predictive distribution. Depending on the choice of prior and the likelihood, there are sometimes analytic solutions to marginalising over $\boldsymbol{\omega}$, such as briefly mentioned in Section 2.1.1 for Bayesian linear regression and in Section 2.1.2 for Gaussian process regression, where the prior is conjugate to the likelihood. However, when this is not the case, one must work around this problem by making approximations.

The following sections describe approaches to approximate inference. The first two sections describe Monte Carlo estimation and variational inference respectively. The sections thereafter move to stochastic variational inference, culminating in a summary of Monte Carlo dropout, which is one of the most commonly used stochastic variational inference technique for BNNs. The last part of this section then alludes to possible extensions to Monte Carlo dropout.

3.2.1 Approximate inference using Monte Carlo estimation

If we start from Equation (3.2), but ignore the normalisation factor, we can write the proportionality relationship as

$$p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}) \propto p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega}) p(\boldsymbol{\omega}). \tag{3.4}$$

Now, if we had a way to draw samples from the functional form on the right-hand side, then we would find ourselves with a set of S samples, $\{\boldsymbol{\omega}^{(s)}\}_{s=1}^{S}$, drawn from the posterior distribution, which would be defined up to a multiplicative constant. These samples can then be used to generate certain statistics such as to approximate the

expectation of the predictive distribution in Equation (3.1):

$$\mathbb{E}_{p(\boldsymbol{\omega}|\mathbf{X},\mathbf{Y})}\left[\mathbf{Y}^* = \mathbf{y}^*|\mathbf{x}^*\right] \approx \frac{1}{S}\sum_{s=1}^{S} p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega}^{(s)}). \tag{3.5}$$

The challenge is how to draw these samples, such that they are representative of the distribution from which we wish to sample. One approach is to use Markov chain Monte Carlo (MCMC) techniques to draw samples [Robert and Casella, 2013]. A Markov chain can be defined by providing the chain with a starting point, e.g. $\boldsymbol{\omega}^{(0)}$, and a transition probability distribution $q(\boldsymbol{\omega}^{(s+1)}|\boldsymbol{\omega}^{(s)})$. Therefore, in order to sample from our distribution of interest $p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})$, the Markov chain must explore the space of $\boldsymbol{\omega}$. One key property is that the Markov chain from which we use to sample must leave the distribution of interest invariant [Bishop, 2006, Page 540], such that

$$p(\boldsymbol{\omega}^{(s+1)}|\mathbf{X}, \mathbf{Y}) = \int_{\boldsymbol{\omega}^{(s)}} q(\boldsymbol{\omega}^{(s+1)}|\boldsymbol{\omega}^{(s)}) p(\boldsymbol{\omega}^{(s)}|\mathbf{X}, \mathbf{Y}) \mathrm{d}\boldsymbol{\omega}^{(s)}. \tag{3.6}$$

The other property is the one of *ergodicity*, which is the requirement that the chain can eventually reach all parts of the space, independent of the initial conditions. I will leave further details of Markov chains to Chapter 5, where we explore Hamiltonian Monte Carlo.

One advantage of Monte Carlo techniques is that no assumption needs to be made on the structure of the posterior. However there remain multiple challenges. We cannot always be sure as to when a chain is fully mixed and is producing representative samples. Furthermore, samples from a Markov chain may be highly correlated and it may not be clear how to account for the correlation when computing expectations. Finally, there are also specific challenges to performing MCMC in neural networks. This not only comes from the requirement of sampling in a high-dimensional space, but also the need for hardware that has the capability to store these samples and use them efficiently when making predictions.

3.2.2 Approximate inference using variational inference

Rather than relying on sampling methods, which come with their aforementioned disadvantages, it may be more attractive to turn the problem of integration to one of optimisation. In order to reformulate the problem, we can introduce a tractable *variational distribution* $q_{\boldsymbol{\theta}}(\boldsymbol{\omega})$, which is dependent on variational parameters $\boldsymbol{\theta}$ [Jordan et al., 1999]. The objective is then to optimise these variational parameters such that $q_{\boldsymbol{\theta}}(\boldsymbol{\omega})$ is as close to the posterior as possible. This is done by minimising the Kullback–Leibler (KL) divergence [Kullback and Leibler, 1951],

$$\mathrm{D}_{\mathrm{KL}}\big(q_{\boldsymbol{\theta}}(\boldsymbol{\omega})\|p(\boldsymbol{\omega}|\mathbf{X},\mathbf{Y})\big) = -\int_{\boldsymbol{\omega}} q_{\boldsymbol{\theta}}(\boldsymbol{\omega}) \log\left(\frac{p(\boldsymbol{\omega}|\mathbf{X},\mathbf{Y})}{q_{\boldsymbol{\theta}}(\boldsymbol{\omega})}\right) \mathrm{d}\boldsymbol{\omega}, \tag{3.7}$$

with respect to $\boldsymbol{\theta}$. Note that the KL divergence is always greater than or equal to zero i.e. $\mathrm{D}_{\mathrm{KL}} \geq 0$. If $\mathrm{D}_{\mathrm{KL}} = 0$ then the two distributions are identical. Hence minimisation of the KL divergence with respect to $\boldsymbol{\theta}$ results in an optimal set of variational parameters $\boldsymbol{\theta}^*$, such that $q_{\boldsymbol{\theta}^*}(\boldsymbol{\omega})$ is as close as possible to $p(\boldsymbol{\omega}|\mathbf{X},\mathbf{Y})$ under this particular divergence.[13]

At this stage it is important to point out a few challenges associated with the minimisation of Equation (3.7). First of all the KL divergence still contains the true posterior distribution, which we do not know. Second, even if we could find an appropriate $q_{\boldsymbol{\theta}^*}(\boldsymbol{\omega})$, how does that help given we still have to perform an integration with the variational distribution? If we go back to the original motivation, our interest is in the predictive distribution from Equation (3.1). If we replace the true posterior with our new approximating distribution the integration takes the form

$$p(\mathbf{y}^*|\mathbf{x}^*,\mathbf{X},\mathbf{Y}) \approx \int p(\mathbf{y}^*|\mathbf{x}^*,\boldsymbol{\omega}) q_{\boldsymbol{\theta}^*}(\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega}. \tag{3.8}$$

[13] It is also possible to minimise the $\mathrm{D}_{\mathrm{KL}}(p\|q)$ instead of $\mathrm{D}_{\mathrm{KL}}(q\|p)$. This also results in $q_{\boldsymbol{\theta}^*}(\boldsymbol{\omega})$ minimising a divergence between the two distributions. This reversed form is less mode-seeking and results in a variational approximation that tends to cover more of the space at the compromise of a worse approximation at the mode.

One could therefore imagine selecting $q_{\boldsymbol{\theta}^*}(\boldsymbol{\omega}) \in \mathcal{Q}$, where $\mathcal{Q}$ is a set of distributions from a family that make this integral tractable. As an example, if the likelihood $p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega})$ were Gaussian then $\mathcal{Q}$ could cover all possible univariate Gaussian distributions.

Finally, going back to the issue of the KL divergence containing the true posterior, we expand the posterior using Bayes' rule and rearrange as follows:[14]

$$
\begin{aligned}
&\mathrm{D}_{\mathrm{KL}}\big(q_{\boldsymbol{\theta}}(\boldsymbol{\omega}) \| p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})\big) \\
&= -\int_{\boldsymbol{\omega}} q_{\boldsymbol{\theta}}(\boldsymbol{\omega}) \log\left(\frac{p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega}) p(\boldsymbol{\omega})}{p(\mathbf{Y}|\mathbf{X}) q_{\boldsymbol{\theta}}(\boldsymbol{\omega})}\right) \mathrm{d}\boldsymbol{\omega} \\
&= \int_{\boldsymbol{\omega}} q(\boldsymbol{\omega}) \log p(\mathbf{Y}|\mathbf{X}) \mathrm{d}\boldsymbol{\omega} - \int_{\boldsymbol{\omega}} q_{\boldsymbol{\theta}}(\boldsymbol{\omega}) \log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega} - \int_{\boldsymbol{\omega}} q_{\boldsymbol{\theta}}(\boldsymbol{\omega}) \log\left(\frac{p(\boldsymbol{\omega})}{q_{\boldsymbol{\theta}}(\boldsymbol{\omega})}\right) \mathrm{d}\boldsymbol{\omega} \\
&= \log p(\mathbf{Y}|\mathbf{X}) - \int_{\boldsymbol{\omega}} q_{\boldsymbol{\theta}}(\boldsymbol{\omega}) \log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega} + \mathrm{D}_{\mathrm{KL}}(q_{\boldsymbol{\theta}}(\boldsymbol{\omega}) \| p(\boldsymbol{\omega})). \qquad (3.9)
\end{aligned}
$$

We can then group the two terms on the right-hand side that contain $q_{\boldsymbol{\theta}}(\boldsymbol{\omega})$ and define the evidence lower bound (ELBO) as

$$
\mathcal{L}_{\mathrm{VI}}(\boldsymbol{\theta}) := \int_{\boldsymbol{\omega}} q_{\boldsymbol{\theta}}(\boldsymbol{\omega}) \log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega} - \mathrm{D}_{\mathrm{KL}}(q_{\boldsymbol{\theta}}(\boldsymbol{\omega}) \| p(\boldsymbol{\omega})). \qquad (3.10)
$$

The reasoning for referring to $\mathcal{L}_{\mathrm{VI}}(\boldsymbol{\theta})$ as the ELBO comes from a further rearrangement of Equation (3.9) to get

$$
\log p(\mathbf{Y}|\mathbf{X}) = \mathcal{L}_{\mathrm{VI}}(\boldsymbol{\theta}) + \mathrm{D}_{\mathrm{KL}}\big(q_{\boldsymbol{\theta}}(\boldsymbol{\omega}) \| p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})\big). \qquad (3.11)
$$

Therefore as $\mathrm{D}_{\mathrm{KL}}\big(q_{\boldsymbol{\theta}}(\boldsymbol{\omega}) \| p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})\big) \geq 0$, then maximising $\mathcal{L}_{\mathrm{VI}}(\boldsymbol{\theta})$ minimises the KL divergence, which was exactly the motivation set out in Equation (3.7). Finally we can write the above as

$$
\log p(\mathbf{Y}|\mathbf{X}) \geq \mathcal{L}_{\mathrm{VI}}(\boldsymbol{\theta}), \qquad (3.12)
$$

[14] $\int_{\boldsymbol{\omega}} q(\boldsymbol{\omega}) \log p(\mathbf{Y}|\mathbf{X}) \mathrm{d}\boldsymbol{\omega} = \log p(\mathbf{Y}|\mathbf{X})$ as $\log p(\mathbf{Y}|\mathbf{X})$ is not dependent on $\boldsymbol{\omega}$.

which shows that $\mathcal{L}_{\text{VI}}(\boldsymbol{\theta})$ is in fact a lower bound on the log evidence.

3.2.3 Stochastic optimisation

For computational efficiency, gradients with respect to the objective (in this case the ELBO) are often split into *mini-batches*, otherwise referred to as a small subset of samples, $\mathcal{S}_k \subseteq \{1, \ldots, N\}$ [Bottou, 2010]. These stochastic gradient optimisers build on the foundations laid by Robbins and Monro [1951]. In the optimisation literature, if $|\mathcal{S}_k| = N$, it is called *batch* optimisation and this requires calculating the derivative of the likelihood over the entire data set, which can be computationally prohibitive, as has previously been mentioned in the context of HMC.[15] Therefore if we redefine the ELBO as explicitly dependent on the data,

$$\mathcal{L}_{\text{VI}}(\boldsymbol{\theta}; \mathbf{X}, \mathbf{Y}) = \sum_{i=1}^{N} \mathcal{L}_{\text{VI}}(\boldsymbol{\theta}; \mathbf{x}_i, \mathbf{y}_i), \tag{3.13}$$

then a single mini-batch that approximates the entire ELBO can be defined as

$$\hat{\mathcal{L}}_{\text{VI}}(\boldsymbol{\theta}) := \frac{N}{|\mathcal{S}_k|} \sum_{i \in \mathcal{S}_k} \mathcal{L}_{\text{VI}}(\boldsymbol{\theta}; \mathbf{x}_i, \mathbf{y}_i), \tag{3.14}$$

where $\mathbb{E}_{\mathcal{S}_k}[\hat{\mathcal{L}}_{\text{VI}}(\boldsymbol{\theta})] = \mathcal{L}_{\text{VI}}(\boldsymbol{\theta}; \mathbf{X}, \mathbf{Y})$ and is therefore an unbiased estimator.

3.2.4 Approximate inference using stochastic variational inference

Finding variational families that are expressive enough but also allow for tractable integrals can present a challenge, especially for neural networks. As already mentioned in Section 3.1, deriving integrals for even the most simple of variational families can

[15] Naïvely, stochastic optimisation originally refers to $\mathcal{S}_k = 1$, hence the requirement of the term *mini-batch* or *data sub-sampling*.

become too complex when working with neural networks that have multiple layers. As noted by Graves [2011] and those thereafter, combining the approaches of Monte Carlo techniques and variational inference, is a scalable way of performing Bayesian inference in BNNs.

Therefore, we can introduce the objective in Equation (3.10), reformulated under SVI by replacing the integral over the data-dependent term with MC estimates drawn according to $\boldsymbol{\omega}^{(s)} \sim q_{\boldsymbol{\theta}}(\boldsymbol{\omega})$:

$$\mathcal{L}_{\text{MC}}(\boldsymbol{\theta}) = \frac{N}{|\mathcal{S}_k|} \sum_{i \in \mathcal{S}_k} \sum_{s} \log p(\mathbf{y}_i | \mathbf{x}_i, \boldsymbol{\omega}^{(s)}) - \text{D}_{\text{KL}}(q_{\boldsymbol{\theta}}(\boldsymbol{\omega}) \| p(\boldsymbol{\omega})). \tag{3.15}$$

The KL term on the right-hand side can be tractable and therefore may not require a stochastic approximation. For example, if the variational distribution is a multivariate Gaussian then the term can be computed exactly as the KL divergence between two Gaussians is known. For further details regarding SVI, see Hoffman et al. [2013].

3.2.5 Dropout: a commonly used inference technique for Bayesian neural networks

We will now briefly look into Monte Carlo dropout as it is a widely used approximate inference technique in BNNs. This summary focusses on a BNN with a single hidden layer, where for a full derivation and an extension to multiple layers, please refer to the supplementary materials of Gal and Ghahramani [2016]. We follow Gal and Ghahramani [2016] and begin from a Gaussian process model.[16] Therefore, we start with a valid covariance function for a GP,

$$\mathbf{K}(\mathbf{x}, \mathbf{x}') = \iint p(\mathbf{w}) p(b) h(\mathbf{w}^\top \mathbf{x} + b) h(\mathbf{w}^\top \mathbf{x} + b) \text{d}\mathbf{w} \text{d}b, \tag{3.16}$$

[16] This can be contrasted with the approach of Neal [1995], who begins with a BNN and shows how it is related to a Gaussian process.

where we are taking the covariance of the non-linear function in a similar manner to Equation (2.6). We can then approximate $\mathbf{K}(\mathbf{x}, \mathbf{x}')$ with $\hat{\mathbf{K}}(\mathbf{x}, \mathbf{x}') = \frac{1}{K}\sum_k h(\mathbf{w}^\top \mathbf{x} + b)h(\mathbf{w}^\top \mathbf{x} + b)$ via K Monte Carlo samples.[17] These samples then correspond to the number of hidden units in the single layer. Following the standard GP methodology [Williams and Rasmussen, 2006], the marginal likelihood can be written as

$$p(\mathbf{Y}|\mathbf{X}) = \int p(\mathbf{Y}|\mathbf{F})p(\mathbf{F}|\mathbf{W}_1, \mathbf{b}, \mathbf{X})p(\mathbf{W}_1)p(\mathbf{b})\mathrm{d}\mathbf{F}\mathrm{d}\mathbf{W}_1\mathrm{d}\mathbf{b}, \tag{3.17}$$

where for regression, we use the same Gaussian likelihood as in Equation (2.2) for Bayesian linear regression. We have now changed from the summation over K to matrix notation. The final step to getting a Gaussian process that has the structure of a neural network is to introduce an auxiliary random variable that will correspond to $\mathbf{W}_2$. This random variable is introduced using the Gaussian identity from Bishop [2006, Page 93],

$$\begin{aligned} p(\mathbf{Y}|\mathbf{F})p(\mathbf{F}|\mathbf{W}_1, \mathbf{b}) &= \mathcal{N}(\mathbf{Y}; \mathbf{0}, \mathbf{\Phi}\mathbf{\Phi}^\top + \lambda^{-1}\mathbf{I}_N) \\ &= \int \prod_d \mathcal{N}(\mathbf{y}_d; \mathbf{\Phi}\mathbf{w_d}, \lambda^{-1}\mathbf{I}_N)\mathcal{N}(\mathbf{w}_d; \mathbf{0}, \mathbf{I}_K)\mathrm{d}\mathbf{w}_d, \end{aligned}$$

which, if we define $\mathbf{W}_2 = [\mathbf{w}_d]_{d=1}^D$ such that $\mathbf{W}_2 \in \mathbb{R}^{K\times D}$, we retrieve the marginal likelihood for the approximate Gaussian process

$$p(\mathbf{Y}|\mathbf{X}) = \int p(\mathbf{Y}|\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}, \mathbf{X})p(\mathbf{W}_1)p(\mathbf{W}_2)p(\mathbf{b})\mathrm{d}\mathbf{W}_1\mathrm{d}\mathbf{W}_2\mathrm{d}\mathbf{b}, \tag{3.18}$$

which is equivalent to a single layer neural network.

The problem remains as to how to perform this integral and therefore Gal and Ghahramani [2016] introduced a variational distribution $q(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}) = q(\mathbf{W}_1)q(\mathbf{W}_2)$ $q(\mathbf{b})$, such that for both $\mathbf{W}_1$ and $\mathbf{W}_2$, the corresponding distribution is $q(\mathbf{W}_l) =$

[17] If we let $\boldsymbol{\phi} = \sqrt{\frac{1}{K}}h(\mathbf{W}_1^\top \mathbf{x} + \mathbf{b})$ then $\hat{\mathbf{K}}(\mathbf{x}, \mathbf{x}') = \mathbf{\Phi}\mathbf{\Phi}^\top$, where $\mathbf{\Phi}_n = \boldsymbol{\phi}(\mathbf{x}_n)$.

$\prod_{q=1}^{Q_{l-1}} q(\mathbf{w}_q)$ where,

$$q(\mathbf{w}_q) = p_l \mathcal{N}(\mathbf{m}_l, \boldsymbol{\sigma}^2 \mathbf{I}_{Q_{l+1}}) + (1 - p_l)\mathcal{N}(0, \boldsymbol{\sigma}^2 \mathbf{I}_{Q_{l+1}}),$$

$q(\mathbf{b}) = \mathcal{N}(\mathbf{m}, \boldsymbol{\sigma}^2 \mathbf{I}_{Q_2})$, and the introduction of the layer-wise dropout parameter p_l is used to parameterise a Bernoulli distribution such that $\mathbf{z}_l \sim \text{Bernoulli}(p_l)$. Therefore $\mathbf{z}_l$ corresponds to the dropout mask of each layer and has the effect of zeroing out a random selection of rows in each layer's weight matrix. The variational parameters to learn are now p_1, p_2, $\mathbf{M}_1$, $\mathbf{M}_2$ and $\mathbf{m}$.

We can learn these variational parameters by deriving the ELBO as in Equation (3.15). Employing the reparameterisation trick [Kingma et al., 2014, Rezende et al., 2014], we allow the parameters to depend on random variables $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\mathbf{z}_l \sim \text{Bernoulli}(p_l)$ such that

$$\begin{aligned} \mathbf{W}_1 &= \mathbf{z}_1(\mathbf{M}_1 + \boldsymbol{\sigma}\boldsymbol{\epsilon}_1) + (1 - \mathbf{z}_1)\boldsymbol{\sigma}\boldsymbol{\epsilon}_1, \\ \mathbf{W}_2 &= \mathbf{z}_2(\mathbf{M}_2 + \boldsymbol{\sigma}\boldsymbol{\epsilon}_2) + (1 - \mathbf{z}_2)\boldsymbol{\sigma}\boldsymbol{\epsilon}_2, \\ \mathbf{b} &= \mathbf{m} + \boldsymbol{\sigma}\boldsymbol{\epsilon}. \end{aligned} \tag{3.19}$$

Following the same notation as Gal and Ghahramani [2016], sampling realisations $\hat{\mathbf{W}}_1$, $\hat{\mathbf{W}}_2$ and $\hat{\mathbf{b}}$, we can write the ELBO of this GP model as:

$$\mathcal{L}_{\text{GP-MC}}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \log p(\mathbf{y}_n | \mathbf{x}_n, \hat{\mathbf{W}}_1, \hat{\mathbf{W}}_2, \hat{\mathbf{b}}) - \text{D}_{\text{KL}}\left(q_{\boldsymbol{\theta}}(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}) \| p(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b})\right). \tag{3.20}$$

We are now able to evaluate the likelihood component and its derivative, which we can use in the the optimisation of the ELBO. The final step is to provide an approximation to the KL divergence component. This approximation is required as there is no exact way of formulating the KL divergence between a mixture of Gaussians. The approximation is reliant on the law of large numbers, which applies when we are using

networks with a large number of weights. In his thesis, Gal [2016b] refers to this as the *KL condition* and when it is satisfied, it results in an objective equivalent to that of optimising a neural network whilst using dropout as a regularisation technique.[18] Therefore we can write

$$\mathcal{L}_{\text{GP-MC}}(\boldsymbol{\theta}) \propto -\frac{\lambda}{2N}\sum_{n=1}^{N}\|\mathbf{y}_n - \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_n)\|_2^2 - \frac{\tau p_1}{2N}\|\mathbf{M}_1\|_2^2 - \frac{\tau p_2}{2N}\|\mathbf{M}_2\|_2^2 - \frac{\tau}{2N}\|\mathbf{m}\|_2^2, \quad (3.21)$$

where τ is the prior precision for the network parameters. Therefore performing dropout by sampling realisations of the weights during training and optimising the standard NN objective (squared error with L2 regularisation) is equivalent to performing SVI over a BNN. This is possible as we are able to evaluate expectations over the derivative $\frac{\partial}{\partial \boldsymbol{\theta}}\mathcal{L}_{\text{GP-MC}}(\boldsymbol{\theta})$, which we use in the optimisation process. Then at test time, we can use MC samples to infer predictive distributions over test points.

As a final note, in the above derivation of Monte Carlo dropout, it is assumed that each layer-wise dropout parameter p_l is selected via a grid-search. However it is also possible to include these variational parameters as part of the ELBO and optimise with respect to both the layer-wise dropout parameters and the variational weights, such that $\boldsymbol{\theta} = \{p_1, p_2, \mathbf{M}_1, \mathbf{M}_2, \mathbf{m}\}$. In order to take derivatives with respect to p_l, we can no longer ignore terms dependent on p_l in the ELBO that were previously treated as constants. Therefore the term corresponding to the entropy of a Bernoulli random variable with probability p_l must be included (please refer back to the supplementary materials of Gal and Ghahramani [2016]). Furthermore, we are required to take gradients with respect to p_l, where these gradients do not exist for a Bernoulli random variable. We can therefore employ a reparameterisation trick for the discrete Bernoulli distribution in a manner similar to that of Equation (3.19) for the variational weights. However, as noted in Gal et al. [2017a], this requires a *'Concrete*

[18] This final step requires a further approximation that results in the standard deviation $\boldsymbol{\sigma}$ in Equation (3.19) to tend to zero. This leads to $\hat{\mathbf{W}}_1 \approx \hat{\mathbf{z}}_1\mathbf{M}_1$, $\hat{\mathbf{W}}_2 \approx \hat{\mathbf{z}}_2\mathbf{M}_2$ and $\hat{\mathbf{b}} \approx \mathbf{b}$. Therefore the sampling of $\boldsymbol{\epsilon}$ is no longer required.

distribution relaxation', where the Concrete distribution is a continuous distribution that can be used to approximate discrete random variables. The Concrete distribution can be written as a function $g(p_l, u)$, where u is distributed according to the uniform distribution and is not dependent on p_l. This reparameterisation leads to $g(p_l, u)$ being a differentiable function with respect to p_l. The result is that we can now automatically tune for the layer-wise dropout parameters jointly as part of the inference scheme, which is an advantage over grid search. See Gal et al. [2017a] for a full derivation of Concrete dropout.

3.2.6 Extending to multiplicative normalising flow

It is possible to draw parallels between dropout and other SVI techniques in BNNs. In fact Louizos and Welling [2017] provide a table that describes the corresponding prior and posterior that each technique uses. For example, a deterministic neural network with L2 regularisation is equivalent to using a normal prior $\mathcal{N}(\mathbf{0}, \mathbf{I})$ and a posterior that is a delta peak, whereas MC dropout uses the same normal prior but has a posterior that is a mixture of zero and delta peaks. Furthermore, in the work by Louizos and Welling [2017], they add to the table by introducing multiplicative normalising flows for performing variational inference in Bayesian neural networks. What is interesting is how it extends on MC dropout and how it aims to provide more flexible posteriors. For MC dropout, $q(\mathbf{z}_l)$ is a Bernoulli distribution and $q(\mathbf{W}_l|\mathbf{z}_l)$ is a Gaussian with zero variance (see Equation (3.19)). For normalising flows, Louizos and Welling [2017] allow the approximate posterior of the weights, $q(\mathbf{W}_l) = \int q(\mathbf{W}_l|\mathbf{z}_l)q(\mathbf{z}_l)\mathrm{d}\mathbf{z}_l$, to be more flexible by applying probability conserving transformations to $q(\mathbf{z}_l)$. In applying normalising flow to $q(\mathbf{z}_l)$, the authors are able to derive an ELBO with a posterior that is built via multiplicative normalising flows. The advantage is shown by better performance over MC dropout in a series of experiments that also compares to Blundell et al. [2015]. Despite normalising flow providing better uncertainty estimates,

MC dropout is still competitive in terms of its performance and also comes with the significant advantage of being easy to implement, both in computational complexity (normalising flow requires a series of invertible Jacobian matrices) and in its wide availability in modern day deep learning libraries [Chollet et al., 2015, Paszke et al., 2017a].

3.3 Conclusion

This chapter has introduced the literature surrounding Bayesian neural networks and pointed out what challenges existed in the past. This started with the task of extending the McCulloch and Pitts model to multiple layers and now the challenge is that of scaling Bayesian neural networks to large data sets without losing the benefits of being Bayesian. The difficulty lies in finding a suitable approach to performing approximate inference, where the two main techniques are Monte Carlo estimation and variational inference. In order to scale variational inference to BNNs with multiple layers, the current solution is to employ SVI, which combines both Monte Carlo estimation and variational inference. In this chapter, these approaches have been summarised within the context of the BNN literature, where MC dropout was introduced, which is a specific example of a commonly used SVI technique for BNNs. Therefore we are now in the position to extend these techniques by improving on their ability to make predictions with appropriate uncertainty estimations, whilst scaling to large models and big data. In particular, Chapter 4 extends MC dropout to real-world decision-making tasks, whilst ensuring that uncertainty estimates are appropriately calibrated to each problem. Chapters 5 and 6 improve on approximate inference using Monte Carlo methods and Chapter 7 extends on BNN loss functions to provide better uncertainty estimates in a real-world astrophysical application.

Chapter 4

Bayesian decision theory with Bayesian neural networks

Bayesian neural networks provide distributions over their outputs and are therefore especially suited for decision-making tasks. However, the framework of Bayesian decision theory is rarely relied upon when implementing BNNs and is therefore an area that has seldom been explored. Furthermore, current approaches in approximate inference typically do not incorporate knowledge of the final application, and therefore cannot guarantee optimal decisions for a given task. To make more suitable task-specific approximations, this chapter introduces a new loss-calibrated evidence lower bound for Bayesian neural networks in the context of supervised learning, informed by Bayesian decision theory. This chapter builds on work that has been presented at 'The theory of deep learning workshop, ICML 2018' [Cobb et al., 2018b].

4.1 Motivation: the utility function

Consider a scenario where we are required to diagnose a patient as either being severely ill, moderately ill or healthy. We can assign a loss associated with each outcome and build a loss matrix as follows:

Table 4.1: Illustration of a loss matrix for a medical example.

		Decision		
	Cost	Healthy	Mild	Severe
True	Healthy	0	40	45
	Mild	50	0	30
	Severe	100	35	0

This loss matrix manages to capture our preferences by assigning the highest cost to diagnosing a patient as healthy when they are severely ill (100). Correct diagnosis results in no penalty (0) and false positives are assigned lower costs than false negatives. Equivalently, we can also represent this loss matrix as a utility matrix, whereby we view utility as a negative loss. In addition, we can shift all the utilities by a constant as our interest is in the relative utilities of decisions. For example the utility in Table 4.2 equally captures the task-specific preferences.

Table 4.2: Illustration of a utility matrix for a medical example.

		Decision		
	Utility	Healthy	Mild	Severe
True	Healthy	100	60	55
	Mild	50	100	70
	Severe	0	65	100

Although in this example we have designed rather arbitrary utilities to capture our preferences, the values in such applications will often be assigned according to requirements set by health organisations. As an example, Leibig et al. [2017] compare their (non-calibrated) results to sensitivity and specificity thresholds set by the NHS (UK National Health Service).

The above example is to demonstrate how we can utilise a utility matrix to encode our preference for making decisions. Selecting the decision with the highest expected utility results in a diagnosis that matches our preferences defined in the utility function. These ideas are made concise in the field of Bayesian decision theory, which is

concerned with making decisions given specified utility functions [Berger, 1985].

4.1.1 Terminology of Bayesian decision theory

We rely on the framework of Bayesian decision theory for our model (see Berger [1985, Chapter 1 & Chapter 5] for more details). The motivation for selecting this framework is due to the way in which it deals with uncertainty. Any prediction we make should involve the uncertainty in our knowledge over the *state of nature*, $\boldsymbol{\omega}$. Specifically to BNNs, we can think of our knowledge of the world state as our confidence in the model parameters (i.e. the function explaining the data), which is given by the posterior over the weights.

Utility. A utility function defines relative preferences between certain outcomes. It may not always be possible to define a utility function as user preferences may not satisfy certain axioms, such as a logical order of preferences, or a user may have an infinite preference for a certain choice.[1] We leave out specific discussion of utility theory but we note that in constructing a utility function, we are effectively describing rational user behaviour. Furthermore, since utilities are the quantification of preferences, scaling and shifting utilities have no effect on the outcome of a decision-making process. In our context, a utility function is defined as $u(\mathbf{h}, \mathbf{y}^*)$, where $\mathbf{h}$ is the decision and $\mathbf{y}^*$ is the prediction. Using our example in Table 4.2, $\mathbf{h}$ corresponds to the columns and $\mathbf{y}^*$ corresponds to the rows. If $\mathbf{y}^*$ is subsequently known then the matrix can be used to report the utility of decision $\mathbf{h}$. However if we instead have a distribution over $\mathbf{y}^*$ (i.e. the predictive distribution) then we must use this uncertainty combined with the utility to make a decision.

Expected utility. The expected utility, or the *expected posterior utility*, is the result of averaging the utility over the predictions $\mathbf{y}^*$. For example, if our prediction

[1] See [Berger, 1985, Page 49] for a list of these axioms. See DeGroot [2005] for existence proofs.

for *healthy*, *mild* and *severe* were $\mathbf{y}^* \sim [0.1, 0.2, 0.7]$, then the expected utility for deciding severe would be

$$\begin{aligned}\mathbb{E}[u|\mathbf{h} = \text{Severe}] &= \int_{\mathbf{y}^*} u\left(\mathbf{h}, \mathbf{y}^*\right) p(\mathbf{y}^*) \mathrm{d}\mathbf{y}^* \\ &= (0.1 \times 55) + (0.2 \times 70) + (0.7 \times 100) \\ &= 89.5,\end{aligned}$$

with $\mathbb{E}[u|\mathbf{h} = \text{Healthy}] = 20.0$ and $\mathbb{E}[u|\mathbf{h} = \text{Mild}] = 71.5$.

Therefore the expected utility is a function of the decision, which we call the *posterior gain*. The ultimate objective is to select the decision that will maximise the posterior gain, i.e. the Bayes optimal action. In the presence of data, we introduce the *predictive conditional-gain*, $\mathcal{G}(\mathbf{h} \mid \mathbf{x}^*)$, which we use in making a decision conditioned on a test input $\mathbf{x}^*$:

$$\mathcal{G}(\mathbf{h} \mid \mathbf{x}^*) = \int_{\mathbf{y}^*} u\left(\mathbf{h}, \mathbf{y}^*\right) p(\mathbf{y}^* \mid \mathbf{x}^*, \mathbf{X}, \mathbf{Y}) \mathrm{d}\mathbf{y}^*. \tag{4.1}$$

The Bayes optimal action, which we will refer to as the optimal decision can then be determined via the optimisation,

$$\begin{aligned}\mathbf{h}^*(\mathbf{x}^*) &= \underset{\mathbf{h} \in \mathcal{H}}{\operatorname{argmax}}\ \mathcal{G}(\mathbf{h} \mid \mathbf{x}^*) \\ &= \underset{\mathbf{h} \in \mathcal{H}}{\operatorname{argmax}}\ \log\left(\mathcal{G}(\mathbf{h} \mid \mathbf{x}^*)\right),\end{aligned} \tag{4.2}$$

where $\mathcal{H}$ is the space of all possible decisions.

Supervised learning. The work within this chapter applies Bayesian decision theory within the context of supervised learning. Therefore we are working with labelled data $\{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$ and we are concerned with inferring a predictive distribution over $\mathbf{y}^*$ conditioned on a test input and all the observed data. We will be concentrating

on classification, whereby the medical diagnosis example given previously was an illustration of a classification problem. For classification in supervised learning using BNNs, $\mathbf{y}_i$ denotes the observed label for input $\mathbf{x}_i$, which takes values $\mathbf{c}$ from the space of all possible classes $\mathcal{C}$ (e.g. class labels 0 to 9 for MNIST experiments LeCun et al. [1998]).[2] We refer to the chosen label for a given input $\mathbf{x}_i$ as the decision, $\mathbf{h}_i$, which can take any label assignment $\mathbf{c} \in \mathcal{C}$. We denote the probability vector output from the model as $\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i)$. The probability vector model output $\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i)$ can be seen as a 'recommendation', where the actual chosen label or action could be different. Therefore the definition of the utility within this context could be explicitly written as $u\left(\mathbf{h} = \mathbf{c}, \mathbf{y}^* = \mathbf{c}'\right)$ (or $u\left(\mathbf{c}, \mathbf{c}'\right)$), which defines what we will gain from different decisions on labels $\mathbf{h}$.

Conditional-gain for supervised classification. It will be important when combining Bayesian decision theory with BNNs to redefine the conditional-gain in terms of the network weights. Therefore, we use the language of supervised classification to rewrite the predictive conditional-gain in terms of an integration with respect to $\boldsymbol{\omega}$:

$$\begin{aligned}
\mathcal{G}(\mathbf{h} = \mathbf{c} \mid \mathbf{x}^*) &= \int_{\mathbf{c}'} u\left(\mathbf{c}, \mathbf{c}'\right) p(\mathbf{y}^* = \mathbf{c}' \mid \mathbf{x}^*, \mathbf{X}, \mathbf{Y}) \mathrm{d}\mathbf{c}' \\
&= \int_{\mathbf{c}'} u\left(\mathbf{c}, \mathbf{c}'\right) \int_{\boldsymbol{\omega}} p(\mathbf{y}^* = \mathbf{c}' \mid \boldsymbol{\omega}, \mathbf{x}^*) p(\boldsymbol{\omega} \mid \mathbf{X}, \mathbf{Y}) \mathrm{d}\boldsymbol{\omega} \mathrm{d}\mathbf{c}' \\
&= \int_{\boldsymbol{\omega}} \left[\int_{\mathbf{c}'} u\left(\mathbf{c}, \mathbf{c}'\right) p(\mathbf{y}^* = \mathbf{c}' \mid \boldsymbol{\omega}, \mathbf{x}^*) \mathrm{d}\mathbf{c}' \right] p(\boldsymbol{\omega} \mid \mathbf{X}, \mathbf{Y}) \mathrm{d}\boldsymbol{\omega} \\
&= \int_{\boldsymbol{\omega}} \mathcal{G}(\mathbf{h} = \mathbf{c} \mid \mathbf{x}^*, \boldsymbol{\omega}) p(\boldsymbol{\omega} \mid \mathbf{X}, \mathbf{Y}) \mathrm{d}\boldsymbol{\omega},
\end{aligned}$$

with the definition

$$\mathcal{G}(\mathbf{h} = \mathbf{c} \mid \mathbf{x}^*, \boldsymbol{\omega}) := \int_{\mathbf{c}'} u\left(\mathbf{c}, \mathbf{c}'\right) p(\mathbf{y}^* = \mathbf{c}' \mid \boldsymbol{\omega}, \mathbf{x}^*) \mathrm{d}\mathbf{c}'. \tag{4.3}$$

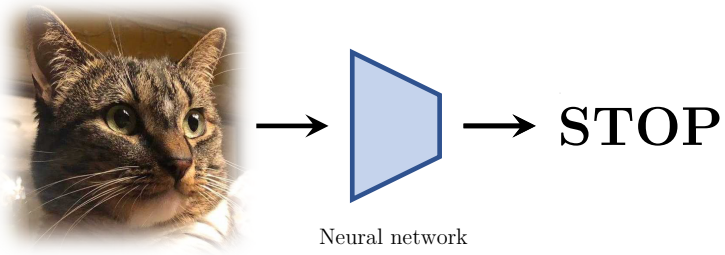
[2] In our notation we use a vector $\mathbf{c}$ to allow for multiple model outputs.

For example, if the likelihood is the negative cross-entropy, with $\mathbf{y}^*$ taking values of possible classes c, then we calculate $\mathcal{G}(\mathbf{h} \mid \mathbf{x}^*, \boldsymbol{\omega})$ by averaging the utility $u(\mathbf{h}, \mathbf{y}^* = c')$ with respect to all classes $c' \in \mathcal{C}$, weighted by the probability of that class.[3]

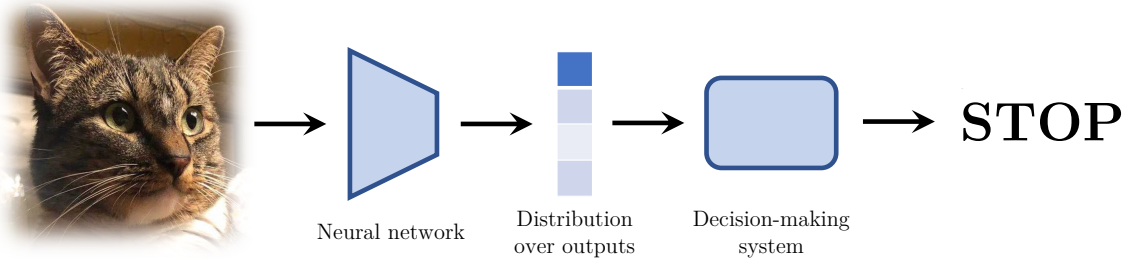
4.1.2 The benefits of Bayesian decision theory

Bayesian decision theory is a framework that seamlessly combines uncertainty with task-specific utility functions to make rational decisions. We can encode any asymmetries into our utility function and rely on the framework of Bayesian decision theory to make a decision that maximises the expected utility. Building a system that relies on Bayesian decision theory provides advantages over systems with end-to-end controllers. Clear preferences can be encoded into the decision-making process that allow for more understandable actions. Furthermore, hand-crafting objectives for machine learning models to do end-to-end control could lead to detrimental results as specifying the relationship between a decision and an observation can be a challenge. For example, constructing a data set and a neural network that can convert observations of the environment into controller actions for an autonomous car is not as obvious as understanding the reasoning behind why a car acts in a certain way becomes harder to determine (see Figure 4.1a). If a cat were to walk out in front of a car, then a computer vision system that can classify cats (as well as other objects) can notify the decision-making system that there is a cat in front of the car with some confidence. Given this knowledge, the decision-making engine can then decide to stop the car, as shown in Figure 4.1b. Separating these two processes allows the neural network to concentrate on pattern recognition tasks, which can then inform the user or a controller to make a decision. End-to-end controllers can obfuscate these two processes, making it hard to understand how decisions were made, or which part of the system requires improving. This example is shown in Figure 4.1.

[3] We note that for regression, if our likelihood were $\mathcal{N}(\mathbf{y}^*; \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}^*), \boldsymbol{\Sigma})$, as is common for regression, we could use MC sampling to approximate the conditional-gain.



(a) Relying on a neural network to provide end-to-end control.



(b) Splitting the network and control to provide more clarity over decisions.

Figure 4.1: Example to display the difference between splitting the decision-making system from the neural network. The end-to-end controller in Figure 4.1a obfuscates decision-making and classification, whereas separating the two processes, as in Figure 4.1b, can help to inform users of the reasoning behind decisions.

4.1.3 Making decisions with variational inference

If we are to use BNNs in real-world applications, such as medical diagnosis [Esteva et al., 2017] and autonomous driving [Sallab et al., 2017], then BNNs must be able to scale to large data sets and networks with millions of parameters. Stochastic variational inference is currently the only viable option in these scenarios to achieve scalable solutions (e.g. Leibig et al. [2017]). Unfortunately using Bayesian decision theory with variational inference is non-trivial. Asymmetric utilities may result in sub-optimal predictions when using an approximate inference method that is task-agnostic. The act of simply minimising a distance metric between an approximate and a true distribution may achieve a high overall predictive accuracy, but it might not be accurate enough over important regions of the output space (such as regions where there might be a high cost for incorrect predictions). As an example, if an engine has

a temperature sensor that we use to predict the possibility of a catastrophic failure, we are predominantly concerned about the accuracy of our model in the space near the temperature threshold. Therefore in previous work by Lacoste-Julien et al. [2011], it was scenarios such as these that led to the argument that models should be aware of the utility during inference, if they are required to make approximations. In their work, they explored toy problems, where their solutions were either tractable (e.g. Gaussian process regression), or used the Laplace approximation to overcome the intractability in Gaussian process classification. In BNNs, we are required to solve issues of intractability in more complex models. Therefore, we must incorporate ideas from modern stochastic variational inference to scale to real-world problems.

We now extend the framework introduced by Lacoste-Julien et al. [2011] with MC dropout for BNNs [Gal and Ghahramani, 2016] to provide a theoretically sound way of making decisions for real-world learning tasks. Therefore a new evidence lower bound loss for Bayesian neural network inference is introduced in the next section that incorporates the utility function. It will be shown that our newly derived lower bound for a BNN is implemented as a standard dropout neural network loss with an additional utility term.

4.2 Loss-calibrated approximate inference in Bayesian neural networks

In introducing Bayesian decision theory, it is now possible to see how approximating the true posterior of a BNN may lead to sub-optimal decisions in terms of a task-specific utility. We may learn a lower bound that is loose in areas where our utility demands a tighter fit. Therefore we extend the loss function of a BNN by deriving a new lower bound that depends on the network weights and the utility.

We define the marginal conditional-gain $\mathcal{G}(\mathbf{H} \mid \mathbf{X})$ for the entire input data using

a conditional independence assumption over our inputs where

$$\begin{aligned}\mathcal{G}(\mathbf{H} \mid \mathbf{X}) &:= \int_{\boldsymbol{\omega}} \prod_{j} \mathcal{G}(\mathbf{h}_j \mid \mathbf{x}_j, \boldsymbol{\omega}) p(\boldsymbol{\omega} \mid \mathbf{X}, \mathbf{Y}) \mathrm{d}\boldsymbol{\omega} \\ &= \int_{\boldsymbol{\omega}} \mathcal{G}(\mathbf{H} \mid \mathbf{X}, \boldsymbol{\omega}) p(\boldsymbol{\omega} \mid \mathbf{X}, \mathbf{Y}) \mathrm{d}\boldsymbol{\omega}\end{aligned} \tag{4.4}$$

and where we assume that given the model parameters, the decision depends only on the input $\mathbf{x}_j$. If this conditional-gain is large, we have assigned high predictive probability to class labels that give a high task-specific utility across our data. Conversely, low values of the conditional-gain imply that our choice of $\mathbf{H}$ has led to an undesirable low task-specific utility over our data. Therefore, given our aim of assigning class labels in a way that maximises the utility, we choose to maximise the conditional-gain. Furthermore, we will show that this is equivalent to minimising a KL divergence between the approximating distribution $q(\boldsymbol{\omega})$ and a *calibrated* posterior, which results in a loss function that is comparable to the BNN loss introduced in Section 3.2.5.

In order to maximise the conditional-gain, we must integrate with respect to the parameters $\boldsymbol{\omega}$ and optimise with respect to the decisions $\mathbf{H}$. However, due to the intractability of the integration, we must define a lower bound to the log conditional-gain, which we maximise instead:

$$\log\left(\mathcal{G}(\mathbf{H} \mid \mathbf{X})\right) \geq \mathcal{L}(q(\boldsymbol{\omega}), \mathbf{H}), \tag{4.5}$$

where we follow the derivation of Lacoste-Julien et al. [2011] by introducing the

approximate posterior $q(\boldsymbol{\omega})$ and applying Jensen's inequality:

$$\begin{aligned}\log\left(\mathcal{G}(\mathbf{H}\mid\mathbf{X})\right) &= \log\left(\int_{\boldsymbol{\omega}} q(\boldsymbol{\omega})\frac{p(\boldsymbol{\omega}\mid\mathbf{X},\mathbf{Y})\mathcal{G}(\mathbf{H}\mid\mathbf{X},\boldsymbol{\omega})}{q(\boldsymbol{\omega})}\mathrm{d}\boldsymbol{\omega}\right)\\ &\geq \int_{\boldsymbol{\omega}} q(\boldsymbol{\omega})\log\left(\frac{p(\boldsymbol{\omega}\mid\mathbf{X},\mathbf{Y})\mathcal{G}(\mathbf{H}\mid\mathbf{X},\boldsymbol{\omega})}{q(\boldsymbol{\omega})}\right)\mathrm{d}\boldsymbol{\omega}\\ &:= \mathcal{L}(q(\boldsymbol{\omega}),\mathbf{H}).\end{aligned} \tag{4.6}$$

We will show that this lower bound can be approximated well and can be reformulated as the *standard optimisation objective* loss for a BNN with an additional penalty term. However, to gain further insight, we can also view the maximisation of this lower bound as equivalent to the minimisation of the KL divergence:

$$\mathrm{KL}(q\mid\mid\tilde{p}_h) = \log\left(\mathcal{G}(\mathbf{H}\mid\mathbf{X})\right) - \mathcal{L}(q,\mathbf{H}), \tag{4.7}$$

where the probability distribution

$$\tilde{p}_h = \frac{p(\boldsymbol{\omega}\mid\mathbf{X},\mathbf{Y})\mathcal{G}(\mathbf{H}\mid\mathbf{X},\boldsymbol{\omega})}{\mathcal{G}(\mathbf{H}\mid\mathbf{X})} \tag{4.8}$$

is the true posterior scaled by the conditional-gain. Therefore we calibrate the approximate posterior to take into account the utility.

We now derive the loss-calibrated ELBO for the BNN by expanding our lower bound:

$$\begin{aligned}\mathcal{L}(q(\boldsymbol{\omega}),\mathbf{H}) = &\underbrace{\int_{\boldsymbol{\omega}} q(\boldsymbol{\omega})\log p(\mathbf{Y}\mid\mathbf{X},\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega} - \mathrm{KL}(q(\boldsymbol{\omega})\mid\mid p(\boldsymbol{\omega}))}_{\text{Same as ELBO in Equation (3.20), i.e. } (\mathcal{L}_{\text{GP-MC}}(\boldsymbol{\theta}))}\\ &+\underbrace{\int_{\boldsymbol{\omega}} q(\boldsymbol{\omega})\log\mathcal{G}(\mathbf{H}\mid\mathbf{X},\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega}}_{\text{New term, requires } \mathbf{H}} + \text{const.}\end{aligned} \tag{4.9}$$

Next, using Monte Carlo integration and the dropout approximating distribution

$q(\boldsymbol{\omega})$, this can be implemented as the standard objective loss of a dropout NN with an additional utility-dependent penalty term

$$\underbrace{-\sum_i \log p(\mathbf{y}_i \mid \mathbf{x}_i, \hat{\boldsymbol{\omega}}_i) + \|\boldsymbol{\omega}\|^2}_{\text{Equivalent to standard dropout loss}} \underbrace{- \sum_i \left(\log \sum_{\mathbf{c}} u\left(\mathbf{h}_i, \mathbf{c}\right) p(\mathbf{y}_i = \mathbf{c}' \mid \mathbf{x}_i, \hat{\boldsymbol{\omega}}_i) \right)}_{\text{Our additional utility-dependent penalty term}} \tag{4.10}$$

where $\hat{\boldsymbol{\omega}}_i \sim q_{\boldsymbol{\theta}}(\boldsymbol{\omega})$. We alternate between one-step minimisation with respect to $\boldsymbol{\theta}$ and setting $\mathbf{h}_i$ using Equation (4.2).

This derivation is influenced by Lacoste-Julien et al. [2011], in which a related objective was derived for a simple tractable model, and by applying variational EM to separately optimise the two sets of parameters. In this section we extend the derivation to solve issues of non-tractability for large complex models, allowing the ideas of Lacoste-Julien et al. [2011] to be applied in real-world applications.

We display our technique for both learning the parameter weights and the decisions in Algorithm 1, where we perform MC dropout by drawing Bernoulli-distributed random variables that correspond to the dropout mask, $\mathbf{z}$, and apply the reparameterisation $\boldsymbol{\omega} = \boldsymbol{\theta}\,\mathrm{diag}(\mathbf{z})$, where $\boldsymbol{\theta}$ are the approximating distribution parameters (weight matrices' means, e.g. $\mathbf{M}_1$, $\mathbf{M}_2$ and $\mathbf{m}$ from Section 3.2.5) [Gal and Ghahramani, 2016].

Algorithm 1 LCBNN optimisation

Given dataset $\mathcal{D} = \{\mathbf{X}, \mathbf{Y}\}$, utility function $u(\mathbf{h}, \mathbf{y})$ and set of all possible labels $\mathcal{C}$

Define learning rate schedule η

Randomly initialise weights $\boldsymbol{\omega}$

repeat

  Sample S index set of training examples

  for $i \in S$ **do**

    for t from 1 to T **do**

      Sample Bernoulli-distributed random variables $\mathbf{z}^t \sim p(\mathbf{z})$ {for each each $\mathbf{x}_i$ we sample T dropout masks $\mathbf{z}^t$}

      $\mathbf{y}_i^t = \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i)$, where $\boldsymbol{\omega} \sim q(\boldsymbol{\omega}|\mathbf{z}^t)$ {Perform a stochastic forward pass with the sampled dropout mask $\mathbf{z}^t$ and $\mathbf{x}_i$}

    end for

    $\mathbf{h}_i^* \leftarrow \underset{\mathbf{h}\in\mathcal{H}}{\operatorname{argmax}} \frac{1}{T}\sum_t u(\mathbf{h}, \mathbf{y}_i^t)$ {Choose class $\mathbf{h} = \mathbf{c} \in \mathcal{C}$ which maximises average utility}

  end for

  Calculate the derivative w.r.t. $\boldsymbol{\omega}$:

$$\nabla\boldsymbol{\omega} \leftarrow -\frac{1}{T}\sum_{i\in S}\frac{\partial}{\partial\boldsymbol{\omega}}\log p\left(\mathbf{y}_i \mid \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i)\right) + \frac{\partial}{\partial\boldsymbol{\omega}}\mathrm{KL}\left(q(\boldsymbol{\omega}) \,||\, p(\boldsymbol{\omega})\right) - \frac{1}{T}\sum_{i\in S}\frac{\partial}{\partial\boldsymbol{\omega}}\log\mathcal{G}\left(\mathbf{h}_i^* \mid \mathbf{x}_i, \boldsymbol{\omega}\right)$$

  with $\mathbf{z}_i \sim p(\mathbf{z})$ a newly sampled dropout mask for each i

  Update $\boldsymbol{\omega}$: $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \eta\nabla\boldsymbol{\omega}$

until $\boldsymbol{\omega}$ has converged

Loss-calibrated KL divergence equivalence. To show that the maximisation of our loss-calibrated evidence lower bound is equivalent to minimising the KL divergence:

$$\mathrm{KL}(q \,||\, \tilde{p}_h) = \log\left(\mathcal{G}(\mathbf{H} \mid \mathbf{X})\right) - \mathcal{L}(q(\boldsymbol{\omega}), \mathbf{H}),$$

we start with $\mathrm{KL}(q \,||\, \tilde{p}_h) = \int q \log \frac{q}{\tilde{p}_h} \mathrm{d}\boldsymbol{\omega}$:

$$\begin{aligned}\mathrm{KL}(q \,||\, \tilde{p}_h) &= \int_{\boldsymbol{\omega}} q(\boldsymbol{\omega}) \log\left(\frac{q(\boldsymbol{\omega})}{\frac{p(\boldsymbol{\omega}|\mathbf{X},\mathbf{Y})\mathcal{G}(\mathbf{H}|\mathbf{X},\boldsymbol{\omega})}{\mathcal{G}(\mathbf{H}|\mathbf{X})}}\right)\mathrm{d}\boldsymbol{\omega} \\ &= \int_{\boldsymbol{\omega}} q(\boldsymbol{\omega}) \log\left(\frac{q(\boldsymbol{\omega})\mathcal{G}(\mathbf{H} \mid \mathbf{X})}{p(\boldsymbol{\omega} \mid \mathbf{X}, \mathbf{Y})\mathcal{G}(\mathbf{H} \mid \mathbf{X}, \boldsymbol{\omega})}\right)\mathrm{d}\boldsymbol{\omega}\end{aligned}$$

We can separate the above term into the log conditional-gain and the lower bound:

$$= \int_{\boldsymbol{\omega}} q(\boldsymbol{\omega}) \log \left(\mathcal{G}(\mathbf{H} \mid \mathbf{X}) \right) \mathrm{d}\boldsymbol{\omega} - \int_{\boldsymbol{\omega}} q(\boldsymbol{\omega}) \log \left(\frac{p(\boldsymbol{\omega} \mid \mathbf{X}, \mathbf{Y}) \mathcal{G}(\mathbf{H} \mid \mathbf{X}, \boldsymbol{\omega})}{q(\boldsymbol{\omega})} \right) \mathrm{d}\boldsymbol{\omega}.$$

As the conditional-gain $\mathcal{G}(\mathbf{H} \mid \mathbf{X})$ does not depend on $\boldsymbol{\omega}$ we recover:

$$\begin{aligned} \mathrm{KL}(q \mid\mid \tilde{p}_h) &= \log \left(\mathcal{G}(\mathbf{H} \mid \mathbf{X}) \right) - \int_{\boldsymbol{\omega}} q(\boldsymbol{\omega}) \log \left(\frac{p(\boldsymbol{\omega} \mid \mathbf{X}, \mathbf{Y}) \mathcal{G}(\mathbf{H} \mid \mathbf{X}, \boldsymbol{\omega})}{q(\boldsymbol{\omega})} \right) \mathrm{d}\boldsymbol{\omega} \\ &= \log \left(\mathcal{G}(\mathbf{H} \mid \mathbf{X}) \right) - \mathcal{L}(q(\boldsymbol{\omega}), \mathbf{H}) \end{aligned}$$

as previously stated.

4.3 Experiments

Utility. For all experiments we bound the utility function to take positive values, such that $\log \mathcal{G}(\mathbf{H} \mid \mathbf{X}, \boldsymbol{\omega})$ is defined for our loss-calibrated lower bound (Equation (4.9)). Therefore throughout our experiments, we define a lower bound M, such that[4]

$$M + \inf_{\mathbf{h} \in \mathcal{H},\ \mathbf{y} \in \mathcal{Y}} u\left(\mathbf{h}, \mathbf{y}\right) > 0. \tag{4.11}$$

We use the transformed utility function $u^*\left(\mathbf{h}, \mathbf{y}\right) = M + \inf_{\mathbf{h} \in \mathcal{H},\ \mathbf{y} \in \mathcal{Y}} u\left(\mathbf{h}, \mathbf{y}\right)$ for all experiments. We refer to Berger [1985, Page 60] for more information regarding transforming utilities.

[4]Note that inf corresponds to infimum.

Baseline. When training a NN, a common technique is to apply a weighting α_i to each class in the cross-entropy loss as follows:

$$\text{loss} = -\sum_{i=1}^{C} \alpha_i \log p(\mathbf{y} = c_i \mid \boldsymbol{\omega}, \mathbf{x}), \tag{4.12}$$

where for each class i, we have a corresponding weight α_i to indicate the size of its contribution to the cross-entropy loss. The term $p(\mathbf{y} = c_i \mid \boldsymbol{\omega}, \mathbf{x})$ is the categorical-likelihood as was defined in Equation (2.14). One reason for using this weighted cross-entropy loss is to overcome large class imbalances by assigning larger weights to less prevalent classes. Therefore this reason, combined with its ease of availability in many deep learning libraries (e.g. Chollet et al. [2015] and Abadi et al. [2015]), has made the weighted cross-entropy loss a widely-used technique.

4.3.1 MNIST: network capacity and label corruption

In this section, we demonstrate the use of our loss-calibrated model to target making decisions on label assignments for corrupted data with a limited capacity model. Furthermore, we show that the utility function forces the network to prioritise certain classes, in order to maximise the utility when the network is limited by capacity. Different noise levels are added to the labels to simulate a scenario where data contains corrupted observations. The corruption takes the form of a certain proportion of labels being reassigned according to a uniform distribution across all the classes. We compare our model to the weighted cross-entropy (defined in Equation (4.12)) and a standard MC dropout BNN model.

We apply our model and the baselines to a modified version of the MNIST data set [LeCun et al., 1998] and focus on maximising the utility for digits $\{3, 8\}$, where our selection consists of classes that often contain ambiguities when trying to distinguish between them. Our utility function is designed such that the maximum utility is

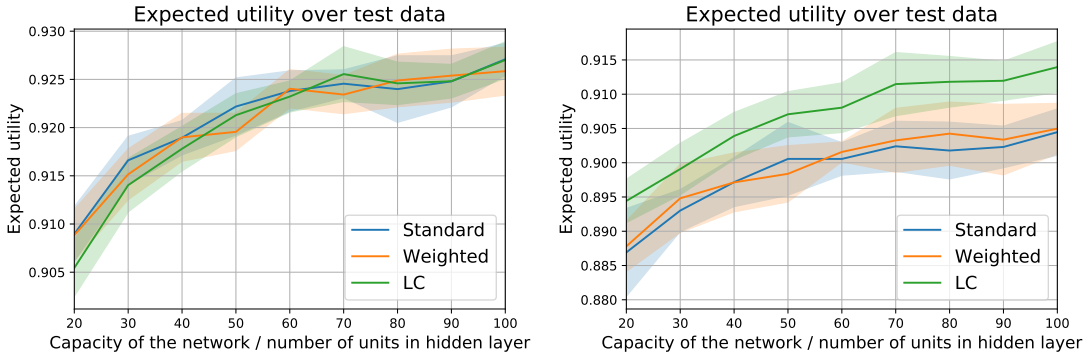
achieved at 100 % accuracy. In addition, the utility function encourages the network to focus on classes $\{3, 8\}$ by discouraging false negatives and accepting a greater number of false positives by assigning a higher relative utility. We display the utility matrix in Table 4.3.

Table 4.3: Utility matrix for the MNIST example. False positives for digits 3 and 8 are encouraged.

		Decision									
	Utility	0	1	2	3	4	5	6	7	8	9
True/Prediction	0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	1	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	2	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	3	0.3	0.3	0.3	1.0	0.3	0.3	0.3	0.3	0.3	0.3
	4	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
	5	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
	6	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
	7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
	8	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	1.0	0.3
	9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

In this experiment the best performing model is the one that consistently achieves the highest expected utility across the test set, where the test set is $10,000$ correctly (uncorrupted) labelled digits. Our set-up is to vary the capacity of all models by increasing the number of neurons in the single hidden layer from 20 to 100. After training all the models, samples are drawn from the predictive distribution for each of the test images and then these samples are used to perform an MC integration over the utility function. Therefore all models use the utility in the decision-making process. All experiments are repeated for 10 random seeds. The weights for the weighted cross entropy were set according to a grid search, where digits 3 and 8 were given a weight of 0.7 and the rest were assigned a weight of 1.0. Finally in the case of the loss-calibrated BNN (LCBNN), we bound the utility function such that $M = 0.0001$ in Equation (4.11).

The results are displayed in Figure 4.2. The LCBNN does better than the baselines



(a) No label noise

(b) 50% uniformly corrupted label noise

Figure 4.2: Each figure displays the expected utility over the test data as the size of the networks is increased. These results are calculated for 10 random seeds and their corresponding one standard deviation bounds are included. For Figure 4.2a, no label noise in training causes all the models to achieve similar utility over the uncorrupted test data.

over the corrupted data and achieves the same performance as the baselines for the data that is not corrupted. These results appear promising as the utility-dependent lower bound enables the LCBNN to train on the noisy labels and capture user preferences better than the baselines. Furthermore, as it achieves comparable performance when the data has no label noise, it shows that the loss-calibrated network is the better choice of model for both scenarios.

We can further highlight a weakness in the weighted cross-entropy by following what is commonly done in scenarios where there is a large imbalance in the data (e.g. Panchapagesan et al. [2016]). It is often the case in data imbalance problems that one would up-weight the less populous classes in the training set, so as to make up for the imbalance. If we up-weight classes $\{3, 8\}$ and re-run the experiment in Figure 4.2b we get the results in Figure 4.3. We see that by increasing the weights applied to classes $\{3, 8\}$ for the weighted cross-entropy, the network actually performs worse due to over-fitting, rather than encouraging the network to focus on these classes and achieve a higher utility.

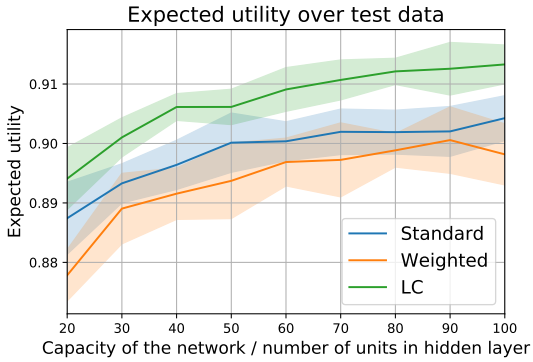


Figure 4.3: Mislabelled training data can lead to severe over-fitting for the weighted cross-entropy model, when important classes are up-weighted.

4.3.1.1 Utility sensitivity

Since the loss-calibrated ELBO requires the log utility, it is important to investigate how sensitive the results are to this bounding. Rather than varying the size of the neural network, we now vary the bound on the utility function from Table 4.3. We then plot the expected utility against the bound in Figure 4.4. The upper plot shows that as the lower bound on the utility becomes tighter to zero, the performance of LCBNN increases in terms of higher expected utility. In addition, the log scale in the figure shows how the LCBNN is insensitive to small perturbations in the bound near zero, where we see the expected utility plateau. Finally, the lower plot in Figure 4.4 shows how this lower bound transforms the utility function used in the loss-calibrated ELBO. The plot shows the corresponding log ratio between the maximum value of the transformed utility, $U_{\max}$, and the minimum value, $U_{\min}$, where

$$\text{log utility ratio} = \log \frac{M + U_{\max}}{M + U_{\min}}.$$

We see that increasing the lower bound reduces the ratio of the transformed utility, which corresponds to the decrease in performance of the LCBNN in the upper plot. We include the standard MC dropout BNN as a baseline.

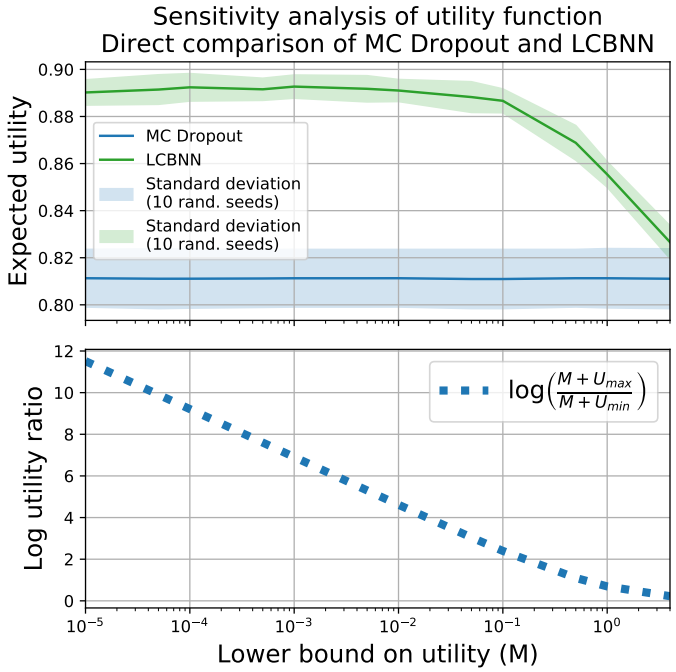


Figure 4.4: Sensitivity experiment to see how the LCBNN behaves as the bounding of the utility function varies above zero. **Upper plot:** a tighter bound results in higher expected utility and therefore better performance for the LCBNN. **Lower plot:** ratio of the maximum value of the bounded log utility over the minimum value. As this ratio decreases the effect of the utility-dependent term in the loss-calibrated ELBO reduces to the MC dropout baseline.

4.3.2 Per-pixel semantic segmentation in autonomous driving

In order to demonstrate that our loss-calibrated model scales to larger networks with real-world applications, we show its performance on a computer vision task of per-pixel semantic segmentation using the data set CamVid [Brostow et al., 2009]. For this task, we design a utility function that captures our preferences for identifying pedestrians, cyclists and cars, over other classes such as trees, buildings and the sky, which is displayed in Table 4.4. We then train our model and the baselines using the Bayesian SegNet-Basic architecture [Kendall et al., 2015]. Our backbone architecture is based on an implementation in Keras with a TensorFlow backend [Chollet et al., 2015, Konrad, 2016], which consists of 9 convolution layers and 5,467,500 parameters.

The data is split into 367 training images, 101 validation images and 233 test images, all with 360×480 resolution.

Table 4.4: Utility matrix for the autonomous driving example. False positives for cars, pedestrians and cyclists are especially encouraged as false positives for these classes are arguably safer than a preference for the alternative.

		Decision											
	Utility	Sk.	Bu.	Po.	Ro.	Pa.	Tr.	Si.	Fe.	Ca.	Pe.	Cy.	Un.
True/Prediction	Sky	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Building	0.0	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Pole	0.2	0.2	0.8	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
	Road	0.2	0.2	0.2	0.8	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
	Pavement	0.2	0.2	0.2	0.2	0.8	0.2	0.2	0.2	0.2	0.2	0.2	0.2
	Tree	0.2	0.2	0.2	0.2	0.2	0.8	0.2	0.2	0.2	0.2	0.2	0.2
	Sign	0.2	0.2	0.2	0.2	0.2	0.2	0.8	0.2	0.2	0.2	0.2	0.2
	Fence	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.8	0.2	0.2	0.2	0.2
	Car	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.8	0.4	0.4	0.4
	Pedestrian	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.8	0.8	0.4
	Cyclist	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.8	0.8	0.4
	Unlabelled	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.8

The same Bayesian decision theoretic procedure is followed as in the previous MNIST experiment. The baselines of the standard MC dropout model and the weighted cross-entropy are also used, where the weighting scheme relates to the class imbalance, such that the rarer classes are given higher weights and the common classes are given lower weights. We use the same weighted cross-entropy formulation as in Kendall et al. [2015]. Once again the objective is to achieve a high expected utility over the unseen test data. This objective may appear more intuitive in this real-world example than for the MNIST experiment, as one could imagine assigning preferences over certain *high-priority* classes.

4.3.2.1 Results

Table 4.5 displays the results for our experiment. The importance of relying on the framework of Bayesian decision theory is highlighted by displaying the results

Table 4.5: Results over the test data for the Bayesian SegNet-Basic architecture. Standard Pred. corresponds to the classifications before integrating over the utility and Decision corresponds to the results after the integration. We show that our utility on the test data greatly improves when assigning labels according to the decision. Furthermore, our LCBNN achieves the highest utility and highlights the benefits of our utility dependent lower bound.

Models	Standard Pred.		Decision	
	Acc.	Exp. Util.	Acc.	Exp. Util.
Standard	81.1 ± 0.6	0.619 ± 0.007	78.1 ± 0.9	0.676 ± 0.003
Weighted	82.1 ± 1.4	0.633 ± 0.010	79.6 ± 2.0	0.682 ± 0.009
LCBNN	82.5 ± 0.4	$\mathbf{0.652 \pm 0.002}$	81.8 ± 0.2	$\mathbf{0.685 \pm 0.003}$

in two headings. The 'standard prediction' gives the classification accuracy and the expected utility over the test data before any integration over the utility function. The 'decision' shows the results of integrating over the utility. These results show that through this integration, we increase our expected utility over the test data across all models and better capture our preferences. Therefore this result advocates for the general use of combining BNNs with Bayesian decision theory.

In addition to highlighting the importance of the Bayesian decision theoretic evaluation scheme, Table 4.5 shows our loss-calibrated model gives a performance boost over the current models. The benefits in incorporating the utility into the lower bound enables the model to achieve a higher utility than the other models that rely on using variational approximations that are learnt with no knowledge of the user's final application. Under normal circumstances separation of utility and prediction is key to Bayesian decision theory, but here we see how incorporation of the utility in the learning stage can provide a performance boost when we cannot be certain that the predictive distribution would otherwise be suited for the task.

The importance of choosing the right evaluation metric is also shown by looking at Table 4.5. Accuracy gives equal weight to all classes, and cannot distinguish important classes from others. Sky pixels, which dominate the dataset, skew this

Table 4.6: Intersection of union (IOU) highlights how classes such as pedestrians and cars are prioritised over lower priority classes such as road, pavement and trees. The IOU results for the our loss-calibrated model clearly demonstrate how our model prioritises the higher utility classes, where these results are calculated from the decisions.

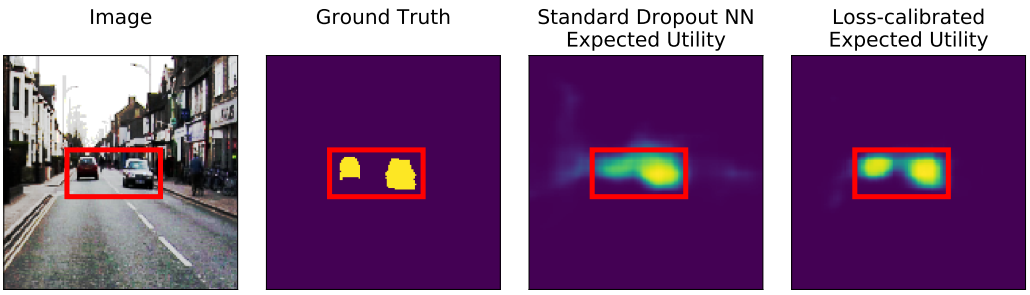
Models	Low Utility Classes IOU			High Utility Classes IOU		Mean IOU
	Road	Pave.	Tree	Car	Ped.	All
Standard	0.85	0.65	0.54	0.28	0.06	0.37
Weighted	0.86	0.66	0.55	0.31	0.09	0.40
LCBNN	0.86	0.65	0.54	**0.39**	**0.13**	0.42

metric unjustifiably. The expected utility metric, on the other hand, down-weights sky pixels in the evaluation and up-weights car and pedestrian pixel classifications. We can directly see exactly where the performance improvement lies from the results of Table 4.6. The table compares the intersection of union (IOU) for a subset of classes and highlights the discrepancy in performance across the classes. This is where the IOU for each class is given by

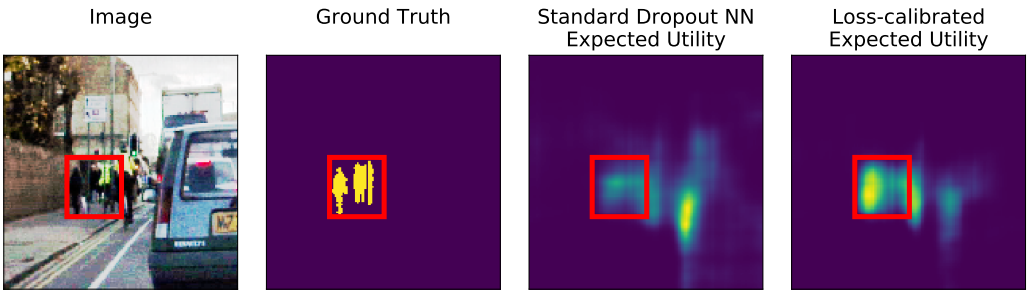
$$\text{IOU} = \frac{\text{Area of overlap between labelled pixels and classified pixels}}{\text{Area of union between labelled pixels and classified pixels}}.$$

Our LCBNN model achieves similar IOUs for the classes with a lower utility, however our model demonstrates a relative increased performance on the more challenging higher priority classes shown in bold. We stress that the aim of this table is not to give state-of-the-art classification accuracy results but rather to demonstrate the performance benefits of a calibrated model on the task of semantic segmentation, in comparison to standard approaches in the field.

Figures 4.5a and 4.5b display utility maps over segments of test images. They give an intuition into how the labels for each model are assigned. These utility maps display the gain each model expects to receive, before knowledge of the ground truth is available (high gain is denoted by bright yellow pixels). For example, Figure 4.5a



(a) Segmentation of cars



(b) Segmentation of pedestrians

Figure 4.5: Utility maps comparing a standard Monte Carlo dropout NN with the LCBNN using the SegNet-Basic architecture [Badrinarayanan et al., 2017]. For **each figure**, the **first column** is a test image taken from the CamVid data set Brostow et al. [2009]. The **second column** is the corresponding ground truth for the car (4.5a) and pedestrian (4.5b) classes. The **third column** is a utility map, given by the standard Bayesian SegNet Monte Carlo dropout implementation, which shows the expected utility in assigning each pixel to the shown class. Yellow corresponds to a high gain, whereas blue correponds to a low gain. The **fourth column** is a utility map given by the LCBNN. The LCBNN model produces a better behaved utility map by placing a higher utility over the areas that contain pedestrians and cars.

shows the per-pixel expected utility that the model believes it would get, if it were to decide upon the car class. Figure 4.5b shows the same but for the pedestrian class. We can see that the loss-calibrated model is able to capture sharper boundaries around the classes of interest. The two cars in Figure 4.5a are clearly identified by the LCBNN in the expected utility map of the car class. In comparison, the standard dropout model seems to merge the two cars together. Figure 4.5b shows a similar result for the pedestrian class, where in addition to sharper gains indicating the pedestrians for the LCBNN, we also see that the cyclist (under the right side of the red box) is also given a high gain for the pedestrian class. If we look at the utility function, this

(a) Standard dropout NN utility map (b) Loss-calibrated BNN utility map

Figure 4.6: In both figures, the utility map from the standard dropout NN (4.6a) and the LCBNN (4.6b) are superimposed on the test image. The expected utility is a superposition of the pedestrian and car classes, where high utility is indicated by the yellow pixels. The red box highlights a key difference between the two models, where the LCBNN assigns a much higher utility to the pedestrian.

behaviour is actually as expected and is encouraged, as the model has been told to treat cyclists and pedestrians equally.

As a further example, we sum the gains for the pedestrian and car classes and superimpose them over a test image. These images are shown in Figures 4.6a and 4.6b for the standard dropout model and for the LCBNN respectively. As before, bright yellow pixels indicate high expected utility. The red box marks the pedestrian, so that it is easier to directly compare performances across the models. The LCBNN prescribes high expected utility over the pedestrian by highlighting them in yellow. On the other hand, the standard dropout model does not assign enough gain for the pedestrian. Both models do well over the car class, although the LCBNN assigns higher gain over the car in the background.

4.3.2.2 Uncertainty behaviour

The utility function in Table 4.4 encourages the loss-calibrated model to put more of its *capacity* into the pedestrian class by defining the utility such that the network is penalised more for not explaining pedestrians well. Therefore, when we look at the

predictive entropy [Shannon, 1948],

$$\mathbb{H}[\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}] = -\sum_c \left(\frac{1}{S}\sum_s p(\mathbf{y}^* = c \mid \boldsymbol{\omega}^{(s)}, \mathbf{x}^*, \mathbf{X}, \mathbf{Y})\right) \log\left(\frac{1}{S}\sum_s p(\mathbf{y}^* = c \mid \boldsymbol{\omega}^{(s)}, \mathbf{x}^*, \mathbf{X}, \mathbf{Y})\right),$$

we would expect to see high values corresponding to pixels with high uncertainty. This could be edge pixels between classes or just objects that are genuinely confusing. Superimposing the predictive entropy for the same test image as before results in Figure 4.7. When comparing the standard dropout model to the LCBNN, the standard model has higher predictive entropy over the pedestrian and is therefore more uncertain. The LCBNN also displays uncertainty around the pedestrian, but predominantly at the boundary pixels. Therefore the LCBNN has better calibrated uncertainties for the task as it is more confident over the pedestrian but correctly displays high uncertainty over the edge pixels. We expect there to be high uncertainty over the edge pixels as it is where the inherent ambiguities lie.

(a) Predictive entropy: standard dropout NN. (b) Predictive entropy: LCBNN.

Figure 4.7: The predictive entropy superimposed over the test image. Yellow pixels indicate high predictive entropy and therefore higher uncertainty. The LCBNN is more certain over the pedestrian and less certain around the edges. The standard model is less certain as it has not been calibrated to explain pedestrians well.

4.4 Conclusion

This chapter introduced a new utility-dependent lower bound for training BNNs, where the LCBNN is able to attain superior performance when learning an approximate distribution over weights for asymmetric utility functions. This was shown via an MNIST label corruption experiment and via an autonomous driving application, where the average expected utility achieved by the LCBNN gave a higher average than the baselines. This new lower bound for BNNs provides a theoretically sound way of incorporating uncertainty and user preferences into real-world applications.

It is further demonstrated that BNNs are well-suited to decision-making applications and should be combined with Bayesian decision theory, where utility functions are known or easy to construct from user preferences. Designing utility functions to encode assumptions not only provides better results over alternative methods, but also results in constructing interpretable models. Therefore as well as being the first to apply loss-calibration to solve issues of non-tractability for large complex models, this work is significant as it demonstrates a novel composition of BNNs and Bayesian decision theory which will be useful in many future applications.

Finally, the chapter showed that if approximate inference is used to infer the BNN weights, then we must calibrate the model according to the utility-dependent lower bound by using a loss-calibrated Bayesian neural network. This results in both a superior performance and a better way for dealing with noise in data.

Chapter 5

Sampling in Bayesian neural networks: alternatives to Euclidean HMC

So far, performing SVI has been relatively cheap and provides ways of dealing with uncertainty in BNNs. However, variational approaches can lead to problems when quantifying uncertainty. Directly sampling from the posterior could be a way of achieving better uncertainty estimations because these are samples taken directly from the model with no assumptions over the structure of the posterior. However sampling NN weights is not easy and in this chapter we will explore ways of improving on the current gold standard of sampling in BNNs, Hamiltonian Monte Carlo. Therefore this chapter introduces a new integration scheme for Riemannian manifold HMC [Cobb et al., 2019b], which allows samplers to adapt to the local geometry of the model.

5.1 Markov chain Monte Carlo (MCMC) methods

In Section 3.2.1 I briefly introduced Markov chains as a technique for performing Monte Carlo sampling. One of the most commonly used and most popularised MCMC

algorithms originated from the statistical mechanics literature: the Metropolis–Hastings (MH) algorithm [Metropolis et al., 1953, Hastings, 1970] includes an acceptance step, whereby proposed samples are accepted according to an acceptance probability,

$$A(\boldsymbol{\omega}^{(s+1)}, \boldsymbol{\omega}^{(s)}) = \min\left(1, \frac{q(\boldsymbol{\omega}^{(s)}|\boldsymbol{\omega}^{(s+1)})p(\boldsymbol{\omega}^{(s+1)})}{q(\boldsymbol{\omega}^{(s+1)}|\boldsymbol{\omega}^{(s)})p(\boldsymbol{\omega}^{(s)})}\right),$$

that ensures the required MCMC properties are satisfied.[1] Despite its success, the MH algorithm can suffer in high-dimensional models due to low acceptance rates and an inability to efficiently explore the space ([Bishop, 2006, Chapter 11]).

5.2 Hamiltonian Monte Carlo

Rather than relying on a random walk to explore the parameter space of a model, HMC employs Hamiltonian dynamics to traverse the parameter space, $\boldsymbol{\omega} \in \mathbb{R}^D$. In order to introduce HMC, we begin with the integrand of interest $f(\boldsymbol{\omega}) = p(\mathbf{y}|\mathbf{x}, \boldsymbol{\omega})\, p(\boldsymbol{\omega})$ and augment the space by introducing the momentum variable $\mathbf{p} \in \mathbb{R}^D$, such that we now have a joint density $f(\boldsymbol{\omega})\, p(\mathbf{p})$. If we let $p(\mathbf{p}) = \mathcal{N}(\mathbf{p}|\mathbf{0}, \mathbf{M})$, where the covariance $\mathbf{M}$ denotes the mass matrix, then the negative log joint probability is given by

$$H(\boldsymbol{\omega}, \mathbf{p}) = -\log\left(f(\boldsymbol{\omega})\right) + \frac{1}{2}\log\left((2\pi)^D|\mathbf{M}|\right) + \frac{1}{2}\mathbf{p}^\top\mathbf{M}^{-1}\mathbf{p}. \tag{5.1}$$

It is then noted that this negative joint likelihood actually consists of a kinetic energy term $\frac{1}{2}\mathbf{p}^\top\mathbf{M}^{-1}\mathbf{p}$ and a potential energy term $-\mathcal{L}(\boldsymbol{\omega}) = -\log\left(f(\boldsymbol{\omega})\right)$. This insight points to using Hamiltonian dynamics to explore this system, where trajectories in this space move along constant energy levels.

[1] This acceptance step was originally related to potential energy of particles in a system, such that a sample configuration of particles was accepted according to this potential energy, rather than the previous method of simply weighting samples by the energy distribution.

Moving in this space requires derivative information:

$$\frac{\mathrm{d}\boldsymbol{\omega}}{\mathrm{d}t} = \frac{\partial H}{\partial \mathbf{p}} = \mathbf{M}^{-1}\mathbf{p}; \quad \frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} = -\frac{\partial H}{\partial \boldsymbol{\omega}} = \nabla_{\boldsymbol{\omega}}\mathcal{L}(\boldsymbol{\omega}). \tag{5.2}$$

The evolution of this system must preserve both volume and total energy, where any transformation that preserves area has the property of *symplecticity* [Hairer et al., 2006, Page 182]. As this Hamiltonian is separable, meaning $\frac{\partial H}{\partial \mathbf{p}}$ and $\frac{\partial H}{\partial \boldsymbol{\omega}}$ are functions of only one of the variables (momentum and parameters, respectively), to traverse the space we can use the Stormer–Verlet or leapfrog integrator (as used in Duane et al. [1987] and Neal [1995]):

$$\mathbf{p}_{t+\epsilon/2} = \mathbf{p}_t + \frac{\epsilon}{2}\frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t}(\boldsymbol{\omega}_t), \quad \boldsymbol{\omega}_{t+\epsilon} = \boldsymbol{\omega}_t + \epsilon\frac{\mathrm{d}\boldsymbol{\omega}}{\mathrm{d}t}(\mathbf{p}_{t+\epsilon/2}), \quad \mathbf{p}_{t+\epsilon} = \mathbf{p}_{t+\epsilon/2} + \frac{\epsilon}{2}\frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t}(\boldsymbol{\omega}_{t+\epsilon}), \tag{5.3}$$

where t corresponds to a leapfrog step iteration and ϵ is the step size. Therefore given this is a numerical solution that uses discrete steps, we must check whether the Hamiltonian energy is conserved by employing a Metropolis–Hastings step with an acceptance probability $\min\{0, H_{t=0} - H_{t=L}\}$ after L leapfrog steps. Finally, the overall sampling process follows the Gibbs sampling structure, where we draw $\mathbf{p} \sim p(\mathbf{p})$ and then draw a new set of parameters $\boldsymbol{\omega}^* \sim p(\boldsymbol{\omega}^*|\mathbf{p}) \propto f(\boldsymbol{\omega}^*)\, p(\mathbf{p} + \delta\mathbf{p})$, where we have followed similar nomenclature to Girolami and Calderhead [2011].

Therefore in repeating this Gibbs sampling scheme, including the implementation of the leapfrog algorithm, we can sample our parameter of interest $\boldsymbol{\omega}$.

5.2.1 Background

HMC has become a popular choice for performing inference in highly complex models, due to its higher acceptance rates and its use of gradient information in exploring the space. It was first introduced as *'Hybrid Monte Carlo'* by Duane et al. [1987] for

computer simulations in lattice field theory.[2] However the term 'Hamiltonian Monte Carlo' was coined from the extensive work by Neal [1995], who introduced HMC to the machine learning community with work on Bayesian neural networks. Despite the success of HMC, there are various hyperparameters that need to be tuned, such as the mass matrix, $\mathbf{M}$, the step size, ϵ, and the number of steps in a given trajectory, L (or leapfrog steps, see Section 5.2). The reliance on heuristics [Neal, 1995], which are heavily dependent on the Bayesian model or the data, often prevent HMC from being exploited to its full potential.

Progress in this area has been made with the introduction of a Hamiltonian-based Monte Carlo scheme that takes into account the geometry of the Bayesian model. This was framed in the work by Girolami and Calderhead [2011], where they introduced Riemannian manifold Hamiltonian Monte Carlo (RMHMC) as an *'overarching geometric framework'* for MCMC methods that included previous work by Roberts and Stramer [2002] and Duane et al. [1987].

5.2.2 Adaptations to Hamiltonian-based Monte Carlo samplers

The challenges associated with setting the hyperparameters of Hamiltonian samplers have seen various solutions offered in the literature. Hoffman and Gelman [2014] focussed on solving the problem of automatically setting the trajectory length L in HMC by introducing the No-U-Turn Sampler, whilst ensuring detailed balance is preserved. This was extended to Riemannian manifolds in Betancourt [2013b] with the aim of reducing unnecessary computation time. Wang et al. [2013] take a different approach to adapting hyperparameters by employing Bayesian optimisation [Snoek et al., 2012] for adaptive tuning within Hamiltonian-based Monte Carlo samplers.

[2]We note that earlier work by Alder and Wainwright [1959] utilised Hamiltonian dynamics to simulate molecular dynamics.

Rather than focussing on techniques for hyperparameter tuning, others have directly worked on the formulation of the Hamiltonian itself. Shahbaba et al. [2014] show that it is possible to split the Hamiltonian in a way that can reduce the computational cost. They use a maximum a posteriori approximation that allows the Hamiltonian to be partially analytically integrated, which can reduce the computational burden when exploring the state space. Previous work by Lan et al. [2012] introduced some of the benefits of explicit integration techniques to RMHMC. However, unlike the new integrator introduced later in this chapter, they derive an integrator that is no longer symplectic and adjust their acceptance step in order to preserve detailed balance. In addition, they introduce two additional matrix inversions that limit efficiency in high-dimensional spaces.

5.3 Moving to a Riemannian manifold

One of the big challenges when implementing HMC is how to set the mass matrix $\mathbf{M}$. The relationship between $\mathbf{M}$ and the dynamics of the Hamiltonian system can be seen in Equation (5.2) (and more directly in Equation (5.3)). The mass matrix acts to scale the step size in the leapfrog integration scheme, whereby the effective step size in each dimension depends on the elements of $\mathbf{M}$. As a result, we look to set $\mathbf{M}$ according to the geometry of the parameter space. In Neal [1995, Appendix 4], one practical solution is offered, which relates the step size for each parameter to the second-order derivative of the log posterior. Interestingly, this is linked to the topic of discussion in this section, where we start to explore ideas of higher-order derivatives.

If our interest lies in measuring distances between distributions, then naïvely treating the space as if it were Euclidean (while computationally convenient) may prevent rapid sampling of the posterior. A common metric for measuring the difference be-

tween distributions is the Kullback–Leibler (KL) divergence:

$$\mathrm{D}_{\mathrm{KL}}\big(p(\mathbf{x},\boldsymbol{\omega})||p(\mathbf{x},\boldsymbol{\omega}')\big) = \iint p(\mathbf{x},\boldsymbol{\omega})\left(\log p(\mathbf{x},\boldsymbol{\omega}) - \log p(\mathbf{x},\boldsymbol{\omega}')\right)\mathrm{d}\mathbf{x}\mathrm{d}\boldsymbol{\omega}. \tag{5.4}$$

If we expand the $\log p(\mathbf{x},\boldsymbol{\omega}')$ term within the integrand using a Taylor expansion around $\boldsymbol{\omega}$ such that $\delta\boldsymbol{\omega} = \boldsymbol{\omega}' - \boldsymbol{\omega}$, we obtain

$$\begin{aligned}\mathrm{D}_{\mathrm{KL}}\big(p(\mathbf{x},\boldsymbol{\omega})||p(\mathbf{x},\boldsymbol{\omega}+\delta\boldsymbol{\omega})\big) \approx &\iint p(\mathbf{x},\boldsymbol{\omega})\log p(\mathbf{x},\boldsymbol{\omega})\mathrm{d}\mathbf{x}\mathrm{d}\boldsymbol{\omega} \\ &- \iint p(\mathbf{x},\boldsymbol{\omega})\Big(\log p(\mathbf{x},\boldsymbol{\omega}) + \left(\frac{\nabla_{\boldsymbol{\omega}} p(\mathbf{x},\boldsymbol{\omega})}{p(\mathbf{x},\boldsymbol{\omega})}\right)^{\top}\delta\boldsymbol{\omega} \\ &\qquad + \frac{1}{2}\delta\boldsymbol{\omega}^{\top}\nabla^2_{\boldsymbol{\omega}}\log p(\mathbf{x},\boldsymbol{\omega})\delta\boldsymbol{\omega} + \ldots\Big)\mathrm{d}\mathbf{x}\mathrm{d}\boldsymbol{\omega}.\end{aligned} \tag{5.5}$$

We note that the first two terms in Equation (5.5) cancel and the first order gradient term reduces to zero as follows:

$$\begin{aligned}-\iint p(\mathbf{x},\boldsymbol{\omega})\left(\frac{\nabla_{\boldsymbol{\omega}} p(\mathbf{x},\boldsymbol{\omega})}{p(\mathbf{x},\boldsymbol{\omega})}\right)^{\top}\delta\boldsymbol{\omega}\mathrm{d}\mathbf{x}\mathrm{d}\boldsymbol{\omega} &= -\iint \nabla_{\boldsymbol{\omega}} p(\mathbf{x},\boldsymbol{\omega})^{\top}\delta\boldsymbol{\omega}\mathrm{d}\mathbf{x}\mathrm{d}\boldsymbol{\omega} \\ &= -\int \nabla_{\boldsymbol{\omega}} p(\boldsymbol{\omega})^{\top}\delta\boldsymbol{\omega}\mathrm{d}\boldsymbol{\omega} \\ &= -\nabla_{\boldsymbol{\omega}}\mathbf{1}^{\top}\delta\boldsymbol{\omega} \\ &= 0.\end{aligned} \tag{5.6}$$

Thus,

$$\begin{aligned}\mathrm{D}_{\mathrm{KL}}\big(p(\mathbf{x},\boldsymbol{\omega})||p(\mathbf{x},\boldsymbol{\omega}+\delta\boldsymbol{\omega})\big) &\approx -\iint p(\mathbf{x},\boldsymbol{\omega})\left(\frac{1}{2}\delta\boldsymbol{\omega}^{\top}\nabla^2_{\boldsymbol{\omega}}\log p(\mathbf{x},\boldsymbol{\omega})\delta\boldsymbol{\omega}\right)\mathrm{d}\mathbf{x}\mathrm{d}\boldsymbol{\omega} \\ &= \iint p(\mathbf{x},\boldsymbol{\omega})\left(\frac{1}{2}\delta\boldsymbol{\omega}^{\top}\mathbf{G}(\mathbf{x},\boldsymbol{\omega})\delta\boldsymbol{\omega}\right)\mathrm{d}\mathbf{x}\mathrm{d}\boldsymbol{\omega}\end{aligned} \tag{5.7}$$

where $\mathbf{G}(\mathbf{x},\boldsymbol{\omega})$ is defined as $-\nabla^2_{\boldsymbol{\omega}}\log p(\mathbf{x},\boldsymbol{\omega})$, which tells us about the curvature for the joint probability space over $\mathbf{x}$ and $\boldsymbol{\omega}$. As per Girolami and Calderhead [2011],

we use a Bayesian perspective and concern ourselves with the joint likelihood of the data and the parameters, therefore $\mathbf{G}(\boldsymbol{\omega})$ is the Fisher information plus the negative Hessian of the log-prior. Furthermore there is a plethora of literature relating to the appropriate metric for measuring distances on a Riemannian manifold, where we refer to either Girolami and Calderhead [2011] or Amari and Nagaoka [2000].

5.3.1 Riemannian manifold Hamiltonian Monte Carlo

Having introduced the need for a metric situated on a Riemannian manifold, we can now go back to the original Hamiltonian of Equation (5.1) but reformulate it to lie on a Riemannian manifold by replacing $\mathbf{M}$ with $\mathbf{G}(\boldsymbol{\omega})$. However, because the mass matrix is now parameterised by $\boldsymbol{\omega}$, the Hamiltonian equations are no longer separable as they contain a coupling between $\boldsymbol{\omega}$ and $\mathbf{p}$:

$$\frac{\mathrm{d}\boldsymbol{\omega}}{\mathrm{d}t} = \frac{\partial H}{\partial \mathbf{p}} = \mathbf{G}(\boldsymbol{\omega})^{-1}\mathbf{p} \quad \text{and} \quad \frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} = -\frac{\partial H}{\partial \boldsymbol{\omega}} = \nabla_{\boldsymbol{\omega}}\mathcal{L}(\boldsymbol{\omega}) - \nabla_{\boldsymbol{\omega}}\mathcal{N}(\mathbf{0}, \mathbf{G}(\boldsymbol{\omega}))\,. \tag{5.8}$$

Since the Hamiltonian is non-separable, the leapfrog integrator defined in Equation 5.3 is no longer applicable due to the volume conservation requirements. Furthermore, the new dependence of the mass matrix on the parameters means that detailed balance would no longer be satisfied.[3]

5.3.2 Hessians with negative eigenvalues

Often, we can calculate the Fisher information matrix with knowledge that the metric will be positive semi-definite. However, once models become more complex, the positive semi-definiteness of the Fisher information metric is no longer guaranteed.

In order to ensure $\mathbf{G}(\boldsymbol{\omega})$ is positive semi-definite, we follow the same *SoftAbs*

[3]At this point, it feels important to highlight that in general, implementations of HMC are often not time-reversible, since that would require an equal chance of ϵ being positive or negative. However, as Hoffman and Gelman [2014] mention, this can be omitted in practice.

metric that Betancourt [2013a] introduced for RMHMC. This allows us to filter the eigenspectrum by passing all the eigenvalues, λ_d, through the function

$$\tilde{\lambda}_d = \lambda_d \coth(\alpha \lambda_d). \tag{5.9}$$

This approximates the absolute values of the eigenvalues and therefore ensures that the new metric $\tilde{\mathbf{G}}(\boldsymbol{\omega}) = \mathbf{Q}\tilde{\boldsymbol{\lambda}}\mathbf{Q}^\top$ is positive semi-definite, by introducing a parameter α which controls the smoothness. As $\alpha \to \infty$ the SoftAbs function becomes the absolute value.[4]

5.4 Explicit RMHMC

5.4.1 Implicit versus explicit integration

The current solution for solving the non-separable Hamiltonian equations in Section 5.3 is to rely on the *implicit* generalised leapfrog algorithm [Leimkuhler and Reich, 2004]. The term *implicit* refers to the computationally expensive first-order implicit Euler integrators, which are fixed-point iterations that are run until convergence. In order to understand the computational cost associated with this generalised leapfrog algorithm, we summarise a single leapfrog step in Algorithm 2.

A single step in the implicit generalised leapfrog contains two 'while' loops, which both require expensive update steps for each iteration. In Betancourt [2013a], the author uses 'while' loops as in Algorithm 2 and sets a threshold δ for breaking out. As an alternative to a conditional break, Girolami and Calderhead [2011] use a 'for' loop and fix the number of iterations to be five or six. Therefore, one step, (at a minimum) requires 11 Hessian calculations along with the additional derivative calculations.

Rather than relying on implicit integration, an *explicit* integration scheme relies

[4] In this work we set $\alpha = 10^6$ as in Betancourt [2013a].

on an explicit evaluation at an already known value [Hairer et al., 2006, Chapter 1, Page 3]. This leads to a deterministic number of operations per leapfrog step, which we will show to be advantageous. We now introduce an explicit integration scheme for RMHMC.

5.4.2 Introducing the new explicit integrator

The challenge in speeding up the symplectic integration for non-separable Hamiltonians has not been applied within the context of Bayesian statistics. However if we delve into the physical sciences, there have been many examples where non-separable Hamiltonians appear [Tao, 2016b, Luo et al., 2017, Zhang et al., 2018]. Therefore it makes sense to draw from the progress made in this area, given the strong relationship between the physical interpretations of HMC within probabilistic modelling applications.

A key advance that moved away from dealing with specific subclasses of non-separable Hamiltonians and moved towards arbitrary non-separable Hamiltonians came in Pihajoki [2015], where the phase space was extended to include an additional momentum term $\tilde{\mathbf{p}}$ and parameter term $\tilde{\boldsymbol{\omega}}$. This extension of the phase space was built on by Tao [2016a] who added an additional binding term that resulted in improved long-term performance. This new augmented Hamiltonian can now be defined as:

$$\tilde{H}(\boldsymbol{\omega}, \mathbf{p}, \tilde{\boldsymbol{\omega}}, \tilde{\mathbf{p}}) = H_1(\boldsymbol{\omega}, \tilde{\mathbf{p}}) + H_2(\tilde{\boldsymbol{\omega}}, \mathbf{p}) + \Omega h(\boldsymbol{\omega}, \mathbf{p}, \tilde{\boldsymbol{\omega}}, \tilde{\mathbf{p}}) \ , \tag{5.10}$$

where the binding term $h(\boldsymbol{\omega}, \mathbf{p}, \tilde{\boldsymbol{\omega}}, \tilde{\mathbf{p}}) = \|\boldsymbol{\omega} - \tilde{\boldsymbol{\omega}}\|_2^2/2 + \|\mathbf{p} - \tilde{\mathbf{p}}\|_2^2/2$ and Ω controls the binding. The two Hamiltonian systems H_1 and H_2 each follow Equation (5.1), where $H(\boldsymbol{\omega}, \tilde{\mathbf{p}})$ and $H(\tilde{\boldsymbol{\omega}}, \mathbf{p})$ have their parameters and momenta mixed. Interestingly, if we set $\Omega = 0$, we retrieve the augmented Hamiltonian from Pihajoki [2015].

The Hamiltonian in Equation (5.10) gives the system:

$$\begin{aligned}
\dot{\boldsymbol{\omega}} &= \quad \partial_{\mathbf{p}} H(\tilde{\boldsymbol{\omega}}, \mathbf{p}) + \Omega(\mathbf{p} - \tilde{\mathbf{p}}) \\
\dot{\mathbf{p}} &= -\partial_{\boldsymbol{\omega}} H(\boldsymbol{\omega}, \tilde{\mathbf{p}}) - \Omega(\boldsymbol{\omega} - \tilde{\boldsymbol{\omega}}) \\
\dot{\tilde{\boldsymbol{\omega}}} &= \quad \partial_{\tilde{\mathbf{p}}} H(\boldsymbol{\omega}, \tilde{\mathbf{p}}) + \Omega(\tilde{\mathbf{p}} - \mathbf{p}) \\
\dot{\tilde{\mathbf{p}}} &= -\partial_{\tilde{\boldsymbol{\omega}}} H(\tilde{\boldsymbol{\omega}}, \mathbf{p}) - \Omega(\tilde{\boldsymbol{\omega}} - \boldsymbol{\omega}).
\end{aligned} \tag{5.11}$$

We now introduce the integrators from Tao [2016a], which when combined make up the symplectic flow for the augmented non-separable Hamiltonian:

$$\phi^{\epsilon}_{H_1} : \begin{bmatrix} \boldsymbol{\omega} \\ \mathbf{p} \\ \tilde{\boldsymbol{\omega}} \\ \tilde{\mathbf{p}} \end{bmatrix} \mapsto \begin{bmatrix} \boldsymbol{\omega} \\ \mathbf{p} - \epsilon \partial_{\boldsymbol{\omega}} H(\boldsymbol{\omega}, \tilde{\mathbf{p}}) \\ \tilde{\boldsymbol{\omega}} + \epsilon \partial_{\tilde{\mathbf{p}}} H(\boldsymbol{\omega}, \tilde{\mathbf{p}}) \\ \tilde{\mathbf{p}} \end{bmatrix}, \quad \phi^{\epsilon}_{H_2} : \begin{bmatrix} \boldsymbol{\omega} \\ \mathbf{p} \\ \tilde{\boldsymbol{\omega}} \\ \tilde{\mathbf{p}} \end{bmatrix} \mapsto \begin{bmatrix} \boldsymbol{\omega} + \epsilon \partial_{\mathbf{p}} H(\tilde{\boldsymbol{\omega}}, \mathbf{p}) \\ \mathbf{p} \\ \tilde{\boldsymbol{\omega}} \\ \tilde{\mathbf{p}} - \epsilon \partial_{\tilde{\boldsymbol{\omega}}} H(\tilde{\boldsymbol{\omega}}, \mathbf{p}) \end{bmatrix},$$

$$\phi^{\epsilon}_{\Omega h} : \begin{bmatrix} \boldsymbol{\omega} \\ \mathbf{p} \\ \tilde{\boldsymbol{\omega}} \\ \tilde{\mathbf{p}} \end{bmatrix} \mapsto \frac{1}{2} \begin{bmatrix} \begin{pmatrix} \boldsymbol{\omega} + \tilde{\boldsymbol{\omega}} \\ \mathbf{p} + \tilde{\mathbf{p}} \end{pmatrix} + \mathbf{R}(\epsilon) \begin{pmatrix} \boldsymbol{\omega} - \tilde{\boldsymbol{\omega}} \\ \mathbf{p} - \tilde{\mathbf{p}} \end{pmatrix} \\ \begin{pmatrix} \boldsymbol{\omega} + \tilde{\boldsymbol{\omega}} \\ \mathbf{p} + \tilde{\mathbf{p}} \end{pmatrix} - \mathbf{R}(\epsilon) \begin{pmatrix} \boldsymbol{\omega} - \tilde{\boldsymbol{\omega}} \\ \mathbf{p} - \tilde{\mathbf{p}} \end{pmatrix} \end{bmatrix}, \tag{5.12}$$

$$\text{where } \mathbf{R}(\epsilon) = \begin{bmatrix} \cos(2\Omega\epsilon)\mathbf{I} & \sin(2\Omega\epsilon)\mathbf{I} \\ -\sin(2\Omega\epsilon)\mathbf{I} & \cos(2\Omega\epsilon)\mathbf{I} \end{bmatrix}.$$

Finally, the numerical symplectic second-order integrator is given by the function composition ($f(g(\cdot)) = f \circ g$)

$$\phi^{\epsilon}_{\tilde{H}} = \phi^{\epsilon/2}_{H_1} \circ \phi^{\epsilon/2}_{H_2} \circ \phi^{\epsilon}_{\Omega h} \circ \phi^{\epsilon/2}_{H_2} \circ \phi^{\epsilon/2}_{H_1}, \tag{5.13}$$

where combining these symmetric mappings is known as Strang splitting [Strang, 1968] and higher-order integrators can be built in a similar manner [Yoshida, 1990].

Here, we can show that this second-order integrator is indeed symplectic.

5.4.3 Checking the symplectomorphism of the augmented Hamiltonian binding term

A symplectic matrix is a matrix $\mathbf{H}$ that satisfies the condition [Channell and Scovel, 1990]:

$$\mathbf{H}^{\top}\mathbf{J}\mathbf{H} = \mathbf{J} \tag{5.14}$$

where,

$$\mathbf{J} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ -\mathbf{I} & \mathbf{0} \end{bmatrix} \tag{5.15}$$

is a non-singular, skew-symmetric matrix and $\mathbf{H} \in \mathbb{R}^{2D\times 2D}$. This definition [Hairer et al., 2006, Chapter 6, Page 183] is derived from the area preserving properties of the linear mapping $\mathbf{H}$, where the structure of $\mathbf{J}$ originates from the form of the determinant in the transformed space. In nonlinear systems, $\mathbf{H}$ can be the Jacobian of a differentiable function [Poincaré, 1899] as shown in Hairer et al. [2006, Chapter 6, Page 184].

To show how the binding term in Equation (5.10) meets this condition, we can transform the Hamiltonian $\tilde{H}(\boldsymbol{\omega}, \mathbf{p}, \tilde{\boldsymbol{\omega}}, \tilde{\mathbf{p}})$ to $\hat{H}(\hat{\boldsymbol{Q}}, \hat{\boldsymbol{P}}, \boldsymbol{\alpha}, \boldsymbol{\beta})$, where $\hat{\boldsymbol{Q}} = \boldsymbol{\omega} + \tilde{\boldsymbol{\omega}}$, $\hat{\boldsymbol{P}} = \mathbf{p} + \tilde{\mathbf{p}}$, $\boldsymbol{\alpha} = \boldsymbol{\omega} - \tilde{\boldsymbol{\omega}}$ and $\boldsymbol{\beta} = \mathbf{p} - \tilde{\mathbf{p}}$, giving

$$\hat{H}(\hat{\boldsymbol{Q}}, \hat{\boldsymbol{P}}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = H_1(\hat{\boldsymbol{Q}} + \frac{\boldsymbol{\alpha}}{2}, \hat{\boldsymbol{P}} - \frac{\boldsymbol{\alpha}}{2}) + H_2(\hat{\boldsymbol{Q}} - \frac{\boldsymbol{\alpha}}{2}, \hat{\boldsymbol{P}} + \frac{\boldsymbol{\alpha}}{2}) + \frac{\Omega}{2}(\|\boldsymbol{\alpha}\|^2 + \|\boldsymbol{\beta}\|^2) + \mathcal{R} \tag{5.16}$$

as shown in Tao [2016a], where $\mathcal{R}$ is the perturbative higher order remainder.

Therefore, if we employ this transformation of variables to the binding term $\phi^{\epsilon}_{\Omega h}$,

the linear system in Equation (5.13) corresponding to $\phi^{\epsilon}_{\Omega h}$ becomes:

$$\begin{bmatrix} \hat{\boldsymbol{Q}} \\ \hat{\boldsymbol{P}} \\ \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos(2\Omega\epsilon) & \sin(2\Omega\epsilon) \\ 0 & 0 & -\sin(2\Omega\epsilon) & \cos(2\Omega\epsilon) \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{Q}} \\ \hat{\boldsymbol{P}} \\ \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{bmatrix} \tag{5.17}$$

after the manipulation of the variables from the linear system in Equation (5.12).

We will now write this as

$$\mathbf{y}' = \mathbf{H}\mathbf{y}, \tag{5.18}$$

where $\mathbf{H}$ is the transformation matrix in Equation (5.17). In order to check this transformation satisfies the condition in Equation (5.15), we perform the matrix operations $\mathbf{H}^{\top}\mathbf{J}\mathbf{H}$. After applying these matrix multiplications and using the trigonometric identity $\sin^2(x) + \cos^2(x) = 1$ we see that $\mathbf{H}^{\top}\mathbf{J}\mathbf{H} = \mathbf{J}$ thus confirming $\phi^{\epsilon}_{\Omega h}$ is a symplectomorphism.

There are two major advantages when comparing the integrator in Equation (5.13) to previous symplectic integrators for non-separable Hamiltonians. First of all, when comparing to the implicit integrator of Section 5.4.1, we no longer need to rely on two fixed-point iteration methods that can often diverge. Furthermore we are guaranteed the same number of derivative calculations of eight per equivalent leapfrog step. This compares to the minimum number of 11 for the implicit generalised leapfrog (although it can often be higher). Secondly, when comparing to the previous explicit integrator of Pihajoki [2015], this integration scheme has better long-term error performance. More specifically, Tao [2016a] showed that until the number of steps, $N = \mathcal{O}(\min(\epsilon^{-2}\Omega^{-1}, \Omega^{1/2}))$, the numerical error is $\mathcal{O}(N\epsilon^2\Omega)$, although empirical results show that larger values of N give favourable results in terms of the sampler's performance. A direct comparison between these two leapfrog schemes is shown in

Algorithms 2 and 3.

Algorithm 2 Implicit Leapfrog Step

Inputs: $\mathbf{p}_0$, $\boldsymbol{\omega}_0, \epsilon,$
$\mathbf{p} = \mathbf{p}_0$
while $\Delta p > \delta$ **do**
  $\mathbf{p}' = \mathbf{p}_0 + \frac{\epsilon}{2}\frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t}(\mathbf{p}, \boldsymbol{\omega}_0)$
  $\Delta p = \max_i\{|p_i - p'_i|\}$
  $\mathbf{p} = \mathbf{p}'$
end while
$\boldsymbol{\omega} = \boldsymbol{\omega}_0$
while $\Delta\omega > \delta$ **do**
  $\boldsymbol{\omega}' = \boldsymbol{\omega}_0 + \frac{\epsilon}{2}\frac{\mathrm{d}\boldsymbol{\omega}}{\mathrm{d}t}(\mathbf{p}, \boldsymbol{\omega}_0) + \frac{\epsilon}{2}\frac{\mathrm{d}\boldsymbol{\omega}}{\mathrm{d}t}(\boldsymbol{p}, \boldsymbol{\omega})$
  $\Delta\omega = \max_i\{|\omega_i - \omega'_i|\}$
  $\boldsymbol{\omega} = \boldsymbol{\omega}'$
end while
$\mathbf{p} = \mathbf{p} + \frac{\epsilon}{2}\frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t}(\mathbf{p}, \boldsymbol{\omega})$

Algorithm 3 Explicit Leapfrog Step

Inputs: $\mathbf{p}, \boldsymbol{\omega}, \tilde{\mathbf{p}}, \tilde{\boldsymbol{\omega}}, \epsilon, \Omega$
$\mathbf{p} = \mathbf{p} - \frac{\epsilon}{2}\partial_{\boldsymbol{\omega}} H(\boldsymbol{\omega}, \tilde{\mathbf{p}})$
$\tilde{\boldsymbol{\omega}} = \tilde{\boldsymbol{\omega}} + \frac{\epsilon}{2}\partial_{\tilde{\mathbf{p}}} H(\boldsymbol{\omega}, \tilde{\mathbf{p}})$
$\tilde{\mathbf{p}} = \tilde{\mathbf{p}} - \frac{\epsilon}{2}\partial_{\tilde{\boldsymbol{\omega}}} H(\tilde{\boldsymbol{\omega}}, \mathbf{p})$
$\boldsymbol{\omega} = \boldsymbol{\omega} + \frac{\epsilon}{2}\partial_{\mathbf{p}} H(\tilde{\boldsymbol{\omega}}, \mathbf{p})$
$c = \cos(2\Omega\epsilon), \quad s = \sin(2\Omega\epsilon)$
$\boldsymbol{\omega} = (\boldsymbol{\omega} + \tilde{\boldsymbol{\omega}} + c(\boldsymbol{\omega} - \tilde{\boldsymbol{\omega}}) + s(\mathbf{p} - \tilde{\mathbf{p}}))/2$
$\mathbf{p} = (\mathbf{p} + \tilde{\mathbf{p}} - s(\boldsymbol{\omega} - \tilde{\boldsymbol{\omega}}) + c(\mathbf{p} - \tilde{\mathbf{p}}))/2$
$\tilde{\boldsymbol{\omega}} = (\boldsymbol{\omega} + \tilde{\boldsymbol{\omega}} - c(\boldsymbol{\omega} - \tilde{\boldsymbol{\omega}}) - s(\mathbf{p} - \tilde{\mathbf{p}}))/2$
$\tilde{\mathbf{p}} = (\mathbf{p} + \tilde{\mathbf{p}} + s(\boldsymbol{\omega} - \tilde{\boldsymbol{\omega}}) - c(\mathbf{p} - \tilde{\mathbf{p}}))/2$
$\tilde{\mathbf{p}} = \tilde{\mathbf{p}} - \frac{\epsilon}{2}\partial_{\tilde{\boldsymbol{\omega}}} H(\tilde{\boldsymbol{\omega}}, \mathbf{p})$
$\boldsymbol{\omega} = \boldsymbol{\omega} + \frac{\epsilon}{2}\partial_{\mathbf{p}} H(\tilde{\boldsymbol{\omega}}, \mathbf{p})$
$\mathbf{p} = \mathbf{p} - \frac{\epsilon}{2}\partial_{\boldsymbol{\omega}} H(\boldsymbol{\omega}, \tilde{\mathbf{p}})$
$\tilde{\boldsymbol{\omega}} = \tilde{\boldsymbol{\omega}} + \frac{\epsilon}{2}\partial_{\tilde{\mathbf{p}}} H(\boldsymbol{\omega}, \tilde{\mathbf{p}})$

5.4.4 Removing the bias and deriving explicit RMHMC

One of the key properties of Hamiltonian-based samplers is that the marginal density of $\boldsymbol{\omega}$ must correspond to the target density such that

$$p(\boldsymbol{\omega}) \propto \int \exp\big(-H(\boldsymbol{\omega}, \mathbf{p})\big)\mathrm{d}\mathbf{p} = \int f(\boldsymbol{\omega})p(\mathbf{p}|\boldsymbol{\omega})\mathrm{d}\mathbf{p} = \exp\{\mathcal{L}(\boldsymbol{\omega})\}, \tag{5.19}$$

where the Hamiltonian, $\exp\big(-H(\boldsymbol{\omega}, \mathbf{p})\big) = f(\boldsymbol{\omega})p(\mathbf{p}|\boldsymbol{\omega}) = p(\mathbf{y}|\mathbf{x}, \boldsymbol{\omega})\, p(\boldsymbol{\omega})p(\mathbf{p}|\boldsymbol{\omega})$, was previously derived in Section 5.2. Unfortunately, if we replace the Hamiltonian in Equation (5.19) with the new augmented Hamiltonian $\tilde{H}(\boldsymbol{\omega}, \mathbf{p}, \tilde{\boldsymbol{\omega}}, \tilde{\mathbf{p}})$, the contribution from the binding term means that the marginal samples of $\boldsymbol{\omega}$ will no longer follow the correct target distribution.

The solution for removing the resulting induced bias comes from only relying on the augmented Hamiltonian for approximating the Hamiltonian dynamics and not using the augmented Hamiltonian in the final Metropolis–Hastings step. Instead,

we can use the true Hamiltonian in the Metropolis–Hastings step, which will have a high acceptance probability if the approximated dynamics closely correspond to the true Hamiltonian dynamics.[5] This solution ensures that we only accept samples that conform to the no-bias regime and still get the computational benefit of the explicit integration scheme. Algorithm 4 shows how explicit RMHMC is implemented using the explicit leapfrog integrations scheme.

Algorithm 4 Explicit RMHMC

Initialise: $\boldsymbol{\omega}, \epsilon, \Omega, L, N$
for n in $1, 2, \dots N$ **do**
  Sample: $\mathbf{p} \sim \mathcal{N}(\mathbf{0}, \mathbf{G}(\boldsymbol{\omega}))$
  # Calculate initial Hamiltonian based on the true Hamiltonian.
  $H_{t=0} = H(\boldsymbol{\omega}, \mathbf{p})$
  # Leapfrog steps:
  $\mathbf{p}^1 = \mathbf{p}$, $\boldsymbol{\omega}^1 = \boldsymbol{\omega}$
  for l in $1, 2, \dots L$ **do**
    Initialise: $\tilde{\mathbf{p}} = \mathbf{p}^l$, $\tilde{\boldsymbol{\omega}} = \boldsymbol{\omega}^l$
    # Use the explicit leapfrog step from Algorithm 3, which uses the augmented Hamiltonian, $\tilde{H}(\boldsymbol{\omega}, \mathbf{p}, \tilde{\boldsymbol{\omega}}, \tilde{\mathbf{p}})$.
    $\boldsymbol{\omega}^l, \mathbf{p}^l \leftarrow \text{leapfrog}(\mathbf{p}^l, \boldsymbol{\omega}^l, \tilde{\mathbf{p}}, \tilde{\boldsymbol{\omega}}, \epsilon, \Omega)$
  end for
  # Calculate final Hamiltonian based on the true Hamiltonian.
  $H_{t=L} = H(\boldsymbol{\omega}^L, \mathbf{p}^L)$
  # Metropolis–Hastings correction
  $u \sim \mathcal{U}(0, 1)$
  if $\log u > \min(0, H_{t=0} - H_{t=L})$ **then**
    $\boldsymbol{\omega}, \mathbf{p} \leftarrow \boldsymbol{\omega}^L, \mathbf{p}^L$
  end if
end for

5.5 Bayesian logistic regression

We will explore the benefits of RMHMC, as well as show that explicit RMHMC is significantly faster than implicit RMHMC. We start with the convex problem of Bayesian logistic regression as the metric tensor is positive semi-definitive and hence

[5] This is similar to the original purpose of the Metropolis–Hastings step to account for the integration error induced by the discretisation of the Hamiltonian dynamics.

well behaved. We define our Hamiltonian as in Equation (5.1), where the log joint probability is given by

$$\mathcal{L}(\boldsymbol{\omega}) = \sum_{i}^{N} \{y_i \log(\boldsymbol{\omega}^\top \mathbf{x}_i) + (1 - y_i) \log(1 - \boldsymbol{\omega}^\top \mathbf{x}_i)\} + \frac{\tau}{2} \boldsymbol{\omega}^\top \boldsymbol{\omega}, \tag{5.20}$$

where the quadratic term in the model parameters corresponds to the prior. We must be careful when defining the precision term τ, as the influence of the prior on the sampling can be significant. Also, we allow the data pairs $\{y_i, \mathbf{x}_i\}_{i=1}^N$ to be augmented such that $x_0 = 1$, which allows ω_0 to correspond to the bias term. We then generate the data by separately sampling $\hat{\boldsymbol{\omega}} \sim \mathcal{N}(\mathbf{m}_{\boldsymbol{\omega}}, \boldsymbol{\Sigma}_{\boldsymbol{\omega}})$ and building a data set through using the logistic $y_i = 1/\left(1 + \exp(-\hat{\boldsymbol{\omega}}^\top \mathbf{x}_i)\right)$.

5.5.1 1D example

In a simple example, where $\boldsymbol{\omega} \in \mathbb{R}^2$ consists of a weight w and a bias term b, we show how RMHMC is able to take advantage of the geometry of the parameter space, whereas HMC does not. We see this in Figure 5.1, where we purposely initialise all samplers with $w = 10$ and $b = 0$ in order to show how each implementation manages when heading towards the known means of $w = 2$ and $b = 0$. There is a clear difference between the standard HMC sampler and both Riemannian samplers. The incorporation of geometry in both RMHMC schemes avoids the slower route that the HMC scheme follows. Furthermore HMC is renowned for being sensitive to the hyperparameters, such as the step size and the number of leapfrog steps. Therefore despite the fact that RMHMC still requires optimisation of these hyperparameters, it is less sensitive since the step size is automatically adapted according to the curvature of the space.

However, despite the obvious advantages of RMHMC, it comes at a severe computational cost. On the same hardware, the samples in Figure 5.1 take 0.5 s, 15.3 s

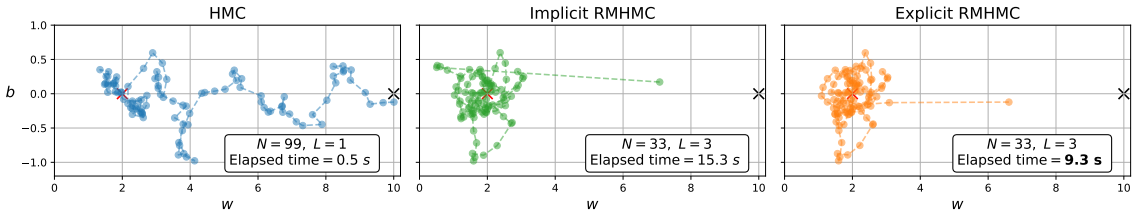


Figure 5.1: Comparison of the three Hamiltonian-based integration schemes, where all inference engines provide the same number of samples and are initialised with the same weight. This is a logistic regression example, where the weight space is two dimensional and samples are drawn from $\mathcal{N}([2, 0], 0.3\mathbf{I})$. The red cross marks the known mean from the generative model. This figure shows how the geometric knowledge of the space in both Riemannian models accelerates the particle to the true distribution, whereas the standard HMC model takes a more circuitous route. We highlight the elapsed time, which shows the advantage of Explicit RMHMC.

and 9.3 s for HMC, Implict RMHMC and Explicit RMHMC respectively.[6] This is when allowing the fixed-point iteration routine to take a maximum number of six iterations. Therefore although RMHMC is expensive, even this small example shows that Explicit RMHMC can help in reducing the computational cost.

5.5.2 Implications of the binding term on performance

For the single 1D case, such that $w \in \mathbb{R}^1$ and $b \in \mathbb{R}^1$, we can test how the performance of the binding term of the explicit integrator (Ω) affects long term performance. If we initialise the samplers at the location $[0, 0]$ in parameter space and set the trajectory length to 40, we can observe how different values of Ω affect the trajectory. In Figure 5.2, we compare the implicit integration scheme in Equation (5.3) to various explicit schemes with different values for Ω. We treat the dashed red line, Implicit RMHMC, as the ground truth and show how all other explicit RMHMC trajectories compare. It is clear from this simple example, that Ω has an important implication on the long

[6] CPU: Intel Core i7 Six Core Processor i7-8700k (3.7GHz) 12MB Cache; GPU: NVIDIA TITAN Xp; Memory (RAM): 32GB Corsair VENGEANCE DDR4 3000MHz (2 x 16GB); OS: Ubuntu 18.04; Python: 3.6.7; PyTorch: version 1.1.0. The elapsed time was measured using the Python module '`time`'. To reproduce all experiments in this chapter please refer to https://github.com/AdamCobb/hamiltorch.

term performance of RMHMC. This was also pointed out in the original paper by Tao [2016a], whereby Ω must be larger than some threshold Ω_0 but also small enough such that the error bound $\mathcal{O}(N\epsilon^2\Omega)$ remains small. Figure 5.2 shows what this means in practice.

In fact the purpose of the additional binding term was to improve on the long term performance of the integration scheme introduced by Pihajoki [2015], which is equivalent to $\Omega = 0$. We can clearly see this is the case, as the trajectory corresponding to $\Omega = 0$ in Figure 5.2 diverges from the implicit scheme shortly after traversing away from the initial condition. Furthermore, the largest value we were able to set for the binding term before the trajectory was rejected, $\Omega = 472$, also diverges from the implicit scheme.

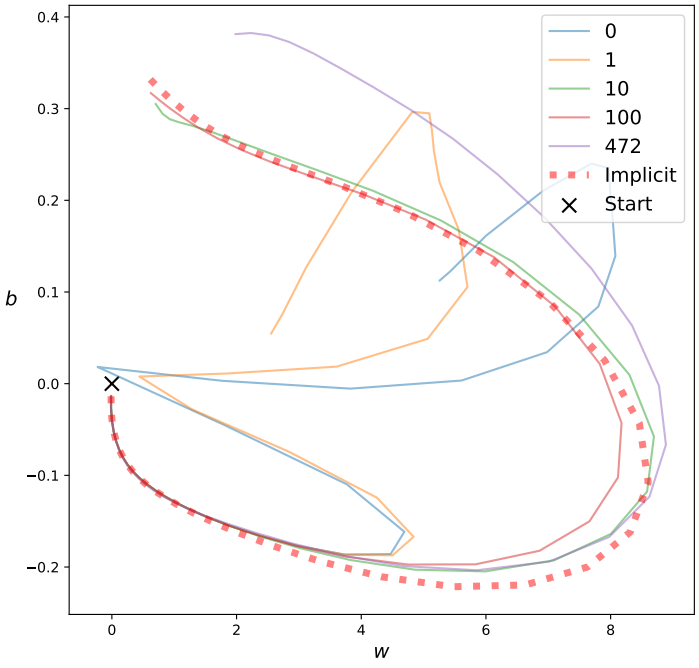


Figure 5.2: Long-term performance of explicit RMHMC when comparing to Implicit RMHMC for a single trajectory ($L = 40, \epsilon = 0.012$). Small values of Ω, as indicated in the legend, diverge shortly after the initial conditions, as well as the largest value.

5.6 Inference in a Bayesian hierarchical model

We now introduce the funnel distribution, which is defined as

$$\prod_i \mathcal{N}(\mathbf{x}_i|0, \exp\{-v\})\mathcal{N}(v|0, 9).$$

This was first introduced by Neal et al. [2003] and is a good example of a model that is challenging for Bayesian inference. Additionally, the marginal distribution of v is $\mathcal{N}(0, 9)$, thus allowing us to make comparisons to the known distribution. To demonstrate the advantages of RMHMC over HMC, we use the KL-divergence, $\mathrm{D}_{\mathrm{KL}}(p(v)\|q(v))$, as our metric of performance, as well as comparing the wall-clock time.[7] To define $q(v)$, we take the first and second moments of the samples to define our approximation to the known normal distribution $p(v)$. The KL-divergence is appropriate because in a real application we would only have access to q, and it is in our interest to know how much information is lost if we rely on q instead of p.

We compare four Hamiltonian-based Monte Carlo schemes, where our baseline is the HMC implementation of the No-U-Turn Sampler [Hoffman and Gelman, 2014], which has been set to adapt its acceptance rate to between $0.65 - 0.9$.[8] We further provide an implementation of HMC that has been tuned with knowledge of the true marginal. Importantly, we compare *implicit RMHMC* with our new *explicit RMHMC* sampler. We further highlight that for both Riemannian schemes the Fisher information matrix is no longer guaranteed to be positive semi-definite as it was for Bayesian logistic regression. Therefore we filter out negative eigenvalues by implementing *Soft-Abs* (see Section 5.3.2).

Table 5.1 shows that both forms of RMHMC outperform HMC in terms of D_{KL},

[7] Wall-clock time was measured using the Python module '`time`'. This module was used to compute the elapsed time in seconds to collect all the samples in one run.

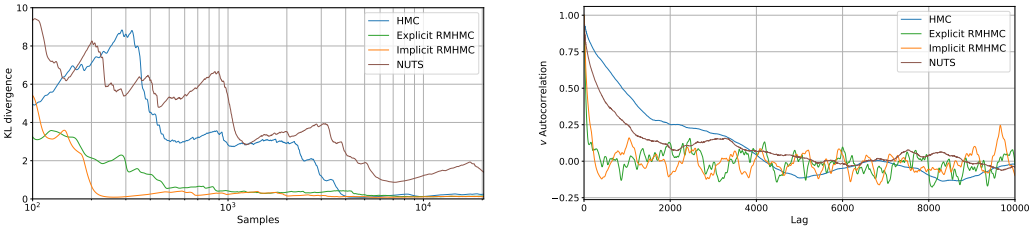
[8] The same parameter settings are followed as in [Hoffman and Gelman, 2014]. The desired acceptance rate is set to 0.75, but this results in large step sizes that cause trajectories to traverse into numerically unstable areas of the log probability space. Therefore the adapted acceptance rate is higher than would normally be desired, although results in the same poor performance.

Table 5.1: $10 + 1$ dimensional funnel: comparison across integration schemes for inference in the funnel distribution. RMHMC outperforms HMC, even when HMC is optimised with a posteriori information. Furthermore explicit RMHMC achieves comparable performance to implicit RMHMC in almost a third of the time.

Scheme	D_{KL}	Wall Time	Acc. Rate	N	L	ϵ	α
HMC - NUTS	1.109	42 min	0.87	10^5	25	0.3896	-
Implicit RMHMC	0.130	3 hr 10 min	0.93	10^3	25	0.1500	10^6
Explicit RMHMC, $\Omega = 10$	0.142	1 hr 12 min	0.81	10^3	25	0.1400	10^6
HMC (Hand-Tuned for KL)	0.270	38 min	0.97	10^5	25	0.2	-

despite providing the optimiser for HMC with a posteriori information by hand-tuning with knowledge of the true distribution and by allowing it to take two orders of magnitude more samples. The No-U-Turn Sampler, which adapts for an appropriate acceptance rate leads to too large a step size and prevents it from exploring the narrow neck of the funnel (Figure 5.4). However, the key result is in the wall-clock time, where our new explicit RMHMC achieves comparable $D_{KL}(p(v)\|q(v))$ to implicit RMHMC in almost a third of the time.[9] We further show both the convergence rate of each sampler in $D_{KL}(p(v)\|q(v))$ and the autocorrelations in Figure 5.3, where both RMHMC samplers display comparable performance in convergence and autocorrelation. The faster convergence of $D_{KL}(p(v)\|q(v))$ for both RMHMC methods demonstrates how the samplers are more efficient at building an empirical representation of $p(v)$, when compared to the other HMC samplers. This is also highlighted by how the autocorrelation between samples plateaus around zero faster for Explicit and Implicit RMHMC. This suggests that the effective sample size of both these samplers is higher than for NUTS and hand-tuned HMC. Finally, the estimated marginal distributions of $p(v)$ are shown in Figure 5.5, where the better performance of the two Riemannian samplers can be seen from the lower KL divergence.

[9]When the maximum number of fixed point iterations for Implicit RMHMC was set to 6 the acceptance rate for the same initial conditions dropped from 0.93 to 0.50, therefore the maximum number was set to 1000, with the convergence threshold for breaking out of the loop set to 0.001. At an acceptance rate of 0.50, the resulting sampler did achieve a similar performance in terms of wall time to Explicit RMHMC but this is clearly not a desirable rate.



(a) KL divergence versus samples. (b) Autocorrelation of v versus the lag.

Figure 5.3: Funnel experiment sampler diagnostics.

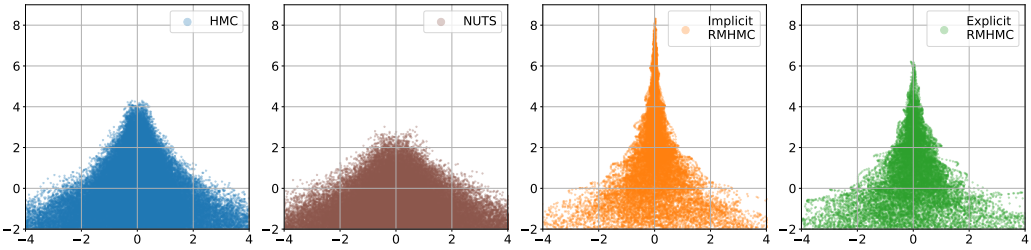


Figure 5.4: Comparison across Hamiltonian-based schemes for inference in a Bayesian hierarchical model. The Riemannian-based integrators have knowledge of the model's complex geometry and therefore more efficiently explore the space and are able to sample in the narrow part of the funnel. The new explicit integration scheme gives a significant speed-up over the previously used implicit scheme for the same number of samples. The No-U-Turn Sampler (NUTS) adapts its step size to a value that prevents it from traversing up the neck of the funnel.

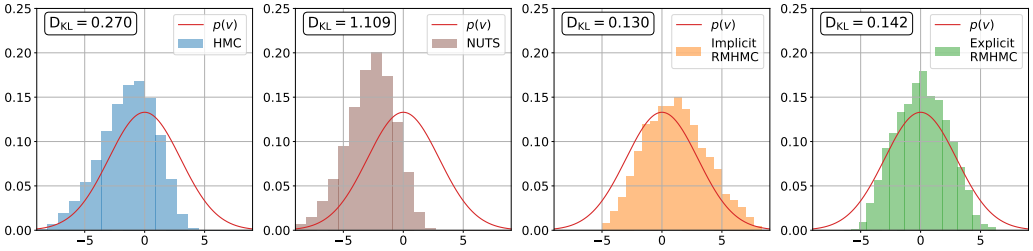


Figure 5.5: The estimated marginal distributions of $p(v)$ for all four samplers. Compared to the known distribution (the red line), the Riemannian samplers provide samples that appear less biased by the narrowness of the funnel. This leads to a lower (better) KL divergence.

5.7 Conclusion

This chapter has introduced a new explicit integration scheme for RMHMC and demonstrated its advantages over the previously used implicit integrator. The advantage of employing RMHMC is that it incorporates the geometry of the parameter

space by capturing the local behaviour of the target distribution. As an example the Riemannian-based samplers were both able to move up the neck of the funnel when sampling the funnel distribution in Section 5.6. This highlights why it is useful to use a different distance metric for performing inference in a Bayesian model when using HMC. However, it can be expensive to rely on implicit integration schemes for computing Hamiltonian dynamics for non-separable Hamiltonians as they require an indeterminate number of higher order derivative computations. Therefore in introducing explicit RMHMC, it has been shown that this new explicit integrator can provide a significant reduction in computation time that makes RMHMC a more viable alternative to Euclidean HMC.

Chapter 6

Semi-separable Hamiltonian Monte Carlo for inference in Bayesian neural networks

As soon as we move to sampling in Bayesian neural networks, we can no longer expect to rely on a Hamiltonian-based sampler that must calculate the Hessian. Therefore we build on ideas from the previous chapter but now focus on a new sampling scheme for Riemannian manifold Hamiltonian Monte Carlo that scales to BNNs. This is based on work that was presented at the 'Bayesian Deep Learning Workshop, NeurIPS 2019' [Cobb et al., 2019a].

6.1 Semi-separable Hamiltonian Monte Carlo

In Chapter 5, we saw the benefits of RMHMC, where we were able to use a metric tensor to better capture the local behaviour of the target distribution. This resulted in more efficient sampling and a rapid convergence to the distribution of interest. Therefore the question arises whether we can scale RMHMC to BNNs, when employing RMHMC requires the computation of the Hessian. In this section we will start by

introducing RMHMC formulated for BNNs and then move to a more suitable approximation in semi-separable HMC. This sampler is especially suited to Bayesian neural networks, and was only previously introduced in the context of Bayesian hierarchical models in Zhang and Sutton [2014].

6.2 Moving to semi-separable HMC in Bayesian neural networks

We can start by explicitly writing the Hamiltonian from RMHMC for a BNN,

$$H(\boldsymbol{\omega}, \boldsymbol{\tau}, \mathbf{p}) = -\log\left(f(\boldsymbol{\omega}, \boldsymbol{\tau})\right) + \frac{1}{2}\log\left((2\pi)^D |\mathbf{G}(\boldsymbol{\omega}, \boldsymbol{\tau})|\right) + \frac{1}{2}\mathbf{p}^\top \mathbf{G}(\boldsymbol{\omega}, \boldsymbol{\tau})^{-1}\mathbf{p}, \quad (6.1)$$

where $f(\boldsymbol{\omega}, \boldsymbol{\tau}) = p(\mathbf{y}|\mathbf{x}, \boldsymbol{\omega})\, p(\boldsymbol{\omega}|\boldsymbol{\tau})p(\boldsymbol{\tau})$ is our new Bayesian hierarchical model. The Gaussian prior over the weights, $p(\boldsymbol{\omega}|\boldsymbol{\tau})$, is parametrised by the precision hyperparameter, $\boldsymbol{\tau}$, which in itself follows a Gamma distribution,

$$p(\tau|\alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)}\tau^{\alpha-1}\exp(-\beta\tau),$$

with shape and rate parameters α and β, and the gamma function $\Gamma(\cdot)$. The precision parameter controls how much the weights in the neural network are allowed to vary from zero. We can tie each group of weights in a given layer to their own precision, such that each layer has a precision associated with the weights, $\tau_{\mathbf{w}}^{(l)}$, and a precision associated with the biases, $\tau_{\mathbf{b}}^{(l)}$. The challenge of performing RMHMC over a BNN comes from calculating the Hessian in building $\mathbf{G}(\boldsymbol{\omega}, \boldsymbol{\tau})$. For example, we are going to be performing RMHMC over a Bayesian CNN with 431,080 parameters and if each element were a single-precision floating point number of 4 bytes (32 bits), then the Hessian alone would require almost a terabyte of memory.

Therefore in order to perform RMHMC over a BNN of this size, we are going

to follow the formulation of semi-separable HMC (SSHMC) that was introduced in Zhang and Sutton [2014], but make it more suited to BNN inference. The key insight from SSHMC is to split $\mathbf{G}(\boldsymbol{\omega}, \boldsymbol{\tau})$ into blocks as follows,

$$\mathbf{G}(\boldsymbol{\omega}, \boldsymbol{\tau}) = \begin{bmatrix} \mathbf{G}_{\boldsymbol{\omega}}(\boldsymbol{\tau}) & 0 \\ 0 & \mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega}) \end{bmatrix}, \tag{6.2}$$

where each block is independent from the other set of parameters. In this case $\mathbf{G}_{\boldsymbol{\omega}}(\boldsymbol{\tau})$ would be independent of $\boldsymbol{\omega}$ and $\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega})$ would be independent of $\boldsymbol{\tau}$. If we keep this block structure for Equation (6.1), then the resulting Hamiltonian can be written as

$$\begin{aligned} H(\boldsymbol{\omega}, \boldsymbol{\tau}, \mathbf{p}) = &- \log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega}) - \log p(\boldsymbol{\omega}|\boldsymbol{\tau}) - \log p(\boldsymbol{\tau}) \\ &+ \frac{1}{2}\left[\mathbf{p}_{\boldsymbol{\omega}}^{\top}\mathbf{G}_{\boldsymbol{\omega}}(\boldsymbol{\tau})^{-1}\mathbf{p}_{\boldsymbol{\omega}} + \mathbf{p}_{\boldsymbol{\tau}}^{\top}\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega})^{-1}\mathbf{p}_{\boldsymbol{\tau}} + \log|\mathbf{G}_{\boldsymbol{\omega}}(\boldsymbol{\tau})| + \log|\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega})|\right] \\ &+ \frac{D}{2}\log 2\pi. \end{aligned} \tag{6.3}$$

Zhang and Sutton [2014] refer to this Hamiltonian as a *semi-separable* Hamiltonian. This Hamiltonian can then be split into two separable Hamiltonians with *auxiliary potential* terms,

$$A(\mathbf{p}_{\boldsymbol{\omega}}|\boldsymbol{\tau}) = \frac{1}{2}\mathbf{p}_{\boldsymbol{\omega}}^{\top}\mathbf{G}_{\boldsymbol{\omega}}(\boldsymbol{\tau})^{-1}\mathbf{p}_{\boldsymbol{\omega}}, \quad A(\mathbf{p}_{\boldsymbol{\tau}}|\boldsymbol{\omega}) = \frac{1}{2}\mathbf{p}_{\boldsymbol{\tau}}^{\top}\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega})^{-1}\mathbf{p}_{\boldsymbol{\tau}}, \tag{6.4}$$

as follows:[1]

$$H_{\boldsymbol{\omega}}(\boldsymbol{\omega}, \mathbf{p}_{\boldsymbol{\omega}}) = - \log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega}) - \log p(\boldsymbol{\omega}|\boldsymbol{\tau}) + A(\mathbf{p}_{\boldsymbol{\omega}}|\boldsymbol{\tau}) + \frac{1}{2}\log|\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega})| + A(\mathbf{p}_{\boldsymbol{\tau}}|\boldsymbol{\omega}), \tag{6.5}$$

[1] Ignoring the constant term that depends on π.

with $\boldsymbol{\tau}, \mathbf{p}_{\boldsymbol{\tau}}$ fixed, and

$$H_{\boldsymbol{\tau}}(\boldsymbol{\tau}, \mathbf{p}_{\boldsymbol{\tau}}) = -\log p(\boldsymbol{\omega}|\boldsymbol{\tau}) - \log p(\boldsymbol{\tau}) + A(\mathbf{p}_{\boldsymbol{\tau}}|\boldsymbol{\omega}) + \frac{1}{2}\log|\mathbf{G}_{\boldsymbol{\omega}}(\boldsymbol{\tau})| + A(\mathbf{p}_{\boldsymbol{\omega}}|\boldsymbol{\tau}), \quad (6.6)$$

with $\boldsymbol{\omega}, \mathbf{p}_{\boldsymbol{\omega}}$ fixed. The auxiliary terms appear in both Hamiltonians, however they play opposite roles in each instance. For example $A(\mathbf{p}_{\boldsymbol{\tau}}|\boldsymbol{\omega})$ is the kinetic energy component for $H_{\boldsymbol{\tau}}(\boldsymbol{\tau}, \mathbf{p}_{\boldsymbol{\tau}})$ but is the auxiliary utility for $H_{\boldsymbol{\omega}}(\boldsymbol{\omega}, \mathbf{p}_{\boldsymbol{\omega}})$. It is this coupling, along with the log-determinant terms in Equations (6.5) and (6.6) that enable energy to be *shared* between the two systems. Without these terms, it would be the same as performing RMHMC with Gibbs sampling. The advantage of jointly integrating over the hyperparameters and parameters is that the trajectory in the hyperparameter space is informed by the current location in the augmented parameter space $(\boldsymbol{\omega}, \mathbf{p}_{\boldsymbol{\omega}})$ and vice-versa.

6.2.1 Implementation

Algorithm 5 consists of the *alternating block-wise leapfrog* algorithm (ALBA), which was first introduced in Zhang and Sutton [2014] for performing semi-separable HMC. When looking at ALBA, the advantage of splitting the Hamiltonian into separable blocks is made clear. We can now rely on the separable HMC sampler, which is both more scalable in the parameter space, and simple to implement. The structure of the semi-separable Hamiltonian means that the original Hamiltonian is preserved if $H_{\boldsymbol{\omega}}(\boldsymbol{\omega}, \mathbf{p}_{\boldsymbol{\omega}})$ and $H_{\boldsymbol{\tau}}(\boldsymbol{\tau}, \mathbf{p}_{\boldsymbol{\tau}})$ are simulated exactly. This observation is going to be useful in Section 6.3, when we reintroduce the generalised leapfrog algorithm. Finally, one further observation is that each leapfrog integrator within the ALBA can take multiple leapfrog steps.

Algorithm 5 Semi-separable HMC by ALBA

Inputs: $\boldsymbol{\omega}, \boldsymbol{\tau}$
Sample: $\mathbf{p}_{\boldsymbol{\omega}} \sim \mathcal{N}(\mathbf{0}, \mathbf{G}_{\boldsymbol{\omega}}(\boldsymbol{\tau}))$, $\mathbf{p}_{\boldsymbol{\tau}} \sim \mathcal{N}(\mathbf{0}, \mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega}))$
$H = H_{\boldsymbol{\tau}} + H_{\boldsymbol{\omega}}$
 # ALBA
for l in $1, 2, \dots L$ **do**
  # Note that the integrators below can take multiple leapfrog steps.
  $\boldsymbol{\omega}^{l+\epsilon/2}, \mathbf{p}_{\boldsymbol{\omega}}^{l+\epsilon/2} \leftarrow \text{leapfrog}(\boldsymbol{\omega}^{l}, \mathbf{p}_{\boldsymbol{\omega}}^{l}, \epsilon/2 \mid \boldsymbol{\tau}^{l}, \mathbf{p}_{\boldsymbol{\tau}}^{l})$
  $\boldsymbol{\tau}^{l+\epsilon}, \mathbf{p}_{\boldsymbol{\tau}}^{l+\epsilon} \leftarrow \text{leapfrog}(\boldsymbol{\tau}^{l}, \mathbf{p}_{\boldsymbol{\tau}}^{l}, \epsilon \mid \boldsymbol{\omega}^{l+\epsilon/2}, \mathbf{p}_{\boldsymbol{\omega}}^{l+\epsilon/2})$
  $\boldsymbol{\omega}^{l+\epsilon}, \mathbf{p}_{\boldsymbol{\omega}}^{l+\epsilon} \leftarrow \text{leapfrog}(\boldsymbol{\omega}^{l+\epsilon/2}, \mathbf{p}_{\boldsymbol{\omega}}^{l+\epsilon/2}, \epsilon/2 \mid \boldsymbol{\tau}^{l+\epsilon}, \mathbf{p}_{\boldsymbol{\tau}}^{l+\epsilon})$
end for
 # MH correction
$H^{L\epsilon} = H_{\boldsymbol{\tau}}^{L\epsilon} + H_{\boldsymbol{\omega}}^{L\epsilon}$
$u \sim \mathcal{U}(0, 1)$
if $\log u > \min(0, H - H^{L\epsilon})$ **then**
  $\boldsymbol{\omega}^*, \mathbf{p}_{\boldsymbol{\omega}}^*, \boldsymbol{\tau}^*, \mathbf{p}_{\boldsymbol{\tau}}^* \leftarrow \boldsymbol{\omega}^{l+\epsilon}, \mathbf{p}_{\boldsymbol{\omega}}^{l+\epsilon}, \boldsymbol{\tau}^{l+\epsilon}, \mathbf{p}_{\boldsymbol{\tau}}^{l+\epsilon}$
else
  $\boldsymbol{\omega}^*, \mathbf{p}_{\boldsymbol{\omega}}^*, \boldsymbol{\tau}^*, \mathbf{p}_{\boldsymbol{\tau}}^* \leftarrow \boldsymbol{\omega}, \mathbf{p}_{\boldsymbol{\omega}}, \boldsymbol{\tau}, \mathbf{p}_{\boldsymbol{\tau}}$
end if
return $\boldsymbol{\omega}^*, \boldsymbol{\tau}^*$

6.3 Choosing the metric tensor and introducing a non-separable Hamiltonian

The remaining task is to select $\mathbf{G}_{\boldsymbol{\omega}}(\boldsymbol{\tau})$ and $\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega})$ for BNNs. This choice appears less obvious than the examples chosen in Zhang and Sutton [2014], where for the funnel distribution they defined $\mathbf{G}_{\mathbf{x}} = -\partial_v^2 \log p(\mathbf{x}, v)^{-1} = e^v \mathbf{I}$ and $G_v = \mathbb{E}_{\mathbf{x}}[\log p(\mathbf{x}, v)]^{-1} = (n + \frac{1}{9})^{-1}$. Due to the possible size of the network, $\mathbf{G}_{\boldsymbol{\omega}}(\boldsymbol{\tau})$ is only feasible to store and compute if it is kept diagonal, whereas we have more flexibility with $\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega})$.

As with the funnel distribution, we will approximate $\mathbf{G}_{\boldsymbol{\omega}}(\boldsymbol{\tau})$ with $\hat{\mathbf{G}}_{\boldsymbol{\omega}}(\boldsymbol{\tau})$ such that

$$\hat{\mathbf{G}}_{\boldsymbol{\omega}}(\boldsymbol{\tau}) = \begin{bmatrix} \tau_{\mathbf{w}}^{(1)}\mathbf{I}_{\mathbf{w}}^{(1)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \tau_{\mathbf{b}}^{(1)}\mathbf{I}_{\mathbf{b}}^{(1)} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \tau_{\mathbf{w}}^{(L)}\mathbf{I}_{\mathbf{w}}^{(L)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \tau_{\mathbf{b}}^{(L)}\mathbf{I}_{\mathbf{b}}^{(L)} \end{bmatrix}, \tag{6.7}$$

where $\mathbf{I}_{\mathbf{w}}^{(l)}$ and $\mathbf{I}_{\mathbf{b}}^{(l)}$ are identity matrices of size equivalent to the number of weights and biases in layer l respectively. This diagonal mass matrix is separable and cheap to compute and can therefore be used efficiently with the leapfrog integrator for separable Hamiltonians.

However, for the $\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega})$ it remains less obvious how to define an appropriate Riemannian metric. Therefore rather than building a separable Hamiltonian for $H_{\boldsymbol{\tau}}(\boldsymbol{\tau}, \mathbf{p}_{\boldsymbol{\tau}})$, we allow $\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega})$ to be dependent on $\boldsymbol{\tau}$, whilst still keeping $\boldsymbol{\omega}$ fixed. Thus, we do not approximate the metric tensor as we did for $\hat{\mathbf{G}}_{\boldsymbol{\omega}}(\boldsymbol{\tau})$. This choice results in $H_{\boldsymbol{\tau}}(\boldsymbol{\tau}, \mathbf{p}_{\boldsymbol{\tau}})$ no longer being separable. We can define the Riemannian metric according to Equation (5.7), where we use the negative Hessian of the joint likelihood $\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega}, \boldsymbol{\tau}) = -\nabla^2_{\boldsymbol{\tau}} \log p(\boldsymbol{\omega}, \boldsymbol{\tau})$. We can write each negative log probability term in Equation (6.6) out explicitly as

$$\text{1.)} \quad -\log p(\boldsymbol{\omega}|\boldsymbol{\tau}) = \frac{1}{2}\left(-\sum_i \log \tau_{\boldsymbol{\omega}_i} + \sum_i \boldsymbol{\omega}_i^2 \tau_{\boldsymbol{\omega}_i} + D_{\boldsymbol{\omega}} \log 2\pi\right),$$

$$\text{2.)} \quad -\log p(\boldsymbol{\tau}) = -\alpha \log \beta - (\alpha - 1)\log \boldsymbol{\tau} + \beta\boldsymbol{\tau} + \log \Gamma(\alpha),$$

This is where $\tau_{\boldsymbol{\omega}_i}$ corresponds to the precision for each network parameter $\boldsymbol{\omega}_i$. We can now form a non-separable $\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega}, \boldsymbol{\tau})$ by looking at $\frac{\partial^2 \log p(\boldsymbol{\omega}, \boldsymbol{\tau})}{\partial^2 \tau_{\boldsymbol{\omega}^{(l)}}}$. Here, we refer to both $\tau_{\mathbf{w}}^{(l)}$ and $\tau_{\mathbf{b}}^{(l)}$ as $\tau_{\boldsymbol{\omega}^{(l)}}$ as they can be used interchangeably. Furthermore, the number of parameters tied to each $\tau_{\boldsymbol{\omega}^{(l)}}$ is defined as Q_l. Therefore

$$
\begin{aligned}
1.) \quad & -\frac{\partial^2}{\partial^2 \tau_{\boldsymbol{\omega}^{(l)}}} \log p(\boldsymbol{\omega}|\boldsymbol{\tau}) = \frac{Q_l}{2\tau^2_{\boldsymbol{\omega}^{(l)}}}, \\
2.) \quad & -\frac{\partial^2}{\partial^2 \tau_{\boldsymbol{\omega}^{(l)}}} \log p(\boldsymbol{\tau}) = \frac{\alpha - 1}{\tau^2_{\boldsymbol{\omega}^{(l)}}}.
\end{aligned}
$$

We then recover a diagonal matrix with elements

$$
\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega}, \boldsymbol{\tau})_{ii} = \frac{1}{\tau^2_{\boldsymbol{\omega}^{(l)}}} \left(\frac{Q_l}{2} + \alpha - 1 \right), \tag{6.8}
$$

where $\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega}, \boldsymbol{\tau})_{ii} > 0$ if $\frac{Q_l}{2} + \alpha \geq 1$. This is always the case[2] if $\alpha > \frac{1}{2}$ as $Q_l \geq 1$. Furthermore $\tau_{\boldsymbol{\omega}^{(l)}} > 0$ by definition. Therefore if these conditions hold, the matrix is positive definite, which is an important result as this matrix can be used as the metric tensor in RMHMC.[3]

The advantage of employing RMHMC over the non-separable version of $\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega}, \boldsymbol{\tau})$ does not come at the same computational cost as calculating the Hessian for $\boldsymbol{\omega}$ as there are fewer hyperparameters, e.g. two per layer. Furthermore the Hessian is positive definite and does not require transforming the eigenvalues as in Section 5.3.2. We will refer to this new method as *semi-semi-separable* HMC or S3HMC as we are using semi-separable HMC, where one of the blocks is no longer separable.

Update equations for $H_{\boldsymbol{\tau}}(\boldsymbol{\tau}, \mathbf{p}_{\boldsymbol{\tau}})$. RMHMC with the Hamiltonian $H_{\boldsymbol{\tau}}(\boldsymbol{\tau}, \mathbf{p}_{\boldsymbol{\tau}})$ requires taking the derivative with respect to both $\boldsymbol{\tau}$ and $\mathbf{p}_{\boldsymbol{\tau}}$ as in the updates of Equation (5.8). If we write out $H_{\boldsymbol{\tau}}(\boldsymbol{\tau}, \mathbf{p}_{\boldsymbol{\tau}})$ again with the terms explicitly

[2] For regression, if we include the output precision, λ, as part of $\boldsymbol{\tau}$, then the condition is actually $\alpha > 1$ as $-\frac{\partial^2}{\partial^2 \tau_{\boldsymbol{\omega}^{(l)}}} \log p(\boldsymbol{\omega}|\boldsymbol{\tau}) = 0$ in this case.

[3] The independence assumption for the prior leads to the matrix being diagonal as there are no second order cross-terms.

labelled,

$$H_{\boldsymbol{\tau}}(\boldsymbol{\tau}, \mathbf{p}_{\boldsymbol{\tau}}) = \underbrace{-\log p(\boldsymbol{\omega}|\boldsymbol{\tau})}_{1.} \underbrace{-\log p(\boldsymbol{\tau})}_{2.} + \underbrace{A(\mathbf{p}_{\boldsymbol{\tau}}|\boldsymbol{\omega})}_{3.} + \underbrace{\frac{1}{2}\log|\hat{\mathbf{G}}_{\boldsymbol{\omega}}(\boldsymbol{\tau})|}_{4.} + \underbrace{A(\mathbf{p}_{\boldsymbol{\omega}}|\boldsymbol{\tau})}_{5.},$$

then we can derive $\frac{\mathrm{d}\boldsymbol{\tau}}{\mathrm{d}t} = \frac{\partial H_{\boldsymbol{\tau}}}{\partial \mathbf{p}_{\boldsymbol{\tau}}}$ and $\frac{\mathrm{d}\mathbf{p}_{\boldsymbol{\tau}}}{\mathrm{d}t} = -\frac{\partial H_{\boldsymbol{\tau}}}{\partial \boldsymbol{\tau}}$. For $\frac{\partial H_{\boldsymbol{\tau}}}{\partial \mathbf{p}_{\boldsymbol{\tau}}}$, the update remains the same as only term (3.) is a function of $\mathbf{p}_{\boldsymbol{\tau}}$. However, for $\frac{\partial H_{\boldsymbol{\tau}}}{\partial \boldsymbol{\tau}}$ the update is slightly more complicated:

$$\begin{aligned}
1.) \quad & -\frac{\partial}{\partial \tau_l}\log p(\boldsymbol{\omega}|\boldsymbol{\tau}) = \frac{1}{2}\left(-\frac{Q_l}{\tau_l} + \sum_q^{Q_l}\omega_q\right), \\
2.) \quad & -\frac{\partial}{\partial \tau_l}\log p(\boldsymbol{\tau}) = -\frac{\alpha-1}{\tau_l} + \beta, \\
3.) \quad & \frac{\partial}{\partial \tau_l}A(\mathbf{p}_{\boldsymbol{\tau}}|\boldsymbol{\omega}) = \frac{1}{2}\sum_l p_{\boldsymbol{\tau},l}^2 \frac{\partial \mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega},\boldsymbol{\tau})}{\partial \tau_l}, \\
4.) \quad & \frac{\partial}{\partial \tau_l}\frac{1}{2}\log|\hat{\mathbf{G}}_{\boldsymbol{\omega}}(\boldsymbol{\tau})| = -\frac{Q_l}{2\tau_l}, \\
5.) \quad & \frac{\partial}{\partial \tau_l}A(\mathbf{p}_{\boldsymbol{\omega}}|\boldsymbol{\tau}) = \frac{1}{2}\sum_q^{Q_l} p_{\boldsymbol{\omega}_q}.
\end{aligned}$$

Therefore

$$\frac{\partial H_{\boldsymbol{\tau}}}{\partial \mathbf{p}_{\boldsymbol{\tau}}} = \frac{1}{2}\left(\sum_q^{Q_l}\omega_q + \sum_l p_{\boldsymbol{\tau},l}^2 \frac{\partial \mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega},\boldsymbol{\tau})}{\partial \tau_l} - \frac{2Q_l}{\tau_l} + \sum_q^{Q_l} p_{\boldsymbol{\omega}_q}\right) - \frac{\alpha-1}{\tau_l} + \beta.$$

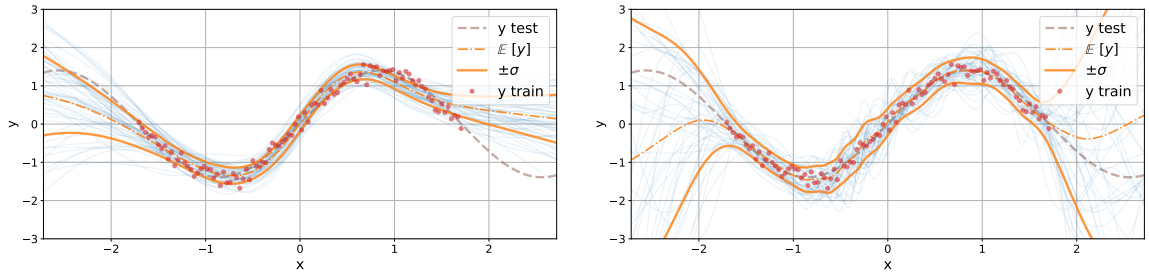
6.4 Regression example

We now look at a simple one-dimensional regression example of fitting a BNN to $y = \sin(x) + \mathcal{N}(0, 0.1^2)$. In this example, S3HMC is applied to two networks for the same data set and with the same initial conditions. One network has an architecture of $[1, 50, 1]$ and the other has an architecture of $[1, 50, 50, 1]$, where both use tanh

non-linearities.

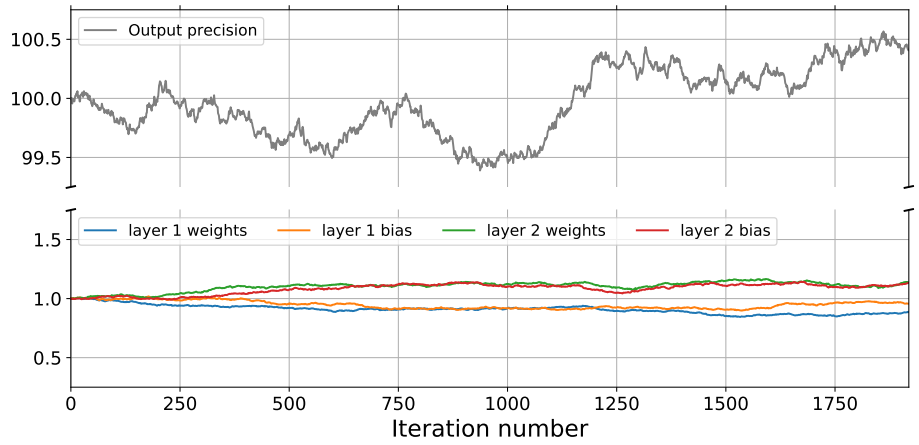
We have already seen that selecting shallower, smaller networks means that our prior assumption over the function results in the same behaviour as for a larger lengthscale in a GP. Therefore we see this behaviour in Figure 6.2, where the predictive samples from the smaller network with 151 parameters are less flexible. In comparison, the samples from the larger network of 2701 parameters have a much higher frequency component (shorter lengthscale) but still pass through the training points. In this case, the estimated marginal likelihood taken over the samples for each model would suggest that since the smaller model explains the data just as well as the larger model, then following along the lines of Occam's razor, we should pick the smaller network for this task ($\log p(\mathbf{Y}|\mathbf{X}) = -7.98$ versus $\log p(\mathbf{Y}|\mathbf{X}) = -52.52$).

One of the advantages of S3HMC is that it allows us to jointly integrate over the hyperparameters. Therefore we show the corresponding hyperparameter samples for the precisions in Figure 6.1. The majority of the layer precisions hover around the initial condition of setting the precisions to the value of 1.0. There is also the additional requirement of either setting or integrating over the output precision in the likelihood $\mathcal{N}(\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{X}), \lambda^{-1})$. In these experiments, the value of λ is initialised at 100 with the corresponding value of α in the Gamma hyperprior set to 100, whilst keeping $\beta = 1.0$. All the remaining hyperpriors parameters are kept at $\alpha = 2.0, \beta = 1.0$. The influence of the hyperprior is important as it ensures that the precisions remain in a high probability part of the space. Poor initial conditions, as well as badly chosen priors can lead to numerical instability. For example selecting a hyperprior that does not penalise sampling precisions too close to zero leads to sampling values that are too small and can cause numerical problems.

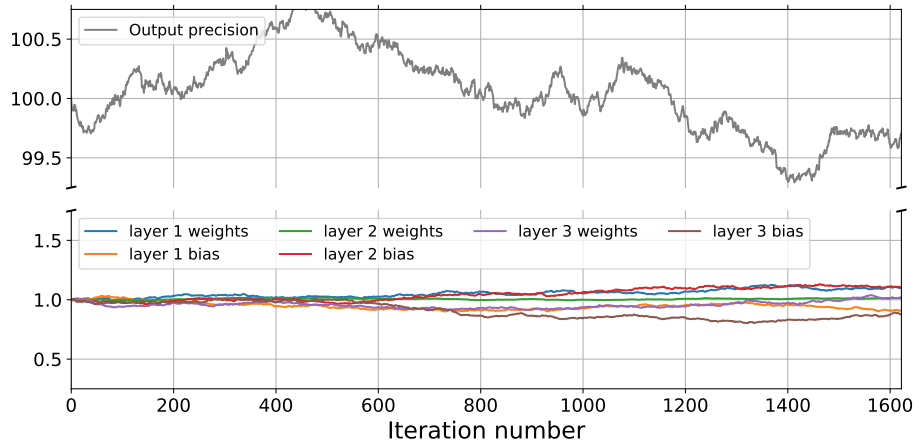


(a) $[1, 50, 1]$ (b) $[1, 50, 50, 1]$

Figure 6.1: Posterior draws from BNNs inferred via S3HMC. The comparison between Figures 6.1a and 6.1b shows how increasing the size of the network results in posterior draws that are more flexible.



(a) $[1, 50, 1]$



(b) $[1, 50, 50, 1]$

Figure 6.2: Corresponding values of $\boldsymbol{\tau}$ drawn for Figures 6.1a and 6.1b. The bias on the last layer tends to vary the most.

6.5 Classification example

We now move on to a classification example and show how S3HMC scales to a larger classification task. The network is the Pytorch MNIST CNN example [Paszke et al., 2017b] based on the LeNet architecture [LeCun et al., 1998]. This network consists of two convolutional layers and two fully connected layers, which contains 431,080 weights (including biases). Although this is not a large number in the context of the deep learning literature (e.g. 60 million [Krizhevsky et al., 2012]), it is significant for performing any form of MCMC over BNNs. In the BNN literature, HMC is often used as a ground truth comparison for ideal uncertainty behaviour with toy examples where the number of weights is small (e.g. 301 weights for HMC in Hernández-Lobato and Adams [2015] and more recently in Krueger et al. [2017]). The performance of S3HMC is compared to two baselines: a deterministic NN, trained via stochastic gradient descent, and MC dropout, with $p = 0.2$. For each inference approach, the networks are trained on the first 10,000 digits of the the MNIST training set. They are validated on the next 1,000 and then tested on the standard 10,000 test digits. Table 6.1 shows how all methods achieve comparable accuracies across the test data, with MC dropout achieving the best accuracy as expected.

Using a similar experimental set-up as in Louizos and Welling [2017], we compare the mean predictive entropy of all models over both the MNIST test data and the notMNIST [Bulatov, 2011] data set. The purpose the notMNIST data set is to test how all the networks perform when predicting over data that lies outside the training distribution (i.e. training on digits (MNIST) and predicting on letters (notMNIST)). In Figure 6.3, the top plot shows the distribution of predictive entropies for all models over the test data. All models achieve good performance over the test set and this is represented in the low entropy predictions appearing on the left-hand side of the plot. However, the interesting result is shown in the bottom plot for notMNIST, where we expect our models to be uncertain due to making predictions over data that do

Table 6.1: Test accuracy (%) for MNIST over $10,000$ digits. All models are trained over the first $10,000$ digits in the training set.

Dataset	Deterministic NN	MC dropout	S3HMC
MNIST	98.20	98.68	98.15

Table 6.2: Mean predictive entropy for notMNIST, when trained over the first $10,000$ digits in the MNIST training set. High values correspond to predictions with high uncertainty.

Dataset	Deterministic NN	MC dropout	S3HMC
notMNIST	0.37 ± 0.38	0.43 ± 0.37	1.21 ± 0.48

not feature in the training data. The deterministic NN's performance is poor with the vast majority of predictions over notMNIST having a low predictive entropy (in blue), which shows overconfidence. Both Bayesian neural networks appear suitably less confident over the notMNIST data. However, S3HMC (in green) outperforms the variational approach of MC dropout (in orange). This is shown by the predictive entropies of S3HMC lying much further to the right than MC dropout. Table 6.2 quantifies this performance by providing the mean and corresponding standard deviation for the predictive entropy for notMNIST. The higher value for S3HMC indicates that it is rightly less certain for the notMNIST data.

Therefore the conclusion from this classification experiment is that S3HMC achieves similar performance in the traditional metric of accuracy, whilst providing superior calibrated uncertainty when making predictions that lie far outside the training distribution. Furthermore, this experiment has shown that MCMC techniques can be a viable alternative to variational approaches for large networks.

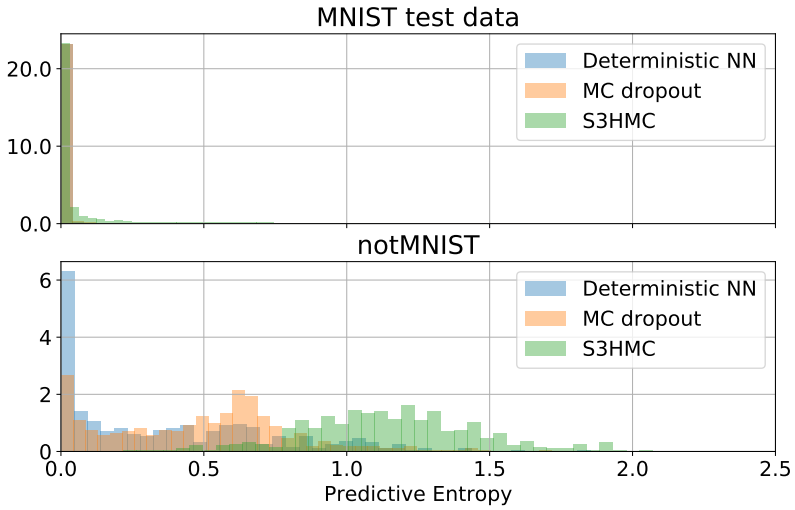


Figure 6.3: A histogram of the mean predictive entropy for both MNIST test data (top) and notMNIST data (bottom). All models get similar accuracy performance over the MNIST test data, however S3HMC is not as overconfident in its predictions, indicated by the broader distribution of higher predictive entropies. For notMNIST, S3HMC puts a higher proportion of its predicted entropies towards the maximum on the right. This is compared to the two baselines, which still show a larger proportion of overconfident predictions.

6.5.1 The challenge of selecting the experiment hyperparameters

For the classification example, it was particularly challenging to select the step size and trajectory length for each integration scheme. The objective was to balance exploration versus the acceptance rate of S3HMC. The trajectory length for the implicit integration scheme for $\boldsymbol{\tau}$ was set to 1, with the step size set to 10^{-4}. The trajectory length for the explicit separable scheme for $\boldsymbol{\omega}$ was set to 1, with a step size of 10^{-2}. α and β were set to 100 and 1000 respectively as a strong prior avoided the issues of the precisions tending to zero.

Further research is required as to appropriate ways of setting these experiment hyperparameters. Building in techniques such as Bayesian optimisation [Snoek et al., 2012] would definitely be desirable.

6.6 Limitation or advantage: scaling to large data with HMC

In the experiments above, the limiting factor on using HMC for BNNs is the size of the data set used for training. As noted by Betancourt [2015], a stochastic form of HMC can be detrimental to the ability of the sampler to explore the full log probability space. However, there are many examples in the real-world where the size of the data set is in fact small and we may still want to leverage techniques such as CNNs. Examples of such instances can be found in medical applications, where vast amounts of data are infeasible to collect [Fruehwirt et al., 2018, Bhattacharya et al., 2018]. In these scenarios, HMC may provide better performance in terms of avoiding over-fitting to small data sets and providing more reliable uncertainty estimates.

Hamiltorch: a PyTorch Python package for sampling. All the Monte Carlo methods implemented in Chapters 5 and 6 were performed using `hamiltorch`. This is a Python package developed to use Hamiltonian Monte Carlo (HMC) to sample from probability distributions. `hamiltorch` is built with a PyTorch [Paszke et al., 2017a] backend to take advantage of the innate automatic differentiation. PyTorch is a machine learning framework that makes modelling with neural networks in Python easier. Therefore since `hamiltorch` is based on PyTorch, we ensured that `hamiltorch` was built to sample directly from neural network models (i.e. objects inheriting from the `torch.nn.Module`).

To use `hamiltorch` to sample a Bayesian neural network one only needs to define a data set $\{\mathbf{X}, \mathbf{Y}\}$ and a neural network model. PyTorch allows us to run much of the code on a GPU, which results in a large speed increase over running on a CPU. An example of this can be shown by running HMC for the CNN in Section 6.5. On the GPU we were able to achieve 116.33 samples per

second, whereas on the CPU the sampling rate was 13.92 samples per second. Finally, it is simple to switch between integration schemes by changing the argument passed to the sampler. Therefore easily switching between HMC and RMHMC is part of our framework. The structure of `hamiltorch` aims to make performing HMC more scalable and accessible to practitioners, as well as enabling researchers to add their own metrics for RMHMC. For the code base please refer to https://github.com/AdamCobb/hamiltorch.

6.7 Conclusion

This Chapter has provided a new way of performing RMHMC in BNNs by adapting semi-separable HMC. This was required due to the computational burden of directly implementing RMHMC from the previous chapter, which used the Hessian. Therefore the aim was to find build a new sampler that has the benefits of RMHMC, but is closer in computational complexity to Euclidean HMC. Therefore this chapter introduced a new application of semi-separable HMC for BNNs. A new metric tensor for the hyperparameters was derived, along with the required conditions for when it is positive definite. It was then demonstrated that S3HMC has the ability to scale to models with over $400,000$ weights, and that it was able to provide better uncertainty estimates when making predictions over out-of-distribution test data. These results show that HMC-based approaches are a promising direction for future research.

Finally, in developing these techniques (both in Chapters 5 and 6), a new Python toolbox has been built that will help others in the community to build on this work. The flexibility of this toolbox, as well as the fact that it uses a PyTorch backend, makes it possible for others to easily experiment with new integration schemes and different metric tensors.

Chapter 7

Learning a covariance with Bayesian neural networks

In many real-world scenarios we may be interested in multi-output regression and how these outputs correlate. Traditional machine learning techniques for solving such problems find the challenge of fitting to both high dimensional data and large data sets difficult to overcome. Therefore BNNs offer one possible solution to this problem. This chapter proposes a novel approach of using BNNs to infer correlations between outputs, motivated with the aim of searching for exoplanets that may be able to support life. This chapter is based on work published in 'The Astronomical Journal' [Cobb et al., 2019c].

7.1 Introduction

In this chapter we must revert back to the variational approach for performing inference in BNNs. Although using HMC is becoming more scalable, unfortunately for larger data sets and models, it can still be infeasible. This is due to the requirement of differentiating the likelihood with respect to the entire data set. Therefore the focus in this chapter is for this scenario, where a variational approach is still the most

scalable option for a real-world application.

7.2 Motivating example: exoplanetary atmospheric retrieval

Exoplanets were briefly introduced in Section 2.4.1, within the context of using regression for exoplanet detection. Here, they are introduced for an application that is concerned with inferring the composition of an exoplanetary atmosphere. This task is not only important from the perspective of a planetary scientist, but also represents a difficult challenge for machine learning researchers due to the demand of ensuring that the uncertainty is suitably calibrated.

7.2.1 Background

Since 2005 we have been able to directly observe light emitted from exoplanets [Charbonneau et al., 2005, Deming et al., 2005]. One consequence of being able to observe light from exoplanets is that it is possible to analyse its frequency components, known as *spectra*. These spectra can then be used to help characterise exoplanetary atmospheres. With this information, one can start to ask questions ranging from what the current climate is, to whether the exoplanet could be habitable.

In general there are two observation techniques available to us when observing exoplanetary spectra. We can either use *transit spectroscopy* or *occultation spectroscopy*. I will broadly summarise the difference here, however for a more detailed summary please see Kreidberg [2017].

Transit spectroscopy. When the exoplanet passes between the host star and the observer, a small proportion of the star's flux is blocked. This reduction in brightness is also partly due to the exoplanet's atmosphere, such that some light will be absorbed

and the rest will pass through. Depending on the frequency band in which the transit is being observed, there will be a variation of absorption and the transmission depth will vary accordingly. A *transmission spectrum* is therefore the measured transit depth plotted against the wavelength. Transit spectroscopy tends to be more commonly used and this will be the form of the observations that we investigate in this chapter.

Occultation spectroscopy. When the exoplanet passes behind the host star (eclipse) and appears again, an increase in flux due to the reflective properties of the exoplanet (thermal or light) would increase the overall flux level.

7.2.2 Atmospheric retrieval

Atmospheric retrieval [Madhusudhan, 2018], is the process of constructing an atmospheric model from an observed spectrum. This is an inverse modelling technique, since the objective is to infer the atmospheric parameters from the observed spectrum, whereby we are only able to use a forward model that constructs a transmission spectrum from a set of atmospheric parameters. A variety of forward models exist that are used in atmospheric retrievals (See Madhusudhan [2018, Table 1]), where the forward model determines the transmission spectrum given the set of atmospheric parameters. Figure 7.1 shows a high level schematic of a forward model. We can see that a forward model is a parametric model, that is used to construct an atmospheric spectrum by setting the atmospheric parameters to certain values.

The second stage of a retrieval is the process of optimising the atmospheric parameters such that the model spectrum in Figure 7.1 closely matches the observed spectrum. The objective is to estimate the atmospheric parameters, as well as their *uncertainties*. This *inverse problem* is the key focus of this chapter, as the task is to learn an inverse model that goes in the opposite direction to the forward model, i.e. from the observed spectrum to the atmospheric parameters. The inverse model must

provide appropriate probability distributions over the atmospheric parameters given the data. Figure 7.2 displays a schematic of how the inverse model is related to the forward model.

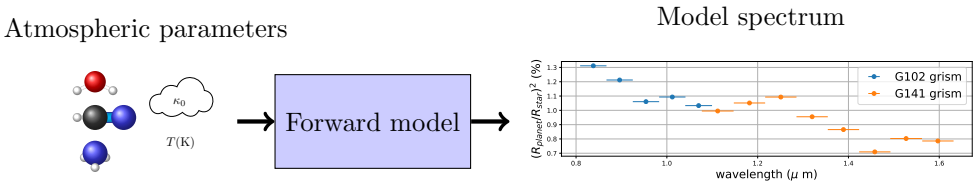


Figure 7.1: A schematic of a forward model. The model spectrum is then compared to the observed transmission spectrum as part of the atmospheric retrieval.

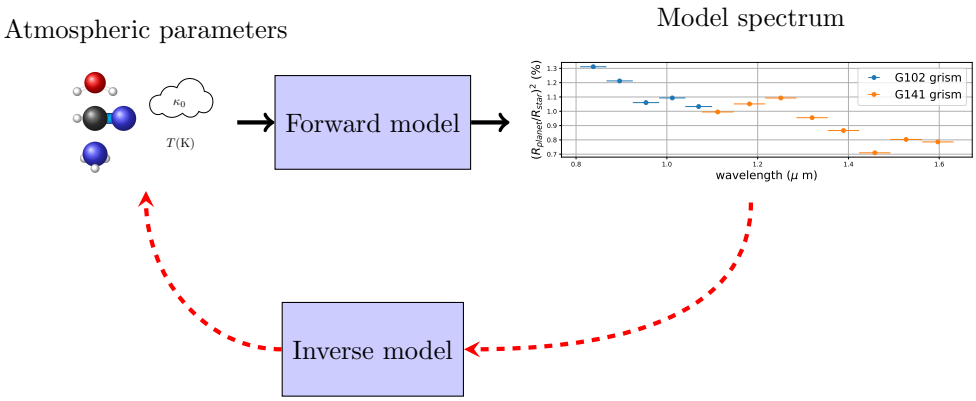


Figure 7.2: A schematic of the inverse model. The inverse model maps the spectra back to the atmospheric parameters that parameterised the forward model. The forward model is based on known physical and chemical processes and the inverse model aims to learn how to retrieve the initial parameters. For example, the inverse model could be a machine learning model such as a random forest or a neural network.

7.3 Machine learning for atmospheric retrieval

Machine learning has an important role to play in atmospheric retrievals as traditional approaches to atmospheric retrievals can be computationally expensive. Early retrievals consisted of a parametric grid search over millions of pre-calculated forward models [Madhusudhan and Seager, 2009]. However since then, Bayesian techniques

such as MCMC and nested sampling have improved on grid searches to provide distributions over atmospheric parameters (for example Madhusudhan and Seager [2010], Line et al. [2014], Waldmann et al. [2015], Oreshenko et al. [2017]).[1] As these parametric atmospheric models become more complex, the computation time required to retrieve probability distributions over parameters can require hundreds of CPU hours, which could be avoided by relying on supervised machine learning techniques to perform the retrievals.[2]

7.3.1 Previously applied machine learning models

The incorporation of supervised machine learning techniques to learn the inverse model of spectra has been recently explored by both Márquez-Neila et al. [2018] and Zingales and Waldmann [2018]. Márquez-Neila et al. [2018] propose a density estimation using random forests, whereas Zingales and Waldmann [2018] combine a generative adversarial network (GAN [Goodfellow et al., 2014]), with a technique called semantic image inpainting [Yeh et al., 2017]. The common theme across both of these techniques is to replace traditional sampling approaches with a learnt inverse model, which can make quick predictions and provide estimations of the atmospheric distributions.

Despite being important steps in the use of machine learning for exoplanetary retrieval, there are a few constraints that they impose on the retrievals. Using the GAN, as in Zingales and Waldmann [2018], requires very specific knowledge of how to divide the feature space. For example, they designed the input 2D array for their model to contain a large area dedicated to the exoplanet's water as this reflected their knowledge of the importance of water. Although impressive, the feature engineering required to achieve their performance makes it hard to reproduce. In comparison,

[1] For nested sampling see Skilling [2004], ter Braak [2006], ter Braak and Vrugt [2008].

[2] For example modelling clouds or increasing the number of atmospheric parameters can increase computation time.

the random forest is a more suited general purpose retrieval technique that can learn directly from the forward model without requiring significant feature engineering. A disadvantage of the random forest is that it does not explicitly learn a distribution over the atmospheric parameters.[3] However in spite of this disadvantage, the random forest performs well at approximating the density function for the retrieved parameters. Therefore, the work by Márquez-Neila et al. [2018] is used as a baseline for comparing to the method introduced in this chapter.

7.3.2 Building the data set

We use the same spectral data set of Márquez-Neila et al. [2018] which is described here. The data set consists of 100,000 synthetic Hubble Space Telescope Wide Field Camera 3 (WFC3) transmission spectra of hot Jupiters, because this allows us to make a direct comparison to their work. In order to create this data, a forward model must be run 100,000 times, where the atmospheric parameters are sampled from a prior distribution, $p(\mathbf{a})$, that aims to cover the space of possible exoplanetary atmospheres in which we are interested. For example, this data set was constructed to retrieve hot Jupiter-like planets.[4] The data set of spectra-parameter pairs was built using the process described in Heng and Kitzmann [2017]. This is based on five atmospheric parameters: an isothermal temperature T; abundances of H_2O, NH_3, and HCN gas; and a grey cloud opacity, κ_0. Each spectrum has 13 channels with bandpasses matching those used in Kreidberg et al. [2015] (0.838 – 1.666 µm). Their prior constraints were: $T = 500$–$2900\,\mathrm{K}$, $X_{\mathrm{H_2O}} = 10^{-13}$–$1$, $X_{\mathrm{NH_3}} = 10^{-13}$–$1$, $X_{\mathrm{HCN}} = 10^{-13}$–$1$, $\kappa_0 = 10^{-13}$–$10^{2}\ \mathrm{cm^2\,g^{-1}}$. Therefore, sampling uniformly within these ranges

[3] In this chapter, we implement the random forest as in Márquez-Neila et al. [2018] However, subsequent work by Raynal et al. [2018] suggests a more rigorous way of achieving predictive distributions with random forests.

[4] Hot Jupiters are exoplanets similar to Jupiter but have short orbital periods. This combination of a short period and being in similar size to Jupiter makes them a class of exoplanets that are more easily detectable. In fact, the first discovered exoplanet around a sun-like star, *51 Pegasi b*, is a hot Jupiter [Mayor and Queloz, 1995].

is used to construct the data set.

The data set is split into 80,000 spectra for training and 20,000 for testing. These are the same splits as is in Márquez-Neila et al. [2018]. In addition, 10,000 are held back from the training for the validation. The structure of the data is as follows. A single spectrum with 13 channels is denoted by the vector $\mathbf{s} \in \mathbb{R}^{D_{\texttt{in}}}$, where $D_{\texttt{in}} = 13$. The vector of five atmospheric parameters is defined by $\mathbf{a} \in \mathbb{R}^{D_{\texttt{out}}}$, where $D_{\texttt{out}} = 5$.

7.3.3 Baseline method: random forest

Here, I briefly summarise the random forest regression model used in Márquez-Neila et al. [2018], where the details of the model are available at https://github.com/exoclime/HELA. The core of their model comes from the `RandomForestRegressor` method in `sklearn` [Pedregosa et al., 2011]. A random forest (RF) consists of multiple decision trees (or regression trees, for the case of continuous data), whereby each tree makes a prediction given an input (see Criminisi et al. [2012]). In their model, they used 1,000 regression trees. They set the number of nodes in each tree via a variance threshold of 0.01. This is a metric that is related to the proportion of the remaining training data that is split at the current node. To produce the density plots for the retrieval, they treated each tree's prediction as a sample from an empirical distribution.

7.4 Learning a covariance matrix with Bayesian neural networks

Our objective is to learn a relationship between transmission spectra and atmospheric parameters. However, learning correlations between these parameters is vital for planetary scientists. These correlations can inform us of characteristics such as the degeneracies that we expect to see between certain parameters. In other words, it

might be difficult to tell the difference between a planet with a high cloud opacity or a planet with low abundances of certain molecules, as either would result in similar spectral features. We will see evidence of these degeneracies later in Section 7.5.2. Furthermore, when certain spectral features are more prominent, we would expect to see distributions that emanate these more prominent features. For example, if the spectral features associated with H_2O are more prevalent in the spectrum, then the retrieved distribution over H_2O should demonstrate this confidence by being more concentrated in its probability mass. This requirement of retrieving atmospheric parameters, whilst quantifying uncertainty points to a technique that learns a covariance. In order to do this, the next section will introduce a new loss function for a Bayesian neural network and discuss its implications.

7.4.1 The new loss function

Our task is to accurately predict the atmospheric parameters and provide predictive[5] distributions over their values. These parameters are expected to covary and since we are aware of this domain knowledge, we can choose to represent their joint distribution by an approximating distribution that captures these correlations. One appropriate distribution is the multivariate normal distribution and this is the one that will be used here. In order to achieve this approximation, the output of the BNN must consist of a lower triangular matrix $\mathbf{L} \in \mathbb{R}^{D_{\text{out}} \times D_{\text{out}}}$ and a mean vector $\boldsymbol{\mu} \in \mathbb{R}^{D_{\text{out}}}$. We can then represent the precision matrix of a multivariate normal via its Cholesky decomposition $\boldsymbol{\Lambda} = \mathbf{L}\mathbf{L}^\top$.

The loss can then be constructed to ensure that the network learns the correlations between the atmospheric parameters. Therefore our loss is the negative log-likelihood

[5]In the atmospheric retrieval literature, the output distribution would be called the posterior distribution over the parameters. However, in introducing a BNN we are inferring the posterior over the weights of the network and then working with this posterior to infer a predictive distribution.

of the multivariate normal, as defined by $\boldsymbol{\mu}$ and $\mathbf{L}$. The loss,

$$\mathcal{L}(\boldsymbol{\omega}, \boldsymbol{\mu}, \mathbf{L}) = -2 \sum_{d=1}^{D_{\text{out}}} \log(l_{dd}) + (\mathbf{y} - \boldsymbol{\mu})^\top \mathbf{L}\mathbf{L}^\top (\mathbf{y} - \boldsymbol{\mu}), \tag{7.1}$$

is defined to be implicitly dependent on the network weights $\boldsymbol{\omega}$ through the lower triangular matrix $\mathbf{L}$ and the inferred mean $\boldsymbol{\mu}$ (see Figure 7.3). As also mentioned in Dorta et al. [2018], we must be careful to ensure that the diagonal elements, l_{dd}, of $\mathbf{L}$ are positive such that $\boldsymbol{\Lambda}$ is positive-definite. To ensure this property of $\boldsymbol{\Lambda}$, we pass the diagonal terms through an exponential function. In comparison to previous loss functions that have been used for BNNs, such as the squared loss and the heteroscedastic squared loss (see Gal [2016b, Chapter 4]), the new loss in Equation (7.1) is able to model correlations between atmospheric parameters. These inferred correlations lead to better uncertainty estimates for the retrieved atmospheric parameters than the previous losses, which is highlighted in Section 7.5.3.

7.4.2 The model

The structure of the model, which will be referred to as `plan-net`, is shown in Figure 7.3. There are four layers of 1024 units, where each of which follows the structure of a concrete dropout layer [Gal et al., 2017a].[6] The advantage of using a concrete dropout layer means that the value of dropout in each layer is jointly optimised.[7] Referring back to Equation (3.20), concrete dropout explicitly incorporates the layer-wise dropout probability p_l within the ELBO of the BNN, rather than just the weight parameters (see end of Section 3.2.5 for more details on concrete dropout).

The result of the `plan-net` configuration is a multivariate normal distribution, where the covariance and mean are dependent on both the network weights and the

[6] For deciding on the architecture, a grid search was implemented over the number of layers and the number of units per layer.

[7] The *Adam* optimisation algorithm [Kingma and Ba, 2014] is used during the training of the model parameters.

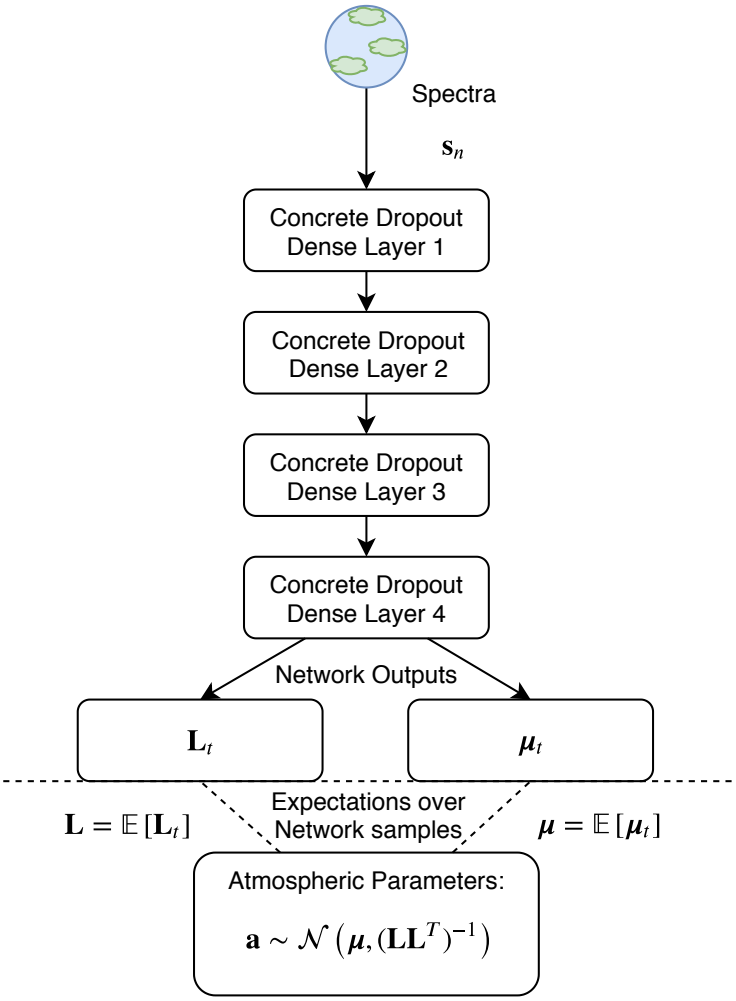


Figure 7.3: `plan-net` model procedure at test time for a given spectrum $\mathbf{s}_n$. T samples are taken from the BNN and the expectations over the lower triangular matrix and the mean are then used to parameterise the multivariate normal distribution. The corresponding atmospheric parameters can then be drawn from this distribution to retrieve the atmospheric parameters. Each concrete dropout layer consists of 1024 units.

observed spectra. The process of a retrieval using `plan-net` can be written as a two-stage process. The first stage is to sample the parameters of the multivariate normal to get the predictive distribution over their values. This stage is the same as in Equation (3.1), however it is included here with the new notation:

$$p(\boldsymbol{\mu}^*, \mathbf{L}^*|\mathbf{s}^*, \mathbf{S}, \mathbf{A}) = \int p(\boldsymbol{\mu}^*, \mathbf{L}^*|\mathbf{s}^*, \boldsymbol{\omega})\, p(\boldsymbol{\omega}|\mathbf{S}, \mathbf{A}) \mathrm{d}\boldsymbol{\omega}, \tag{7.2}$$

where $\mathbf{S}$ and $\mathbf{A}$ correspond to the observed spectra and atmospheric parameters (equivalent to $\mathbf{X}$ and $\mathbf{Y}$ as before). If we sample from the `plan-net` model by implementing T forward passes, we can build a set of predictive samples $\{\boldsymbol{\mu}^{*(t)}, \mathbf{L}^{*(t)}\}_{t=1}^{T}$. The second stage of the process is to evaluate these samples from the multivariate Gaussian that they form. The objective is to sample from the probability distribution over the atmospheric parameters, as defined as

$$p(\mathbf{a}^*|\boldsymbol{\mu}^*, \mathbf{L}^*, \mathbf{s}^*, \mathbf{S}, \mathbf{A}) = \mathcal{N}\left(\mathbf{a}^*|\boldsymbol{\mu}^*, (\mathbf{L}^*\mathbf{L}^{*\top})^{-1}\right). \tag{7.3}$$

One way to estimate the probability distribution over the atmospheric parameters is to take the expectations over the samples,

$$\mathbb{E}[\boldsymbol{\mu}^*] = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{\mu}^*(\mathbf{s}_n)^{(t)}, \quad \mathbb{E}[\boldsymbol{\Lambda}^*] = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{\Lambda}^*(\mathbf{s}_n)^{(t)}, \tag{7.4}$$

where the dependence of these parameters on the input spectra has been indicated in the equations. Then retrieving from the predictive distribution over the atmospheric parameters can be done according to

$$\mathbf{a}^* \sim \mathcal{N}\left(\mathbb{E}[\boldsymbol{\mu}^*], \mathbb{E}[\boldsymbol{\Lambda}^*]^{-1}\right). \tag{7.5}$$

Alternatively, the other option is to sample from each instance of a probability dis-

tribution as given by

$$\mathbf{a}^{*(t)} \sim p(\mathbf{a}^* | \boldsymbol{\mu}^{*(t)}, \mathbf{L}^{*(t)}, \mathbf{s}^*, \mathbf{S}, \mathbf{A}). \tag{7.6}$$

There are advantages and disadvantages to implementing either Equation (7.5) or (7.6). If we follow the formulation that relies on the expectation, there is the advantage of selecting a predetermined value for T and then sampling becomes cheap as one only needs to sample from the multivariate Gaussian. In comparison, sampling from each instance of a probability distribution as in Equation (7.6) requires a forward pass through the network for a each sample. For simple models, each forward pass may not be too computationally expensive but as models get larger this method could end up being the less attractive choice. On the other hand, a disadvantage of making an approximation by using parameter expectations is that information may be lost. More specifically, variations in the mean and in the scale are no longer conserved as they are when using the other option of sampling from each instance of the distribution.

7.4.3 Ensemble of Bayesian neural networks

Instead of relying on a single `plan-net` model, we also look at the possibility of employing an ensemble of models. Previous work has shown that there are advantages in using an ensemble of neural networks, especially as they can offer more accurate estimations of the predictive uncertainty than a single network [Lakshminarayanan et al., 2017, Gal and Smith, 2018]. The additional benefit is that an ensemble is more robust to changes in weight initialization and to the path taken during stochastic optimization. However, for every model added to the ensemble there is an associated memory and computational cost. I will discuss how to decide on the number of models in Section 7.5.1.

Another challenge in using an ensemble is in how the outputs from the individual

models are combined. As before, this combination depends on whether the retrieval follows Equation (7.5) or (7.6). If we use Equation (7.6), then the simple extension is to treat the samples from all models equally and follow the same routine as for one model. However, if we use the expectations for each individual model, then combining the models may require an extra step. In our case, each output is the mean and covariance of a multivariate normal distribution. Therefore in combining these distributions together, we can treat the overall output from the ensemble as a Gaussian mixture model, whereby each component's weight corresponds to $^1/_M$, where M is the number of models in the ensemble.

To calculate the expectation of this mixture model, $\boldsymbol{\mu}^*_{\text{ens}}$, we take the average of the individual component means such that

$$\boldsymbol{\mu}^*_{\text{ens}} = \frac{1}{M} \sum_{m=1}^{M} \mathbb{E}[\boldsymbol{\mu}^*]_m. \tag{7.7}$$

The variance of the mixture model $\boldsymbol{\Sigma}^*_{\text{ens}}$ can be calculated by employing the law of total variance:[8]

$$\boldsymbol{\Sigma}^*_{\text{ens}} = \frac{1}{M} \sum_{m=1}^{M} \left(\mathbb{E}[\boldsymbol{\mu}^*]_{\text{m}} - \boldsymbol{\mu}_{\text{ens}}\right)^2 + \frac{1}{M} \sum_{m=1}^{M} \mathbb{E}[\boldsymbol{\Lambda}^*]_m^{-1}. \tag{7.8}$$

This combines the variance in the component means with the expectation of the variance of the individual models, thus taking into account how unsure each model is and how far each model's mean lies from the ensemble mean. Therefore the atmospheric parameters retrieved via the ensemble $\mathbf{a}^*_{\text{ens}}$ are distributed according to $\mathbf{a}^*_{\text{ens}} \sim \mathcal{N}\left(\boldsymbol{\mu}^*_{\text{ens}}, \boldsymbol{\Sigma}^*_{\text{ens}}\right)$.

[8] Also known as Eve's law as the combination of $\mathbb{E}$'s and *'var'*s spell EVVE [Blitzstein and Hwang, 2014, Page 401]

7.5 Atmospheric retrieval: results for hot Jupiter-like exoplanets

We now look at the results of performing the atmospheric retrieval using our model and comparing to the baseline of the random forest. The performance is first compared over the synthetic test spectra before comparing the results on the Hubble Space Telescope Wide Field Camera 3 transmission spectrum of the exoplanet *WASP-12b*.

To remain consistent with Márquez-Neila et al. [2018], the metric for comparison is the coefficient of determination, R^2, where,

$$R^2 = 1 - \frac{\sum_{n=1}^{N}\sum_{d=1}^{D_{\texttt{out}}}\left(a_{d,n} - \mu^*_{\text{ens}_{d,n}}\right)^2}{\sum_{n=1}^{N}\sum_{d=1}^{D_{\texttt{out}}}\left(a_{d,n} - \tilde{a}_d\right)^2} \tag{7.9}$$

as defined in the `sklearn.metrics` Python package, where the summation is over both the size of the data set N and the output dimension $D_{\texttt{out}}$. $\tilde{a}_d$ is the data mean for each atmospheric parameter and the prediction for each data point is given by $\mu^*_{\text{ens}_{d,n}}$. This can be viewed as a ratio between the residuals for the model prediction and the total sum of squares. The numerator is small when the predictions are close to the true values and the denominator is large when the variance of the data is large. Therefore if the data has a high variance but the model is still able to make close predictions (in squared error) then the R^2 score will be close to 1.0, indicating the model is explaining the data well. If the variance of the predictions closely matches the variance of the data then R^2 score will be close to 0.0, which suggests the model is not able to explain the data.

7.5.1 Synthetic test data

Table 7.1 displays a model comparison of R^2 values over the 20,000 test spectra. The results show that both the single `plan-net` model and the ensemble outperform the

Table 7.1: Table reports R^2 values for each atmospheric parameter. Values near 1 indicate high correlation between model prediction and the known atmospheric parameters. The single `plan-net` model already achieves a higher overall mean R^2 as well as being higher for each individual parameter when compared to the random forest. In addition, the performance boost from the ensemble shows the advantage of combining multiple models. Bold indicates the best R^2 value for each parameter.

	T(K)	$\log X_{H_2O}$	$\log X_{HCN}$	$\log X_{NH_3}$	κ_0	Mean
Our Ran. Forest R^2	0.746	0.608	0.466	0.700	0.736	0.651
Ran. Forest[9] R^2	0.746	0.608	0.467	0.700	0.737	0.652
`plan-net` R^2	0.765	0.629	0.482	0.723	0.748	0.669
Ens. 5 `plan-net` R^2	**0.770**	**0.631**	**0.488**	**0.724**	**0.751**	**0.673**

random forest. The random forest results are both reproduced and reported from the paper to show that this is a fair comparison. Figure 7.4 provides further insight into the selection process for the number of models in the ensemble. We can see how both the R^2 and mean squared error (denoted by MSE) suggest that an ensemble of five models is an appropriate choice, as both metrics plateau at this ensemble number.

Figure 7.5 is a scatter plot of predicted log abundances of H_2O against the true values from the test set. The dashed red line corresponds to when predicted equals true and represents ideal performance. This scatter plot emphasises the difficulty in making predictions on this data set. Despite achieving the best performance, many test planets lie far away from the dashed red line. Therefore it is even more important that the uncertainties provided by the model account for this behaviour.

7.5.2 Transmission spectrum of WASP-12b

Given the performance over the synthetic test data set, we can further test the models on the WFC3 transmission spectrum of WASP-12b. Figure 7.6 shows the posterior plots for the random forest and the `plan-net` ensemble. In the case of WASP-12b, the `plan-net` ensemble retrieves marginalised predictive distributions similar to the

[9]Reported from Márquez-Neila et al. [2018].

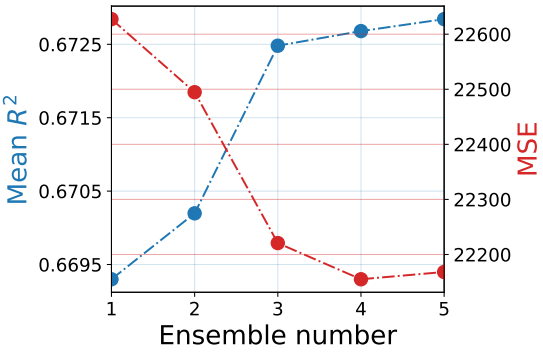


Figure 7.4: Shows the R^2 performance in blue and the mean squared error (MSE) performance in red, both plotted against the number of models in the ensemble. Performance plateaus at an ensemble number of 5, which is the number selected.

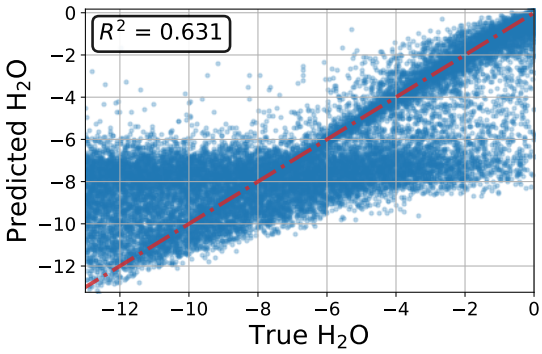


Figure 7.5: Predicted log abundances of H_2O against the true values from the test set. These are the predictions from the ensemble of 5 `plan-net` models.

random forest for the cloud opacity (κ_0) and for the abundance of NH_3 and H_2O. For temperature and HCN, the `plan-net` ensemble has a distribution that is consistent with the retrieval performed in Kreidberg et al. [2015], while the random forest favours cooler temperatures. Both models favour low ($\leq 10^{-7}$) abundances for HCN and NH_3, indicating a non-detection of these molecules. These numerical comparisons can be seen in Table 7.2.[10] In general the `plan-net` ensemble displays expected uncertainty behaviour. Fisher and Heng [2018] found that WFC3 transmission spectra are often adequately explained by H_2O, T and κ_0 only. Therefore the results in Figure 7.6b agree with this statement, as shown by the narrower marginal distributions, when comparing to the less confident distributions over the abundances of HCN and NH_3.

Finally, the retrieval performed over WASP-12b by the `plan-net` ensemble follows the direct sampling method of Equation (7.6). Both methods achieved identical results to two significant figures for the retrieval, therefore only one is listed in Table 7.2. Further experiments would be required to determine how the two methods compare

[10] Márquez-Neila et al. [2018] utilise a constant-opacity cloud parameterisation, while Kreidberg et al. [2015] use a cloud and haze model that assumes an opaque grey cloud deck, which introduces degeneracies between the cloud and haze parameters. Consequently, a direct comparison between the two models cannot be made in Table 7.2.

Table 7.2: Retrieved atmospheric parameters for WASP-12b. All retrievals are consistent, with the ensemble `plan-net` model achieving closer agreement with the temperature and H_2O abundance retrieved by Kreidberg et al. [2015]. We note that Kreidberg et al. [2015] did not retrieve for $\log X_{\mathrm{HCN}}$ and $\log X_{\mathrm{NH_3}}$. They also used a different cloud parameterization that makes κ_0 not applicable to their model. Errors are reported for one standard deviation, where we report the median and equivalent asymmetric posterior percentiles for the random forest and for Kreidberg et al. [2015].

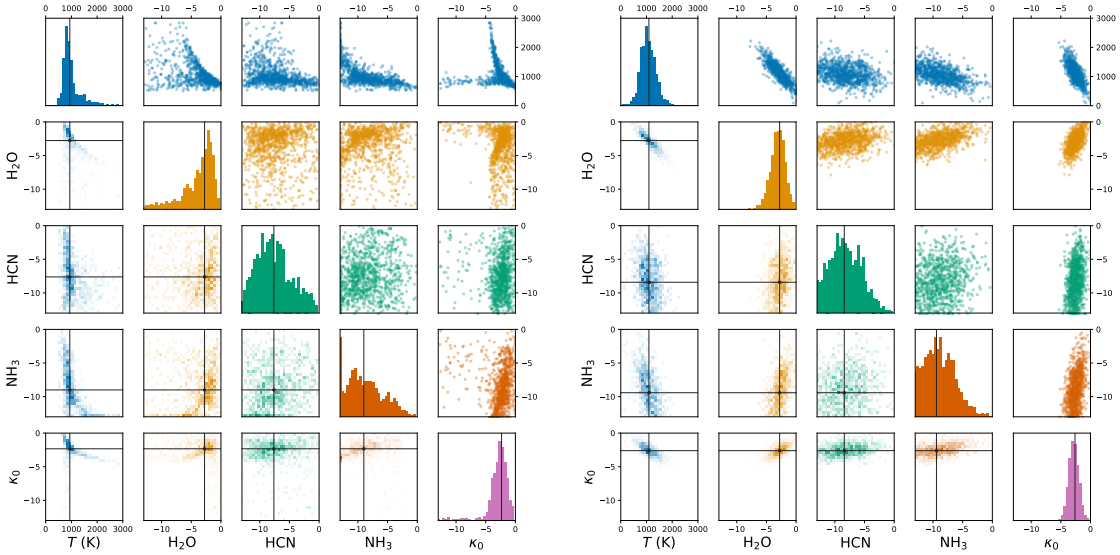
	T(K)	$\log X_{\mathrm{H_2O}}$	$\log X_{\mathrm{HCN}}$	$\log X_{\mathrm{NH_3}}$	κ_0
Kreidberg et al. [2015]	1371^{+466}_{-343}	$-2.7^{+1.0}_{-1.1}$	-	-	-
Márquez-Neila et al. [2018] nested sampling	1105^{+545}_{-287}	$-3.0^{+2.0}_{-1.9}$	$-8.5^{+3.8}_{-2.9}$	$-8.4^{+3.1}_{-2.9}$	-2.8 ± 0.9
Our Rand. Forest	937^{+410}_{-146}	$-2.8^{+1.5}_{-3.4}$	$-7.5^{+3.4}_{-2.9}$	$-9.2^{+4.1}_{-2.7}$	$-2.3^{+1.1}_{-1.6}$
Ens. 5 `plan-net`	1123 ± 301	-2.8 ± 1.2	-8.4 ± 3.0	-9.5 ± 3.2	-2.6 ± 0.8

in their retrieval distributions. However in terms of wall-clock time, directly sampling is an order of magnitude slower than following the ensemble expectation procedure of Equations 7.7 and 7.8. Directly sampling requires 1000 forward passes through the model (1.03 ± 0.10 s), whereas we can afford to take fewer samples when taking the expectations and hence produce the distributions faster (e.g. 30 samples per model in ensemble, such that 150 samples takes 0.17 ± 0.02 s).[11] We then would perform a retrieval by sampling from the inferred multivariate normal, which is equivalent for both methods.

7.5.3 The advantage of the new loss function

In order to highlight why the new loss function is preferred over previous loss functions, Figure 7.7 displays a Bayesian neural network that has been trained using a heteroscedastic squared loss as in Gal [2016b, Chapter 4]. This loss assumes a diagonal covariance and therefore can only learn independent variances for each parameter. Therefore the predictive distributions in Figure 7.7 no longer include correlations between atmospheric parameters. This can be seen by comparing Figure 7.7 with Figure

[11] CPU: Intel Core i7 Six Core Processor i7-8700k (3.7GHz) 12MB Cache; GPU: NVIDIA TITAN Xp; Memory (RAM): 32GB Corsair VENGEANCE DDR4 3000MHz (2 x 16GB); OS: Ubuntu 18.04; Python: 3.6.7; Tensorflow: version 1.8.0. Wall-clock time was measured using the in-built Jupyter Notebook (version 6.0) '`%timeit`' command.



(a) Random forest (b) `plan-net` ensemble

Figure 7.6: Retrieval analysis of the WFC3 transmission spectrum of WASP-12b, where we compare the random forest to the `plan-net` ensemble. The black cross denotes the median for the samples, where we report the results in Table 7.2. We note consistent results across all models.

7.6b. As an example, the heteroscedastic squared loss cannot capture the correlation between H_2O and T, which is captured by the newly introduced loss function. Therefore this new loss offers a clear advantage over previously used loss functions for BNNs.

7.5.4 Limitations for atmospheric retrieval

As long as the data set used to train the model contains all relevant molecules, BNNs can inform which molecules should be considered in a traditional retrieval analysis based on retrieved abundances and their uncertainties. A single `plan-net` must be trained once for a certain class of planets, e.g., WFC3 transmission spectra of hot Jupiters. Once the model has been trained, all inferences with that model are fast and repeatable, for the class of planets represented in the training set. Therefore, although training the model can be computationally expensive, this only needs to be done once. In this example, each `plan-net` model takes 20 minutes to train over the

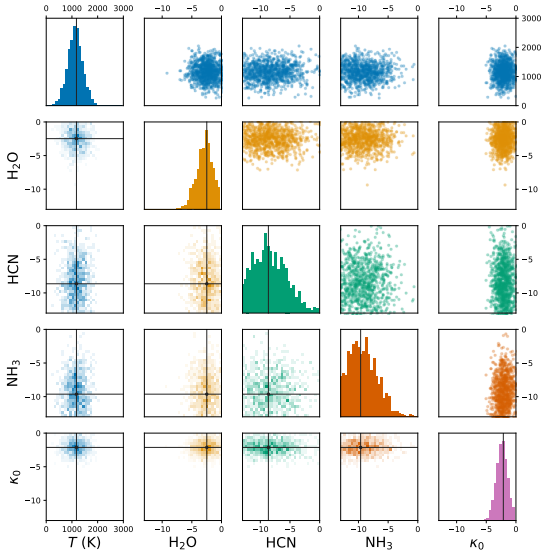


Figure 7.7: Retrieved atmospheric parameters for WASP-12b, where a `plan-net` model is trained using a heteroscedastic squared loss rather than the new loss function. This model is unable to learn the correlations as in Figure 7.6b.

WFC3 transmission spectra. Thus, despite the limitations of BNNs, their results are valuable and help save computation time spent on retrieval analyses.

7.6 The generalisation of this method

The loss function introduced in Equation (7.1) along with the sampling routines provides a general method for learning correlations for multivariate regression problems. Although we focus on the exoplanetary atmospheric retrieval in this chapter, any task which requires learning a covariance will benefit from this model.

Other methods such as the Gaussian process that was introduced in Section 2.1.2, are especially suited to learning covariances. However, their dependence on the size of the data makes this task computational expensive for data sets much larger than 10,000 due to the inversion of an $N \times N$ covariance matrix. Current research has shown that exact Gaussian process regression on a data set of order 10^5 is possible, but it would require access to 8 GPUs [Wang et al., 2019]. This would be too resource

heavy for our task. Therefore the requirement for a model such as a BNN is necessary in this task where there are 80,000 training points. However, there are many examples where we do not have access to large data sets of exoplanets. It is in these scenarios where Gaussian processes may be more suitable.

7.7 Conclusion

This chapter introduced a BNN model for multivariate regression that assumes a full covariance Gaussian distribution as its noise model. This work is significant because it offers a new technique for modelling correlations between outputs when using BNNs. This technique was motivated by the example of exoplanetary atmospheric retrieval and represents the first study that employs BNNs for atmospheric retrievals, setting the foundation for further research in this area.[12] Furthermore, the `plan-net` model is not limited to one application domain; rather, it is a new general technique for using BNNs to learn covariances between outputs.

Future work could investigate learning more complicated distributions by, for example, using normalising flow [Papamakarios et al., 2017]. Furthermore, another promising avenue for research is the performance of atmospheric retrievals in low data regimes, conditions under which techniques like Gaussian processes might be more appropriate.

[12] All the code in this chapter is available at https://github.com/exoml/plan-net.

Chapter 8

Conclusion and future research

8.1 Future work

8.1.1 Decision-making with Bayesian neural networks

There are various further avenues that can be explored when taking the work from Chapter 4 forward. One possible area to explore is Bayesian optimisation [Snoek et al., 2012] and ways in which loss-calibrated BNNs can be leveraged in this application domain. There are a few examples of Bayesian optimisation that do use BNNs but their success has been limited [Snoek et al., 2015, Springenberg et al., 2016] because neural networks are not conditioned on data in the same manner that Gaussian processes are. However, BNNs allow us to employ Bayesian optimisation over high-dimensional data in a way that Gaussian processes do not.

A further avenue could be to find other ways that loss-calibrated Bayesian neural networks could be applied to real-world problems. However, for each application we would need to overcome the challenge of designing an appropriate utility function, which may not always be straightforward. Therefore in future work it may be desirable to find ways of automatically learning utility functions rather than designing them ourselves. Examples of similar problems exist in literature, such as in inverse

reinforcement learning [Ng et al., 2000], where the task is to learn a reward for a reinforcement learning task.

8.1.2 MCMC in Bayesian neural networks

Currently available hardware and software makes it possible to scale gradient-based sampling techniques to large networks. The work in Chapter 6 has shown that it is possible for MCMC techniques to compete with SVI approaches to BNNs. Not only are they able to compete, but they are also able to provide better calibrated uncertainties that can inform us more consistently of when the model is unsure of its output. However, there is more to be done before we see HMC approaches consistently outperforming the state of the art in large-scale applications.

Two approaches may alleviate some of the problems associated with performing HMC in BNNs. One extension would be to use ensemble techniques by partitioning the data in a way that allows for larger data sets, whilst avoiding a form of stochastic gradient HMC. This extension would look for appropriate ways to split the data and to properly combine the predictive distributions of each of the individual models. The second extension would combine optimisation-based inference (e.g. SVI) with MCMC techniques. This is especially true for Riemannian-based HMC samplers, where initialising large networks far from the optimum can result in a failure of RMHMC.

The natural limitation of HMC to smaller data sets could offer an advantage for real-world applications, where data may be innately sparse or expensive to collect. However, these may still be applications that could benefit from a large convolutional neural network that is suited to computer vision tasks. Therefore an MCMC technique that does not over-fit to the data may provide a solution in this scenario. For example, a medical application may heavily rely on computer vision but only have a limited number of training examples available. In this scenario we may want to take advantage

of the inductive bias of a CNN, but also want to avoid the need for access to large data sets.

8.1.3 Real-world applications

Chapter 7 displayed a real-world application, where Bayesian neural networks provided a solution to the problem of exoplanetary atmospheric retrieval. However, applications like autonomous driving (see Chapter 4) and medicine [Leibig et al., 2017, Fruehwirt et al., 2018] have already benefited from the use of Bayesian neural networks. Therefore the key challenge in applying BNNs to real-world applications is balancing the need for meaningful distributions with the ability to scale to large networks and large data. Overcoming this challenge requires communicating with domain experts and understanding what their requirements are and how they can be achieved. Chapter 7 is an example of this kind of problem and shows how the challenges in other fields, such as astrophysics, can drive improvements in the machine learning techniques themselves.

8.2 Conclusion

This thesis has introduced new ways of scaling Bayesian neural networks to real-world problems in a manner that ensures their distributions remain meaningful through appropriate calibration. Chapter 1 outlined the objectives of this thesis as well as the motivation of the work. Chapter 2 gave an overview of three key components of machine learning: the data, the model and uncertainty. In particular, it highlighted how models such as linear regression and Gaussian processes relate to BNNs and how the structural form of these models can provide insights into BNNs. Chapter 3 gave an overview of existing techniques and outlined how previous work has tried to tackle the problem of scaling BNNs. This chapter was also to summarise common

approximate inference techniques.

Chapter 4 highlighted the importance of including BNNs within the broader framework of Bayesian decision theory and how current metrics such as accuracy are often not the sole determinant of performance. Rather than using metrics such as accuracy, utility functions were introduced specifically designed for each task. Chapter 4 also introduced a new ELBO for variational approaches to BNNs that is better suited to Bayesian decision theory. This is the first piece of work to apply loss-calibration to solve issues of non-tractability in BNNs. Finally, by introducing a way of scaling loss-calibrated approximate inference to BNNs, others have been able to extend this work to applications with continuous utilities [Kuśmierczyk et al., 2019b] or to offer other solutions for correcting for inaccurate posterior approximations [Kuśmierczyk et al., 2019a]. Therefore the significance of this work means it will be useful in many future decision-making applications.

Chapter 5 introduced a more efficient way of performing integration within RMHMC that is computationally cheaper in terms of in the number of higher order derivatives required. This yielded a 2 to 3 times speed-up over the previously-used implicit integration scheme. In Chapter 6, these ideas culminated in a new application of RMHMC to BNNs that results in superior uncertainty quantification over variational methods. This was demonstrated by showing that S3HMC was able to appropriately determine inconsistencies between the training and test data by reporting high uncertainty where previous competing techniques failed. Importantly, this ability to identify out-of-sample data points also yielded a similar accuracy performance over the test data when comparing to the previous approaches. Finally, to develop these techniques, a Python package has been built that allows others in the community to continue building on this work. The flexibility of this package, as well as the fact that it is specifically designed to sample from neural network models, makes it possible for others to experiment with new integration schemes and different metric tensors.

Finally, Chapter 7 proposed a new loss function for a BNN that was motivated by a real-world astrophysical example. This resulted in the first study that used BNNs for atmospheric retrieval, whereby a novel formulation of the loss function enabled us to retrieve atmospheric parameters of a real exoplanet orbiting a star that is around 1,400 light years away [Caltech, 2019].

Bibliography

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

S. Aigrain, H. Parviainen, S. Roberts, S. Reece, and T. Evans. Robust, open-source removal of systematics in Kepler data. *Monthly Notices of the Royal Astronomical Society*, 471(1):759–769, 2017.

B. J. Alder and T. E. Wainwright. Studies in molecular dynamics. I. General method. *The Journal of Chemical Physics*, 31(2):459–466, 1959.

M. A. Alvarez, L. Rosasco, N. D. Lawrence, et al. Kernels for vector-valued functions: A review. *Foundations and Trends in Machine Learning*, 4(3):195–266, 2012.

S.-I. Amari and H. Nagaoka. Methods of information geometry. 2000.

A. Amini, A. Soleimany, S. Karaman, and D. Rus. Spatial Uncertainty Sampling for End-to-End Control. In *Neural Information Processing Systems (NIPS); Bayesian Deep Learning Workshop*, 2017.

J. A. Anderson and E. Rosenfeld. *Talking nets: An oral history of neural networks*. MIT Press, 2000.

S. Arangio and F. Bontempi. Structural health monitoring of a cable-stayed bridge with Bayesian neural networks. *Structure and Infrastructure Engineering*, 11(4): 575–587, 2015.

V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

D. Barber and C. M. Bishop. Ensemble learning in Bayesian neural networks. *Nato ASI Series F Computer and Systems Sciences*, 168:215–238, 1998.

T. Bayes. LII. An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, FRS communicated by Mr. Price, in a letter to John Canton, AMFR S. *Philosophical transactions of the Royal Society of London*, (53):370–418, 1763.

J. O. Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer Science & Business Media, 1985.

M. Betancourt. A general metric for Riemannian manifold Hamiltonian Monte Carlo. In *International Conference on Geometric Science of Information*, pages 327–334. Springer, 2013a.

M. Betancourt. The fundamental incompatibility of scalable Hamiltonian Monte Carlo and naive data subsampling. In *International Conference on Machine Learning*, pages 533–540, 2015.

M. J. Betancourt. Generalizing the no-U-turn sampler to Riemannian manifolds. *arXiv preprint arXiv:1304.1920*, 2013b.

S. Bhattacharya, A. G. C. Ramos, F. Kawsar, N. D. Lane, L. M. Gionta, J. Manidis, G. Silvesti, and M. Vegreville. Monitoring Daily Activities of Multiple Sclerosis Patients with Connected Health Devices. In *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*, pages 666–669. ACM, 2018.

C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

A. Blaas, A. D. Cobb, J.-P. Calliess, and S. J. Roberts. Scalable Bounding of Predictive Uncertainty in Regression Problems with SLAC. In *International Conference on Scalable Uncertainty Management*, pages 373–379. Springer, 2018.

J. K. Blitzstein and J. Hwang. *Introduction to probability*. Chapman and Hall/CRC, 2014.

C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural networks. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37*, pages 1613–1622. JMLR. org, 2015.

W. J. Borucki, D. Koch, G. Basri, N. Batalha, T. Brown, D. Caldwell, J. Caldwell, J. Christensen-Dalsgaard, W. D. Cochran, E. DeVore, et al. Kepler planet-detection mission: introduction and first results. *Science*, 327(5968):977–980, 2010.

L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

G. J. Brostow, J. Fauqueur, and R. Cipolla. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 30(2):88–97, 2009.

A. E. Bryson. A gradient method for optimizing multi-stage allocation processes. In *Proc. Harvard Univ. Symposium on digital computers and their applications*, 1961.

Y. Bulatov. notMNIST data set. http://yaroslavvb.blogspot.com/2011/09/notmnist-dataset.html, 2011. Accessed: 8th September 2019.

Caltech. NASA Exoplanet Archive. https://exoplanetarchive.ipac.caltech.edu/, The California Institute of Technology, under contract with the National Aeronautics and Space Administration under the Exoplanet Exploration Program, 2019. Accessed: 20th September 2019.

P. J. Channell and C. Scovel. Symplectic integration of Hamiltonian systems. *Nonlinearity*, 3(2):231, 1990.

D. Charbonneau, L. E. Allen, S. T. Megeath, G. Torres, R. Alonso, T. M. Brown, R. L. Gilliland, D. W. Latham, G. Mandushev, F. T. O'Donovan, and A. Sozzetti. Detection of Thermal Emission from an Extrasolar Planet. *APJ*, 626:523–529, June 2005. doi: 10.1086/429991.

T. Chen, E. Fox, and C. Guestrin. Stochastic gradient Hamiltonian Monte Carlo. In *International Conference on Machine Learning*, pages 1683–1691, 2014.

F. Chollet et al. Keras. https://github.com/keras-team/keras, 2015.

A. D. Cobb. Exoplanet Detection in Large Astronomical Data Sets. *Master's thesis*, 2015.

A. D. Cobb, R. Everett, A. Markham, and S. J. Roberts. Identifying Sources and Sinks in the Presence of Multiple Agents with Gaussian Process Vector Calculus. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1254–1262. ACM, 2018a.

A. D. Cobb, S. J. Roberts, and Y. Gal. Loss-calibrated approximate inference in Bayesian neural networks. *arXiv preprint arXiv:1805.03901*, 2018b.

A. D. Cobb, A. G. Baydin, I. Kiskin, A. Markham, and S. J. Roberts. Semi-separable hamiltonian monte carlo for inference in bayesian neural networks. *Fourth workshop on Bayesian Deep Learning, NeurIPS*, 2019a.

A. D. Cobb, A. G. Baydin, A. Markham, and S. J. Roberts. Introducing an Explicit Symplectic Integration Scheme for Riemannian Manifold Hamiltonian Monte Carlo. *arXiv preprint arXiv:1910.06243*, 2019b.

A. D. Cobb, M. D. Himes, F. Soboczenski, S. Zorzan, M. D. OBeirne, A. G. Baydin, Y. Gal, S. D. Domagal-Goldman, G. N. Arney, D. Angerhausen, et al. An Ensemble of Bayesian Neural Networks for Exoplanetary Atmospheric Retrieval. *The Astronomical Journal*, 158(1):33, 2019c.

A. Criminisi, J. Shotton, E. Konukoglu, et al. Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Foundations and Trends in Computer Graphics and Vision*, 7 (2–3):81–227, 2012.

A. Damianou and N. Lawrence. Deep Gaussian processes. In *Artificial Intelligence and Statistics*, pages 207–215, 2013.

M. H. DeGroot. *Optimal statistical decisions*, volume 82. John Wiley & Sons, 2005.

D. Deming, S. Seager, L. J. Richardson, and J. Harrington. Infrared radiation from an extrasolar planet. *nat*, 434:740–743, Mar. 2005. doi: 10.1038/nature03507.

J. Denker, D. Schwartz, B. Wittner, S. Solla, R. Howard, L. Jackel, and J. Hopfield. Large automatic learning, rule extraction, and generalization. *Complex Systems*, 1 (5):877–922, 1987.

J. S. Denker and Y. LeCun. Transforming Neural-net Output Levels to Probability Distributions. *Technical Memorandum TM11359-901120-05, AT&T Bell Laboratories, Holmdel NJ 07733*, 1990.

J. S. Denker and Y. LeCun. Transforming neural-net output levels to probability distributions. In *Advances in Neural Information Processing Systems*, pages 853–859, 1991.

L. Devroye. Sample-based non-uniform random variate generation. In *Proceedings of the 18th conference on Winter simulation*, pages 260–265. ACM, 1986.

P. Domingos. Bayesian averaging of classifiers and the overfitting problem. In *International Conference on Machine Learning*, volume 2000, pages 223–230, 2000.

G. Dorta, S. Vicente, L. Agapito, N. D. Campbell, and I. Simpson. Structured Uncertainty Prediction Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5477–5485, 2018.

S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth. Hybrid Monte Carlo. *Physics letters B*, 195(2):216–222, 1987.

A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115, 2017.

R. Everett, A. Cobb, A. Markham, and S. Roberts. Optimising Worlds to Evaluate and Influence Reinforcement Learning Agents. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1943–1945. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

A. Filos, S. Farquhar, A. N. Gomez, T. G. J. Rudner, Z. Kenton, L. Smith, M. Alizadeh, A. de Kroon, and Y. Gal. Benchmarking Bayesian Deep Learning with Diabetic Retinopathy Diagnosis. https://github.com/OATML/bdl-benchmarks, 2019.

C. Fisher and K. Heng. Retrieval analysis of 38 WFC3 transmission spectra and resolution of the normalization degeneracy. *MNRAS*, 481:4698–4727, Dec. 2018.

W. Fruehwirt, A. D. Cobb, M. Mairhofer, L. Weydemann, H. Garn, R. Schmidt, T. Benke, P. Dal-Bianco, G. Ransmayr, M. Waser, et al. Bayesian deep neural networks for low-cost neurophysiological markers of Alzheimer's disease severity. *Machine Learning for Health (ML4H) Workshop, NeurIPS*, 2018.

Y. Gal. Bayesian Deep Learning Workshop. http://bayesiandeeplearning.org, 2016a. Accessed: 8th September 2019.

Y. Gal. *Uncertainty in deep learning*. PhD thesis, University of Cambridge, 2016b.

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

Y. Gal and L. Smith. Sufficient conditions for idealised models to have no adversarial examples: a theoretical and empirical study with Bayesian neural networks. *arXiv preprint arXiv:1806.00667*, 2018.

Y. Gal, R. McAllister, and C. E. Rasmussen. Improving PILCO with Bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, 2016.

Y. Gal, J. Hron, and A. Kendall. Concrete dropout. In *Advances in Neural Information Processing Systems*, pages 3581–3590, 2017a.

Y. Gal, R. Islam, and Z. Ghahramani. Deep Bayesian active learning with image data. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1183–1192. JMLR. org, 2017b.

M. Girolami and B. Calderhead. Riemann manifold Langevin and Hamiltonian Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(2):123–214, 2011.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

A. Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356, 2011.

S. F. Gull. Developments in maximum entropy data analysis. In *Maximum entropy and Bayesian methods*, pages 53–71. Springer, 1989.

E. Hairer, C. Lubich, and G. Wanner. *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*, volume 31. Springer Science & Business Media, 2006.

W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. 1970.

K. Heng and D. Kitzmann. The theory of transmission spectra revisited: a semi-analytical method for interpreting WFC3 data and an unresolved challenge. *MNRAS*, 470:2972–2981, Sept. 2017.

J. M. Hernández-Lobato and R. Adams. Probabilistic backpropagation for scalable learning of Bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869, 2015.

G. Hinton and D. Van Camp. Keeping neural networks simple by minimizing the description length of the weights. In *in Proc. of the 6th Ann. ACM Conf. on Computational Learning Theory*. Citeseer, 1993.

G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

M. D. Hoffman and A. Gelman. The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15 (1):1593–1623, 2014.

M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

N. Houlsby, F. Huszár, Z. Ghahramani, and M. Lengyel. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.

E. T. Jaynes. *Probability theory: The logic of science*. Cambridge University Press, 2003.

M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.

H. J. Kelley. Gradient theory of optimal flight paths. *ARS Journal*, 30(10):947–954, 1960.

A. Kendall and Y. Gal. What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision? *Advances in Neural Information Processing Systems*, 2017.

A. Kendall, V. Badrinarayanan, and R. Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv preprint arXiv:1511.02680*, 2015.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589, 2014.

Y. Konrad. Keras-SegNet-Basic. https://github.com/0bserver07/Keras-SegNet-Basic, 2016.

L. Kreidberg. *Exoplanet Atmosphere Measurements from Transmission Spectroscopy and Other Planet Star Combined Light Observations*, page 100. Springer International Publishing, 2017. doi: 10.1007/978-3-319-30648-3_100-1.

L. Kreidberg, M. R. Line, J. L. Bean, K. B. Stevenson, J.-M. Désert, N. Madhusudhan, J. J. Fortney, J. K. Barstow, G. W. Henry, M. H. Williamson, and A. P. Showman. A Detection of Water in the Transmission Spectrum of the Hot Jupiter WASP-12b and Implications for Its Atmospheric Composition. *APJ*, 814:66, Nov. 2015. doi: 10.1088/0004-637X/814/1/66.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

D. Krueger, C.-W. Huang, R. Islam, R. Turner, A. Lacoste, and A. Courville. Bayesian Hypernetworks. *STAT*, 1050:13, 2017.

S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

T. Kuśmierczyk, J. Sakaya, and A. Klami. Correcting Predictions for Approximate Bayesian Inference. *arXiv preprint arXiv:1909.04919*, 2019a.

T. Kuśmierczyk, J. Sakaya, and A. Klami. Variational Bayesian Decision-making for Continuous Utilities. *arXiv preprint arXiv:1902.00792*, 2019b.

S. Lacoste-Julien, F. Huszár, and Z. Ghahramani. Approximate inference for the loss-calibrated Bayesian. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 416–424, 2011.

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

S. Lan, V. Stathopoulos, B. Shahbaba, and M. Girolami. Lagrangian Dynamical Monte Carlo. *arXiv preprint arXiv:1211.3759*, 2012.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

C. Leibig, V. Allken, M. S. Ayhan, P. Berens, and S. Wahl. Leveraging uncertainty information from deep neural networks for disease detection. *Scientific reports*, 7 (1):17816, 2017.

B. Leimkuhler and S. Reich. *Simulating Hamiltonian dynamics*, volume 14. Cambridge University Press, 2004.

M. R. Line, H. Knutson, A. S. Wolf, and Y. L. Yung. A Systematic Retrieval Analysis of Secondary Eclipse Spectra. II. A Uniform Analysis of Nine Planets and their C to O Ratios. *APJ*, 783:70, Mar. 2014. doi: 10.1088/0004-637X/783/2/70.

C. Louizos and M. Welling. Multiplicative normalizing flows for variational Bayesian neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2218–2227. JMLR. org, 2017.

J. Luo, X. Wu, G. Huang, and F. Liu. Explicit Symplectic-like Integrators with Midpoint Permutations for Spinning Compact Binaries. *The Astrophysical Journal*, 834(1):64, 2017.

D. J. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, 1992.

D. J. MacKay. Developments in probabilistic modelling with neural networksensemble learning. In *Neural Networks: Artificial Intelligence and Industrial Applications*, pages 191–198. Springer, 1995.

N. Madhusudhan. Atmospheric Retrieval of Exoplanets. *ArXiv e-prints*, Aug. 2018.

N. Madhusudhan and S. Seager. A Temperature and Abundance Retrieval Method for Exoplanet Atmospheres. *APJ*, 707:24–39, Dec. 2009. doi: 10.1088/0004-637X/707/1/24.

N. Madhusudhan and S. Seager. On the Inference of Thermal Inversions in Hot Jupiter Atmospheres. *APJ*, 725:261–274, Dec. 2010. doi: 10.1088/0004-637X/725/1/261.

P. Márquez-Neila, C. Fisher, R. Sznitman, and K. Heng. Supervised machine learning for analysing spectra of exoplanetary atmospheres. *Nature Astronomy*, June 2018. doi: 10.1038/s41550-018-0504-2.

M. Mayor and D. Queloz. A Jupiter-mass companion to a solar-type star. *Nature*, 378(6555):355, 1995.

W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

T. P. Minka. Expectation propagation for approximate Bayesian inference. In *Proceedings of the Seventeenth conference on Uncertainty in Artificial Intelligence*, pages 362–369. Morgan Kaufmann Publishers Inc., 2001.

M. Minsky and A. P. Seymour. *Perceptrons; an introduction to computational geometry*. MIT Press, Cambridge, Mass, 1969. ISBN 9780262130431.

E. Mizutani, S. E. Dreyfus, and K. Nishio. On derivation of MLP backpropagation from the Kelley-Bryson optimal-control gradient formula and its application. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 2, pages 167–172. IEEE, 2000.

R. M. Neal. *Bayesian Learning for Neural Networks*. PhD thesis, University of Toronto, 1995.

R. M. Neal et al. Slice sampling. *The Annals of Statistics*, 31(3):705–767, 2003.

R. M. Neal et al. MCMC using Hamiltonian dynamics. *Handbook of Markov chain Monte Carlo*, 2(11):2, 2011.

A. Y. Ng, S. J. Russell, et al. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, volume 1, page 2, 2000.

A. Nguyen, J. Yosinski, and J. Clune. Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 427–436, 2015.

M. Opper and C. Archambeau. The variational Gaussian approximation revisited. *Neural computation*, 21(3):786–792, 2009.

M. Oreshenko, B. Lavie, S. L. Grimm, S.-M. Tsai, M. Malik, B.-O. Demory, C. Mordasini, Y. Alibert, W. Benz, S. P. Quanz, R. Trotta, and K. Heng. Retrieval Analysis of the Emission Spectrum of WASP-12b: Sensitivity of Outcomes to Prior Assumptions and Implications for Formation History. *APJL*, 847:L3, Sept. 2017. doi: 10.3847/2041-8213/aa8acf.

S. Panchapagesan, M. Sun, A. Khare, S. Matsoukas, A. Mandal, B. Hoffmeister, and S. Vitaladevuni. Multi-Task Learning and Weighted Cross-Entropy for DNN-Based Keyword Spotting. In *Interspeech*, pages 760–764, 2016.

G. Papamakarios, T. Pavlakou, and I. Murray. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pages 2338–2347, 2017.

D. B. Parker. Learning logic. Invention report S81-64, File 1, Office of Technology Licensing. *October, Stanford University*, 1982.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic Differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017a.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. PyTorch MNIST example. https://github.com/pytorch/examples/blob/master/mnist/main.py, 2017b. Accessed: 8th September 2019.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Courna-

peau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

P. Pihajoki. Explicit methods in extended phase space for inseparable Hamiltonian problems. *Celestial Mechanics and Dynamical Astronomy*, 121(3):211–231, 2015.

H. Poincaré. Les Méthodes nouvelles de la Mécanique Céleste. *Paris, Grauthier-Viltars*, 3:46, 1899.

L. Raynal, J.-M. Marin, P. Pudlo, M. Ribatet, C. P. Robert, and A. Estoup. ABC random forests for Bayesian parameter inference. *Bioinformatics*, 35(10):1720–1728, 2018.

D. Rezende and S. Mohamed. Variational Inference with Normalizing Flows. In *International Conference on Machine Learning*, pages 1530–1538, 2015.

D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In *International Conference on Machine Learning*, pages 1278–1286, 2014.

H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.

C. Robert and G. Casella. *Monte Carlo statistical methods*. Springer Science & Business Media, 2013.

G. O. Roberts and O. Stramer. Langevin diffusions and Metropolis-Hastings algorithms. *Methodology and computing in applied probability*, 4(4):337–357, 2002.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.

F. Rosenblatt. Principles of neurodynamics: perceptions and the theory of brain mechanisms. 1962.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.

B. Shahbaba, S. Lan, W. O. Johnson, and R. M. Neal. Split Hamiltonian Monte Carlo. *Statistics and Computing*, 24(3):339–349, 2014.

C. E. Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

J. Skilling. Nested Sampling. In R. Fischer, R. Preuss, and U. V. Toussaint, editors, *American Institute of Physics Conference Series*, volume 735 of *American Institute of Physics Conference Series*, pages 395–405, Nov. 2004. doi: 10.1063/1.1835238.

J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.

J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, and R. Adams. Scalable Bayesian optimization using deep neural networks. In *International Conference on Machine Learning*, pages 2171–2180, 2015.

J. T. Springenberg, A. Klein, S. Falkner, and F. Hutter. Bayesian optimization with robust Bayesian neural networks. In *Advances in Neural Information Processing Systems*, pages 4134–4142, 2016.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

S. M. Stigler. Who discovered Bayes's theorem? *The American Statistician*, 37(4a): 290–296, 1983.

G. Strang. On the construction and comparison of difference schemes. *SIAM Journal on Numerical Analysis*, 5(3):506–517, 1968.

M. Tao. Explicit symplectic approximation of nonseparable Hamiltonians: Algorithm and long time performance. *Physical Review E*, 94(4):043303, 2016a.

M. Tao. Explicit high-order symplectic integrators for charged particles in general electromagnetic fields. *Journal of Computational Physics*, 327:245–251, 2016b.

C. ter Braak. A Markov chain Monte Carlo version of the genetic algorithm differential evolution: easy Bayesian computing for real parameter spaces. *Statistics and Computing*, 16:239–249, 2006. ISSN 0960-3174. URL http://dx.doi.org/10.1007/s11222-006-8769-1.

C. J. F. ter Braak and J. A. Vrugt. Differential evolution Markov chain with snooker updater and fewer chains. *Statistics and Computing*, 18(4):435–446, 2008. ISSN 0960-3174. doi: 10.1007/s11222-008-9104-9. URL http://dx.doi.org/10.1007/s11222-008-9104-9.

N. Tishby, E. Levin, and S. A. Solla. Consistent inference of probabilities in layered networks: Predictions and generalization. In *International Joint Conference on Neural Networks*, volume 2, pages 403–409, 1989.

I. P. Waldmann, M. Rocchetto, G. Tinetti, E. J. Barton, S. N. Yurchenko, and J. Ten-

nyson. Tau-REx II: Retrieval of Emission Spectra. *APJ*, 813:13, Nov. 2015. doi: 10.1088/0004-637X/813/1/13.

K. Wang, G. Pleiss, J. Gardner, S. Tyree, K. Q. Weinberger, and A. G. Wilson. Exact Gaussian processes on a million data points. In *Advances in Neural Information Processing Systems*, pages 14622–14632, 2019.

Z. Wang, S. Mohamed, and N. Freitas. Adaptive Hamiltonian and Riemann Manifold Monte Carlo. In *International Conference on Machine Learning*, pages 1462–1470, 2013.

M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 681–688, 2011.

P. Werbos. Beyond Regression: New tools for Prediction and Analysis in the Behavioral Sciences. *Ph.D. thesis, Harvard University*, 1974.

B. Widrow and M. E. Hoff. Adaptive Switching Circuits. In *1960 IRE WESCON Convention Record, Part 4*, pages 96–104, New York, 1960. IRE.

B. Widrow and M. A. Lehr. 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415–1442, 1990.

C. K. Williams. Computing with infinite networks. In *Advances in Neural Information Processing Systems*, pages 295–301, 1997.

C. K. Williams and C. E. Rasmussen. *Gaussian Processes for Machine Learning*, volume 2. MIT press Cambridge, MA, 2006.

R. A. Yeh, C. Chen, T. Yian Lim, A. G. Schwing, M. Hasegawa-Johnson, and M. N. Do. Semantic image inpainting with deep generative models. In *Proceedings of the*

IEEE Conference on Computer Vision and Pattern Recognition, pages 5485–5493, 2017.

H. Yoshida. Construction of higher order symplectic integrators. *Physics letters A*, 150(5-7):262–268, 1990.

R. Zhang, Y. Wang, Y. He, J. Xiao, J. Liu, H. Qin, and Y. Tang. Explicit symplectic algorithms based on generating functions for relativistic charged particle dynamics in time-dependent electromagnetic field. *Physics of Plasmas*, 25(2):022117, 2018.

Y. Zhang and C. Sutton. Semi-separable Hamiltonian Monte Carlo for inference in Bayesian hierarchical models. In *Advances in Neural Information Processing Systems*, pages 10–18, 2014.

T. Zingales and I. P. Waldmann. ExoGAN: Retrieving Exoplanetary Atmospheres Using Deep Convolutional Generative Adversarial Networks. *ArXiv e-prints*, June 2018.